Lecture 6

Projection methods

Ivan Rudik
AEM 7130

Roadmap

1. What is projection
2. How we approximate functions

Our basic dynamic model

An arbitrary infinite horizon problem can be represented using a Bellman equation

$$V(S_t) = \max_{q_t} u(q_t) + \beta V(S_{t+1}(q_t))$$

Our basic dynamic model

An arbitrary infinite horizon problem can be represented using a Bellman equation

$$V(S_t) = \max_{q_t} u(q_t) + \beta V(S_{t+1}(q_t))$$

We can start from any arbitrary initial state vector and simulate the optimal policy paths in deterministic or stochastic settings

Our basic dynamic model

An arbitrary infinite horizon problem can be represented using a Bellman equation

$$V(S_t) = \max_{q_t} u(q_t) + \beta V(S_{t+1}(q_t))$$

We can start from any arbitrary initial state vector and simulate the optimal policy paths in deterministic or stochastic settings

We know that this problem has a fixed point (solution) V^* from previous lectures, but *how* do we approximate V^*?

Our basic dynamic model

An arbitrary infinite horizon problem can be represented using a Bellman equation

$$V(S_t) = \max_{q_t} u(q_t) + \beta V(S_{t+1}(q_t))$$

We can start from any arbitrary initial state vector and simulate the optimal policy paths in deterministic or stochastic settings

We know that this problem has a fixed point (solution) V^* from previous lectures, but *how* do we approximate V^*?

One way to do this is via **projection**

Our basic dynamic model

An arbitrary infinite horizon problem can be represented using a Bellman equation

$$V(S_t) = \max_{q_t} u(q_t) + \beta V(S_{t+1}(q_t))$$

We can start from any arbitrary initial state vector and simulate the optimal policy paths in deterministic or stochastic settings

We know that this problem has a fixed point (solution) V^* from previous lectures, but *how* do we approximate V^*?

One way to do this is via **projection**

Or perturbation, or discretization, or some analytic approaches for specific models

Our basic dynamic model

An arbitrary infinite horizon problem can be represented using a Bellman equation

$$V(S_t) = \max_{q_t} u(q_t) + \beta V(S_{t+1}(q_t))$$

We can start from any arbitrary initial state vector and simulate the optimal policy paths in deterministic or stochastic settings

We know that this problem has a fixed point (solution) V^* from previous lectures, but *how* do we approximate V^*?

One way to do this is via **projection**

Or perturbation, or discretization, or some analytic approaches for specific models

We'll focus on value functions here, but we can approximate policy functions as well

Projection methods

Main idea: build some function $\hat{V}$ indexed by coefficients that approximately solves the Bellman

Projection methods

Main idea: build some function $\hat{V}$ indexed by coefficients that approximately solves the Bellman

What do I mean by approximately?

Projection methods

Main idea: build some function $\hat{V}$ indexed by coefficients that approximately solves the Bellman

What do I mean by approximately?

The coefficients of $\hat{V}$ are selected to minimize some residual function that tells us how far away our solution is from solving the Bellman

Projection methods

Main idea: build some function $\hat{V}$ indexed by coefficients that approximately solves the Bellman

What do I mean by approximately?

The coefficients of $\hat{V}$ are selected to minimize some residual function
that tells us how far away our solution is from solving the Bellman

Rearrange the Bellman equation and define a new functional H that maps the problem into a more general framework

$$H(V) = V(S_t) - \max_{q_t} u(q_t) + \beta V(S_{t+1}(q_t))$$

Projection methods

Main idea: build some function $\hat{V}$ indexed by coefficients that approximately solves the Bellman

What do I mean by approximately?

The coefficients of $\hat{V}$ are selected to minimize some residual function that tells us how far away our solution is from solving the Bellman

Rearrange the Bellman equation and define a new functional H that maps the problem into a more general framework

$$H(V) = V(S_t) - \max_{q_t} u(q_t) + \beta V(S_{t+1}(q_t))$$

We can find some function V that solves $H(V) = \mathbf{0}$

Projection methods

How do we do this?

Projection methods

How do we do this?

We solve this by specifying some linear combination of **basis functions** $\Psi_i(\mathbf{S})$

$$V^j(\mathbf{S}|\theta) = \sum_{i=0}^{j} \theta_i \Psi_i(\mathbf{S})$$

with coefficients $\theta_0, \ldots, \theta_j$

Projection methods

We then define a residual

$$R(\mathbf{S}|\theta) = H(V^j(\mathbf{S}|\theta))$$

and select the coefficient values to minimize the residual, given some measure of distance

Projection methods

We then define a residual

$$R(\mathbf{S}|\theta) = H(V^j(\mathbf{S}|\theta))$$

and select the coefficient values to minimize the residual, given some measure of distance

This step of selecting the coefficients is called **projecting** H against our basis

Projection methods

We then define a residual

$$R(\mathbf{S}|\theta) = H(V^j(\mathbf{S}|\theta))$$

and select the coefficient values to minimize the residual, given some measure of distance

This step of selecting the coefficients is called **projecting** H against our basis

We still have some choices to make:

What basis do we select?

How do we project (select the coefficients)?

Projection methods

At first this may be a pretty confusing concept

Projection methods

At first this may be a pretty confusing concept

Simple example to get intuition?

Projection methods

At first this may be a pretty confusing concept

Simple example to get intuition?

Ordinary least squares linear regression

Projection methods

At first this may be a pretty confusing concept

Simple example to get intuition?

Ordinary least squares linear regression

We can think of the problem as searching for some unknown conditional expectation $E[Y|X]$, given outcome variable Y and regressors X

Projection methods

We don't know the true functional form, but we can approximate it using the first two monomials on X: 1 and X

$$E[Y|X] \approx \theta_0 + \theta_1 X$$

Projection methods

We don't know the true functional form, but we can approximate it using the first two monomials on X: 1 and X

$$E[Y|X] \approx \theta_0 + \theta_1 X$$

These are the first two elements of the *monomial basis*

Projection methods

We don't know the true functional form, but we can approximate it using the first two monomials on X: 1 and X

$$E[Y|X] \approx \theta_0 + \theta_1 X$$

These are the first two elements of the *monomial basis*

One residual function is then: $R(Y, X|\theta_0, \theta_1) = abs(Y - \theta_0 - \theta_1 X)$

Projection methods

We don't know the true functional form, but we can approximate it using the first two monomials on X: 1 and X

$$E[Y|X] \approx \theta_0 + \theta_1 X$$

These are the first two elements of the *monomial basis*

One residual function is then: $R(Y, X|\theta_0, \theta_1) = abs(Y - \theta_0 - \theta_1 X)$

In econometrics we often minimize the square of this residual, which is just OLS

Projection methods

We don't know the true functional form, but we can approximate it using the first two monomials on X: 1 and X

$$E[Y|X] \approx \theta_0 + \theta_1 X$$

These are the first two elements of the *monomial basis*

One residual function is then: $R(Y, X|\theta_0, \theta_1) = abs(Y - \theta_0 - \theta_1 X)$

In econometrics we often minimize the square of this residual, which is just OLS

So OLS is within the class of projection methods

Projection methods

We don't know the true functional form, but we can approximate it using the first two monomials on X: 1 and X

$$E[Y|X] \approx \theta_0 + \theta_1 X$$

These are the first two elements of the *monomial basis*

One residual function is then: $R(Y, X|\theta_0, \theta_1) = abs(Y - \theta_0 - \theta_1 X)$

