Tools: Functional Approximation and Non-Linear Equations

Emile Alexandre Marin

emarin@ucdavis.edu

October 12, 2023

Simulation

Let's warm up with pseudo-code to simulate an economy:

Example: Suppose you want to run an experiment of tossing a fair coin a hundred times and record the results (this will take some long time, but is feasible). Then suppose you need to repeat this experiment many times (say 1000). The cost of doing this is very large if you do it manually

Simulation

Let's warm up with pseudo-code to simulate an economy:

Example: Suppose you want to run an experiment of tossing a fair coin a hundred times and record the results (this will take some long time, but is feasible). Then suppose you need to repeat this experiment many times (say 1000). The cost of doing this is very large if you do it manually

Construct algorithm that will simulate the experiments and record results. If the coin is fair then the probability of getting heads (H) or tails (T) is 50%.

Simulation

Let's warm up with pseudo-code to simulate an economy:

Example: Suppose you want to run an experiment of tossing a fair coin a hundred times and record the results (this will take some long time, but is feasible). Then suppose you need to repeat this experiment many times (say 1000). The cost of doing this is very large if you do it manually

Construct algorithm that will simulate the experiments and record results. If the coin is fair then the probability of getting heads (H) or tails (T) is 50%.

- Just as easily, agent following rule of thumb response for economic action (e.g. if market downturn, sell equity 50% of the time)

Simulation

Let's warm up with pseudo-code to simulate an economy:

Example: Suppose you want to run an experiment of tossing a fair coin a hundred times and record the results (this will take some long time, but is feasible). Then suppose you need to repeat this experiment many times (say 1000). The cost of doing this is very large if you do it manually

Construct algorithm that will simulate the experiments and record results. If the coin is fair then the probability of getting heads (H) or tails (T) is 50%.

- Just as easily, agent following rule of thumb response for economic action (e.g. if market downturn, sell equity 50% of the time)

Use RAND to produce values that are uniformly distributed in $[0, 1]$ and represent (H) with a value less than 0.5 and (T) with a value more than 0.5.

Simulation

Let's warm up with pseudo-code to simulate an economy:

Example: Suppose you want to run an experiment of tossing a fair coin a hundred times and record the results (this will take some long time, but is feasible). Then suppose you need to repeat this experiment many times (say 1000). The cost of doing this is very large if you do it manually

Construct algorithm that will simulate the experiments and record results. If the coin is fair then the probability of getting heads (H) or tails (T) is 50%.

- Just as easily, agent following rule of thumb response for economic action (e.g. if market downturn, sell equity 50% of the time)

Use RAND to produce values that are uniformly distributed in $[0, 1]$ and represent (H) with a value less than 0.5 and (T) with a value more than 0.5.

Pseudocode for Simulation

```
repeat 1000 times
for i = 1 up to 100
r = RAND
if r < 0.5
print H
else
print T
end
stop when i becomes 100
stop after repeating 1000 times
```

Pseudocode for Simulation

repeat 1000 times
for i = 1 up to 100
r = RAND
if r $<$ 0.5
print H
else
print T
end
stop when i becomes 100
stop after repeating 1000 times

What do you think the output will look like?

Functional Approximation

Take some data (can be simulated) $\{x_i, y_i\}_{i=1,\ldots N}$ and assume there exists a relationship $y = g(x)$. From the data, approximate $\tilde{g}(x)$.

Functional Approximation

Take some data (can be simulated) $\{x_i, y_i\}_{i=1,\ldots N}$ and assume there exists a relationship $y = g(x)$. From the data, approximate $\tilde{g}(x)$.

Generally, powerful mathematics and algorithms teach us that:

$$\tilde{g}(x) = \sum_{n=0}^{N} c_n T_n(x)$$

where n is the order of the approximation, c_n are coefficients we can solve for, and $T_n(x)$ is a set of basis functions.

Functional Approximation

Take some data (can be simulated) $\{x_i, y_i\}_{i=1,\ldots N}$ and assume there exists a relationship $y = g(x)$. From the data, approximate $\tilde{g}(x)$.

Generally, powerful mathematics and algorithms teach us that:

$$\tilde{g}(x) = \sum_{n=0}^{N} c_n T_n(x)$$

where n is the order of the approximation, c_n are coefficients we can solve for, and $T_n(x)$ is a set of basis functions.

More surprisingly, $\lim_{N\to\infty} \tilde{g}(x) = g(x)$.

Functional Approximation

Take some data (can be simulated) $\{x_i, y_i\}_{i=1,\ldots N}$ and assume there exists a relationship $y = g(x)$. From the data, approximate $\tilde{g}(x)$.

Generally, powerful mathematics and algorithms teach us that:

$$\tilde{g}(x) = \sum_{n=0}^{N} c_n T_n(x)$$

where n is the order of the approximation, c_n are coefficients we can solve for, and $T_n(x)$ is a set of basis functions.

More surprisingly, $\lim_{N\to\infty} \tilde{g}(x) = g(x)$.
Any continuous function on a closed and bounded interval can be uniformly approximated on that interval by polynomials to any degree of accuracy (Weierstrass Theorem)

Analytic functions def– taylor approx. see screenshots on phone

3/30

Types of approximation

Local Approximations (accurate only around a point):

- Taylor approximations

Types of approximation

Local Approximations (accurate only around a point):

- Taylor approximations

Global approximations:

- Ordinary regression + interpolation
- Orthogonal polynomials (Chebyshev polynomials)
- Splines

Types of approximation

Local Approximations (accurate only around a point):

- Taylor approximations

Global approximations:

- Ordinary regression + interpolation
- Orthogonal polynomials (Chebyshev polynomials)
- Splines

More generally:

- Finite element methods: use basis functions that are zero on most of the domain (linear interpolation and spline interpolation)
- Spectral methods: use basis functions that are nonzero on most of the domain (Chebyshev polynomials)

"Solving" a Model

Solving a model $\equiv$ getting good approximations for functions (e.g. value functions, policy functions, etc.) that are not known and cannot be derived in closed form (analytically).

"Solving" a Model

Solving a model $\equiv$ getting good approximations for functions (e.g. value functions, policy functions, etc.) that are not known and cannot be derived in closed form (analytically).

- Local approx: Taylor's Theorem: any sufficiently smooth function can be locally approximated by polynomials
  - Advantage: Gives us the exact approximation
  - Disadvantage: Holds locally; e.g. as we move away from the steady state, approximation may deteriorate a lot

"Solving" a Model

Solving a model $\equiv$ getting good approximations for functions (e.g. value functions, policy functions, etc.) that are not known and cannot be derived in closed form (analytically).

- Local approx: Taylor's Theorem: any sufficiently smooth function can be locally approximated by polynomials
  - Advantage: Gives us the exact approximation
  - Disadvantage: Holds locally; e.g. as we move away from the steady state, approximation may deteriorate a lot
- Global approx: any continuous function defined on an interval $[a, b]$ can be uniformly approximated as closely as desired by a polynomial function
  - Disadvantage: Does not tell us which polynomial to pick for approximating the function (tricky)
  - Advantage: Generates a global approximation

Taylor Approximation

Univariate: For $g : \mathbb{R} \to \mathbb{R}$ where $g \in C^{n+1}$ and $x^* \in \mathbb{R}$, n-th order approx:

$$g(x) \approx g(x^*) + (x - x^*)g^{(1)}(x^*) + \frac{1}{2}(x - x^*)^2 g^{(2)}(x^*) + \ldots + \frac{1}{n!}(x - x^*)^n g^{(n)}(x^*) + e_{n+1}$$

where $\lim_{N \to \infty} e_{n+1} = 0$

Required info for approx: $g(x^*)$, $\{g^{(n)}(x^*)\}$.

Taylor Approximation

Univariate: For $g : \mathbb{R} \to \mathbb{R}$ where $g \in C^{n+1}$ and $x^* \in \mathbb{R}$, n-th order approx:

$$g(x) \approx g(x^*) + (x - x^*)g^{(1)}(x^*) + \frac{1}{2}(x - x^*)^2 g^{(2)}(x^*) + \ldots + \frac{1}{n!}(x - x^*)^n g^{(n)}(x^*) + e_{n+1}$$

where $\lim_{N \to \infty} e_{n+1} = 0$

Required info for approx: $g(x^*)$, $\{g^{(n)}(x^*)\}$.

so: $c_n = \frac{g^{(n)}(a)}{n!}$ and $T^n(x) = (x - a)^n$, $c_0 = g(a)$.

Multivariate: For $g : \mathbb{R}^m \to \mathbb{R}$ where $g \in C^{n+1}$ and $x^* \in \mathbb{R}^m$, n-th order approx:

$$g(x) \approx g(x^*) + \sum_{i=1}^{m}(x_i - x_i^*)\frac{\partial g(x_i)}{\partial x_i} +$$
$$\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}(x_i - x_i^*)(x_j - x_j^*)\frac{\partial^2 g(x_i)}{\partial x_i \partial x_j} + ...$$
$$+\frac{1}{n!}\sum_{i_1=1}^{m}...\sum_{i_n=1}^{m}\Pi_{k=1}^{n}(x_{ik} - x_{ik}^*)^n\frac{\partial^n g(x^*)}{\partial x_{i_1}, ...\partial x_{i_n}} + e_{n+1}$$

where $\lim_{n\to\infty} e_{n+1} = 0$.

How good is a local approximation?

- designed to give good approximations near a point x^* but its accuracy may deteriorate very rapidly as we move away

How good is a local approximation?

