Probabilistic Agent Based Modelling: Mathematical foundations

Daniel Tang
Deselby research institute

Abstract

In order to do modelling well we need to do calibration, data assimilation and uncertainty quantification properly. All of these problems can be posed in terms of probabilistic inference. At present, tools to do probabilistic inference on agent based models are lacking. In this paper I introduce the idea of an agent based dynamical system and develop a mathematical foundation for thinking about and working with probabilistic inference in these systems.

1 Introduction: The problem

In this paper we will consider the following problem: Suppose we have a complex system that can be described in terms of interacting agents. Suppose also that we have some (possibly incomplete) knowledge of the behaviour of those agents. Suppose we also have a set of real-world, noisy observations of the aggreagate behaviour of the system as a whole and are able to define an observation operator for each observation type that goes from the state of the system to the observation. We would like to know something about a (hitherto unobserved) observable at some time which may be in the past, present or future, given our knowledge of the behaviour of the agents, our observations and our observation operators. That something may be an expectation value, a modal value, or a distribution chosen from a family that minimises the KL divergence. (or the modal trajectory in time)

1.1 Defining agents

We will proceed by specifying our knowledge of the behaviour of an agent as a computer program written from the point of view of an agent. In order to express any incompleteness in our knowledge of the behaviour, we suppose there is a set of global paramaters, which we do not know but which we have a prior distribution for, that may infulence an agent's behaviour. In addition, we may make the agent's behaviour stochastic in order to express our uncertainty in the agent's behaviour. By doing this, we can track the effect of our uncertainty in the agent's behaviour on the uncertainty in the aggregate behaviour of the system.

1.2 Defining observation operators

Similarly, we will define observation operators as computer programs that take a state of the system as input and return an observation (i.e. a real number or vector of reals). The observation operator will itself be stochastic in order to represent the noise in the observation.

We can connect our data to the observation operators by storing the oputput of an observation operator in a variable and allowing an `observe(x,v)` command which should be interpreted as "we observe the variable `x` to have value `v`".

In this way, our whole problem can be specified in a single computer program that looks very much like a normal simulation which, if executed, would generate an instance of our data and an instance of the observable we're interested in. The values generated would probably not match our actual data, but there is a positive, if small, probability that they would match.

Our problem can now be cast as one of doing inference on this computer program. That is, finding the expectation value of the observable of interest given that the generated data matches our actual observations.

2 Agent based dynamical systems

We introduce the concept of an *agent based dynamical system* which consists of a set of interacting *agents*. Each agent has a state and can interact with other agents. At any given time, t, each agent has a propensity, r, to perform an action such that, in the infinitesimal slice of time, $[t, t+dt]$, there is a probability rdt that the agent performs the action. The only actions an agent can perform are to remove an agent and to create a new agent. Surprisingly, this will be enough to express all agent behaviours.

3 Object oriented implementation of an agent based dynamical system

In order to implement an agent based dynamical system in an object oriented programming language, we have a class, `ABM`, which is templated on the agent state type `T`. This class represents a probability distribution over ABM states and has three methods:

`add(s)`	add an agent in state `s` to the ABM and return the modified ABM
`foreach(a)`	execute action `a` on every agent and return the result
`observe(o,v)`	observe that the observable `o` has value `v` and return the resulting `ABM`

An `action` is a lambda function that takes an agent state (the state of the agent that is performing the action) and an `ABM` (the environment the agent finds itself in) and returns an `ABM` that is the result of performing the action. When passed to a `foreach` method, the `ABM` passed to the action is the `ABM` with the current agent removed.

An `observable<T>` is a lambda function that takes an `ABM` and returns an observation of type `T`.

A `behaviour` is a set of `<rate, action>` pairs, where `rate` is a lambda function that takes an agent state and the `ABM` and returns a floating point number giving the rate of the behaviour.

Time in the model can be moved forward using the `Behaviour::integrateTime(abm, Dt)` method, which integrates the abm forward in thim by where `Dt` units of time and returns the result.