In econometrics we often minimize the square of this residual, which is just OLS

So OLS is within the class of projection methods

In OLS we use observed data, but in theory we use the operator $H(V)$

Projection classes are defined by metrics

Projection methods are separated into several broad classes by the type of residual we're trying to shrink to zero

Projection classes are defined by metrics

Projection methods are separated into several broad classes by the type of residual we're trying to shrink to zero

We need to select some metric function ρ, that determines how we project

Projection classes are defined by metrics

Projection methods are separated into several broad classes by the type of residual we're trying to shrink to zero

We need to select some metric function ρ, that determines how we project

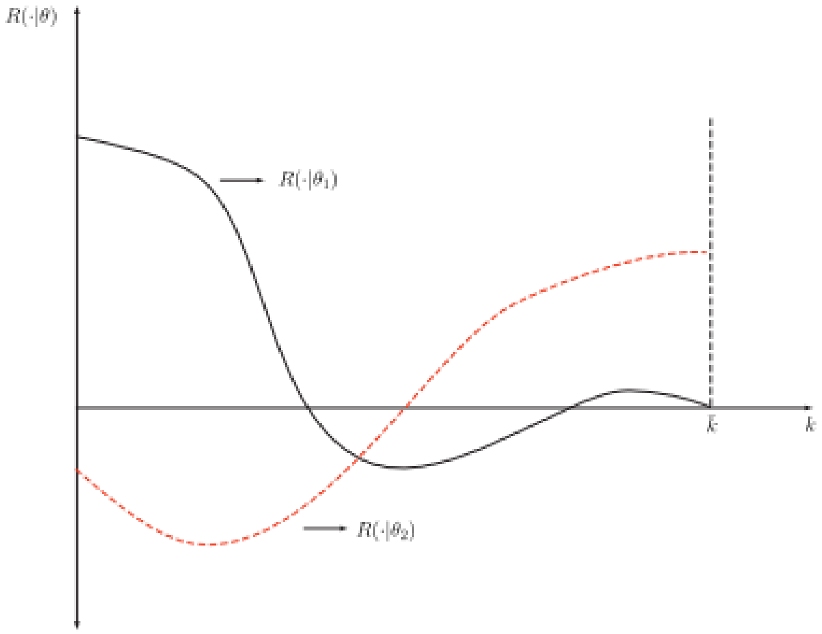
ρ tells us how close our residual function is to zero over the domain of our state space

Example residuals given different projections

Example: The figure shows two different residuals on some capital domain of $[0, \bar{k}]$

The residual based on the coefficient vector θ_1 is large for small values of capital but near-zero everywhere else

Figure 2: Residual Functions



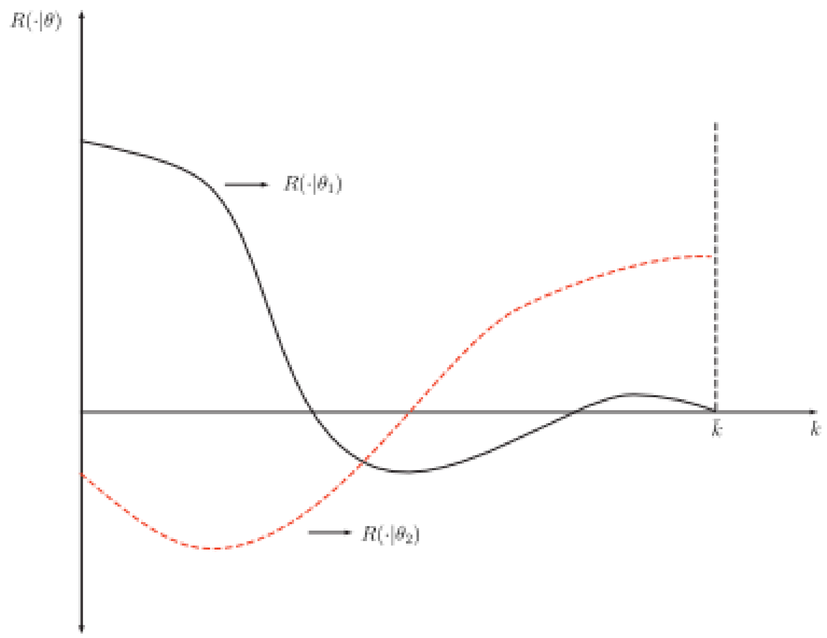
Example residuals given different projections

Example: The figure shows two different residuals on some capital domain of $[0, \bar{k}]$

The residual based on the coefficient vector θ_1 is large for small values of capital but near-zero everywhere else

The residual based on θ_2 has medium values just about everywhere

Figure 2: Residual Functions



Example residuals given different projections

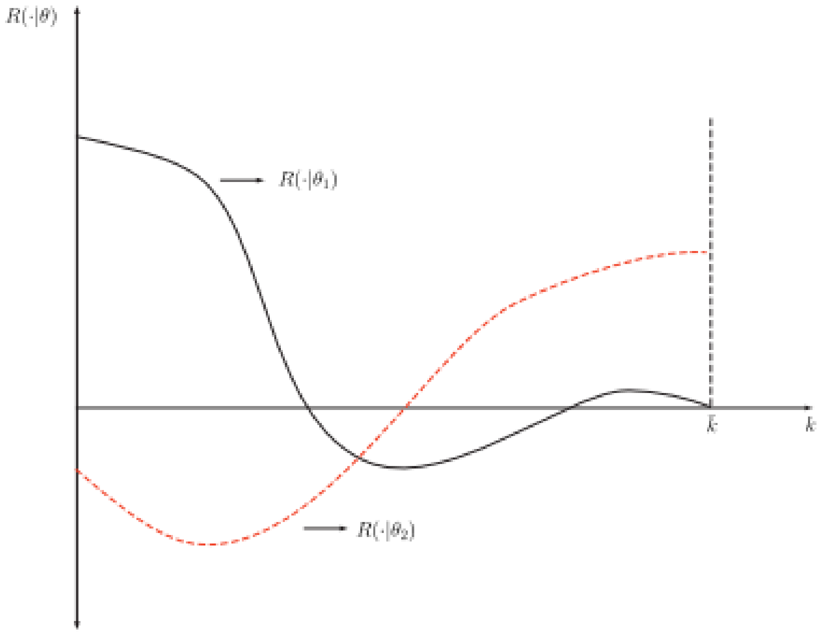
Example: The figure shows two different residuals on some capital domain of $[0, \bar{k}]$

The residual based on the coefficient vector θ_1 is large for small values of capital but near-zero everywhere else

The residual based on θ_2 has medium values just about everywhere

Which is closer to zero over the interval? It will depend on our selection of ρ

Figure 2: Residual Functions



We want to use weighted residuals

We move from the plain residual to ρ because we want to set a *weighted residual* equal to zero (more general)

We want to use weighted residuals

We move from the plain residual to ρ because we want to set a *weighted residual* equal to zero (more general)

Suppose we have some weight functions $\phi_i : \Omega \to \mathbb{R}$ that map from our state space to the real line

We want to use weighted residuals

We move from the plain residual to ρ because we want to set a *weighted residual* equal to zero (more general)