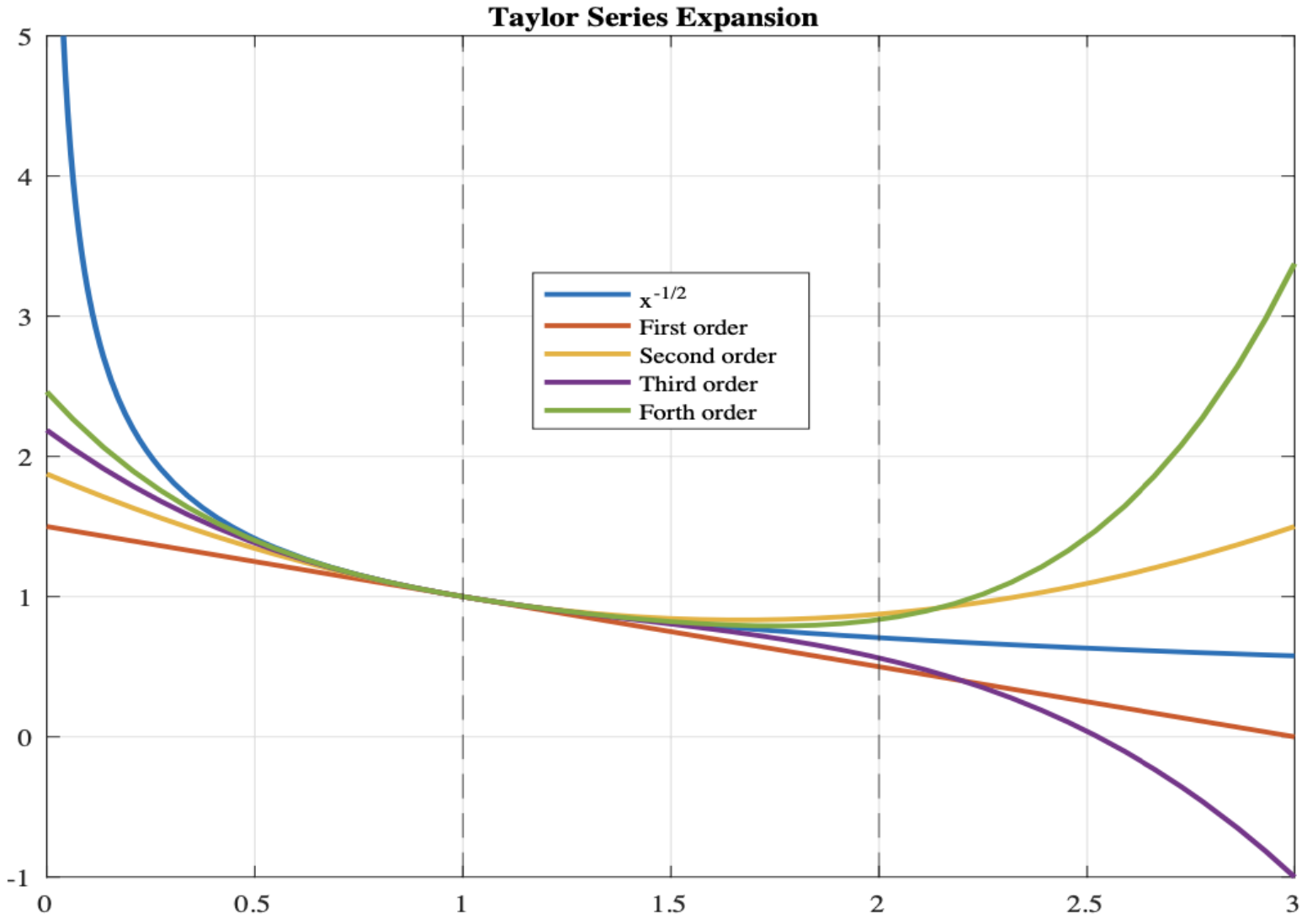
- designed to give good approximations near a point x^* but its accuracy may deteriorate very rapidly as we move away
- how far away can we go from x^* before we start losing accuracy?

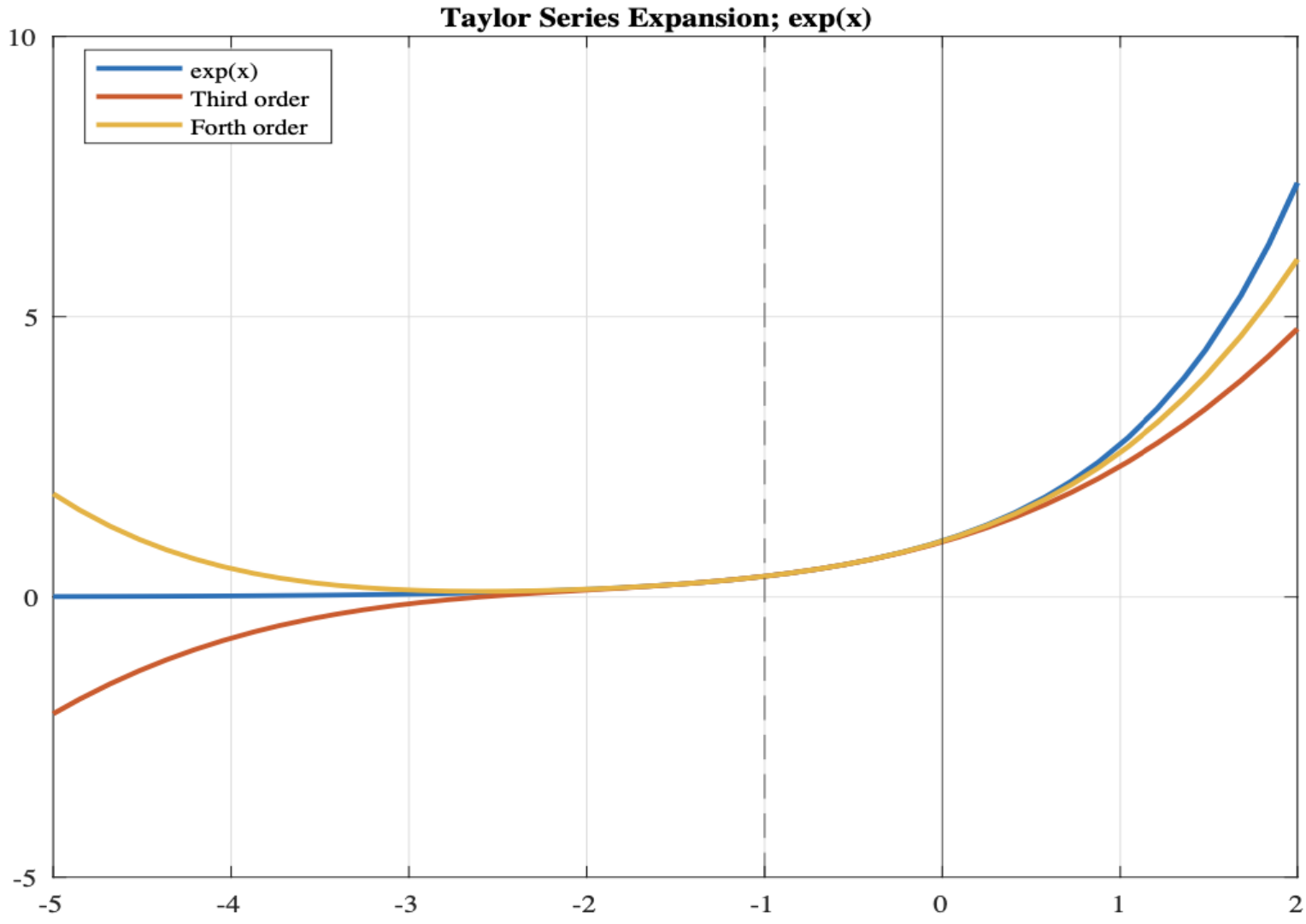
How good is a local approximation?

- designed to give good approximations near a point x^* but its accuracy may deteriorate very rapidly as we move away
- how far away can we go from x^* before we start losing accuracy?

Suppose that we know that the function f (or some derivative of it) has a singularity at a point y. Then, the Taylor approximation around x^* is reliable only within an interval $[x^* - d, x^* + d]$, whered is the distance (in absolute terms) of the two points, x^* and y (see[Judd, 1998, ch. 6]).

Example: Take $g(x) = x^{-1/2}$ that has a singularity at $x = 0$, and approximate around $x^* = 1$. We expect that the approximations will deteriorate close to the singularity, but also according to the theorem for any $x > 2$.


Taylor Series Expansion
5
4
3
2
1
0
-1
0
0.5
1
1.5
2
2.5
3
x^-1/2
First order
Second order
Third order
Forth order


Taylor Series Expansion; exp(x)
exp(x)
Third order
Forth order
10
5
0
-5
-5
-4
-3
-2
-1
0
1
2

Global Approximations

Consider a function $g : [a, b] \to \mathbb{R}$ taking value $g(x)$. A global approximation generates a new function $\tilde{g}(x)$ that is close to $g(x)$ over the domain $[a, b]$, or some subdomain we are interested in.

Global Approximations

Consider a function $g : [a, b] \to \mathbb{R}$ taking value $g(x)$. A global approximation generates a new function $\tilde{g}(x)$ that is close to $g(x)$ over the domain $[a, b]$, or some subdomain we are interested in.

- 'close' $\equiv$ based on some appropriate norm (=distance), e.g. the $p - norm$

Global Approximations

Consider a function $g : [a, b] \to \mathbb{R}$ taking value $g(x)$. A global approximation generates a new function $\tilde{g}(x)$ that is close to $g(x)$ over the domain $[a, b]$, or some subdomain we are interested in.

