Numerical Methods Bootcamp

Lecture 2
Functional approximation and solving nonlinear equations

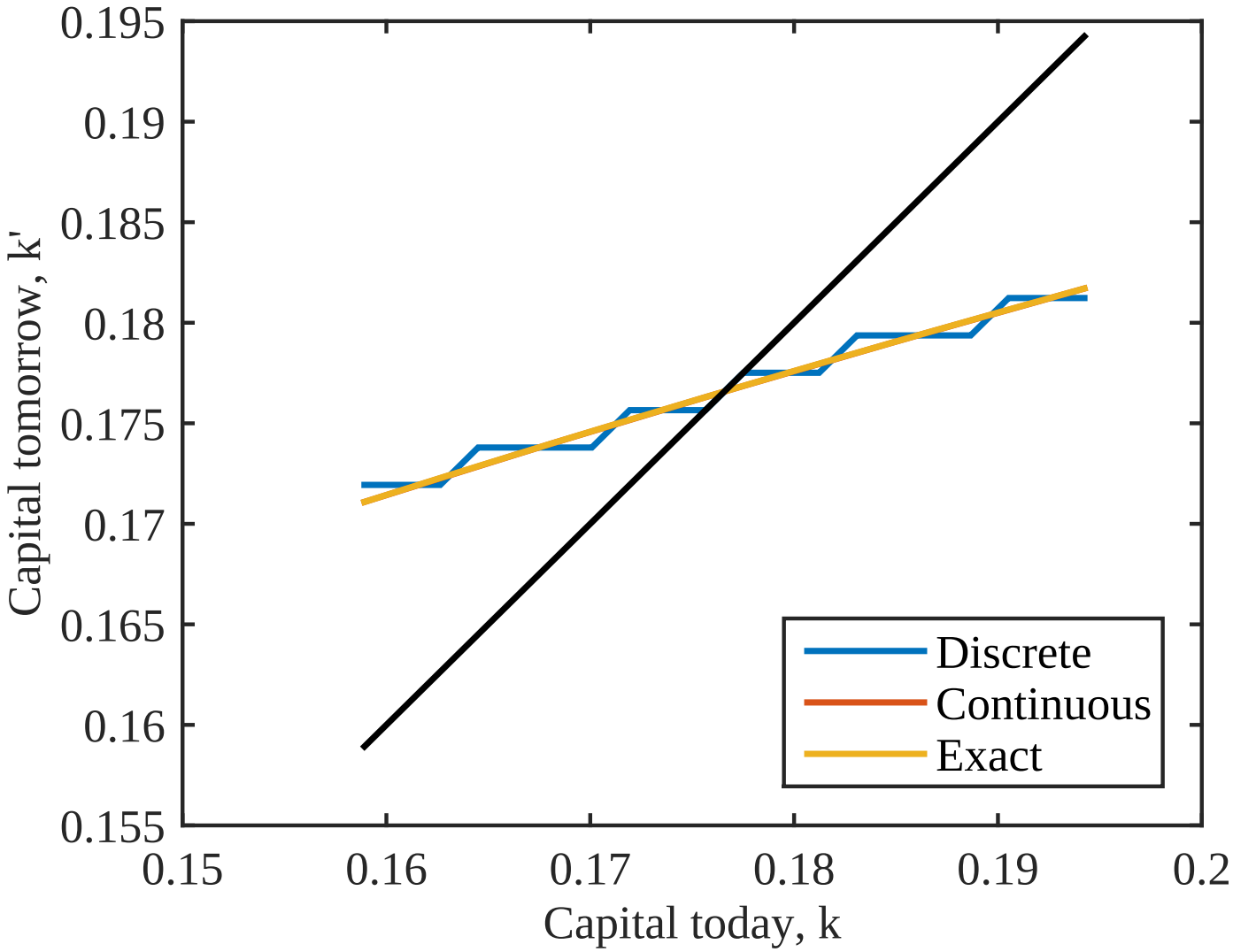
Pontus Rendahl
rpk22@cam.ac.uk

2018

Introduction

- If everything went according to plan, you managed to solve the stochastic growth model yesterday, using what is called discretised value function iteration.
- I did too but I compared two different techniques
  1. First, the one you used
  2. And second, on in which choices k' are not restricted to belong to the grid itself.
- Let's take a look at the difference in the deterministic version

Ramsey growth model, $\delta = 1$



Introduction

- This is quite something.
- It turns out that using 5 nodes for the “continuous” case is more accurate as using 500 nodes in the discrete. And even more accurate than using 1,000 nodes in some metrics.
- So it really seems like we should favour these methods!

Introduction

- How did I do this?
- Recall the Bellman equation

$$V(k) = \max_{k' \in \mathcal{K}} \{u(f(k) + (1-\delta)k - k') + \beta V(k')\}$$

- The only thing I did was relaxing the constraint $k' \in \mathcal{K}$
- But how can I do that?
  - Using functional approximations together nonlinear equation solvers.

Introduction

- True, I do not know what $V(k')$ is for some k' not in $\mathcal{K}$.
- But I can perhaps approximate $V(k')$ for those situations.
- Call this approximation $\hat{V}(k')$.
- I then used a nonlinear equation solver to find the solution to

$$u'(f(k) + (1 - \delta)k - k') = \beta \hat{V}'(k')$$

- And iterated until convergence!

Introduction

- To learn how to do things like this you will need to know two things
  1. Functional approximations, and
  2. Nonlinear equation solvers.

Functional approximations

- The idea behind functional approximations is quite simple
  - We are given some kind of data $(x_i, y_i)_{i=1}^{N}$ from which we know there exist a relation $y = g(x)$
  - From this data we try to find an approximation of $g(\cdot)$, let's call it $\tilde{g}(\cdot)$
- Every approximation I have seen can be boiled down to the form

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

where c_i are coefficients to be solved for in order to have a "good" fit, and $T_n(x)$ are the basis functions.

Functional approximations

- An Nth order Taylor approximation around $x = a$ is, for instance given by the above approximation schedule with

$$c_n = \frac{g^{(n)}(a)}{n!} \quad \text{and} \quad T_n(x) = (x - a)^n$$

and $c_0 = g(a)$

Functional approximations

- There are two categories of functional approximations
  - Finite element methods which use basis functions that are zero on most of the domain. Examples are linear interpolation and spline interpolation.
  - Spectral methods which use basis functions that are nonzero on most of the domain. Examples are Chebyshev polynomials.