Suppose we have some weight functions $\phi_i : \Omega \to \mathbb{R}$ that map from our state space to the real line

The one-dimensional metric is defined as

$$\rho(R \cdot |\theta, 0) = \begin{cases} 0 & \text{if } \int_\Omega \phi_i(\mathbf{S}) R(\cdot|\theta) d\mathbf{S} = 0, i = 1, \ldots, j+1 \\ 1 & \text{otherwise} \end{cases}$$

We want to use weighted residuals

We move from the plain residual to ρ because we want to set a *weighted residual* equal to zero (more general)

Suppose we have some weight functions $\phi_i : \boldsymbol{\Omega} \rightarrow \mathbb{R}$ that map from our state space to the real line

The one-dimensional metric is defined as

$$\rho(R \cdot |\theta, 0) = \begin{cases} 0 & \text{if } \int_\Omega \phi_i(\mathbf{S}) R(\cdot|\theta) d\mathbf{S} = 0, i = 1, \dots, j+1 \\ 1 & \text{otherwise} \end{cases}$$

Where we want to solve for $\theta = \mathbf{argmin}\, \rho(R(\cdot|\theta), 0)$

We want to use weighted residuals

We can then change our problem to simply solving a system of integrals ensuring the metric is zero

$$\int_{\Omega} \phi_i(\mathbf{S}) R(\cdot|\theta) d\mathbf{S} = 0, i = 1, \ldots, j+1.$$

We want to use weighted residuals

We can then change our problem to simply solving a system of integrals ensuring the metric is zero

$$\int_{\Omega} \phi_i(\mathbf{S}) R(\cdot|\theta) d\mathbf{S} = 0, i = 1, \ldots, j+1.$$

We can solve this using standard rootfinding techniques

We want to use weighted residuals

We can then change our problem to simply solving a system of integrals ensuring the metric is zero

$$\int_{\Omega} \phi_i(\mathbf{S}) R(\cdot|\theta) d\mathbf{S} = 0, i = 1, \ldots, j+1.$$

We can solve this using standard rootfinding techniques

Big remaining question: how do we choose our $j+1$ weight functions?

We want to use weighted residuals

We can then change our problem to simply solving a system of integrals ensuring the metric is zero

$$\int_{\Omega} \phi_i(\mathbf{S}) R(\cdot|\theta) d\mathbf{S} = 0, i = 1, \ldots, j+1.$$

We can solve this using standard rootfinding techniques

Big remaining question: how do we choose our $j+1$ weight functions?

First lets begin with a simple example before moving into the most commonly used weight functions

Least squares projection

Suppose we selected the weight function to be

$$\phi_i(\mathbf{S}) = \frac{\partial R(\mathbf{S}|\theta)}{\partial \theta_{i-1}}$$

Least squares projection

Suppose we selected the weight function to be

$$\phi_i(\mathbf{S}) = \frac{\partial R(\mathbf{S}|\theta)}{\partial \theta_{i-1}}$$

Then we would be performing least squares! Why?

Least squares projection

Recall the objective of least squares is

$$\min_{\theta} \int R^2(\cdot|\theta) d\mathbf{S}$$

Least squares projection

Recall the objective of least squares is

$$\min_{\theta} \int R^2(\cdot|\theta) d\mathbf{S}$$

The FOC for a minimum is

$$\int \frac{\partial R(\mathbf{S}|\theta)}{\partial \theta_{i-1}} R(\cdot|\theta) d\mathbf{S} = 0, i = 1, \ldots, j+1$$

Least squares projection

Recall the objective of least squares is

$$\min_{\theta} \int R^2(\cdot|\theta)d\mathbf{S}$$

The FOC for a minimum is

$$\int \frac{\partial R(\mathbf{S}|\theta)}{\partial \theta_{i-1}} R(\cdot|\theta)d\mathbf{S} = 0, i = 1, \ldots, j+1$$

So the first order condition sets the weighted average residual to zero
where the weights are determined by the partial derivatives of the residual

Least squares projection

Recall the objective of least squares is

$$\min_{\theta} \int R^2(\cdot|\theta) d\mathbf{S}$$

The FOC for a minimum is

$$\int \frac{\partial R(\mathbf{S}|\theta)}{\partial \theta_{i-1}} R(\cdot|\theta) d\mathbf{S} = 0, i = 1, \ldots, j+1$$

So the first order condition sets the weighted average residual to zero
where the weights are determined by the partial derivatives of the residual

The partial derivative weight function yields a metric function that solves the least squares problem!

Collocation

The most commonly used weight function gives us a methodology called **collocation**

Collocation

The most commonly used weight function gives us a methodology called **collocation**

Here our weight function is

$$\phi_i(\mathbf{S}) = \delta(\mathbf{S} - \mathbf{S}_i)$$

Where δ is the *Dirac delta function* and $\mathbf{S}_i$ are the j+1 *collocation points* or *collocation nodes* selected by the researcher

Collocation

The most commonly used weight function gives us a methodology called **collocation**

Here our weight function is

$$\phi_i(\mathbf{S}) = \delta(\mathbf{S} - \mathbf{S}_i)$$

Where δ is the *Dirac delta function* and $\mathbf{S}_i$ are the j+1 *collocation points* or *collocation nodes* selected by the researcher

The Dirac delta function is zero at all $\mathbf{S}$ except at $\mathbf{S} = \mathbf{S}_i$

Collocation

The most commonly used weight function gives us a methodology called **collocation**

Here our weight function is

$$\phi_i(\mathbf{S}) = \delta(\mathbf{S} - \mathbf{S}_i)$$

Where δ is the *Dirac delta function* and $\mathbf{S}_i$ are the j+1 *collocation points* or *collocation nodes* selected by the researcher

The Dirac delta function is zero at all $\mathbf{S}$ except at $\mathbf{S} = \mathbf{S}_i$

What does this weight function mean?

Collocation

Before we even select our coefficients, this means that the residual can only be non-zero at a finite set of points $\mathbf{S_i}$

Collocation

Before we even select our coefficients, this means that the residual can only be non-zero at a finite set of points $\mathbf{S_i}$

So the solution to our problem must set the residual to zero at these collocation points

Collocation

Before we even select our coefficients, this means that the residual can only be non-zero at a finite set of points $\mathbf{S_i}$

So the solution to our problem must set the residual to zero at these collocation points

Since we have a finite set of points we do not need to solve difficult integrals but only a system of equations

$$R(\mathbf{S}_i|\theta) = 0, i = 1, \ldots, j + 1$$

Rough idea of how we proceed

Collocation methods aim to approximate the value function $V(S_t)$ with some linear combination of known, and usually orthogonal ($\int f(x)g(x)w(x) = 0$), basis functions

Rough idea of how we proceed

Collocation methods aim to approximate the value function $V(S_t)$ with some linear combination of known, and usually orthogonal ($\int f(x)g(x)w(x) = 0$), basis functions

One example of a possible class of basis functions is the monomial basis: $x, x^2, x^3, \ldots$

Rough idea of how we proceed

Collocation methods aim to approximate the value function $V(S_t)$ with some linear combination of known, and usually orthogonal ($\int f(x)g(x)w(x) = 0$), basis functions

One example of a possible class of basis functions is the monomial basis: $x, x^2, x^3, \ldots$

But how do we implement collocation?