- 'close' $\equiv$ based on some appropriate norm (=distance), e.g. the $p - norm$

Three types of approx:

- L^p approximation: $g(x)$ is known for all (infinite) values $x \in [a, b]$. Typically not of interest since f not known.
- Regression: some information for $g(x)$ is known, giving us m data points that pin down $n < m$ free parameters that generate an approximation
- Interpolation: some information for $g(x)$ is known, giving us n data points that pin down n free parameters that generate an approximation

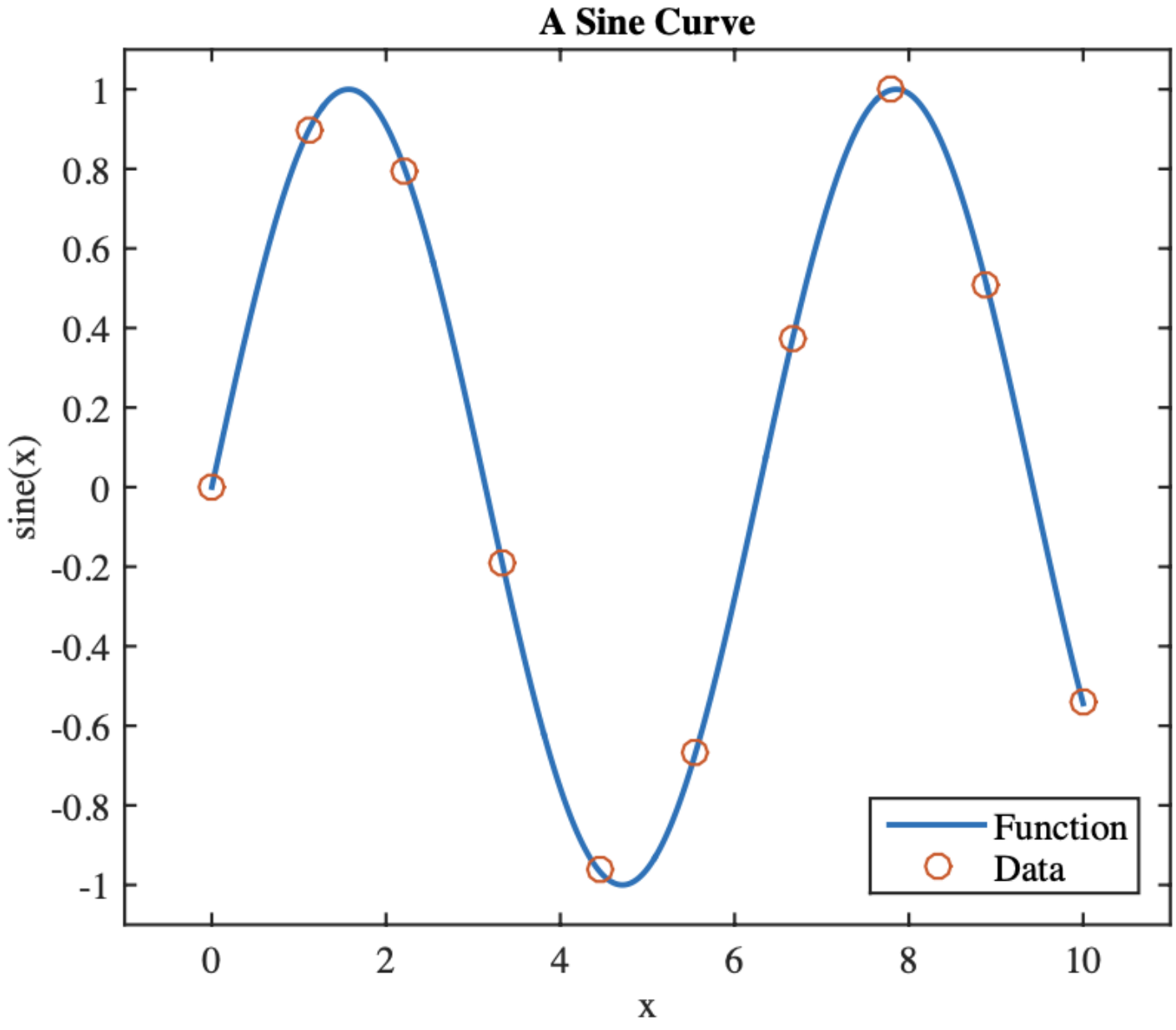
Linear interpolation and Splines

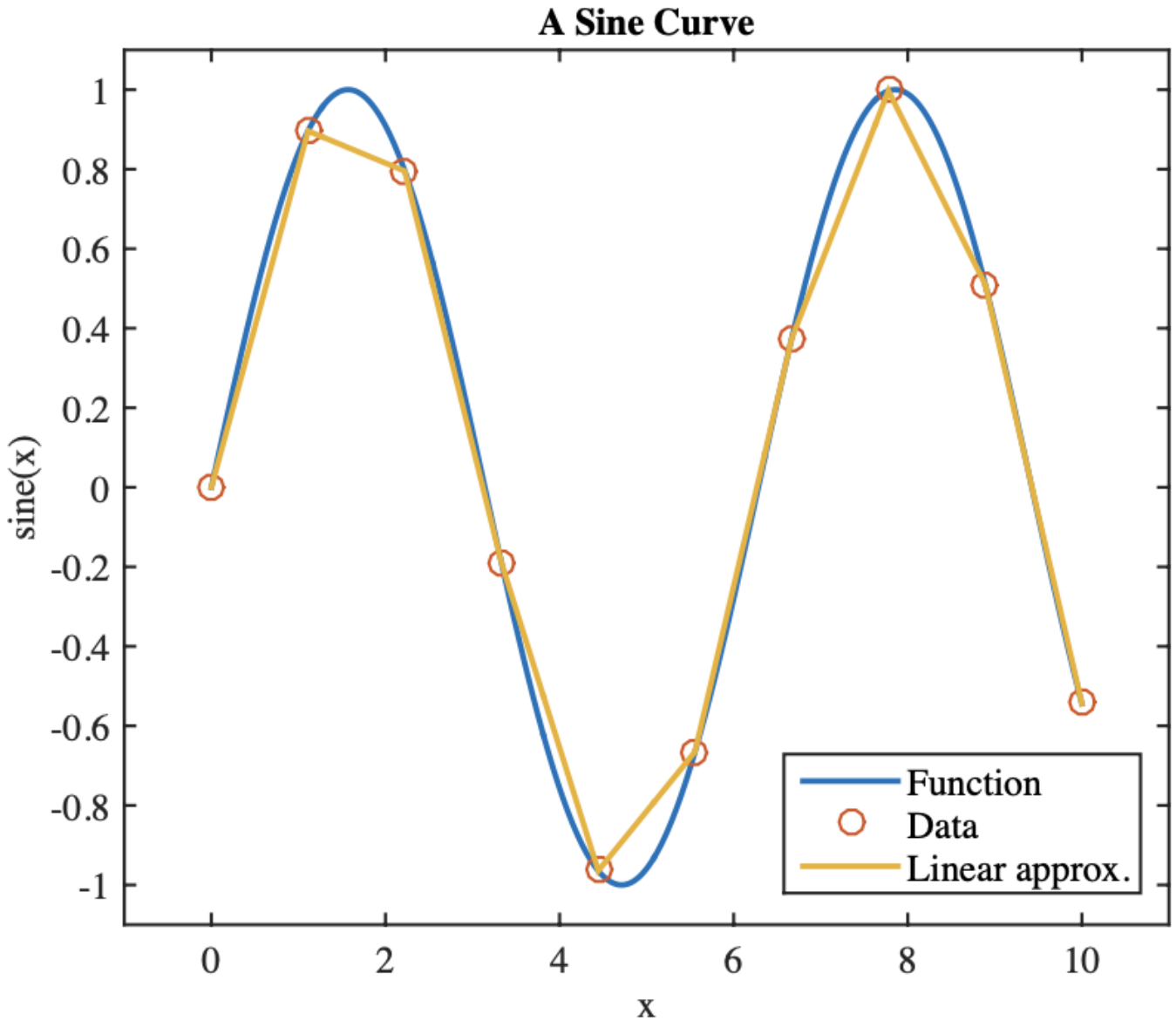
Let's start with simple shape-preserving approximations.