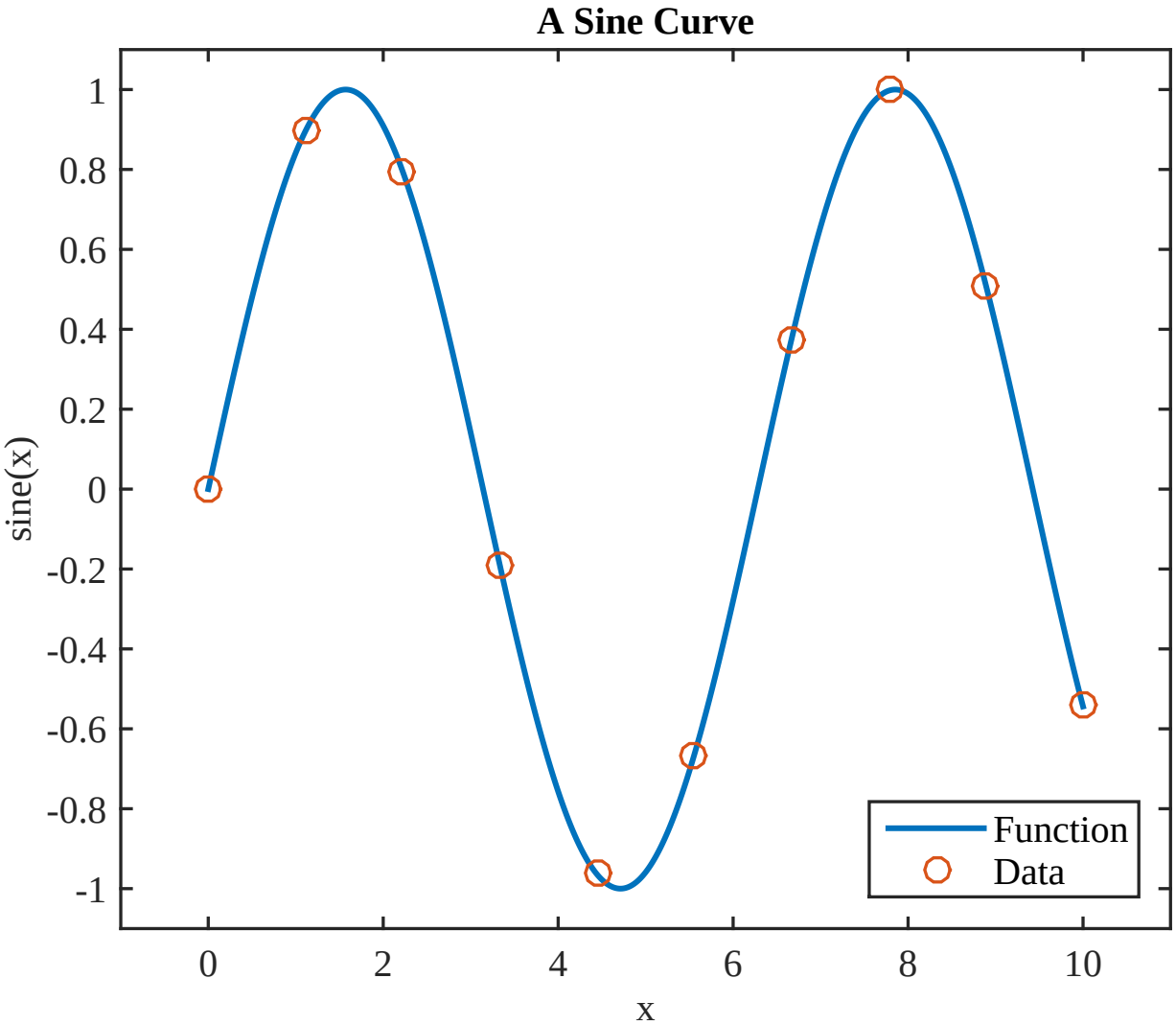
Functional approximations: Linear interpolation

- Given a collection of data $(x_i, y_i)_{i=1}^N$, linear interpolation is given by

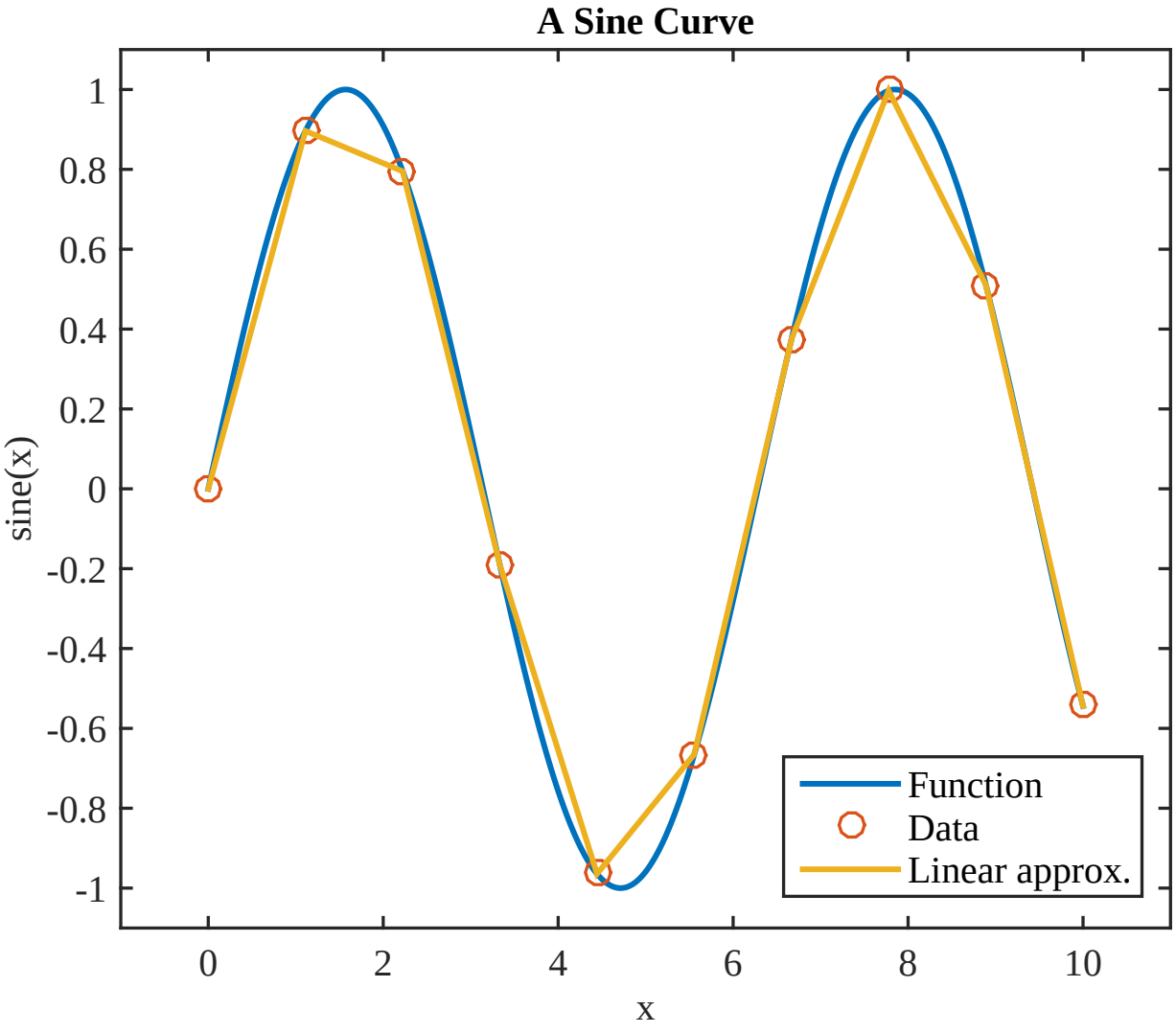
$$\tilde{g}_i(x) = \frac{x - x_i}{x_{i+1} - x_i} y_{i+1} + (1 - \frac{x - x_i}{x_{i+1} - x_i}) y_i$$

  if $x \in [x_i, x_{i+1}]$
- Take the sine function for instance.