Rough idea of how we proceed

Recall we solve for coefficients θ by setting the residual to be zero at all of our collocation points

Rough idea of how we proceed

Recall we solve for coefficients θ by setting the residual to be zero at all of our collocation points

But the residual is a function of V, and thus will be a function of θ, a loop seemingly impossible to escape

Rough idea of how we proceed

Recall we solve for coefficients θ by setting the residual to be zero at all of our collocation points

But the residual is a function of V, and thus will be a function of θ, a loop seemingly impossible to escape

We can solve this problem by *iterating* on the problem,
continually setting the residuals equal to zero, recovering new θs, and repeating

Rough idea of how we proceed

In any given iteration, we:

Rough idea of how we proceed

In any given iteration, we:

1. Begin with a vector of coefficients on the basis functions (e.g. a random guess)

Rough idea of how we proceed

In any given iteration, we:

1. Begin with a vector of coefficients on the basis functions (e.g. a random guess)

2. Use a linear combination of the basis functions as an approximation to the value function on the right hand side of the Bellman

Rough idea of how we proceed

In any given iteration, we:

1. Begin with a vector of coefficients on the basis functions (e.g. a random guess)
2. Use a linear combination of the basis functions as an approximation to the value function on the right hand side of the Bellman
3. Solve the Bellman with the approximated value function in it, at our set of collocation points

Rough idea of how we proceed

In any given iteration, we:

1. Begin with a vector of coefficients on the basis functions (e.g. a random guess)
2. Use a linear combination of the basis functions as an approximation to the value function on the right hand side of the Bellman
3. Solve the Bellman with the approximated value function in it, at our set of collocation points
4. Recover a set maximized continuation values at these points in our state space conditional on the previous value function approximant we used

Rough idea of how we proceed

In any given iteration, we:

1. Begin with a vector of coefficients on the basis functions (e.g. a random guess)
2. Use a linear combination of the basis functions as an approximation to the value function on the right hand side of the Bellman
3. Solve the Bellman with the approximated value function in it, at our set of collocation points
4. Recover a set maximized continuation values at these points in our state space conditional on the previous value function approximant we used
5. Use these new maximized values to obtain updated coefficients solving the system of linear equations, and repeat the process until we have "converged"

Still have some questions

By the contraction mapping theorem this is guaranteed to converge from any arbitrary initial set of coefficients! (conditional on satisfying the assumptions and there being no numerical error..)

Still have some questions

By the contraction mapping theorem this is guaranteed to converge from any arbitrary initial set of coefficients! (conditional on satisfying the assumptions and there being no numerical error..)

This still leaves several questions unanswered

Still have some questions

By the contraction mapping theorem this is guaranteed to converge from any arbitrary initial set of coefficients! (conditional on satisfying the assumptions and there being no numerical error..)

This still leaves several questions unanswered

- Where in the state space do we place the collocation points?

Still have some questions

By the contraction mapping theorem this is guaranteed to converge from any arbitrary initial set of coefficients! (conditional on satisfying the assumptions and there being no numerical error..)

This still leaves several questions unanswered

- Where in the state space do we place the collocation points?
- What basis functions do we use?

Still have some questions

By the contraction mapping theorem this is guaranteed to converge from any arbitrary initial set of coefficients! (conditional on satisfying the assumptions and there being no numerical error..)

This still leaves several questions unanswered

- Where in the state space do we place the collocation points?
- What basis functions do we use?

Do your answers to the previous two questions really matter?

Still have some questions

By the contraction mapping theorem this is guaranteed to converge from any arbitrary initial set of coefficients! (conditional on satisfying the assumptions and there being no numerical error..)

This still leaves several questions unanswered

- Where in the state space do we place the collocation points?
- What basis functions do we use?

Do your answers to the previous two questions really matter?

Yes they are crucial

Still have some questions

Schemes exist to generate high quality approximations, and to obtain the approximation at low computational cost

Still have some questions

Schemes exist to generate high quality approximations, and to obtain the approximation at low computational cost

Once we have recovered an adequate approximation to the value function, we have effectively solved the entire problem!

Still have some questions

Schemes exist to generate high quality approximations, and to obtain the approximation at low computational cost

Once we have recovered an adequate approximation to the value function, we have effectively solved the entire problem!

We know how the policymaker's expected discounted stream of future payoffs changes as we move through the state space

Still have some questions

Schemes exist to generate high quality approximations, and to obtain the approximation at low computational cost

Once we have recovered an adequate approximation to the value function, we have effectively solved the entire problem!

We know how the policymaker's expected discounted stream of future payoffs changes as we move through the state space

We can solve the Bellman at some initial state and know that solving the Bellman will yield us optimal policies

Still have some questions

Schemes exist to generate high quality approximations, and to obtain the approximation at low computational cost

Once we have recovered an adequate approximation to the value function, we have effectively solved the entire problem!

We know how the policymaker's expected discounted stream of future payoffs changes as we move through the state space

We can solve the Bellman at some initial state and know that solving the Bellman will yield us optimal policies

Therefore we can simulate anything we want and recover optimal policy functions given many different sets of initial conditions or realizations of random variables

Interpolation

What points in our state space do we select to be collocation points?

Interpolation

What points in our state space do we select to be collocation points?

We often have continuous states in economics (capital, temperature, oil reserves, technology shocks, etc.), so we must have some way to reduce the infinity of points in our state space into something more manageable

Interpolation

What points in our state space do we select to be collocation points?

We often have continuous states in economics (capital, temperature, oil reserves, technology shocks, etc.), so we must have some way to reduce the infinity of points in our state space into something more manageable

We do so by selecting a specific finite number of points in our state space and use them to construct a *collocation grid* that spans the domain of our problem

Interpolation

Using our knowledge of how the value function behaves at the limited set of points on our grid, we can interpolate our value function approximant at all points off the grid points, but *within* the domain of our grid

Interpolation

Using our knowledge of how the value function behaves at the limited set of points on our grid, we can interpolate our value function approximant at all points off the grid points, but *within* the domain of our grid

Note: The value function approximant is not very good outside the grid's domain since that would mean extrapolating beyond whatever information we have gained from analyzing our value function on the grid

Basis functions

Let V be the value function we wish to approximate with some $\hat{V}$

Basis functions

Let V be the value function we wish to approximate with some $\hat{V}$

$\hat{V}$ is constructed as a linear combination of n linearly independent (i.e. orthogonal) basis functions

$$\hat{V}(x) = \sum_{j=1}^{n} c_j \psi_j(x)$$

Basis functions

Let V be the value function we wish to approximate with some $\hat{V}$

$\hat{V}$ is constructed as a linear combination of n linearly independent (i.e. orthogonal) basis functions

$$\hat{V}(x) = \sum_{j=1}^{n} c_j \psi_j(x)$$

Each $\psi_j(x)$ is a basis function, and the coefficients c_j determine how they are combined at some point $\bar{x}$ to yield our approximation $\hat{V}(\bar{x})$ to $V(\bar{x})$

Basis functions

The number of basis functions we select, n, is the degree of interpolation

Basis functions

The number of basis functions we select, n, is the degree of interpolation

In order to recover n coefficients, we need at least n equations that must be satisfied at a solution to the problem

Basis functions

The number of basis functions we select, n, is the degree of interpolation

In order to recover n coefficients, we need at least n equations that must be satisfied at a solution to the problem