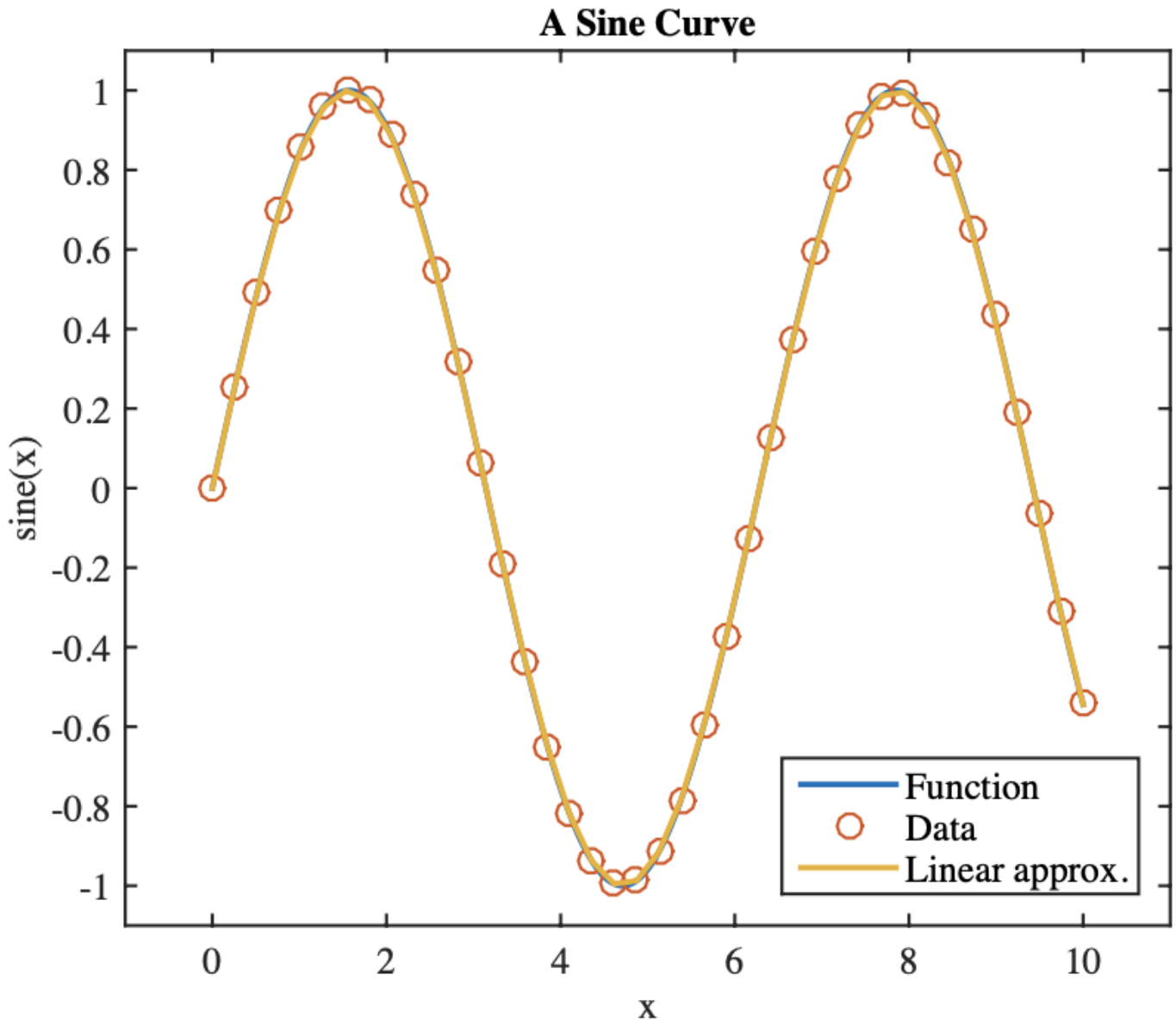
Given data $\{x_i, y_i\}_{i=1}^{M}$, linear interpolation is given by:

$$\tilde{g}(x) = \frac{x - x_i}{x_{i+1} - x_i} y_{i+1} + \left(1 - \frac{x - x_i}{x_{i+1} - x_i}\right) y_i$$

for $x \in [x_i, x_{i+1}]$.


A Sine Curve
sine(x)
x
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
2
4
6
8
10
Function
Data


A Sine Curve
sine(x)
x
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
2
4
6
8
10
Function
Data
Linear approx.


A Sine Curve
sine(x)
x
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
2
4
6
8
10
Function
Data
Linear approx.

Splines

Linear interpolation consists of a series of **local**, **linear** approximations.

Splines follow the same logic, but apply a non-linear local approximation at each point. For any point $x_i \in [x_i, x_{i+1}]$, a cubic spline approx $\tilde{g}(x)$ is given by:

$$\tilde{g}_i(x) = a + bx + cx^2 + dx^3$$

Coefficients given by:

- $\tilde{g}_i'(x_{i+1}) = \tilde{g}_{i+1}'(x_{i+1})$
- $\tilde{g}_i''(x_{i+1}) = \tilde{g}_{i+1}''(x_{i+1})$
- $\tilde{g}_1''(x_1) = \tilde{g}_N''(x_N) = 0$ (end conditions)

Splines in practice

There are $m = 4^{*}\#$ of intervals unknowns.

Lets do this in practice. Data points $(x_1, y_1), (x_2, y_2)$, i.e m=4. What is the system of equations?

$$
\begin{aligned}
a + bx_1 + cx_1^2 + dx_1^3 &= y_1, \\
a + bx_2 + cx_2^2 + dx_2^3 &= y_2, \\
2c + 6dx_1 &= 0, \\
2c + 6dx_2 &= 0
\end{aligned}
$$

4 equations used to solve for 4 coefficients.

Splines in practice

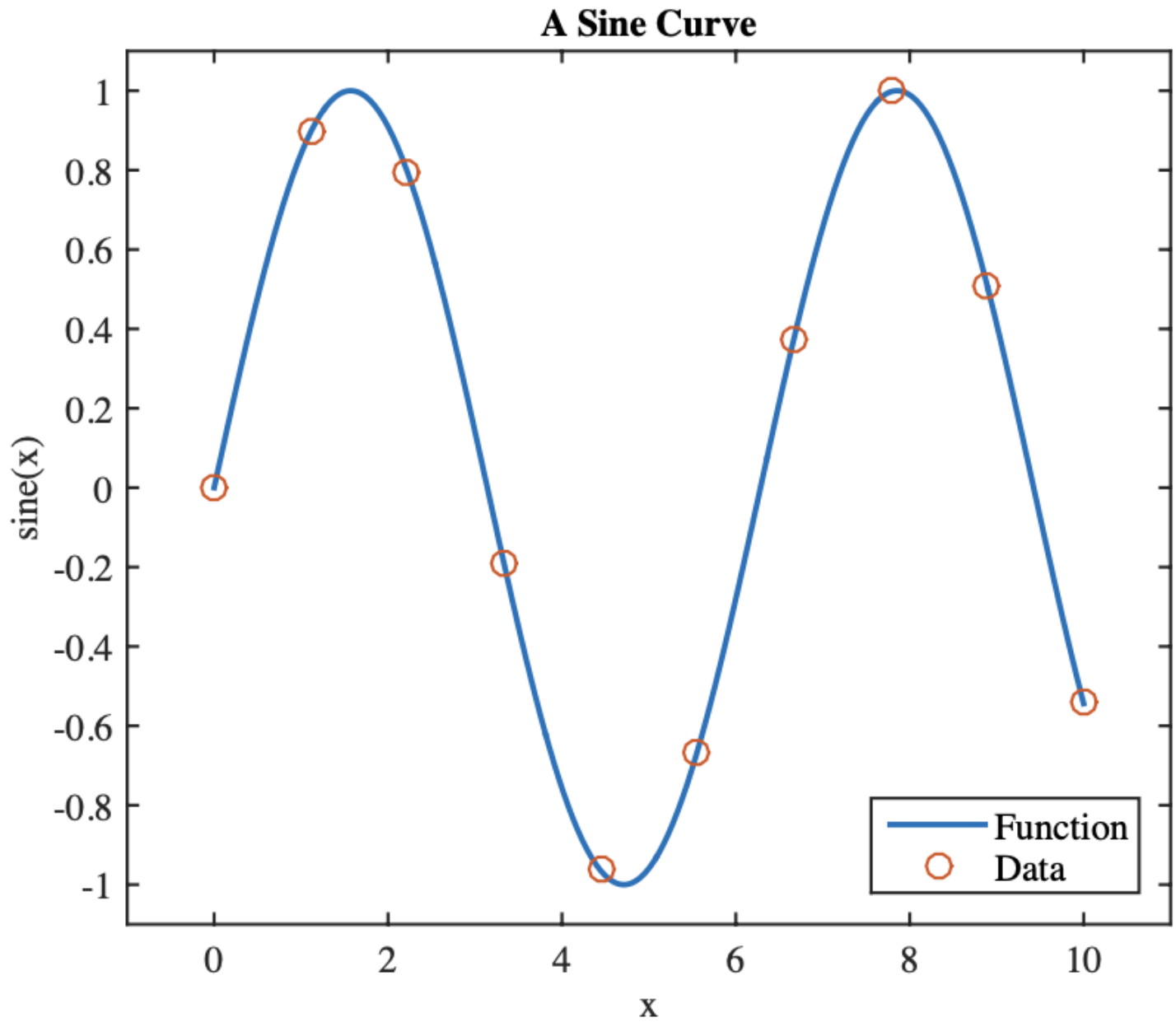
There are $m = 4^{*}\#$ of intervals unknowns.