Functional approximations: Linear interpolation



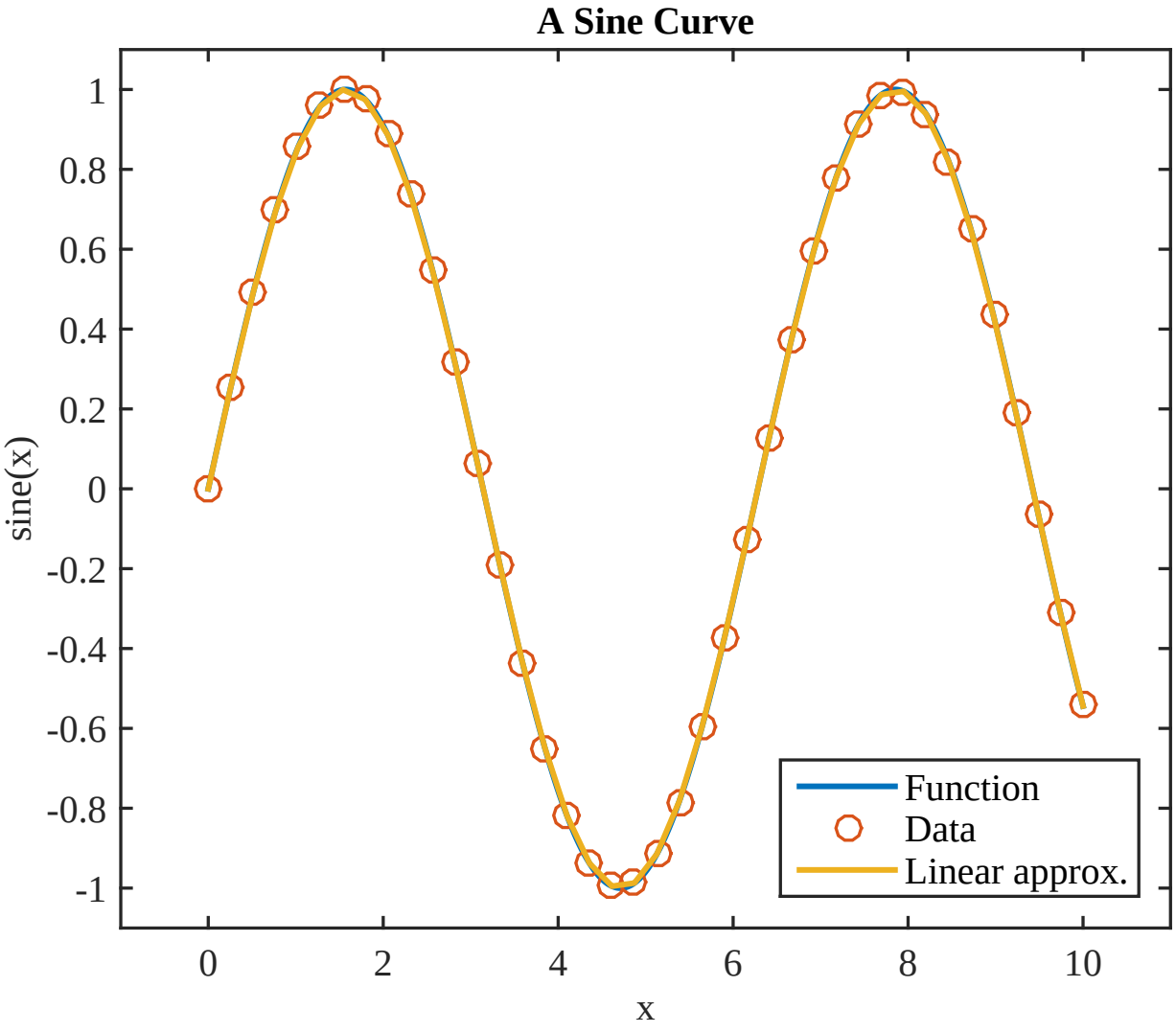
Functional approximations: Linear interpolation



Functional approximations: Linear interpolation

- Perhaps that doesn't look too great.
- But I can always add more data and improve my approximation.
- Before I had 10 data points. How about 40?

Functional approximations: Linear interpolation



Functional approximations: Splines

- As with linear interpolation, splines use local approximations.
- The difference is that the local approximation is not linear.
- In particular, for any $x \in [x_i, x_{i+1}]$ a cubic spline approximation $\tilde{g}$ is given by

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

- How can we identify these (four) parameters?