If we have precisely n equations, we are just solving a simple system of linear equations: we have a perfectly identified system and are solving a collocation problem

Basis functions

The number of basis functions we select, n, is the degree of interpolation

In order to recover n coefficients, we need at least n equations that must be satisfied at a solution to the problem

If we have precisely n equations, we are just solving a simple system of linear equations: we have a perfectly identified system and are solving a collocation problem

This is what happens we select our number of grid points in the state space to be equal to the number of coefficients (which induces a Dirac delta weighting function)

Basis functions

Solve a system of equations, *linear in* $\boldsymbol{c_j}$ that equates the function approximant at the grid points to the recovered values

$$\Psi \mathbf{c} = \mathbf{y}$$

where $\boldsymbol{\Psi}$ is the matrix of basis functions, $\boldsymbol{c}$ is a vector of coefficients, and $\boldsymbol{y}$ is a vector of the recovered values

Basis functions

Solve a system of equations, *linear in* $\boldsymbol{c_j}$ that equates the function approximant at the grid points to the recovered values

$$\boldsymbol{\Psi}\mathbf{c} = \mathbf{y}$$

where $\boldsymbol{\Psi}$ is the matrix of basis functions, $\boldsymbol{c}$ is a vector of coefficients, and $\boldsymbol{y}$ is a vector of the recovered values

We can recover $\boldsymbol{c}$ by left dividing by $\boldsymbol{\Psi}$ which yields

$$\mathbf{c} = \boldsymbol{\Psi}^{-1}\mathbf{y}$$

Basis functions

Solve a system of equations, *linear in* c_j that equates the function approximant at the grid points to the recovered values

$$\Psi\mathbf{c} = \mathbf{y}$$

where $\boldsymbol{\Psi}$ is the matrix of basis functions, $\boldsymbol{c}$ is a vector of coefficients, and $\boldsymbol{y}$ is a vector of the recovered values

We can recover $\boldsymbol{c}$ by left dividing by $\boldsymbol{\Psi}$ which yields

$$\mathbf{c} = \Psi^{-1}\mathbf{y}$$

If we have more equations, or grid points, than coefficients, then we can just use OLS to solve for the coefficients by minimizing the sum of squared errors

Basis functions

Solve a system of equations, *linear in* c_j that equates the function approximant at the grid points to the recovered values

$$\Psi \mathbf{c} = \mathbf{y}$$

where $\boldsymbol{\Psi}$ is the matrix of basis functions, $\boldsymbol{c}$ is a vector of coefficients, and $\boldsymbol{y}$ is a vector of the recovered values

We can recover $\boldsymbol{c}$ by left dividing by $\boldsymbol{\Psi}$ which yields

$$\mathbf{c} = \Psi^{-1}\mathbf{y}$$

If we have more equations, or grid points, than coefficients, then we can just use OLS to solve for the coefficients by minimizing the sum of squared errors

$$\mathbf{c} = (\Psi'\Psi)^{-1}\Psi'\mathbf{y}$$

(Pseudo-)spectral methods

Spectral methods apply all of our basis functions to the entire domain of our grid: they are global

(Pseudo-)spectral methods

Spectral methods apply all of our basis functions to the entire domain of our grid: they are global

When using spectral methods we virtually always use polynomials

(Pseudo-)spectral methods

Spectral methods apply all of our basis functions to the entire domain of our grid: they are global

When using spectral methods we virtually always use polynomials

Why?

(Pseudo-)spectral methods

Spectral methods apply all of our basis functions to the entire domain of our grid: they are global

When using spectral methods we virtually always use polynomials

Why?

The Stone-Weierstrass Theorem which states (for one dimension)

Suppose f is a continuous real-valued function defined on the interval $[a, b]$.
For every $\epsilon > 0, \ \exists$ a polynomial $p(x)$ such that for all $x \in [a, b]$ we have
$||f(x) - p(x)||_{sup} \leq \epsilon$

(Pseudo-)spectral methods

Spectral methods apply all of our basis functions to the entire domain of our grid: they are global

When using spectral methods we virtually always use polynomials

Why?

The Stone-Weierstrass Theorem which states (for one dimension)

Suppose f is a continuous real-valued function defined on the interval $[a, b]$.
For every $\epsilon > 0,\ \exists$ a polynomial $p(x)$ such that for all $x \in [a, b]$ we have
$||f(x) - p(x)||_{sup} \leq \epsilon$

What does the SW theorem say in words?

(Pseudo-)spectral methods

For any continuous function $f(x)$, we can approximate it arbitrarily well with some polynomial $p(x)$, as long as $f(x)$ is continuous

(Pseudo-)spectral methods

For any continuous function $f(x)$, we can approximate it arbitrarily well with some polynomial $p(x)$, as long as $f(x)$ is continuous

This means the function can even have kinks

(Pseudo-)spectral methods

For any continuous function $f(x)$, we can approximate it arbitrarily well with some polynomial $p(x)$, as long as $f(x)$ is continuous

This means the function can even have kinks

Unfortunately we do not have infinite computational power so solving kinked problems with spectral methods is not advised

(Pseudo-)spectral methods

For any continuous function $f(x)$, we can approximate it arbitrarily well with some polynomial $p(x)$, as long as $f(x)$ is continuous

This means the function can even have kinks

Unfortunately we do not have infinite computational power so solving kinked problems with spectral methods is not advised

Note that the SW theorem *does not* say what kind of polynomial can approximate f arbitrarily well, just that some polynomial exists

Basis choice

What would be your first choice of basis?

Basis choice

What would be your first choice of basis?

Logical choice: the monomial basis: $\mathbf{1}, \boldsymbol{x}, \boldsymbol{x}^{2}, \ldots$

Basis choice

What would be your first choice of basis?

Logical choice: the monomial basis: $1, x, x^2, \ldots$

It is simple, and SW tells us that we can uniformly approximate any continuous function on a closed interval using them

Basis choice

What would be your first choice of basis?

Logical choice: the monomial basis: $1, x, x^2, \ldots$

It is simple, and SW tells us that we can uniformly approximate any continuous function on a closed interval using them

Practice

code up a function `project_monomial(f, n, lb, ub)` that takes in some function `f`, degree of approximation `n`, lower bound `lb` and upper bound `ub`, and constructs a monomial approximation on an evenly spaced grid via collocation

Basis choice

What would be your first choice of basis?