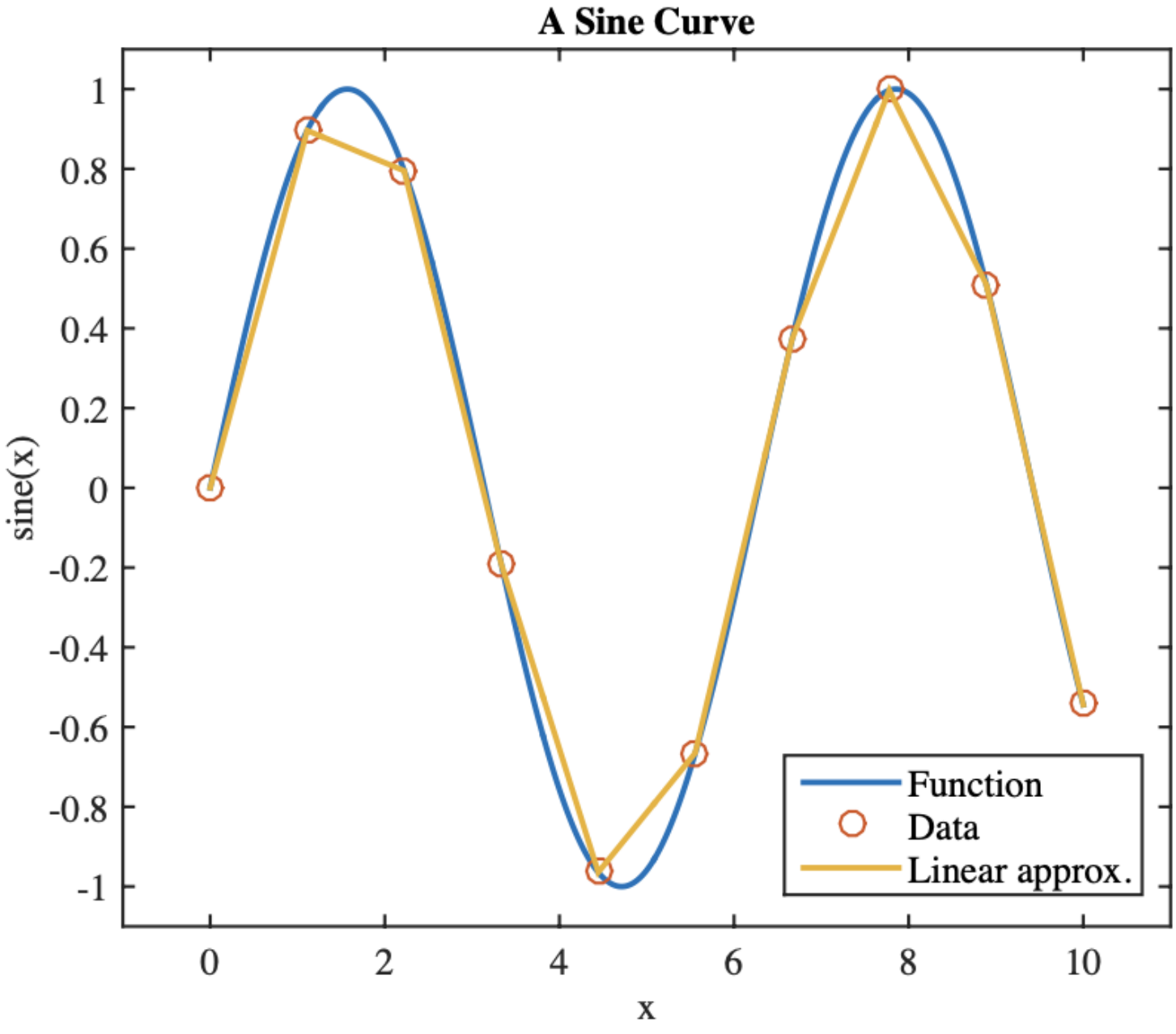
Lets do this in practice. Data points $(x_1, y_1), (x_2, y_2)$, i.e m=4. What is the system of equations?

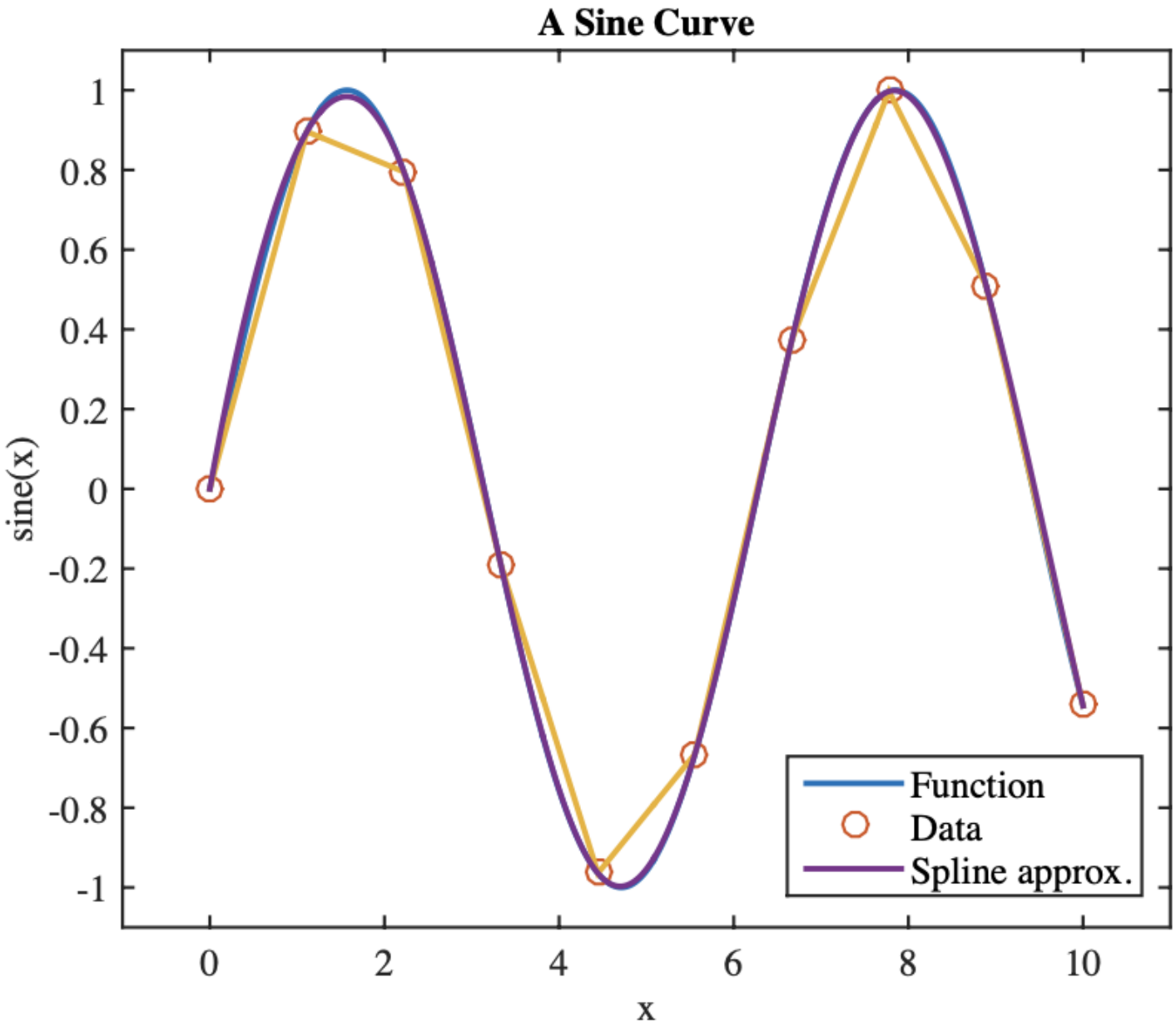
$$
\begin{aligned}
a + bx_1 + cx_1^2 + dx_1^3 &= y_1, \\
a + bx_2 + cx_2^2 + dx_2^3 &= y_2, \\
2c + 6dx_1 &= 0, \\
2c + 6dx_2 &= 0
\end{aligned}
$$

4 equations used to solve for 4 coefficients.

Start with linear interpolation (more robust to kinks). If true functions looks smooth, switch to spline to get better results.


A Sine Curve
sine(x)
x
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
2
4
6
8
10
Function
Data


A Sine Curve
sine(x)
x
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
2
4
6
8
10
Function
Data
Linear approx.


A Sine Curve
sine(x)
x
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
2
4
6
8
10
Function
Data
Spline approx.

Implementation and Matlab HELP

https://uk.mathworks.com/help/matlab/ref/interp1.html

Implementation and Matlab HELP

https://uk.mathworks.com/help/matlab/ref/interp1.html

The command is:

- *interp1(x,y,z,'linear')* for linear interpolation
- *interp1(x,y,z,'spline')* for spline interpolation
- *interp1(x,y,z,'spchip')* for pchip interpolation

Implementation and Matlab HELP

https://uk.mathworks.com/help/matlab/ref/interp1.html

The command is:

- *interp1(x,y,z,'linear')* for linear interpolation
- *interp1(x,y,z,'spline')* for spline interpolation
- *interp1(x,y,z,'spchip')* for pchip interpolation

Recently, Matlab is phasing this out and replacing with g =*griddedInterpolant(x,y)*. Then g(z) to generate approximation.

Description

`vq = interp1(x,v,xq)` returns interpolated values of a 1-D function at specific query points using linear interpolation. Vector `x` contains the sample points, and `v` contains the corresponding values, $v(x)$. Vector `xq` contains the coordinates of the query points.

If you have multiple sets of data that are sampled at the same point coordinates, then you can pass `v` as an array. Each column of array `v` contains a different set of 1-D sample values.

