Numerical Methods-Lecture IV: Bellman Equations: Solutions

Trevor Gallen

Preliminaries

- We've seen the abstract concept of Bellman Equations
- Now we'll talk about a way to solve the Bellman Equation: Value Function Iteration
- This is as simple as it gets!

Value Function Iteration

- Bellman equation:

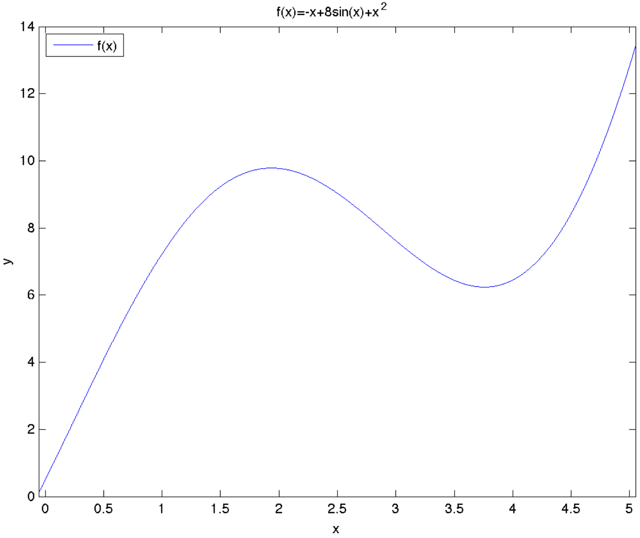
$$V(x) = \max_{y \in \Gamma(x)} \{F(x,y) + \beta V(y)\}$$

- A solution to this equation is a function V for which this equation holds $\forall$ x
- What we'll do instead is to assume an initial V_0 and define V_1 as:

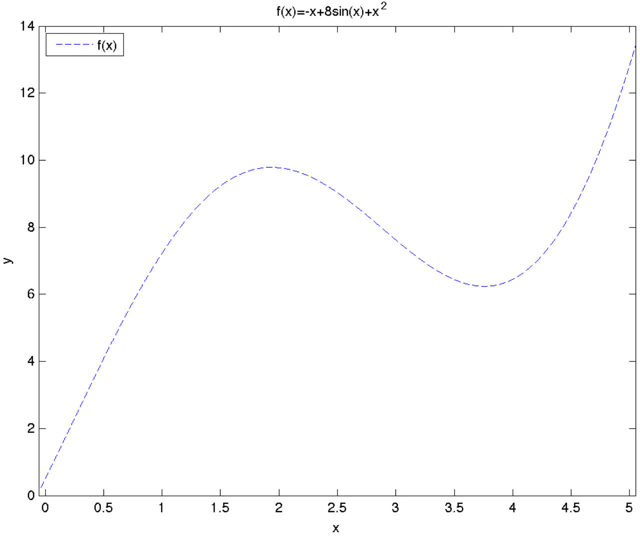
$$V_1(x) = \max_{y \in \Gamma(x)} \{F(x,y) + \beta V_0(y)\}$$

- Then redefine $V_0 = V_1$ and repeat
- Eventually, $V_1 \approx V_0$
  - But V is typically continuous: we'll discretize it
  - Make function continuous by connecting the dots

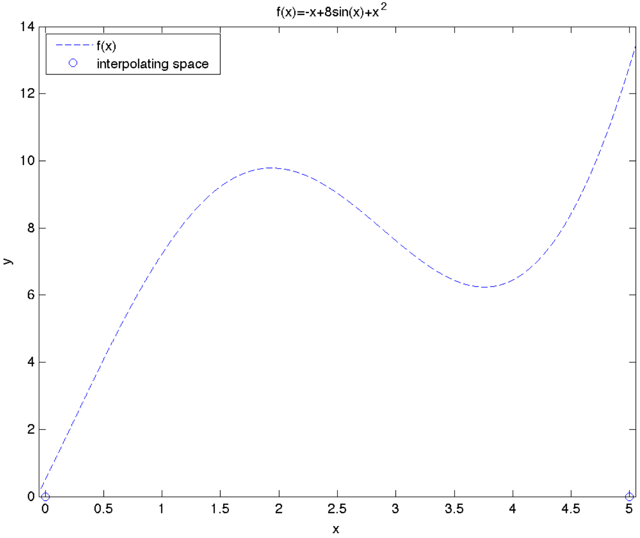
Aside: Approximating f(x)