4 A probabilistic interpretation of an `ABM`

A probability distribution over the states of an agent based model is formally a member of a symmetric Fock space on the state of the agents, T, which we'll write as $\mathscr{F}(T)$. Surprisingly, we can express everything we need to using just two operators on this space; the *annihilation operator* a_k and the *creation operator* $a_k^\dagger$.

The creation operator, $a_k^\dagger$, represents the addition of an agent in state k to every state in the probability distribution. By definition, the probability of the ABM being empty after the action of a creation operator is zero.

Creation operators always commute with other creation operators

$$a_k^\dagger a_q^\dagger = a_q^\dagger a_k^\dagger \tag{1}$$

We define the annihilation operator a_k as the operator that has the following properties

$$a_k a_q = a_q a_k \tag{2}$$

$$a_k a_q^\dagger - a_q^\dagger a_k = \delta_{kq} \tag{3}$$

$$a_k \mathbf{1} = \mathbf{0} \tag{4}$$

where δ_{kq} is the Kronecker delta function and we use $\mathbf{1}$ to represent the ABM probability distribution representing an empty ABM with probability 1 and $\mathbf{0}$ to represent the distribution with all states having zero probability.

For a given (non-probabilistic) ABM, we define the *occupation number* of agent state s to be the number of agents in the ABM that are in that state. The annihilation operator a_s multiplies the probability of each ABM state by the occupation number of s and removes one agent in state s.

We can see that this behaviour follows from the properties above using a recursive proof. Suppose that $a_k a_k^\dagger S = nS$ then, from equation 3

$$a_k a_k^\dagger a_k^\dagger S = (1 + a_k^\dagger a_k) a_k^\dagger S = (n+1) a_k^\dagger S$$

We can start the recursion using equations 3 and 4

$$a_k a_k^\dagger \mathbf{1} = (1 + a_k^\dagger a_k)\mathbf{1} = \mathbf{1}$$

4.1 Single agent

We start with the simplest possible agent based dynamical system: a system which is either empty or there is a single, stateless agent. Suppose that at time $t = 0$ the agent is definitely present and between time t and $t + dt$ the agent leaves the system with probability rdt. Here's the program that represents this system

```
abm = ABM()
abm.add(void)

b = Behaviour()
```

```
b.add({1.0}, (state, abm) -> {
  return abm
})
return b.integrateTime(abm, T)
```

remember that the ABM passed to the action in the bahaviour is the ABM minus the agent whose behaviour we're considering. So in this case, it's the empty ABM and returning this is equivalent to the agent removing itself from the system.

Starting from the first line of the program, we create a new ABM which is definitely empty, so is in the state $\mathbf{1}$. We then add an agent, after which the ABM will be in state $a^\dagger\mathbf{1}$. Next we create a behaviour. The action is just the "do nothing" action, which we can think of as the identity operator, 1. We'll think of a behaviour as an operator that takes a probability distribution over ABMs and returns a rate of change of that probability distribution.

———————— TODO ————-

Given an empty state and an agent that is currently outside of the system that has a propensity, r, to enter the system, then in time slice $[t,t+dt]$ there is a probability rdt that the state will switch from empty to having an agent in it. So in time dt the probability of the empty state goes down by rdt and the probability of there being an agent goes up by rdt. So the rate of change of the state is givcen by

$$H_{a^\dagger} = \frac{\partial}{dt} = (a^\dagger - 1)r$$

Similarly, for the annihilation operator. If we start with an agent in the system, the probability of there being an agent decreases by rdt while the probability of the state being empty increases by the same amount. We can achieve this with the operator

$$H_a = \frac{\partial}{\partial t} = (a - a^\dagger a)r$$

We'll call $H_{a^\dagger}$ and H_a the *Hamiltonians* for their respective operators. Again, we'll see later that by choosing this form for the Hamiltonians we make things very elegant when we come to generalise to many agents.

