7. Markov chains and the master equation

Last time

- Random walks using Julia objects
- Monte Carlo methods

Goals for today

- Tips and ideas from Problem Set 2
- Exact time evolution: Master equation
- Exact enumeration: Numerical solution of master equation
- Markov chains

Tips from Problem Set 2

Shapes of graphs

- The answer to "how does this behave" is *never* "it is increasing"
- We want to know *how fast*? What *shape* is it
- I.e. $y = f(x)$ *for which* f
- "Right-skewed graph" is useless (some people said it was left-skewed)
- How can we work this out?

Using different scales on a graph

- Firstly: Guess
- Change scales of axes to look for a straight line
- Since then can write down (apparent) relationship
- Use log-scale for y-axis with `yscale=:log10`
- Then data point (x_i, y_i) is shown as $(x_i, \log(y_i))$

Interpreting log scales

- If result is linear relationship on log–linear graph, then

$$\log(y) \simeq a\,x + b$$

  so have **exponential growth / decay**:

$$y \simeq Ce^{a\,x},$$

- If linear on log–log graph then

$$\log(y) \simeq a\ \log(x) + b$$

  so have **power law** or **polynomial** growth / decay:

$$y \simeq Cx^{a}$$

Programming tips:

- If you find yourself writing a piece of code over and over again, **put it in a function**!
- Don’t Repeat Yourself
- “I changed it in the function and then changed it back”
- Never do this: Make it an argument to the function instead

Julia tips:

- Use spaces around `=` and other operators for readability
- Use _ rather than capital letters in function names
- Types have CamelCase
- Prefer array comprehensions if readable
- `sortperm` gives permutation that sorts a vector
- `Iterators.product` and `Iterators.repeated`

Performance

- Use tuples instead of short arrays for performance
- E.g. `rand( (-1, +1) )` vs `rand( [-1, +1] )`
- Each array is allocated on heap; tuples are allocated on stack
- Allocations are reported by `@time` and `@btime` from `BenchmarkTools.jl`

Types for performance

- Performance in Julia relies on **types** being known
- Using untyped data structures like `v = [ ]` and `d = Dict()` *kills* performance
- Typed empty vector: `v = Float64[]`
- Typed empty `Dict`: `d = Dict{Int, Int}()`
- Prefer `v = [x0]` – type is inferred
- Prefer `d = Dict(0 => x0)` – type is inferred

Parametrized types

- E.g. `counts` function from PS2:

```
function counts(v::Vector{T}) where {T}
    d = Dict{T, Int}()
    ...
end
```

- Or don't assume that data is a vector: `julia function counts(data) T = eltype(data) d = Dict{T, Int}() end`
- `eltype` gives element type of a container

Probability distributions

- Recall: **probability distribution** of discrete random variable X is set of probabilities $\mathbb{P}(X = i)$
- So far: calculated by counting discrete occurrences of outcomes of X
- In Monte Carlo simulation, or exactly (PS2 for sum of dice)
- Is there a more general approach to calculating exact probability distributions?

Back to simple random walk in 1D

- Suppose have simple random walker starting at site 0
- Notation: X_t := state (position) at time t
- Notation: $P_i^t := \mathbb{P}(X_t = i) :=$ probability that at state i at time t
- Notation: $\mathbf{P}^t$:= probability distribution at time t
- Know $\mathbf{P}^0$ = prob. dist. of X_0
- What is probability distribution of X_1? Of X_2?

Simple 1D random walk II

- Initial condition:

$$P_i^0 = \delta_i = \begin{cases} 1 & \text{if } i = 0 \\ 0 & \text{otherwise} \end{cases}$$

- **Master equation** (terrible name) gives time evolution of probability distribution:

$$P_i^{t+1} = \tfrac{1}{2} P_{i-1}^t + \tfrac{1}{2} P_{i+1}^t \quad \forall i$$

- **Exact enumeration**: Name for "solve this equation numerically"
- "*Dual*" to Monte Carlo
- Gives result of infinite number of runs, but no individual trajectories

Exact enumeration: Set-up

- Use `Vector` (for states numbered 1 to n)
- e.g. random walk on $1, \ldots, L$, starts at $L/2$:

```
L = 20
T = 10

P₀ = zeros(L)    # P\_0<TAB>
P₀[L ÷ 2] = 1

P = copy(P₀)        # time t
next_P = copy(P₀)   # time t+1
Ps = [copy(P₀)]
```