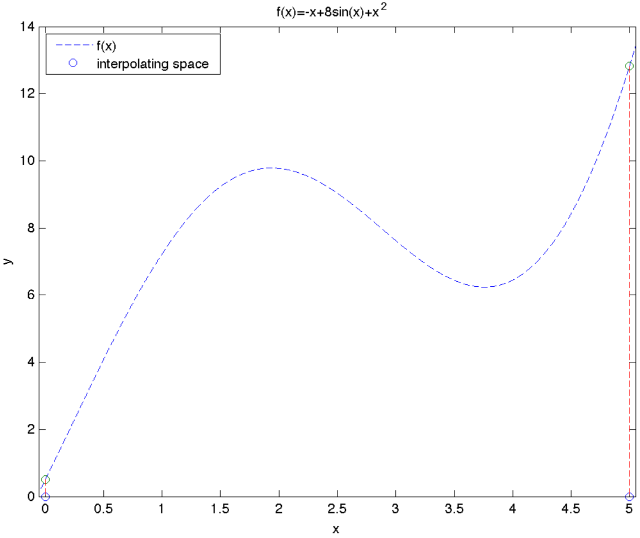
Aside: Approximating f(x)



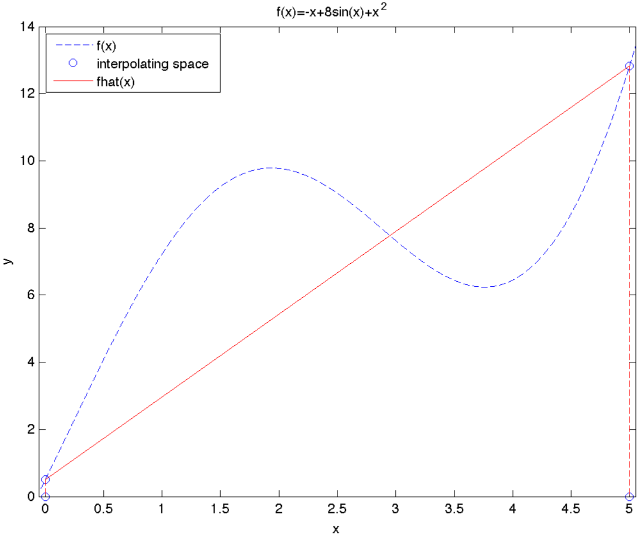
Aside: Approximating f(x)



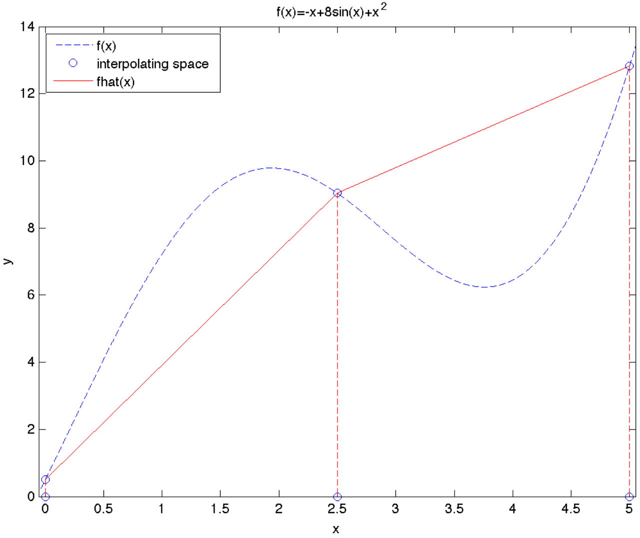
Aside: Approximating f(x)



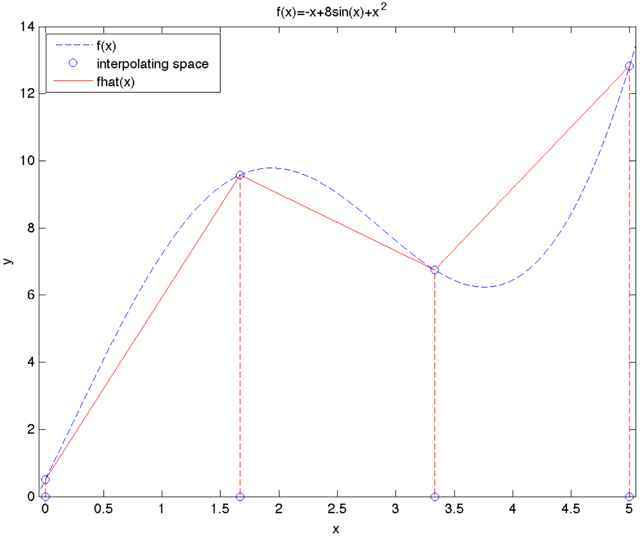
Aside: Approximating f(x)