Examples

Interpolation of Coarsely Sampled Sine Function

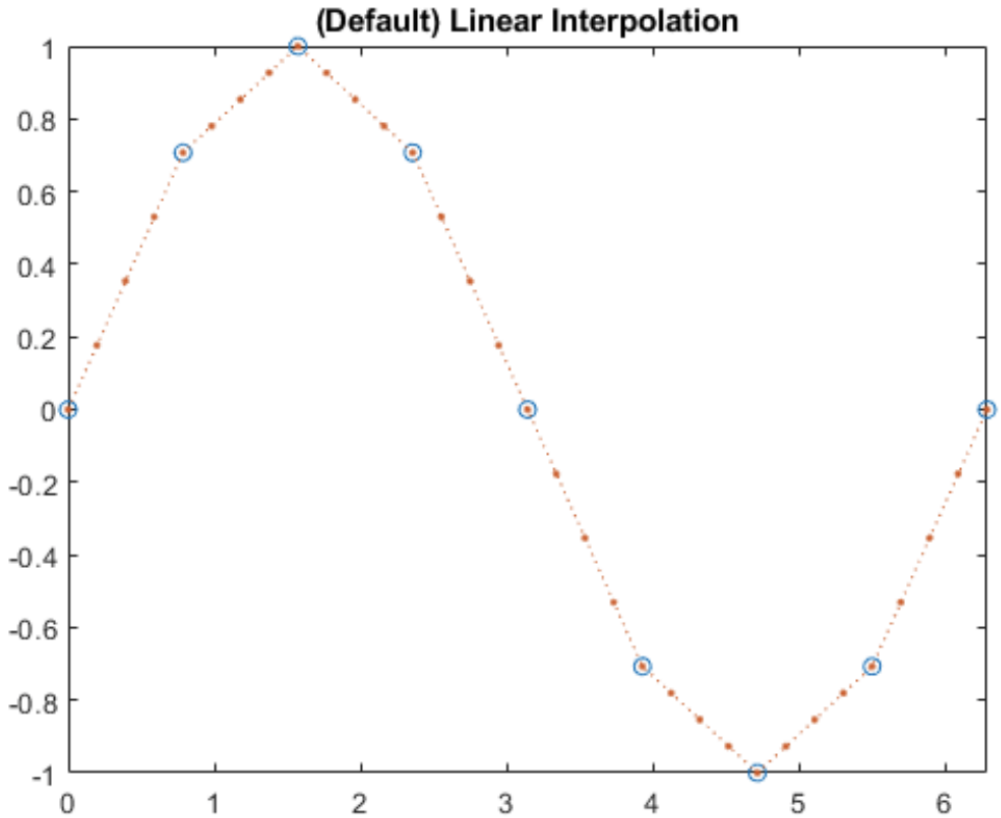
Define the sample points, x, and corresponding sample values, v.

```
x = 0:pi/4:2*pi;
v = sin(x);
```

Define the query points to be a finer sampling over the range of x.

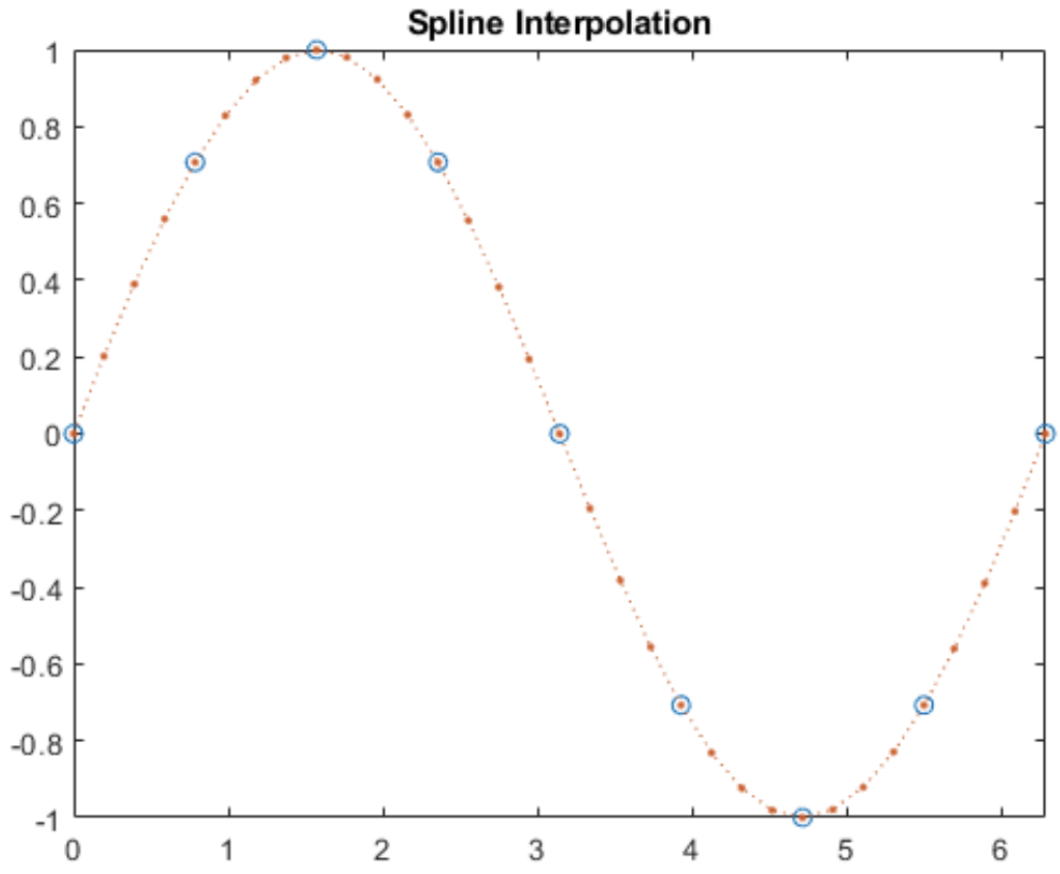
```
xq = 0:pi/16:2*pi;
```

```
figure
vq1 = interp1(x,v,xq);
plot(x,v,'o',xq,vq1,':.');
xlim([0 2*pi]);
title('(Default) Linear Interpolation');
```



Now evaluate v at the same points using the `'spline'` method.

```
figure
vq2 = interp1(x,v,xq,'spline');
plot(x,v,'o',xq,vq2,':.');
xlim([0 2*pi]);
title('Spline Interpolation');
```



Spectral Methods

Linear interpolation and splines are piece-wise (local) approximations. Each basis functions is 0 on most of the domain.

Spectral methods uses instead one large polynomial for the entire domain:

- Benefits: For "well-behaved" functions, these will do quite well with very few parameters to be identified $\Longrightarrow$ speed
- Costs: Can display oscillating and very weird behavior

Monomials

To approximate $f : [a, b] \to \mathbb{R}$, $f \in C[a, b]$. The vector space is spanned by all monomials: $\{1, x, x^2, ..., x^i, ...\}$.

Then $g(x) = \sum_{i=0}^{N} c_i x^i$.

* Monomials rarely ok due to multicollinearity

Monomials

To approximate $f : [a, b] \to \mathbb{R}$, $f \in C[a, b]$. The vector space is spanned by all monomials: $\{1, x, x^2, ..., x^i, ...\}$.

Then $g(x) = \sum_{i=0}^{N} c_i x^i$.

* Monomials rarely ok due to multicollinearity

$\Longrightarrow$ Families of orthogonal polynomials do a good job at avoiding this problem.

Monomials

To approximate $f : [a, b] \to \mathbb{R}$, $f \in C[a, b]$. The vector space is spanned by all monomials: $\{1, x, x^2, ..., x^i, ...\}$.

Then $g(x) = \sum_{i=0}^{N} c_i x^i$.

* Monomials rarely ok due to multicollinearity

$\Longrightarrow$ Families of orthogonal polynomials do a good job at avoiding this problem.

Define $\omega : [a, b] \to \mathbb{R}$, $\omega(x) > 0 and \int_{[a,b]} \omega(x) dx < \infty$. A family of polynomials $\mathscr{P} = \{\phi_1(x), \phi_2(x), ...\}$ is orthogonal with respect to the weight function omega $\omega(x)$ if:

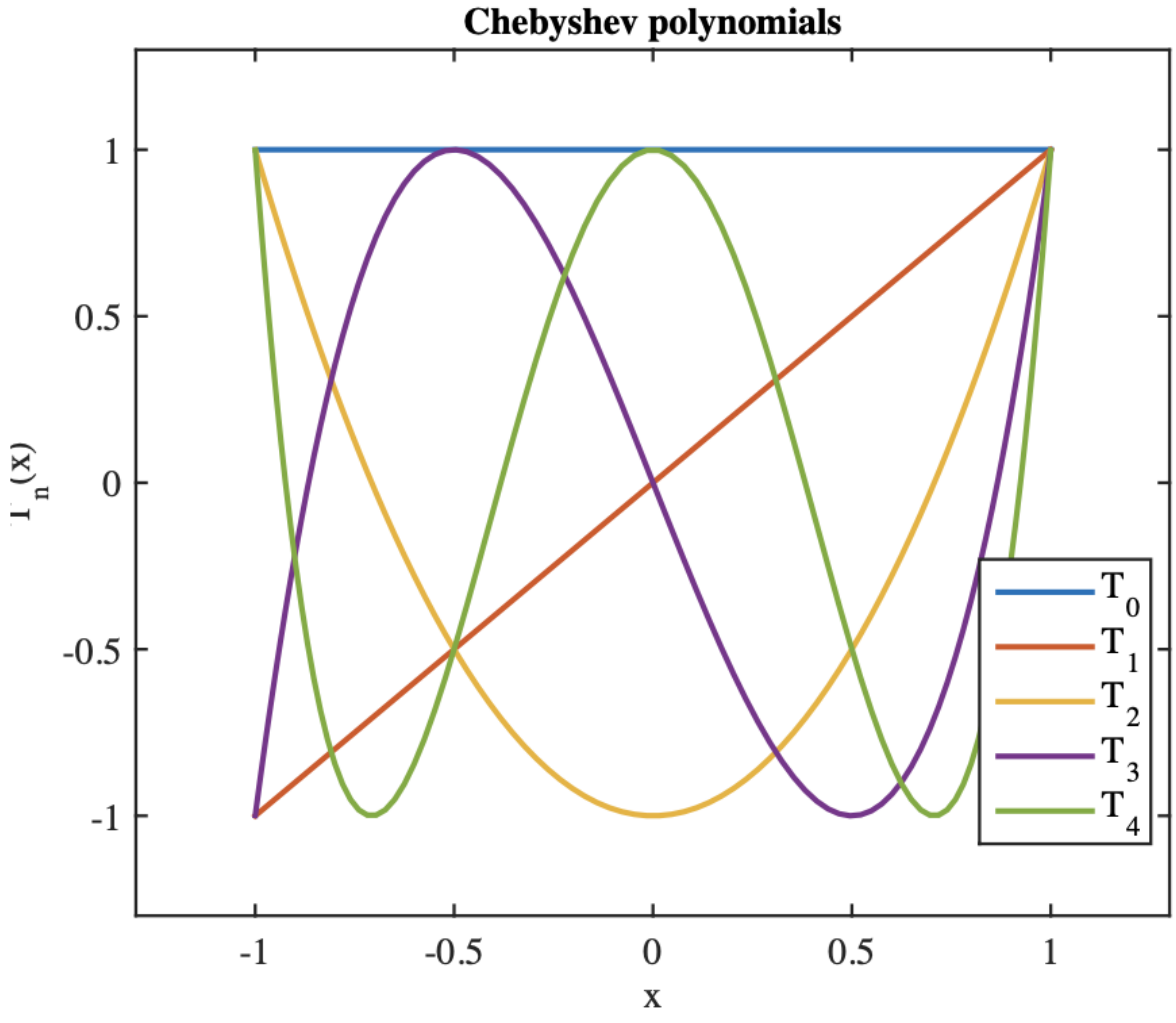
$$\int_{[a,b]} \omega(x)\phi_i(x)\phi_j(x) = 0,$$

for $\phi_i \neq \phi_j$.

