A simple example

- Inventory ordering problem
- Data:
  - Starting stock level
  - Maximum stock level
  - Demand in each time period (deterministic)
  - Total cost of ordering n units of stock = *F(n)*, for some given function F
    - Typically non-linear, increasing
  - Cost per unit per time period of holding stock
  - Stock out cost
- Determine:
  - Quantity of stock to order in each period

Let's get our notation straight

S Discrete state space

A Discrete action space

Stages indexed by t : This often corresponds to time periods

Transition function: $S_{t+1} = S^M(S_t, a_t)$

The state we are in at stage t+1 depends on the previous state and action
In our example, the new inventory level depends on the previous level, quantity ordered and the demand

Contribution function: $C_t(S_t, a_t)$

The immediate cost (or value) of making decision a_t in state S_t. *F* in our example.

Value function: $V_t(S_t) = \min_{a_t} \left\{ C_t(S_t, a_t) + V_{t+1}(S_{t+1}) \right\}$

Total cost (or value) of making all the best decisions from here to the end

Solving simple dynamic programming problems is easy

- Code recursive function directly
- “Memo-ise” values for efficiency
- Return value function and the argument that achieves the optimal value

Extending to stochastic problems is sometimes easy

- Our example:
  - Demand is no longer constant, but rather given by a known, discrete probability distribution
- Notation:
  - Transition matrix: $p_t(S_{t+1} | S_t, a_t)$
    Probability that if we are in state S_t and take action a_t that we will next be in state S_{t+1}
    For our example this can be calculated from the demand distribution and our action
  - Value function: $V_t(S_t) = \min_{a_t} \left\{ C_t(S_t, a_t) + \sum_{s' \in S} p_t(s' | S_t, a_t) V_{t+1}(s') \right\}$

    alternatively: $V_t(S_t) = \min_{a_t} \{ C_t(S_t, a_t) + \mathrm{E}[V_{t+1}(S_{t+1}) | S_t, a_t] \}$

Solving simple stochastic dynamic programming problems is easy

- Code recursive function directly
- “Memo-ise” values for efficiency
- Return value function and the argument that achieves the optimal value

- The answer is an expected value and a **policy**. For each possible state, it specifies the optimal action.

Some problems are too hard to solve exactly

- Consider our example expanded to 10 product types:
  - If we have 1000 maximum units in stock for each product, we have 1001^{10} possible states
  - If demand for each can range from 0 to 200, we have 201^{10} possible outcomes
  - If we can order between up to 500 units at a time, we have 501^{10} possible actions
- These are the three curses of dimensionality:
  - State space – traditional curse of dimensionality for deterministic problems
  - Outcome space – we may not be able to compute our expectation
  - Decision space – LP and MIP regularly handle very large decision spaces