Once we have the Hamiltonian for a probability distribution, the time evolution of that probability distribution is completely defined by the equation

$$\frac{\partial P}{\partial t} = HP$$

which is the probabilistic dynamics' analogue of Schrödinger's equation. This equation has a very general solution

$$P(t) = e^{tH}P(0)$$

where the exponent of an operator is defined as

$$e^{tH} = \sum_{k=0}^{\infty} \frac{(tH)^k}{k!}$$

and eponentiation of an operator is just repeated application (taking care to respect the order of non-commutative operators).

Going back now to the agent that has propensity r to annihilate itself, the equation of motion is

$$\frac{\partial P}{\partial t} = H_a P = \left(\frac{\partial}{\partial a^\dagger} - a^\dagger \frac{\partial}{\partial a^\dagger}\right) rP$$

Solving this for $P(0) = a^\dagger$ gives

$$P(t) = e^{rt((1-a^\dagger)\frac{\partial}{\partial a^\dagger})} a^\dagger$$

$$= \sum_{k=0}^{\infty} \frac{\left(rt(1-a^\dagger)\frac{\partial}{\partial a^\dagger}\right)^k}{k!} a^\dagger$$

$$= \sum_{k=0}^{\infty} \frac{(-rt)^k \left((a^\dagger - 1)\frac{\partial}{\partial a^\dagger}\right)^k}{k!} a^\dagger$$

but since $\frac{\partial^n}{\partial a^{\dagger n}} = 0$ for $n > 1$

$$\left((a^\dagger - 1)\frac{\partial}{\partial a^\dagger}\right)^k = (a^\dagger - 1)\frac{\partial}{\partial a^\dagger}$$

for $k > 0$, so

$$P(t) = 1 + \sum_{k=0}^{\infty} \frac{(-rt)^k}{k!}(a^\dagger - 1)$$

where the initial 1 accounts for the case when $k = 0$. We can now turn the sum back into an exponent to give

$$P(t) = (1 - e^{-rt}) + e^{-rt}a^\dagger$$

So the probability that the agent exists reduces exponentially as time progresses. This is exactly what we would expect.

4.2 Many agents

We can now show how our operators extend very elegantly to systems with more than one agent. In this case, we represent the state that there are definitely n agents in the system as $P = a^{\dagger n}$. A probability distribution for a set of agents can therefore represented by a polynomial

$$P = \sum_{n=0}^{\infty} p_n a^{\dagger n}$$

where p_n is the probability that there are n agents, and

$$\sum_{n=0}^{\infty} p_n = 1$$

By choosing this representation, we can see immediately that our creation operator works on definite states:

$$a^\dagger a^{\dagger n} = a^{\dagger n+1}$$

Even better, it works on probabilistic states too. For example, suppose there's a 0.5 probability that the system is empty and a 0.5 probability that there's one agent, so $P = 0.5 + 0.5a^\dagger$. In this case

$$a^\dagger(0.5 + 0.5a^\dagger) = 0.5a^\dagger + 0.5a^{\dagger 2}$$

So, after applying the creation operator, there's an equal probability of there being either one or two agents, i.e. the operator simultaneously adds an agent to all possible states.

If we start with an empty system (i.e. $P(0) = 1$) and add agents at a constant rate we'd expect the Hamiltonian to be given by

$$\frac{\partial P}{\partial t} = H_{a^\dagger} P = (a^\dagger - 1) r P$$

This can again be solved analytically:

$$P(t) = e^{tH} = e^{-rt(1-a^\dagger)} = e^{-rt} e^{rta^\dagger}$$

$$= e^{-rt} \sum_k \frac{(rt)^k}{k!} a^{\dagger k}$$

so, the probability of there being k agents (i.e. the coefficient of the k^{th} power of $a^\dagger$) is

$$c_k = \frac{(rt)^k e^{-rt}}{k!}$$

which is just the Poisson distribution, as we would expect.