Chebyshev Polynomials

* $T^n : [-1, 1] \to [-1, 1]$
* Recursively defined, $T_0 = 1, T_1 = x,\ T_2 = 2xT_{n-1} - T_{n-2}...$

Advantages: Guarantee uniform convergence and provide grid points for approximation!


Chebyshev polynomials
1
0.5
0
-0.5
-1
T_n(x)
-1
-0.5
0
0.5
1
x
T0
T1
T2
T3
T4

The general Chebyshev polynomials order n are defined as

$$\tilde{T}^n[a,b] \to \mathbb{R},$$
$$\tilde{T}^n(x) = T^n(h(x)) = T^n\left(\frac{2(x-a)}{b-a} - 1\right)$$

How do we calculate coefficients such that $g(x) = \sum_n c_n \tilde{T}^n(x)$, and the weighted squared error between $g(x)$ and $\tilde{g}(x)$ is minimized?

- based on numerical analysis theorems, omitted here, see Judd

$\Longrightarrow$ cookbook approach on next slide

Chebyshev polynomial algorithm

For $\tilde{g}(x) = \sum_{i=0}^{n} c_i \tilde{T}(x) = \sum_{i=0}^{n} c_i T(h(x))$:

1. Choose order n and number of grid points (nodes), $m > n + 1$
2. Collocation points (roots of polynomial) given by:

$$z_j = cos(\frac{(2j-1)\pi}{2m}),\ j = 1, ..., m$$

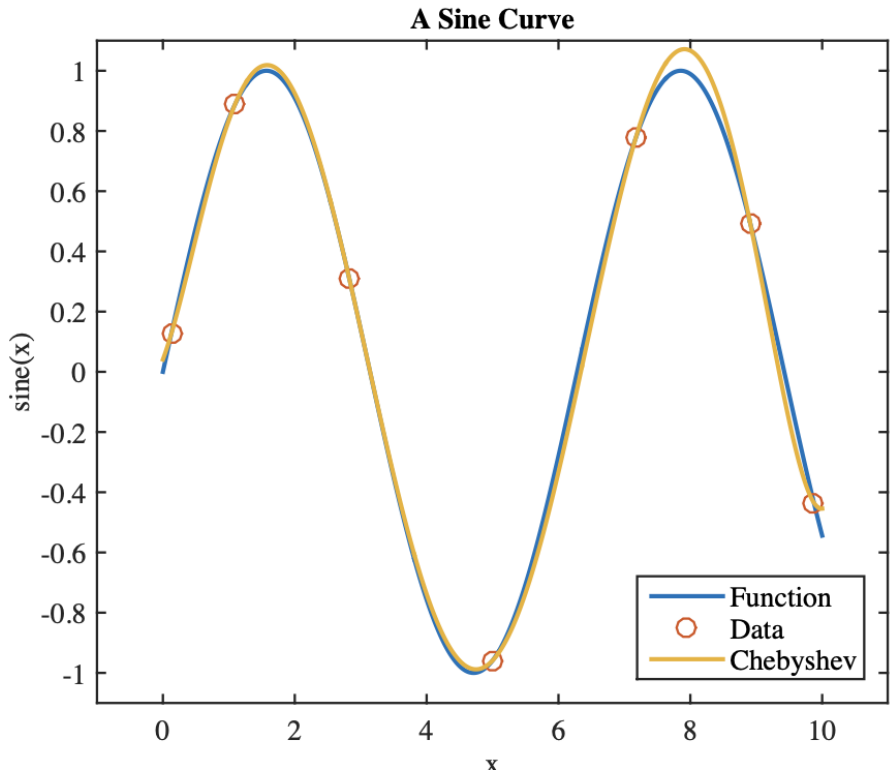
3. Transform these points from $[-1, 1]$ to $[a, b]$:

$$x_j = \frac{(z_j + 1)(b - a)}{2} + a$$

4. Get $y_j = f(x_j)$ (can be a guess within a computational algo)
5.

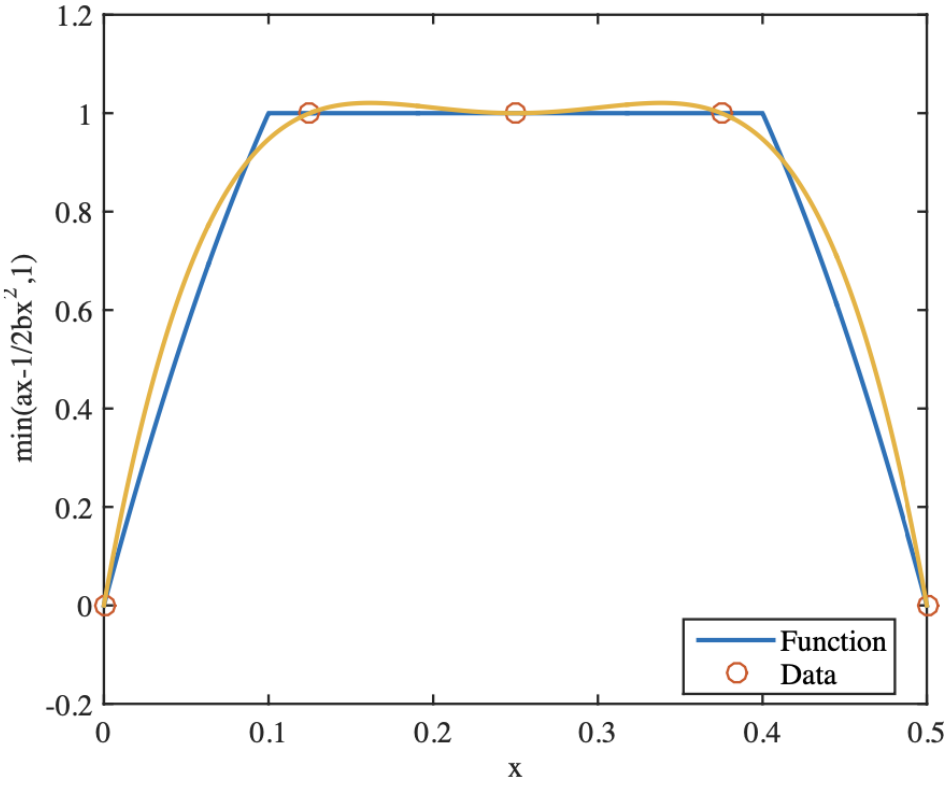
$$c_0 = \frac{1}{m} \sum_{j=1}^{m} g(x_j),\ c_i = \frac{2}{m} \sum_{j=1}^{m} g(x_j) T_i(x_j),\quad i = 2, ...n$$