Logical choice: the monomial basis: $1, x, x^2, \ldots$

It is simple, and SW tells us that we can uniformly approximate any continuous function on a closed interval using them

Practice

code up a function `project_monomial(f, n, lb, ub)` that takes in some function `f`, degree of approximation `n`, lower bound `lb` and upper bound `ub`, and constructs a monomial approximation on an evenly spaced grid via collocation

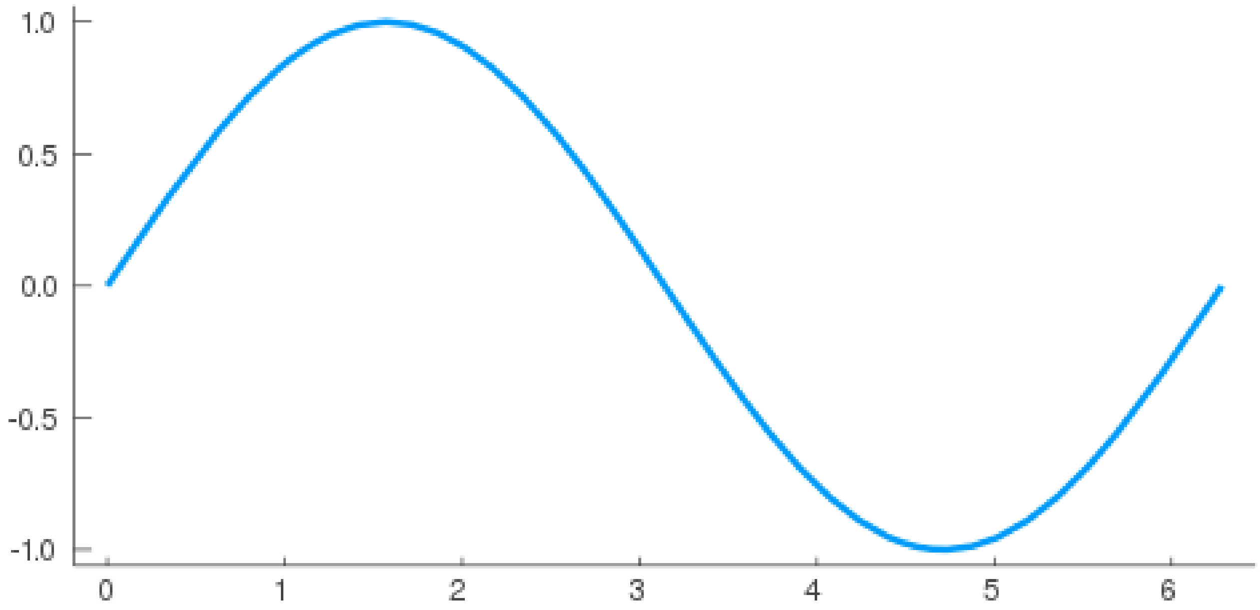
We will be plotting stuff, see http://docs.juliaplots.org/latest/generated/gr/ for example code using the `GR` backend

The monomial basis

Let's approximate `sin(x)`

```
using Plots
gr();
f(x) = sin.(x);

Plots.plot(f, 0, 2pi, line = 4, grid = false, legend = false, size = (500, 250))
```



Approximating sin(x)

```
function project_monomial(f, n, lb, ub)
    # solves Ψc = y → c = Ψ\y
    # Ψ = matrix of monomial basis functions evaluted on the grid

    coll_points = range(lb, ub, length = n)                          # collocation points
    y_values = f(coll_points)                                        # function values on the grid
    basis_functions = [coll_points.^degree for degree = 0:n-1]      # vector of basis functions
    basis_matrix = hcat(basis_functions...)                          # basis matrix

    coefficients = basis_matrix\y_values                             # c = Ψ\y

    return coefficients

end;
coefficients_4 = project_monomial(f, 4, 0, 2pi);
coefficients_5 = project_monomial(f, 5, 0, 2pi);
coefficients_10 = project_monomial(f, 10, 0, 2pi)
```

```
## 10-element Array{Float64,1}:
##   0.0
##   0.9990725797458863
##   0.004015857153649684
##  -0.1738437387373486
```

Approximating sin(x)

Now we need to construct a function `f_approx(coefficients, plot_points)` that takes in the `coefficients` vector, and an arbitrary vector of points to evaluate the approximant at (for plotting)

Approximating sin(x)

Now we need to construct a function `f_approx(coefficients, plot_points)` that takes in the `coefficients` vector, and an arbitrary vector of points to evaluate the approximant at (for plotting)

```
function f_approx(coefficients, points)
    n = length(coefficients) - 1
    basis_functions = [coefficients[degree + 1] * points.^degree for degree = 0:n] # evaluate basis functions
    basis_matrix = hcat(basis_functions...)                                        # transform into matrix
    function_values = sum(basis_matrix, dims = 2)                                  # sum up into function value
    return function_values
end;
```

Approximating sin(x)

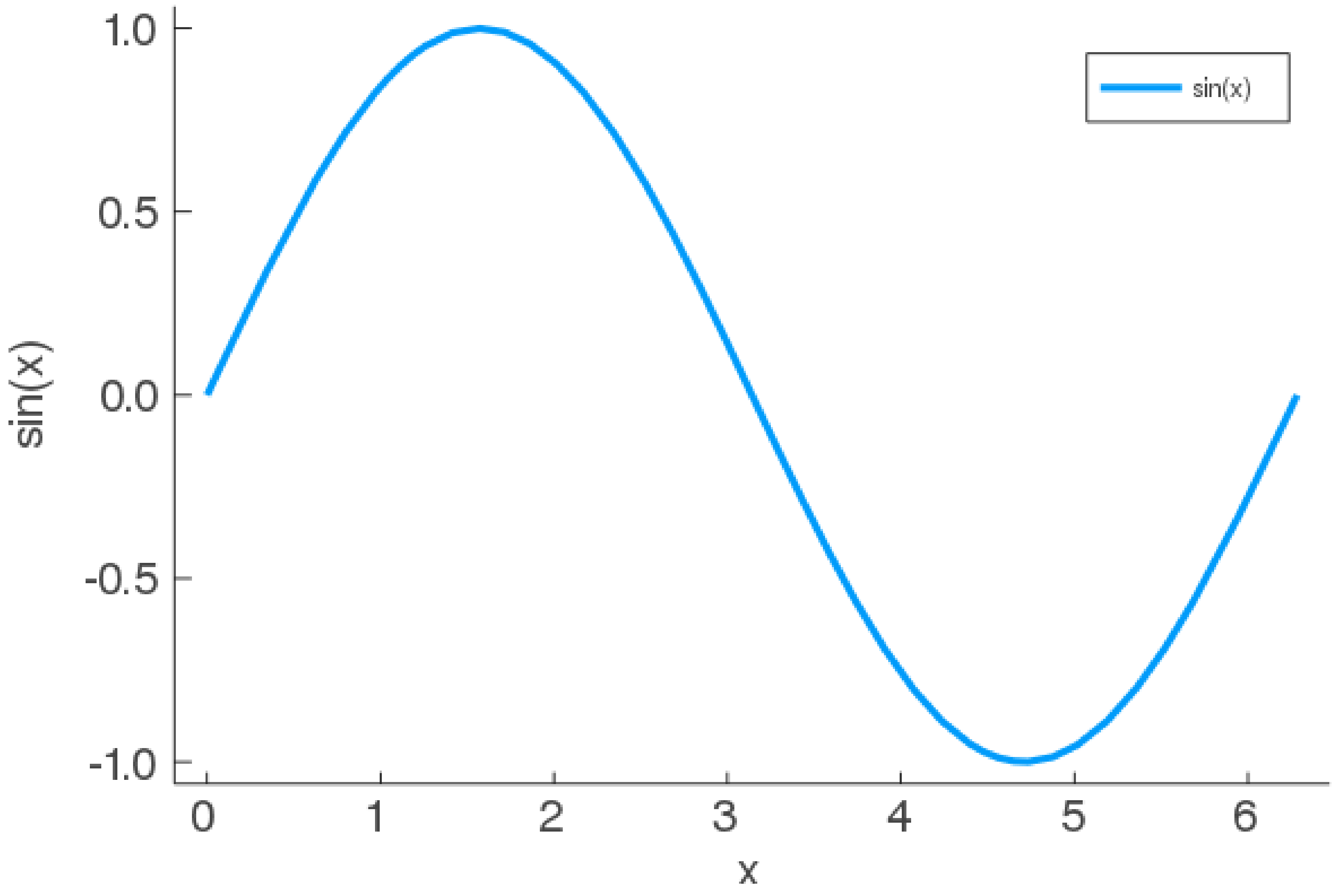
Now we need to construct a function `f_approx(coefficients, plot_points)` that takes in the `coefficients` vector, and an arbitrary vector of points to evaluate the approximant at (for plotting)

```
function f_approx(coefficients, points)
    n = length(coefficients) - 1
    basis_functions = [coefficients[degree + 1] * points.^degree for degree = 0:n] # evaluate basis functions
    basis_matrix = hcat(basis_functions...)                                        # transform into matrix
    function_values = sum(basis_matrix, dims = 2)                                  # sum up into function value
    return function_values
end;
```