Now let's try the annihilation operator on $P = 0.5a^\dagger + 0.5a^{\dagger 2}$:

$$a(0.5a^\dagger + 0.5a^{\dagger 2}) = 0.5 + a^\dagger$$

We don't get $0.5 + 0.5a^\dagger$ as you may have expected. Instead, for a distribution $P = \sum_n p_n a^{\dagger n}$ we get

$$a \sum_n p_n a^{\dagger n} = \sum_n n p_n a^{\dagger n-1}$$

which isn't even a valid probability distribution. However, this behaviour can be harnessed to make a very useful operator

$$N = a^\dagger a$$

which in quantum field theory is called the number operator because it multiplies the probability of each state by the number of agents in that state

$$a^\dagger a \sum_n p_n a^{\dagger n} = \sum_n n p_n a^{\dagger n}$$

We can now understand the Hamiltonian of the annihilation operator in a new light

$$H_a = (a - a^\dagger a) r = (a^{\dagger -1} - 1) N r$$

where $a^{\dagger -1}$ is the inverse of the creation operator, which we'll call the agent-based annihilation operator. In this form, we can understand the $a^{\dagger -1}$ term as the transition operator for a single agent's propensity. The number operator N has the effect of applying this operator to all the agents in the system, as if we'd executed a `foreach` loop in a computer program.

This works for any operator that represents the transition function for an agent's propensity. Because it's so useful, well define a higher order function which takes an agent action T and returns the Hamiltonian that applies this propensity to all agents in a probabilistic state

$$\mathscr{H}_r(T) = (T - 1) N r$$

we'll call this the *foreach* operator.

To illustrate this, suppose we start with n agents and each agent has a propensity, r, to annihilate itself. In pseudocode this may look something like

```
foreach(a in S) {
  if(random() < r*dt) delete(a)
}
```

We can create the Hamiltonian for this

$$\frac{\partial}{\partial t} = \mathscr{H}(a^{\dagger -1}) = (a^{\dagger -1} - 1) N r$$

Expanding this gives

$$H = (a^{\dagger -1} - 1) a^\dagger a r = (a - a^\dagger a) r$$

which is exactly the Hamiltonian from the single agent case, but now we understand why it has this form and why it will also work in the case of many agents.

The only difference here is the boundary condition. Instead of $S(0) = a^\dagger$ as in the single agent case, we now have $P(0) = a^{\dagger n}$. The solution to this is

$$P(t) = \sum_m \binom{n}{m} e^{-rtm} (1 - e^{-rt})^{n-m} a^{\dagger m}$$

The coefficients of $a^\dagger$ (i.e. the probabilities that there are m agents) make up a Binomial distribution, which, again, is what we would expect.

The above examples, although very simple, demonstrate how we can define a Hamiltonian operator in terms of the behaviour of a single agent then apply it to each agent in a system and ultimately describe the aggregate dynamics of a probability distribution over the whole system.

4.3 Agent states

So far, the agents we've been considering haven't had any internal state. We'll start simple again by giving agents a binary state which can be either 0 or 1. We'll represent an agent in state 0 as $a_0^\dagger$ and an agent in state 1 as $a_1^\dagger$. A system with n agents in state $a_0^\dagger$ and m agents in state $a_1^\dagger$ will be represented as $a_0^{\dagger n} a_1^{\dagger m}$, so a probability distribution over all states is again a polynomial, but this time in two variables. We define annihilation operators for each agent state:

$$a_0 = \frac{\partial}{\partial a_0^\dagger}$$

$$a_1 = \frac{\partial}{\partial a_1^\dagger}$$

This induces two number operators $N_0 = a_0^\dagger a_0$ and $N_1 = a_1^\dagger a_1$ and two foreach operators $\mathscr{H}_{0r}(T) = (T-1)N_0 r$ and $\mathscr{H}_{1r} = (T-1)N_1 r$ which loop over agents in state 0 and 1 respectively.

A change of state from x to y can be represented as an agent in state x annihilating itself while simultaneously creating a new agent in state y, $a_y^\dagger a_x^{\dagger -1}$.