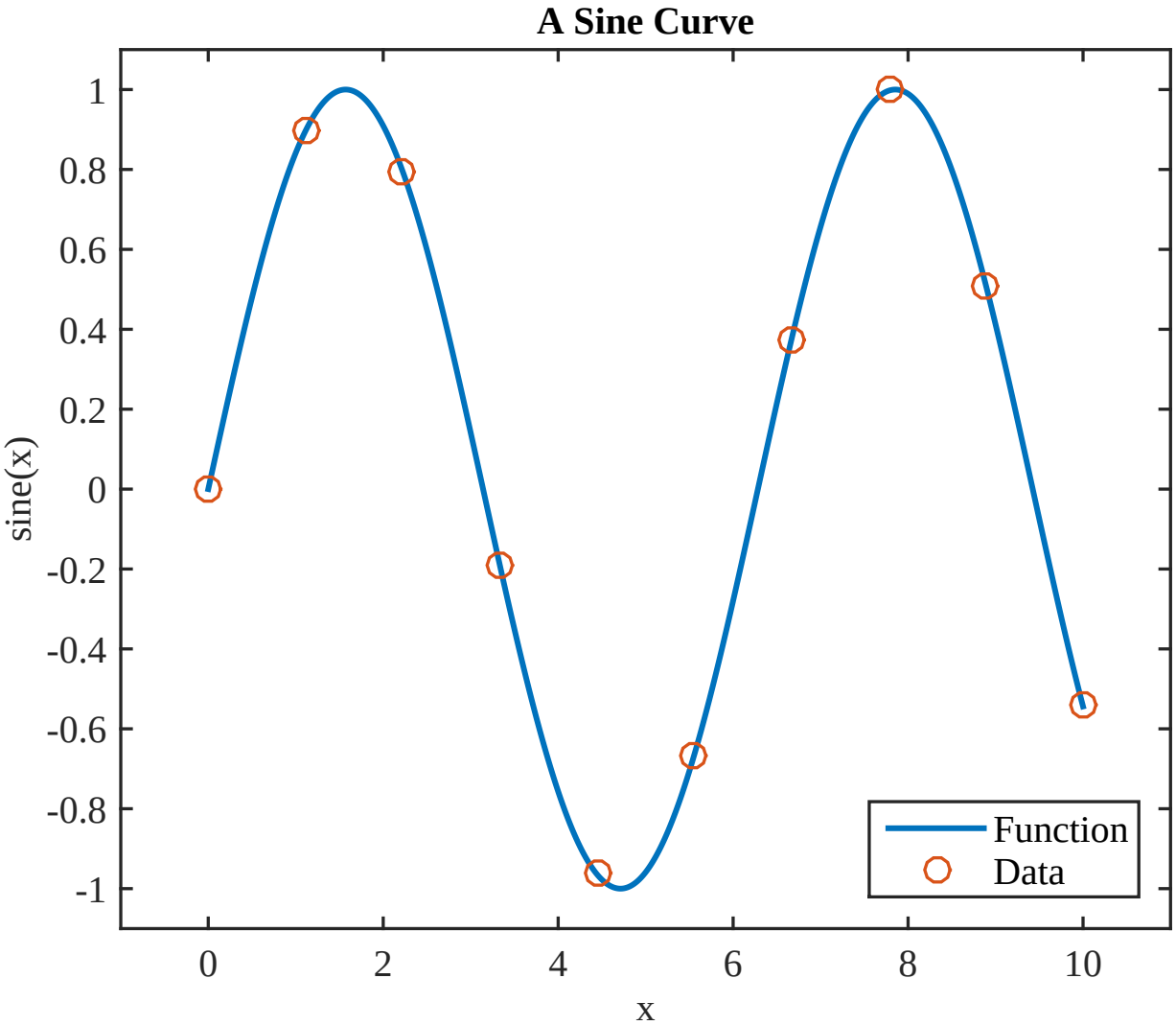
Functional approximations: Splines

- Quite straightforward. For instance, if we were only given two data points (x_1, x_2) and (y_1, y_2), two parameters out of four are identified as $y_i = x_i$ for $i = 1, 2$.
- Then we set $\tilde{g}''(x_i) = 0$ at $i = 1, 2$ which gives two more conditions.

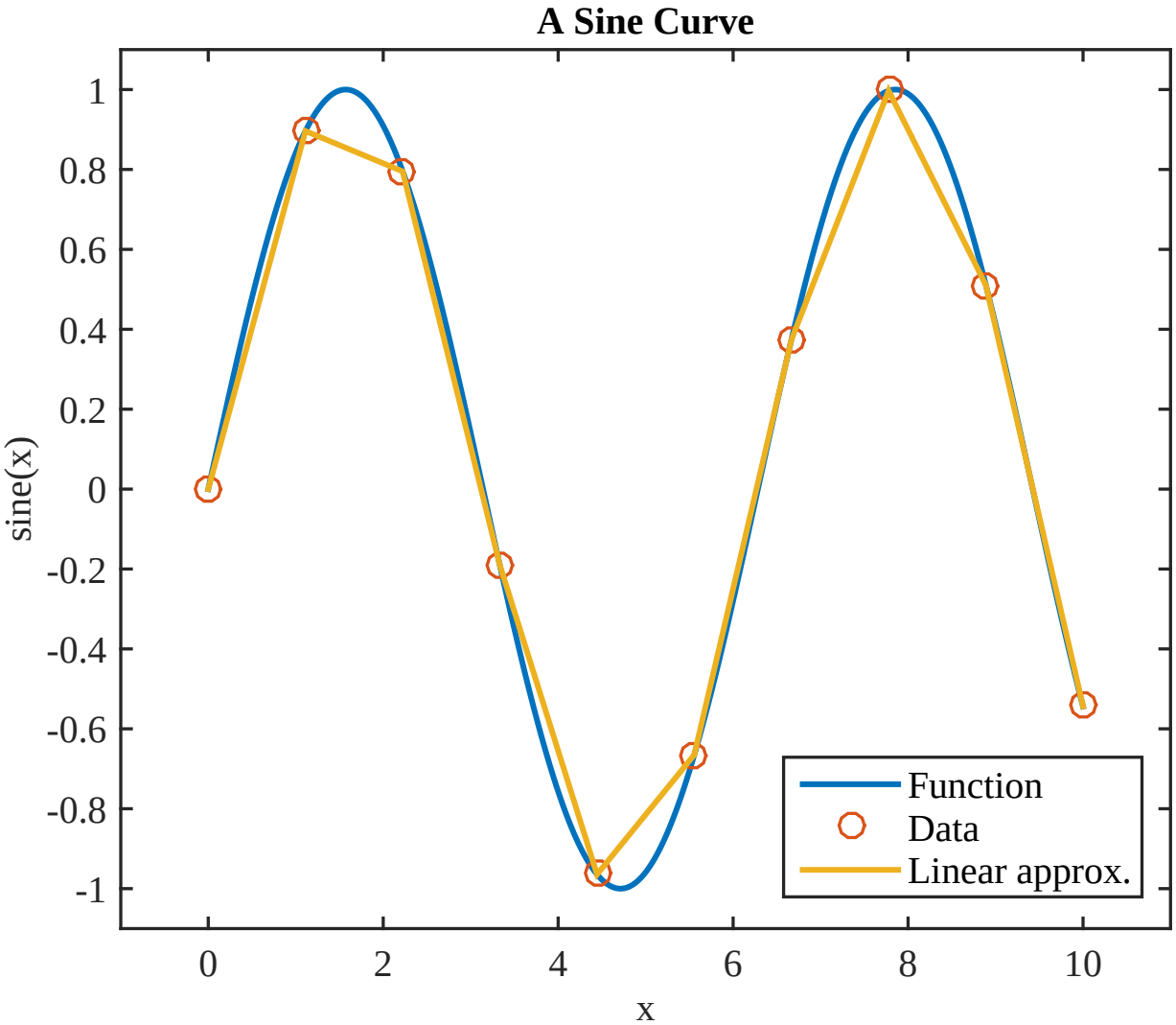
Functional approximations: Splines

- Quite straightforward. For instance, if we were only given two data points (x_1, x_2) and (y_1, y_2), two parameters out of four are identified as $y_i = x_i$ for $i = 1, 2$.
- Then we set $\tilde{g}''(x_i) = 0$ at $i = 1, 2$ which gives two more conditions.
- But normally we have more than two data points; say N data points.
- Then we identify parameters by ensuring that $\tilde{g}_i'(x_{i+1}) = \tilde{g}_{i+1}'(x_{i+1})$, and $\tilde{g}_i''(x_{i+1}) = \tilde{g}_{i+1}''(x_{i+1})$ at all nodes.
- In addition, we use the "not-a-knot" end-condition $\tilde{g}_1''(x_1) = \tilde{g}_N''(x_N) = 0$.