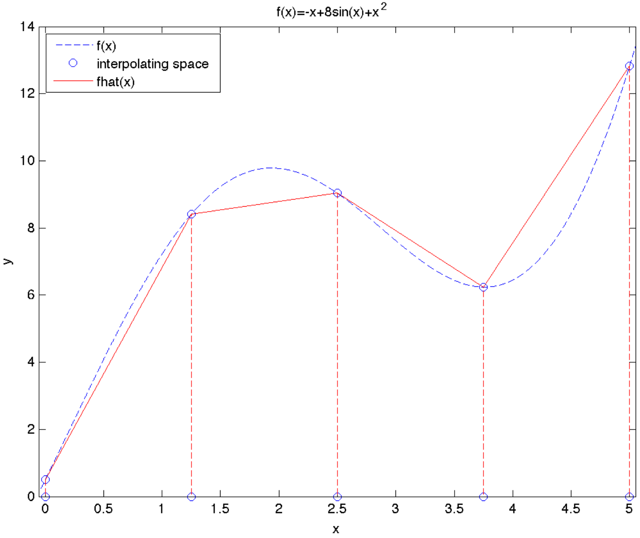
Aside: Approximating f(x)



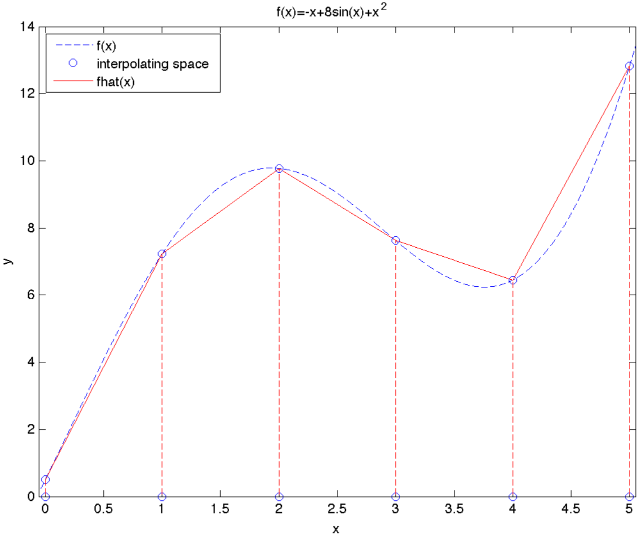
Aside: Approximating f(x)



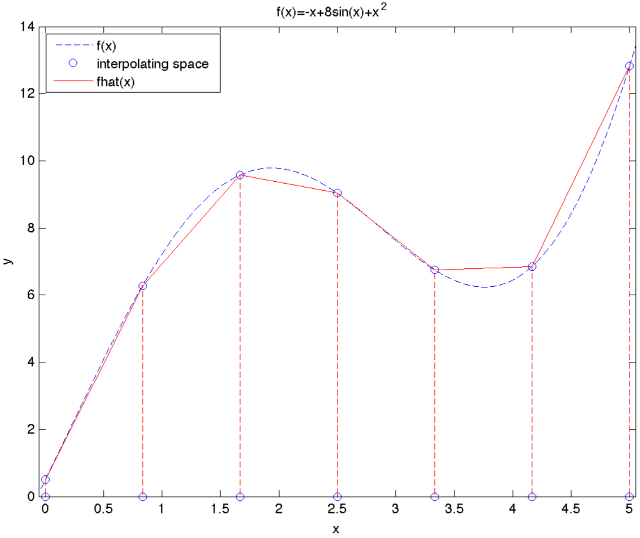
Aside: Approximating f(x)



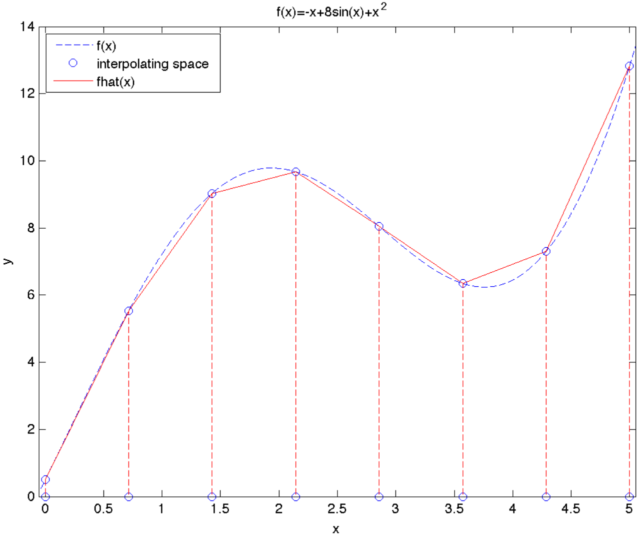
Aside: Approximating f(x)