Let's illustrate this with a system of n agents who each have a propensity r to flip to the opposite state. In pseudocode we want something like

```
S0  = S
foreach(x in S0 suchthat S0.state(x)==0) {
  if(random() < r*dt) S.state(x) = 1
}
foreach(x in S0 suchthat S0.state(x)==1) {
  if(random() < r*dt) S.state(x) = 0
}
```

This can be expressed as a Hamiltonian

$$H = r\mathscr{H}_{0r}(a_1^\dagger a_0^{\dagger -1}) + r\mathscr{H}_{1r}(a_0^\dagger a_1^{\dagger -1})$$

expanding this gives

$$H = r\left((a_1^\dagger - a_0^\dagger)\frac{\partial}{\partial a_0^\dagger} + (a_0^\dagger - a_1^\dagger)\frac{\partial}{\partial a_1^\dagger}\right)$$

If we suppose that at time $t = 0$ all agents are in state $a_1^\dagger$ then we can solve the governing equation $\frac{\partial P}{\partial t} = HP$ to give

$$P(t) = 2^{-n}\sum_{m=0}^{n}\binom{n}{m}(1-e^{-2rt})^{n-m}(1+e^{-2rt})^m a_0^{\dagger n-m} a_1^{\dagger m}$$

[TODO: add proof]

We can represent an agent with more than one variable in it's state quite naturally by allowing vectors in the subscripts of $a^\dagger$. So, for example, an agent with two binary values, a and b could be referred to as $a_{[a,b]}^\dagger$.

4.4 Interacting agents

Now let's get agents to interact. Suppose we have a compartmental model of infectious disease spread. Suppose there's a population of n people. Each person can be one of suseptible, infected or recovered. Infected people have a propensity, β, to meet and infect a suseptible person, in which case the suseptible person becomes infected. Also, infected people have a propensity, γ to recover.

In order to represent the interaction of infected people with suseptible, we want a Hamiltonian that would be equivalent to pseudocode something like

```
foreach(x in S0 suchthat
        S0.state(x)==Suseptible) {
  foreach(y in S0 suchthat
          S0.state(y) == Infected) {
    if(random() < beta*dt)
      S.state(x) = Infected
  }
}
```

It turns out that the number operator can be used in a quite natural way to represent this nested `foreach` operator.

If we let $a_S^\dagger$, $a_I^\dagger$ and $a_R^\dagger$ represent suseptible, infected and recovered people respectively, then the Hamiltonian we're after is just

$$H = \mathscr{H}_{S\beta}(a_I^\dagger a_S^{\dagger -1} N_I) = (a_I^\dagger a_S^{\dagger -1} N_I - 1)N_S\beta$$

so, the Hamiltonian for the whole model is just

$$H = \mathscr{H}_{S\beta}(a_I^\dagger a_S^{\dagger -1} N_I) + \mathscr{H}_{I\gamma}(a_R^\dagger a_I^{\dagger -1})$$

and the equation of motion for the probability distribution over states is, as always

$$\frac{\partial P}{\partial t} = HP$$

Although this is a differential equation, it's worth pointing out explicitly that it is representing discrete interactions between agents. So, it captures things like disease eradication when the number of infected agents is small, unlike the standard differential version of the SIR model, which is only accurate for large numbers of agents.

Notice also that the different number operators are able to single out agents based on their state. If we allow higher-order operators, we can extend this idea to any predicate on the state of an agent, so that we can loop over any describable subset of agents. For example, suppose agents are located on a 2D grid and each agent has a state $\mathbf{v} = (x, y)$ giving the agent's position on that grid. We could express the program

```
foreach(a in S0) {
  foreach(b in S0 suchthat
    norm(a.v-b.v)<4) {
    call F(a.v,b.v)
  }
}
```

as

$$H = \sum_{(a_v \in S)} \sum_{(b_v \in \{v \mid |a_v - v| < 4\})} \mathscr{H}_{a_v}(F(a_v, b_v) N_{b_v})$$

4.5 Arithmetic

If an agent has numerical variables in its state, we can represent arithmetic on those variables with a change of state operation. For example, if an agent has two integers, a and b in its state, we can represent the operation $a = a + b$ as

$$a^{\dagger}_{[a+b,b]} a^{\dagger -1}_{[a,b]}$$

and so the Hamiltonian for an agent that has a unit propensity to perform this operation is

$$H = \mathscr{H}_{[a,b]}(a^{\dagger}_{[a+b,b]} a^{\dagger -1}_{[a,b]}) = (a^{\dagger}_{[a+b,b]} a^{\dagger -1}_{[a,b]} - 1) N_{[a,b]}$$

The same method can be used to apply any operator on any number of variables.