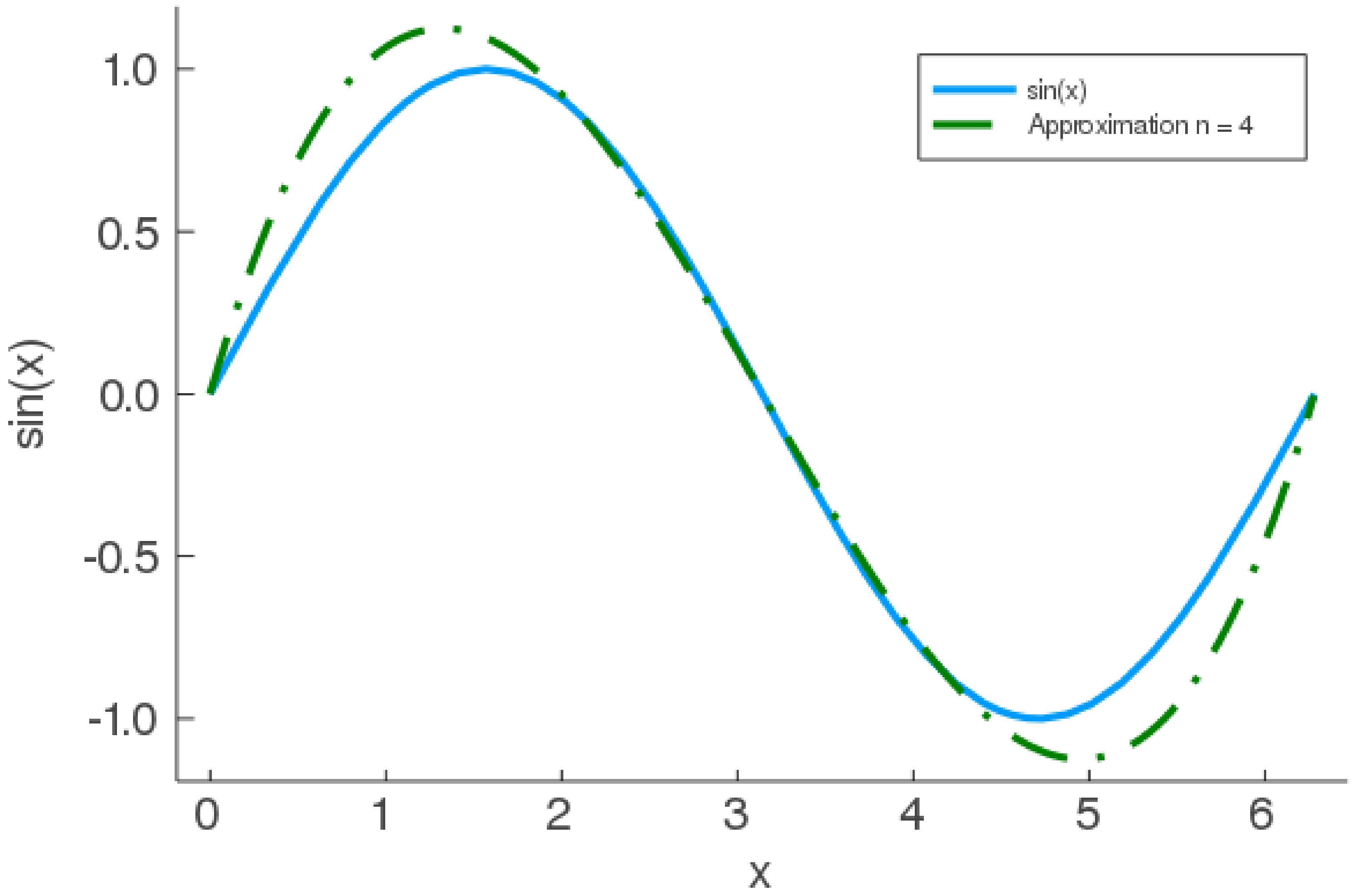
```
plot_points = 0:.01:2pi;
f_values_4 = f_approx(coefficients_4, plot_points);
f_values_5 = f_approx(coefficients_5, plot_points);
f_values_10 = f_approx(coefficients_10, plot_points)
```

```
## 629×1 Array{Float64,2}:
##   0.0
##   0.009999095361059786
```