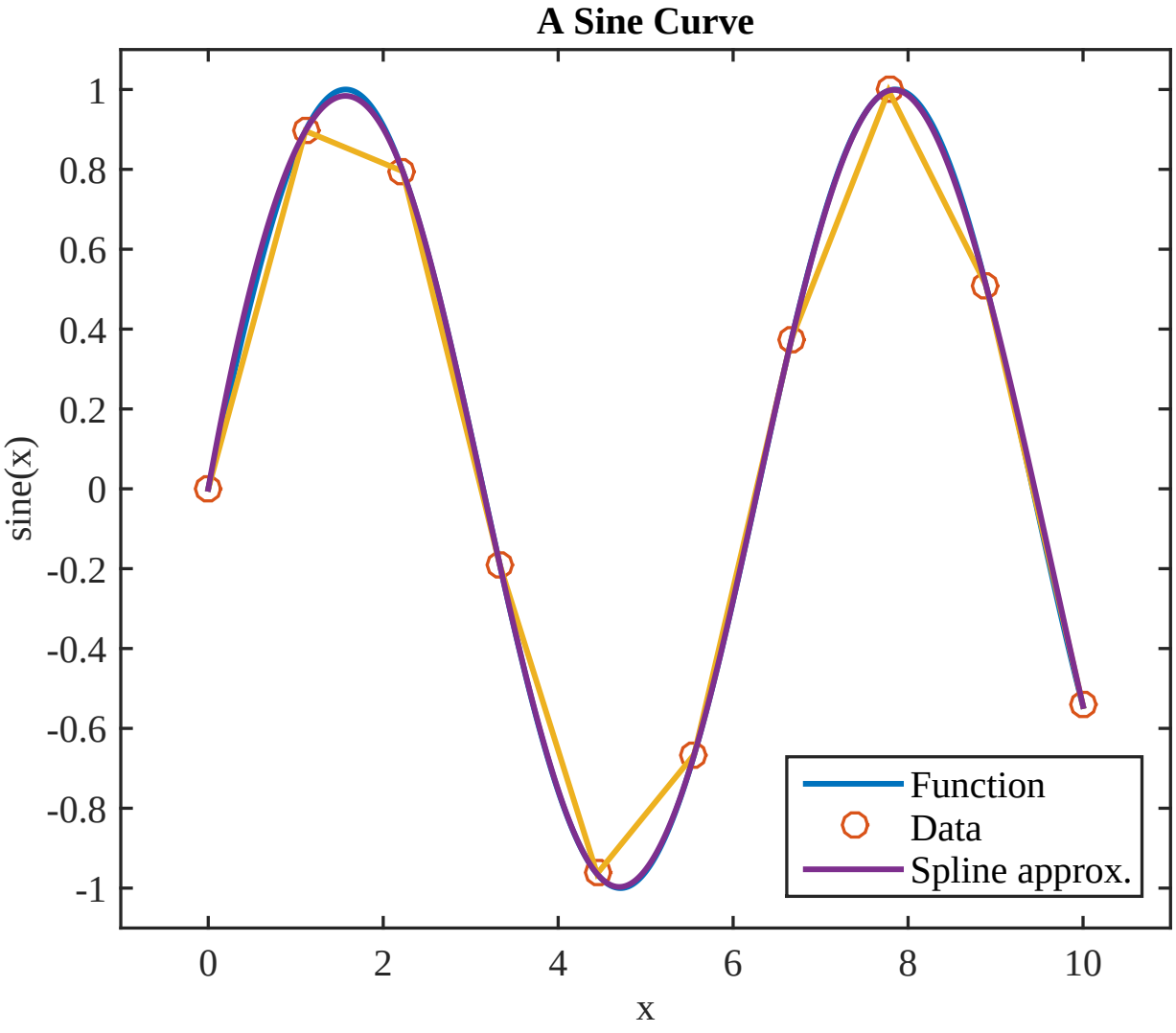
Functional approximations: Spline interpolation



Functional approximations: Spline interpolation



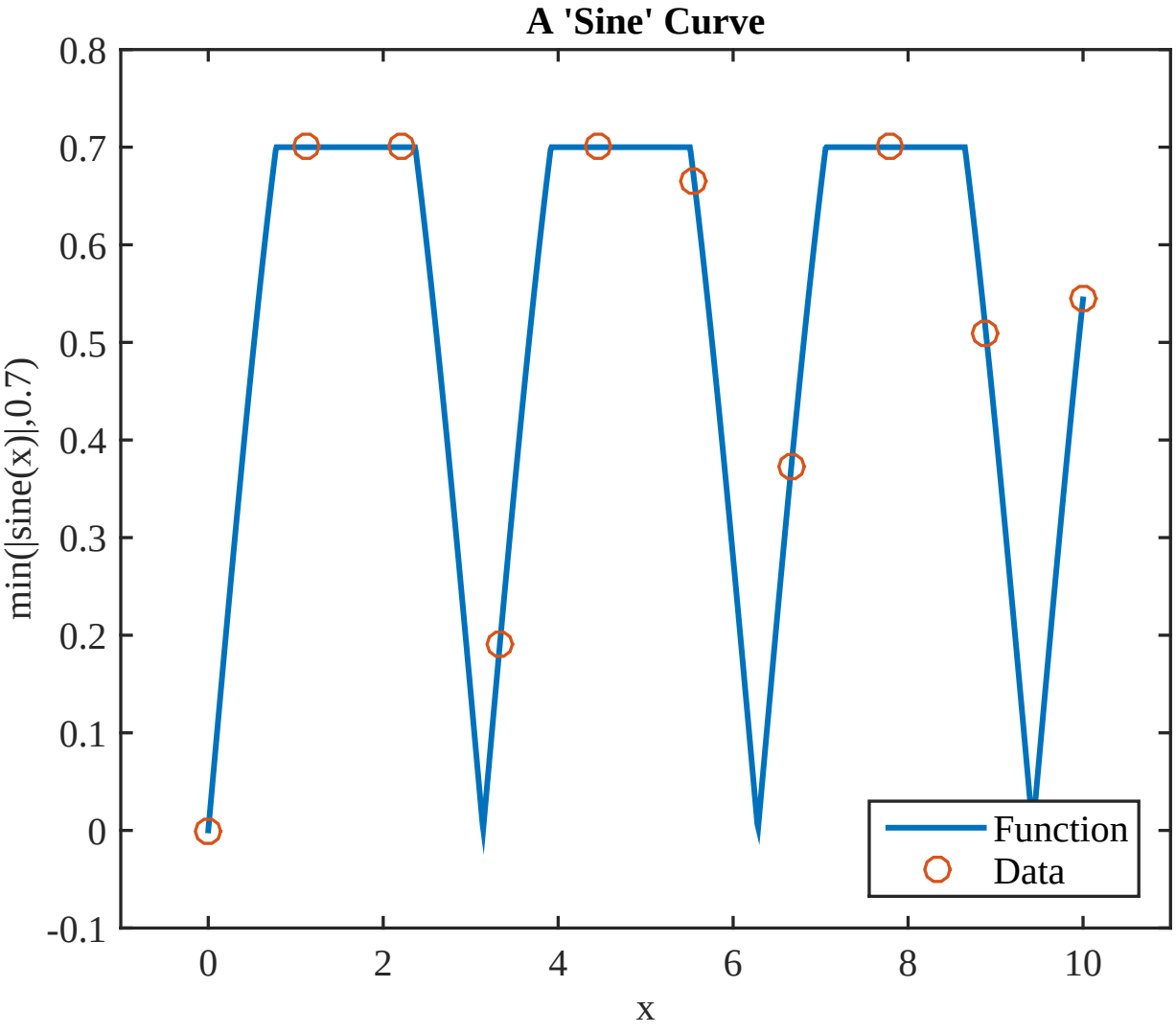
Functional approximations: Spline interpolation