4.6 Continuous states and the uncertainty principle

So far, we've only considered agents with a finite number of discrete states. We may want to endow the agent with a real number as part of its state, in which case the agent can be in any one of an uncountable infinity of states and we'd need a polynomial in an infinite number of variables to represent this.

The idea of using a polynomial to represent a probability distribution carries over to the infinitesimal case quite naturally. The transition is made easier if we think of the coefficients as a function from monomials to coefficient values. For example, for a monovariate polynomial

$$P = \sum_n c_n a^{\dagger n} = \sum_n f(a^{\dagger n}) a^{\dagger n}$$

In this format we can go over to the infinitesimal limit by replacing the monomials with sets of real numbers. The coefficients can now be represented as a function from sets of real numbers to probability densities (i.e. a PDF over $\mathscr{P}(\mathbb{R})$, the power set of the real numbers). The probability distribution now becomes

$$P = \int_{\mathbf{z} \in \mathscr{P}(\mathbb{R})} f(\mathbf{z})$$

Note that because we chose sets rather than bags to represent monomials, we can now no longer represent multiple agents in the same state, but because each state is infinitesimally small the probability of finding more than one agent in a state is of second order.

Multiplication of a state by $\mathbf{z}_n = \left\{a^{\dagger}_n\right\}$ becomes

$$\mathbf{z_n} P \equiv \int_{\mathbf{z} \in \mathscr{P}(\mathbb{R})} f(\mathbf{z} \setminus \mathbf{z_n})$$

while differentiation becomes

$$\frac{\partial P}{\partial \mathbf{z_n}} \equiv \int_{\mathbf{z} \in \mathscr{P}(\mathbb{R})} f(\mathbf{z} \cup \left\{a^{\dagger}_n\right\})$$

[TODO: make these integrals more formally correct. Define as differential ring/algebra?]

Consider an agent with a 1-dimensional, real valued state. We'd naturally represent a probability distribution over this value as a density function $P : \mathbb{R} \to \mathbb{R}$. For such a density function, we define the information of the function to be

$$I = \int_{-\infty}^{\infty} P(x) \log(P(x)) dx$$

We can escape the infinitude of variables by noting that in all realistic scenarios, the amount of information we have about an agent is finite. That is, whatever observations we make of a variable, there always remains some inherent uncertainty. This is analogous to Heisenberg's uncertainty principle: the more precicely we measure the location of a particle, the more momentum it must have. In our case, momentum is replaced with information and we are only interested in cases of finite information.

Given this, we can represent any probability density that has finite information as a probability mass function over a finite number of variables. For example, if the PDF has a compact support, a convenient representation would be

as a Bernstein polynomial, a Deselby polynomial or the spectral modes.

[TODO: find good bases that make the mapping simple from agent to aggregate behaviour]

4.7 Mixed real/integer states

We can create a fully general system state that can deal with mixed real and integer agent states by defining the state to be a function from bags of state vectors to real coefficients.

$$P = \int_{\Psi \in \mathscr{P}(\psi)} f(\Psi)$$

5 Making observations and doing inference

If we have a prior $P(S)$ over system states and an observation operator O that goes from a probability distribution over a system state to a probability distribution over the reals and we observe that the obervation operator has value o then we can define the posterior

$$P(S|OS = o) \propto P(OS = o|S)P(S)$$

i,e, we multiply the prior probability distribution over S by the likelihood of the observation. We can represent likelihoods as polynomials in a similar way to probability distributions

$$P(o|.) = \sum_n P(o|a^{\dagger n})a^{\dagger n}$$

However, multiplication of two probability distributions is not the same as polynomial multiplication. Instead, it is multiplication of the coefficients:

$$\left(\sum_n c_n a^{\dagger n}\right) \times \left(\sum_m d_m a^{\dagger m}\right) = \sum_n c_n d_n a^{\dagger n}$$