Exact enumeration: Time evolution

- Time evolution: “‘julia for t in 1:100 for i in 2:L-1 next_P[i] = 0.5 * (P[i-1] + P[i+1]) end

```
    push!(Ps, copy(next_P))
    global P, next_P = next_P, P
end
```
```
- Trick: *swap* `P` and `next_P` – no new memory allocated
```

Exact enumeration II

- Must *take care with boundary conditions*
- Random walk can go arbitrarily far
- So *infinite* number of possible states
- Truncate at finite boundaries
- What happens to probability that reaches boundary in simple random 1D walk?

General exact enumeration

- When can we do this?
- For *any* (discrete-time) **Markov chain**
- **Markov chain**:
  - Stochastic process
  - Discrete state space (set of possible states)
  - Probability of being in state depends only on previous time step
- **Master equation**: Time evolution of probability distribution

Master equation

- **Master equation**: probability at j at $t+1$ is
  - sum over i of (prob. at i) * (prob. to jump $i \to j$):

$$P_j^{t+1} = \sum_i P_i^t \, p(i \to j) \quad \forall j$$

- If you know some linear algebra, you will recognize a **matrix–vector product** here
- In any case, you should take 18.06 Linear Algebra, taught by Prof. Edelman!

"Atmosphere walk"

- Recall "atmosphere" walk from PS2:
- Probability to decrease height is $p > 0.5$
- Reflecting boundary
- E.g. "atmosphere" walker from PS2: truncate at finite height and be careful with reflecting boundary:

$$P_i^{t+1} = p\,P_{i+1}^t + (1-p)\,P_{i-1}^t \quad \forall i \geq 2$$

$$P_1^{t+1} = p\,P_2^t + p\,P_1^t$$

- Need to add extra boundary condition at finite height H

Exact limit distribution

- PS2: "atmosphere" Markov chain (random walk) has a **distribution** that **converges** to a **limiting distribution**
- $P_i^t \to \pi_i$ as $t \to \infty$, $\forall i$
- $\boldsymbol{\pi} := (\pi_i)_{i=1}^{\infty}$ is **stationary distribution**
- *New concept* since probability distribution is an (infinite) *vector*
- Simple example of **Markov Chain Monte Carlo** algorithm:
- Run Markov chain to get given probability distribution
- Can we calculate limiting distribution directly?

Exact limit distribution II

- Take limit $t \to \infty$ in above equations to get

$$\pi_i = p\,\pi_{i+1} + (1-p)\,\pi_{i-1} \quad \forall i \geq 2$$

$$\pi_1 = p\,\pi_2 + p\,\pi_1$$

- Not hard to solve exactly *analytically* – exercise

Exact limit distribution III

- *System of linear equations*
- Can / should write in matrix language (18.06 again)
- There are efficient numerical methods to solve
- One solution method: iterate *dynamical* equations above until convergence!
- Another example of **fixed-point iteration**
- Application: Google PageRank algorithm

Exact enumeration for first-passage problems

- PS2: **first-passage problem**: first time to exit box
- Can we solve exactly?
- Can look for **mean exit time** at each point
- Get system of linear equations – linear algebra again!

Exact enumeration for distribution of exit times

- Exit time τ is random variable
- What is its distribution?
- Calculated it using Monte Carlo in PS2
- Can we calculate $\mathbb{P}(\tau = n)$ *exactly*?

Exact enumeration for first-passage

- Idea: Use exact enumeration to evolve probability distribution
- Until some probability hits the window and "exits"
- What should we do then?

Exact enumeration for first-passage II

- Idea: $\mathbb{P}(\tau = n)$ = probability that exits at time n

  is just sum of probabilities that hit exit sites i at time n!
- Set those values to 0 in P_i^t

Example: Time to hit 0 in simple random walk

- Example: Start simple random walk at $X = 1$
- How long will it take to hit 0?
- Takes time 1 with probability $1/2$
- What is mean time to reach origin?

Code for exact enumeration

- As before, must make state space finite
- Impose reflecting boundary at $x = L$
- Increase $L \to \infty$ for infinite system.

Review

- Can calculate numerically exact evolution of probability distribution for discrete-time Markov chains
- Can calculate