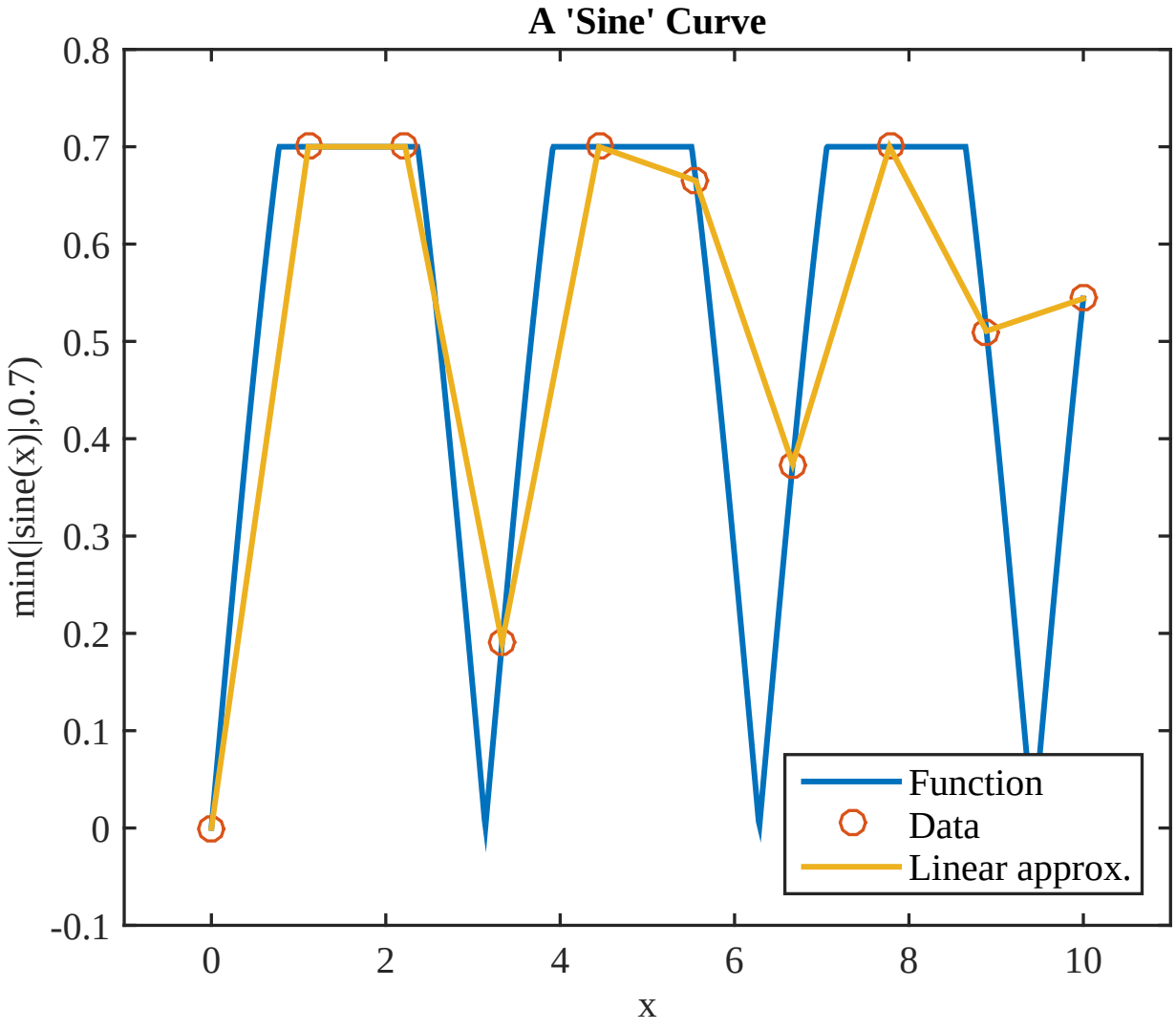
Functional approximations: Splines

- Splines are normally much better at approximating smooth functions than linear interpolation.
- Yet, they are slower, and less robust to kinky function behavior.
- I usually program everything using linear interpolation, and then, if things are smooth, I switch to splines at a final run to get prettier graphs.