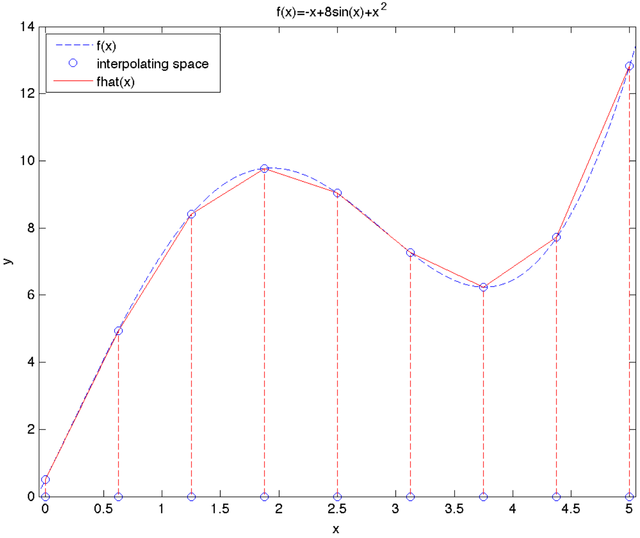
Aside: Approximating f(x)



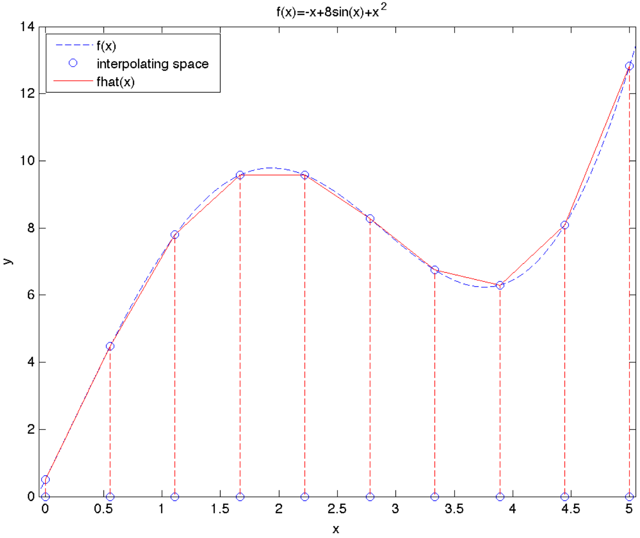
Aside: Approximating f(x)



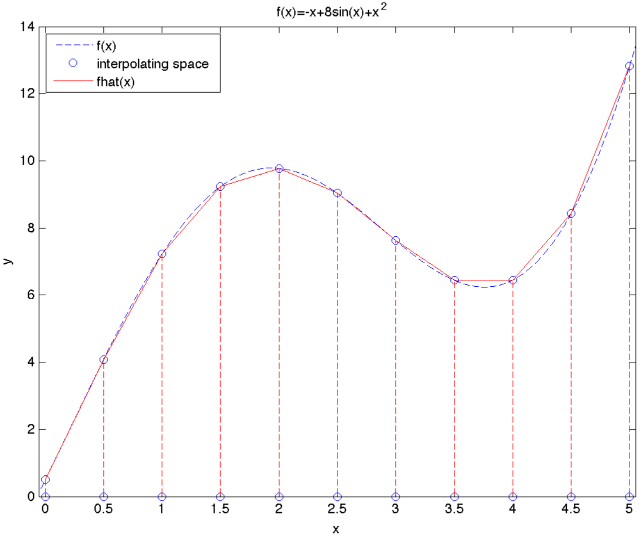
Aside: Approximating f(x)