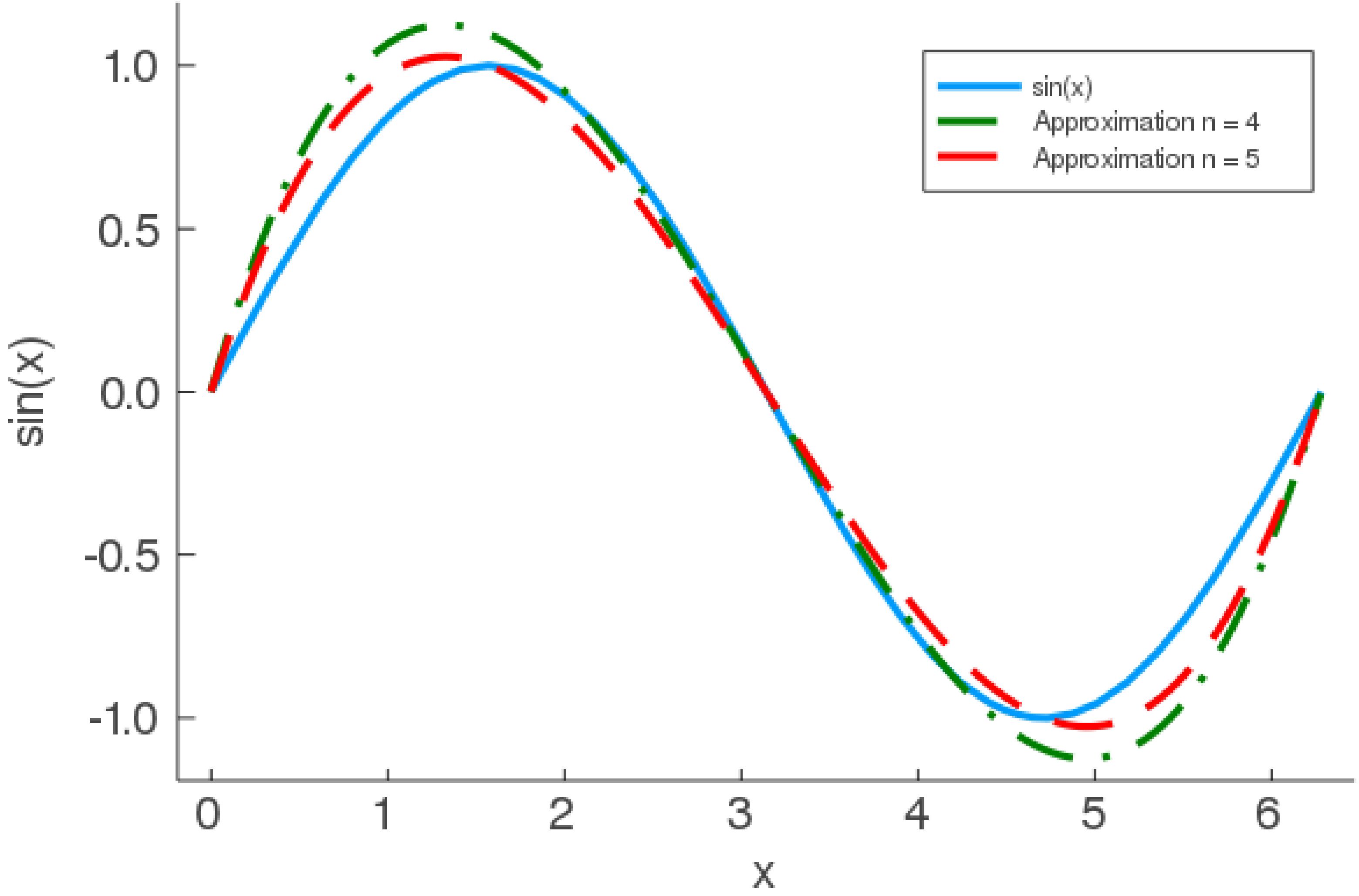
Plot



Plot

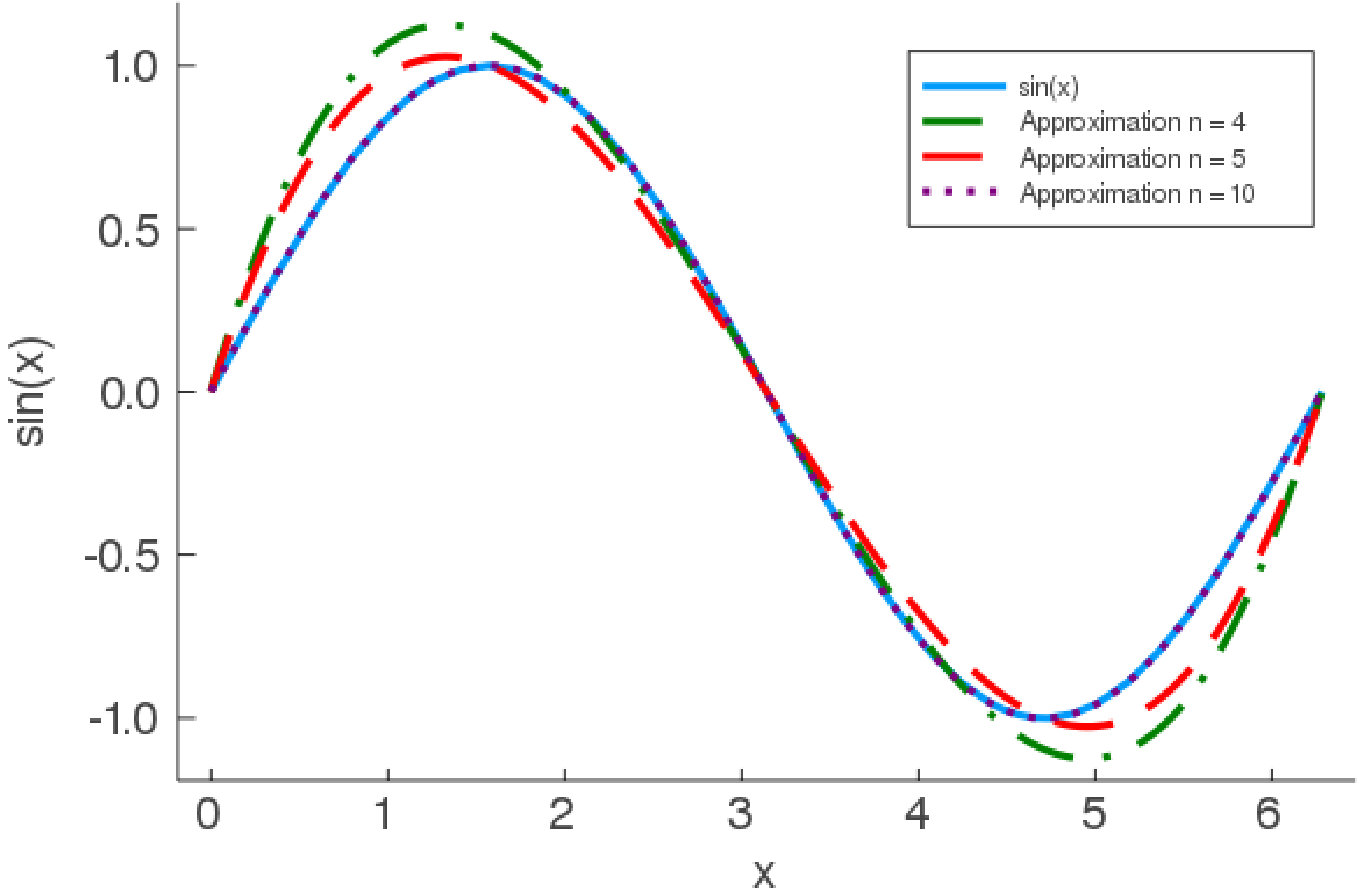


Plot



sin(x)
Approximation n = 4
Approximation n = 5
1.0
0.5
0.0
-0.5
-1.0
sin(x)
0
1
2
3
4
5
6
x

Plot



1.0
0.5
0.0
-0.5
-1.0
sin(x)
0
1
2
3
4
5
6
x
sin(x)
Approximation n = 4
Approximation n = 5
Approximation n = 10

Cool!

We just wrote some code that exploits Stone-Weierstrauss and allows us to (potentially) approximate any continuous function arbitrarily well as `n` goes to infinity

Cool!

We just wrote some code that exploits Stone-Weierstrauss and allows us to (potentially) approximate any continuous function arbitrarily well as `n` goes to infinity

To approximate any function we'd need to feed in some basis function `g(x, n)` as opposed to hard-coding it like I did in the previous slides

Cool!

We just wrote some code that exploits Stone-Weierstrauss and allows us to (potentially) approximate any continuous function arbitrarily well as `n` goes to infinity

To approximate any function we'd need to feed in some basis function `g(x, n)` as opposed to hard-coding it like I did in the previous slides

Turns out we never use the monomial basis though

Cool!

We just wrote some code that exploits Stone-Weierstrauss and allows us to (potentially) approximate any continuous function arbitrarily well as `n` goes to infinity

To approximate any function we'd need to feed in some basis function `g(x, n)` as opposed to hard-coding it like I did in the previous slides

Turns out we never use the monomial basis though