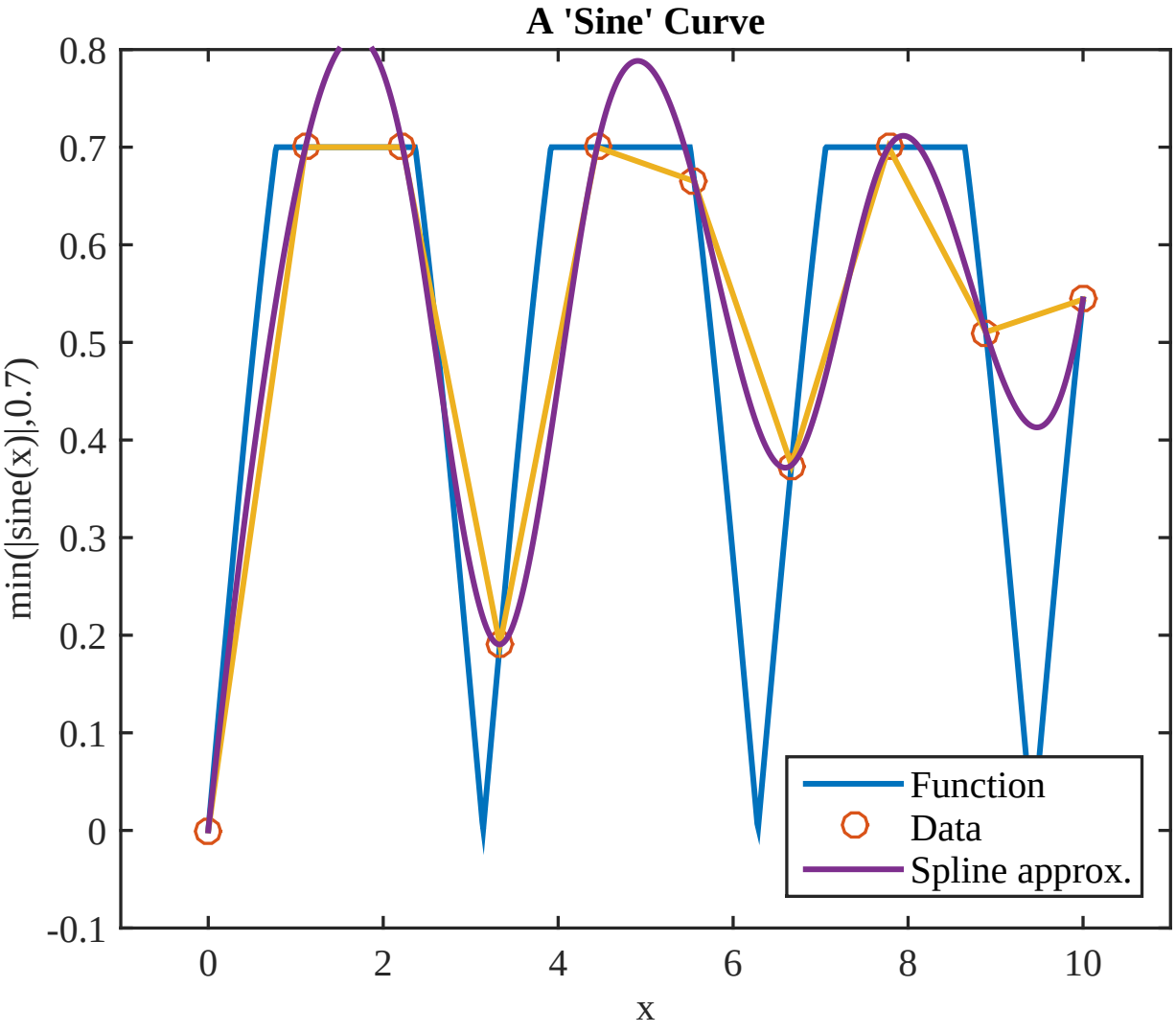
Functional approximations: Spline vs. linear



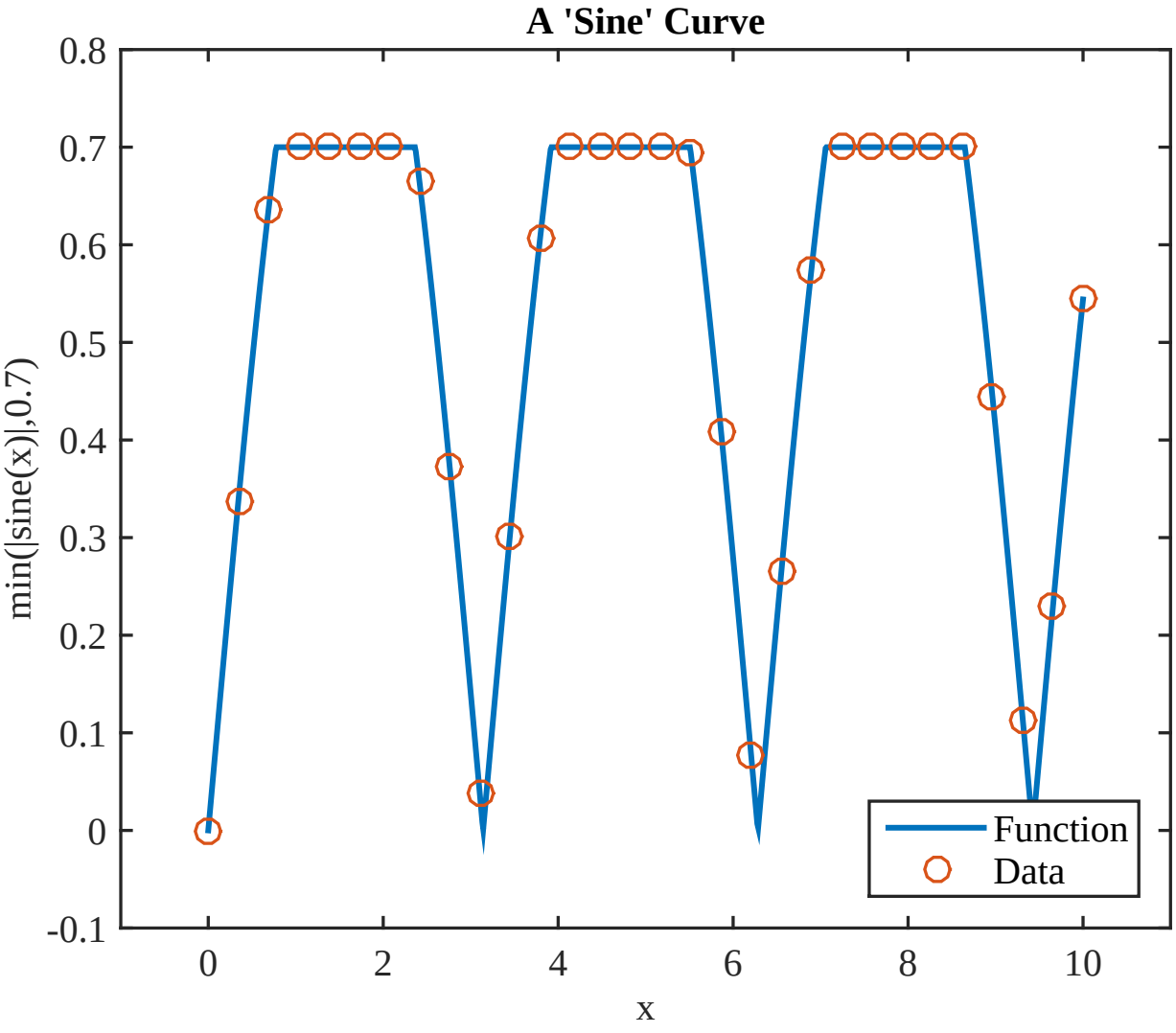
Functional approximations: Spline vs. linear