Going back to the example of a single agent that has a unit propensity to annihilate itself. If we start with a prior that there's a 0.5 probability that the agent exists, then observe that it doesn't exist at time t, what's the posterior probability that it existed at time $t = 0$? The equation of motion is

$$\frac{\partial P}{\partial t} = HP = \left(\frac{\partial}{\partial a^\dagger} - a^\dagger \frac{\partial}{\partial a^\dagger}\right) P$$

Integrating this for the case $P(0) = (1-p) + pa^\dagger$

$$P(t) = e^{\frac{\partial}{\partial a^\dagger} - a^\dagger \frac{\partial}{\partial a^\dagger}} S$$

$$= 1 - pe^{-rt} + e^{-rt} pa^\dagger$$

So,

$$P(o|.) = 1 + (1 - e^{-rt})a^\dagger$$

So

$$P(.|o) \propto (1-p) + (1-e^{-rt})pa^\dagger$$

normalising

$$\int P(.|o)dp = (1 - 0.5e^{-rt})$$

$$P(.|o) = \frac{(1-p) + (1-e^{-rt})pa^\dagger}{(1-0.5e^{-rt})}$$

So, if the prior $P(0) = 0.5 + 0.5a^\dagger$ then the posterior is given by

$$P(.|o) = \frac{0.5 + 0.5(1-e^{-rt})a^\dagger}{(1-0.5e^{-rt})}$$

6 Numerical methods

So far, we've considered simple systems that have analytic solutions. However, almost all of the systems of interest will not have an analytic solution. We'll now introduce some techniques to help us to numerically approximate solutions.

Because the number of coefficients in the probabilistic state of a system expands exponentially with the number of variables in an agent's state and with the number of agents, for most real world models it isn't feasible to explicitly represent the probabilistic state as a list of coefficients. There are three ways around this. We can represent the state symbolically, we can approximate the state or we can abstract over the state.

6.1 Symbolic representation of probability distributions

In almost all cases of interest a probability distribution, although very high dimensional and complex, will be the result of a (potentially large but manageable) number of operations. What's more, we can construct these states from the empty set of agents $P = 1$. So, instead of representing a system state as a list of monomial coefficients we can represent it as a list of operations on 1:

$$P = \prod_n O_n 1$$

all of which are constructed from the creation and annihilation operators.

When thought of like this, we never have to explicitly represent a probability distribution as a polynomial or as

any other kind of representation. Instead, all we ever need to deal with is creation and annihilation operators on probability distributions.

We can reduce the necessary behaviour of these operators to an abstract algebra that obeys the following axiomatic rules:

$$a^\dagger 0 = 0 \tag{5}$$

where d_{kq} is the Kronecker delta function.

Using these rules, we can show that everything acts as expected. Starting with the number operator, $a^\dagger a$, we have, from equations 4 and 5

$$a^\dagger a 1 = 0$$

and more generally, if $a^\dagger aS = nS$ then, from equation 3

$$a^\dagger a a^\dagger S = a^\dagger (a^\dagger a + 1)S = (n+1)a^\dagger S$$

so the number operator acts as expected.

Given this we can show that the annihilation operator also acts as expected. Suppose again that $a^\dagger aS = nS$ then

$$aa^\dagger S = (a^\dagger a + 1)S = (n+1)S$$

Equations 2 and 1 ensure that these results extend to multivariate cases.

6.2 State approximation: Projecting to lower dimensional spaces

We can also often approximate a high-dimensional probabilistic state in a lower dimensional space that is more practical to work with.

6.3 State abstraction: Projection to lower dimensional spaces without approximation

7 Integrating over propensities: The path integral

8 Data assimilation