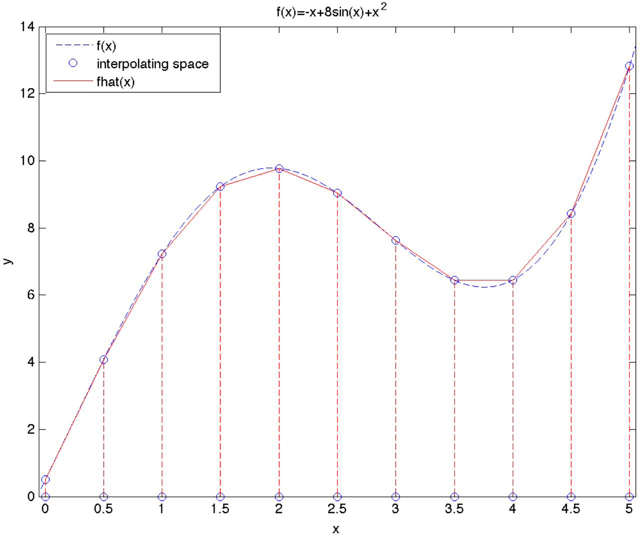
Aside: Approximating f(x)



Aside: Approximating f(x)



Aside: Approximating f(x)



Basic Steps

1. Choose grid of states X and a stopping threshold ϵ
2. Assume an initial V_0 for each $x \in X$
3. For each $x \in X$, solve the problem:

$$\max_{y \in \Gamma(x)} \{F(x, y) + \beta V_0(y)\}$$

4. Store the solution as $V_1(x)$
5. Redefine $V_0 = V_1$
6. Repeat steps 3-5 until $abs(V_1 - V_0) < \epsilon$.
7. Now, for all your relevant points, the Bellman equation holds
8. Solve the system one last time, storing the policy function

How do I solve the problem?

- Step 3 requires you to solve:

$$\max_{y \in \Gamma(x)} \{F(x, y) + \beta V_0(y)\}$$

- How do we do it?
- How do we maximize?
- We'll learn good ways
- For now, discretize all your choices like you discretized your states
- Pick best choice, store utility
- If you allow for choices to imply states that aren't defined, interpolate linearly

Aside: Intuition for VFI

- In the iteration period, all future states are the same: we don't care what happens.
- In a "cake-eating" example, this means eat everything.
- In such a scenario, we eat all the cake: we're happier with more cake.
- When we iterate once more, now tomorrow is the last day on earth: we now prefer saving a little cake.
- When we iterate again, tomorrow's tomorrow is the last day...
- Because we discount, as we iterate more, whatever we do on the last day matters less and less
- Eventually, we're all but immortal: $\lim_{t \to \infty} \beta^t = 0$

Aside: Intuition for VFI

- In the iteration period, all future states are the same: we don't care what happens.
- In a "cake-eating" example, this means eat everything.
- In such a scenario, we eat all the cake: we're happier with more cake.
- When we iterate once more, now tomorrow is the last day on earth: we now prefer saving a little cake.
- When we iterate again, tomorrow's tomorrow is the last day...
- Because we discount, as we iterate more, whatever we do on the last day matters less and less
- Eventually, we're all but immortal: $\lim_{t\to\infty} \beta^t = 0$ (really, $\lim_{t\to\infty} \beta^t u_2(x_t, x_{t+1}) x_{t+1} = 0$)

Let's do a concrete example

$$U(c_t) = \log(c_t)$$

$$c_t + i_t = k_t^{0.7}$$

$$k_{t+1} = 0.93k_t + i_t$$

- Discretize states
  - Minimum: $\underline{k} = 0$
  - Maximum: $\bar{k} = 0.93\bar{k} + \bar{k}^{0.7} \Rightarrow \bar{k} = 7075$
  - Choose 10 possible steps
- Allow choice of feasible discrete k
- Choose best, store it.
- Repeat

Solving in Matlab

```
alpha = 0.7;
delta = 0.07;
k_min = 0;
k_max = 7075;
k_num = 10;
k_space = linspace(k_min,k_max,k_num);
V_1 = 0.*k_space;
V_0 = V_1;
error = Inf;
while error > 1e-10
   for k_index = 1:k_num
    k = k_space(k_index);
    kchoice_index = find(k_space < 0.93k+k.^0.7);
    k_choices = k_space(kchoice_index);
    c_choices = 0.93*k+k.^0.7-k_choices;
    utility = log(c_choices) + beta V_0(find(kchoice_index));
    [V,ind] = max(utility);
    V_1(k_index) = V;
    k_best(k_index) = k_choices(ind);
    end
  error = max(abs(V_1-V_0))
end
```

Simulating in Matlab

```
num_i = k_num
num_t = 50;
k_sim = NaN(num_i,num_t);
k_sim(:,1) = NaN(num_i,num_t);
for i = 1:num_i
for t = 1:num_t
k_sim(i,t+1) = k_best(find(k_space)==k_sim(i,t))
end end
```