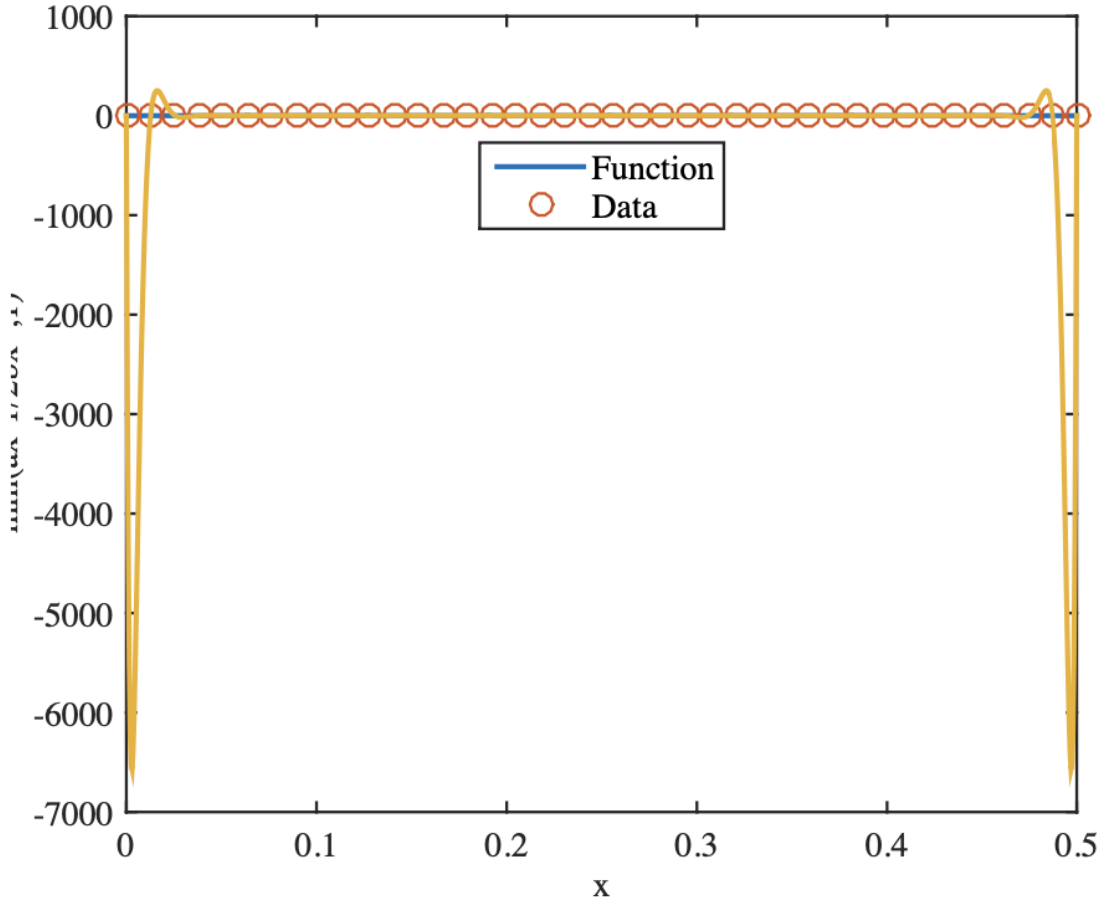
Chebyshev Polynomials



Go to page 67

Chebyshev Polynomials




1000
0
-1000
-2000
-3000
-4000
-5000
-6000
-7000
0
0.1
0.2
0.3
0.4
0.5
x
Function
Data

Non-Linear Equations: A practical approach

* Most economic models can be represented as:

$$f(x, y) = 0,$$

  where $f(\cdot)$ is a known function.

* Moreover, there exists a mapping $y = g(x)$,

Non-Linear Equations: A practical approach

* Most economic models can be represented as:

$$f(x, y) = 0,$$

where $f(\cdot)$ is a known function.

* Moreover, there exists a mapping $y = g(x)$,

So, an economic model can often be understood as:

$$f(x, g(x)) = 0$$

Non-Linear Equations: A practical approach

* Most economic models can be represented as:

$$f(x, y) = 0,$$

  where $f(\cdot)$ is a known function.

* Moreover, there exists a mapping $y = g(x)$,

So, an economic model can often be understood as:

$$f(x, g(x)) = 0$$

and, useful for our algorithms, $f(x, \hat{y}) \neq 0$ for $\hat{y} \neq y$.

Non-Linear Equations: A practical approach

* Most economic models can be represented as:

$$f(x, y) = 0,$$

where $f(\cdot)$ is a known function.

* Moreover, there exists a mapping $y = g(x)$,

So, an economic model can often be understood as:

$$f(x, g(x)) = 0$$

and, useful for our algorithms, $f(x, \hat{y}) \neq 0$ for $\hat{y} \neq y$.

- Where $g(x)$ is not known, use a functional approximation $\tilde{g}(x)$!

Two-period consumption-savings problem

$$\begin{aligned} \max\ & u(c_1) + \beta u(c_2) \\ s.t.\quad & c_1 + a_1 = m, \\ & c_2 = (1+r)a_1 \end{aligned}$$

Two-period consumption-savings problem

$$\begin{aligned} \max\ & u(c_1) + \beta u(c_2) \\ s.t.\quad & c_1 + a_1 = m, \\ & c_2 = (1+r)a_1 \end{aligned}$$

The unconstrained problem is :

$$\max\ u(m - a_1) + \beta u((1+r)a_1)$$

Two-period consumption-savings problem

$$\begin{aligned} \max\, & u(c_1) + \beta u(c_2) \\ s.t. \quad & c_1 + a_1 = m, \\ & c_2 = (1+r)a_1 \end{aligned}$$

The unconstrained problem is :

$$\max u(m - a_1) + \beta u((1+r)a_1)$$

The FOC wrt a_1 is :

$$u'(m - a_1) + \beta u'((1+r)a_1)(1+r) = 0$$

Two-period consumption-savings problem

$$\begin{aligned} \max\ & u(c_1) + \beta u(c_2) \\ s.t.\quad & c_1 + a_1 = m, \\ & c_2 = (1+r)a_1 \end{aligned}$$

The unconstrained problem is :

$$\max u(m - a_1) + \beta u((1+r)a_1)$$

The FOC wrt a_1 is :

$$u'(m - a_1) + \beta u'((1+r)a_1)(1+r) = 0$$

or $f(a_1) = 0$ as promised, where $a_1 = g(m)$ or $c_1 = h(m) = m - g(m)$.

The gist of computational methods in Econ:

1. Solve $f(a_1) = 0$ on a grid for m to get data $\{a_{1,i}, m_{1,i}\}$
2. Approximate g with $\tilde{g}$ to get policy function, using $\{a_{1,i}, m_{1,i}\}$

and sometimes in reverse order.

NLE Solvers: Bisection

$f : \mathbb{R} \to \mathbb{R}$, $f(a) < 0, f(b) > 0$ for some $[a, b]$.

NLE Solvers: Bisection

$f : \mathbb{R} \to \mathbb{R}$, $f(a) < 0, f(b) > 0$ for some $[a, b]$.

IVT $\implies$ $f(x^*) = 0$ for $x^* \in [a, b]$

NLE Solvers: Bisection

$f : \mathbb{R} \to \mathbb{R}$, $f(a) < 0, f(b) > 0$ for some $[a, b]$.

IVT $\implies$ $f(x^*) = 0$ for $x^* \in [a, b]$

* **Bisection Method.**
  1. Find $a, b : f(a)f(b) < 0$, choose error tolerance ϵ
  2. $x' = (a + b)/2$
  3.
     - if $f(x') = 0$ done
     - if $f(x')f(b) < 0$, $a = x'$
     - if $f(x')f(a) < 0$, $b = x'$
  4. repeat until $|a - b| < \epsilon$

NLE Solvers: Newton

* **Newton Method.**

1. Choose error tolerance δ and ϵ (change and level)
2. Make a guess x^{old}
3. Take first order approx of f at x^{old}:

$$f(x) \approx v(x) = f(x^{old}) + f'(x^{old})(x - x^{old})$$

4. Set to zero and solve for $x^{new} = x^{old} - \dfrac{f(x^{old})}{f'(x^{old})}$
5. repeat until $\dfrac{|x^{new} - x^{old}|}{1 + x^{new}} < \delta$ and $f(x^{new}) < \epsilon$

NLE Solvers: Newton

* **Newton Method.**

1. Choose error tolerance δ and ϵ (change and level)
2. Make a guess x^{old}
3. Take first order approx of f at x^{old}:

$$f(x) \approx v(x) = f(x^{old}) + f'(x^{old})(x - x^{old})$$

4. Set to zero and solve for $x^{new} = x^{old} - \dfrac{f(x^{old})}{f'(x^{old})}$
5. repeat until $\dfrac{|x^{new} - x^{old}|}{1 + x^{new}} < \delta$ and $f(x^{new}) < \epsilon$

Converges quickly, but reliant on a good initial guess.

Back to Consumption-Savings

$$f(x) = u'(m - x) + \beta u'((1 + r)x)(1 + r) = 0$$

Back to Consumption-Savings

$$f(x) = u'(m - x) + \beta u'((1 + r)x)(1 + r) = 0$$

$$f'(x) = u''(m - x) + \beta u''((1 + r)x)(1 + r)^2 = 0$$

Back to Consumption-Savings

$$f(x) = u'(m-x) + \beta u'((1+r)x)(1+r) = 0$$
$$f'(x) = u''(m-x) + \beta u''((1+r)x)(1+r)^2 = 0$$

1. choose an m and x_0
2. $x_{k+1} = x_k - \dfrac{f(x_k)}{f'(x_k)}$ for $k = \{0, ..., K\}$
3. until convergence (see previous slide)

Multi-dimensional Newton Method

$$f(x) = \begin{Bmatrix} f_1(x) & f_2(x) & \cdots & f_N(x) \end{Bmatrix}'$$

Take the first order approximation:

$$f(x) \approx v(x) = f(x^{old}) + J_{x^{old}}(x^{old})(x - x^{old})$$

where J_x is a Jacobian:

$$J_x = \begin{Bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \frac{\partial f_2(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} & \cdots & \frac{\partial f_2(x)}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_n(x)}{\partial x_1} & \frac{\partial f_n(x)}{\partial x_2} & \cdots & \frac{\partial f_n(x)}{\partial x_n} \end{Bmatrix}$$

Finally, $x^{new} = x^{old} - J^{-1}_{x^{old}} \frac{f(x^{old})}{f(x^{new})}$.

Once More, Consumption-Savings

$$f(x) = u'(m - x) + \beta u'((1 + r)x)(1 + r) = 0$$

Once More, Consumption-Savings

$$f(x) = u'(m - x) + \beta u'((1 + r)x)(1 + r) = 0$$

Solved for $a_1 = x$ at a given m.

Once More, Consumption-Savings

$$f(x) = u'(m - x) + \beta u'((1 + r)x)(1 + r) = 0$$

Solved for $a_1 = x$ at a given m.

Now:

1. find $a_{1,i}$ for $m_i \in \{m_1, ..., m_N\}$
2. now we have data points $(a_{1,1}, ..., a_{1,N})$ and $(m_1, ..., m_N)$
3. use functional approximation techniques from before to construct policy function $a_1(m)$ and $c_1(m) = m - a_1(m)$.