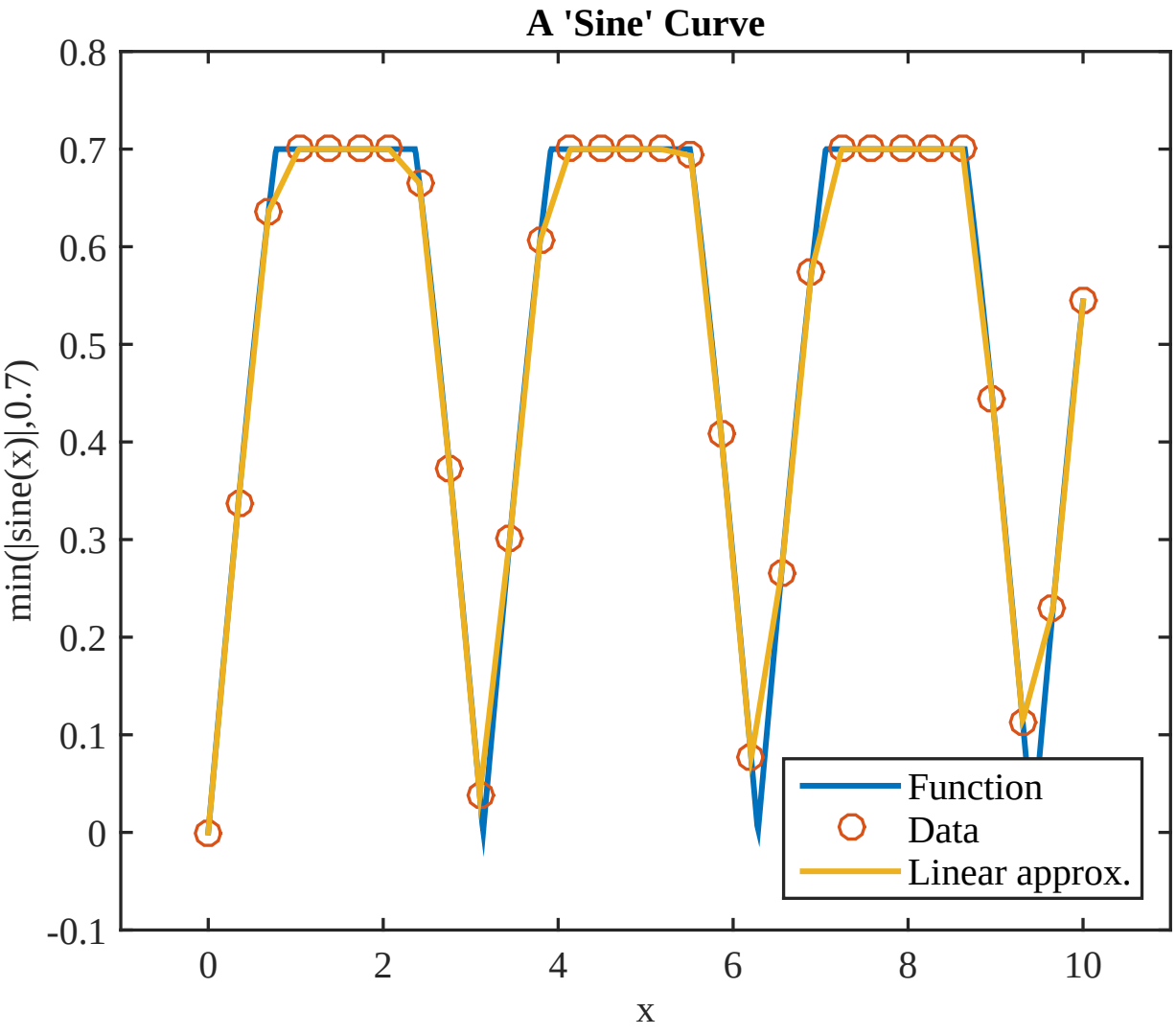
Functional approximations: Spline vs. linear



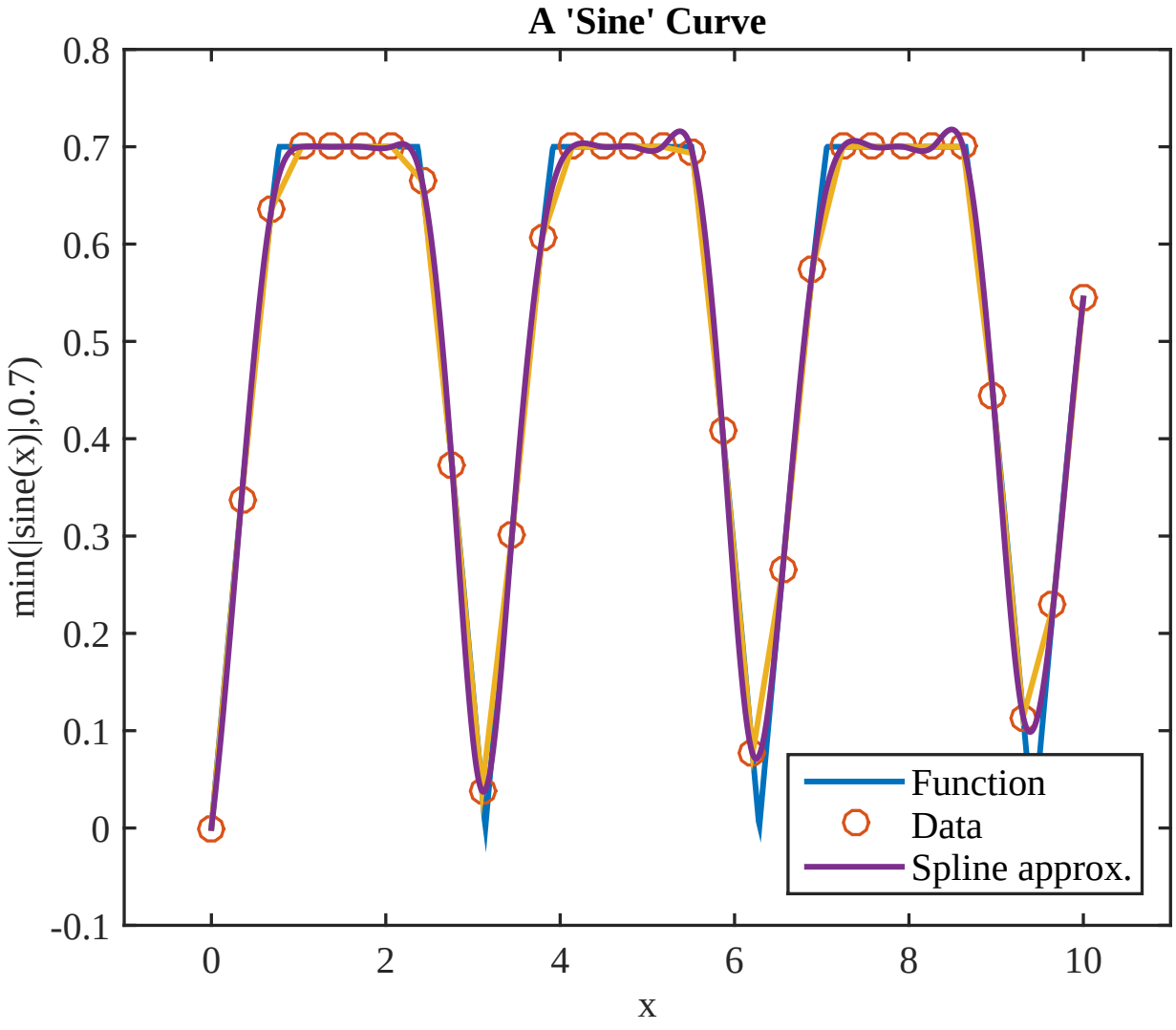
Functional approximations: Spline vs. linear



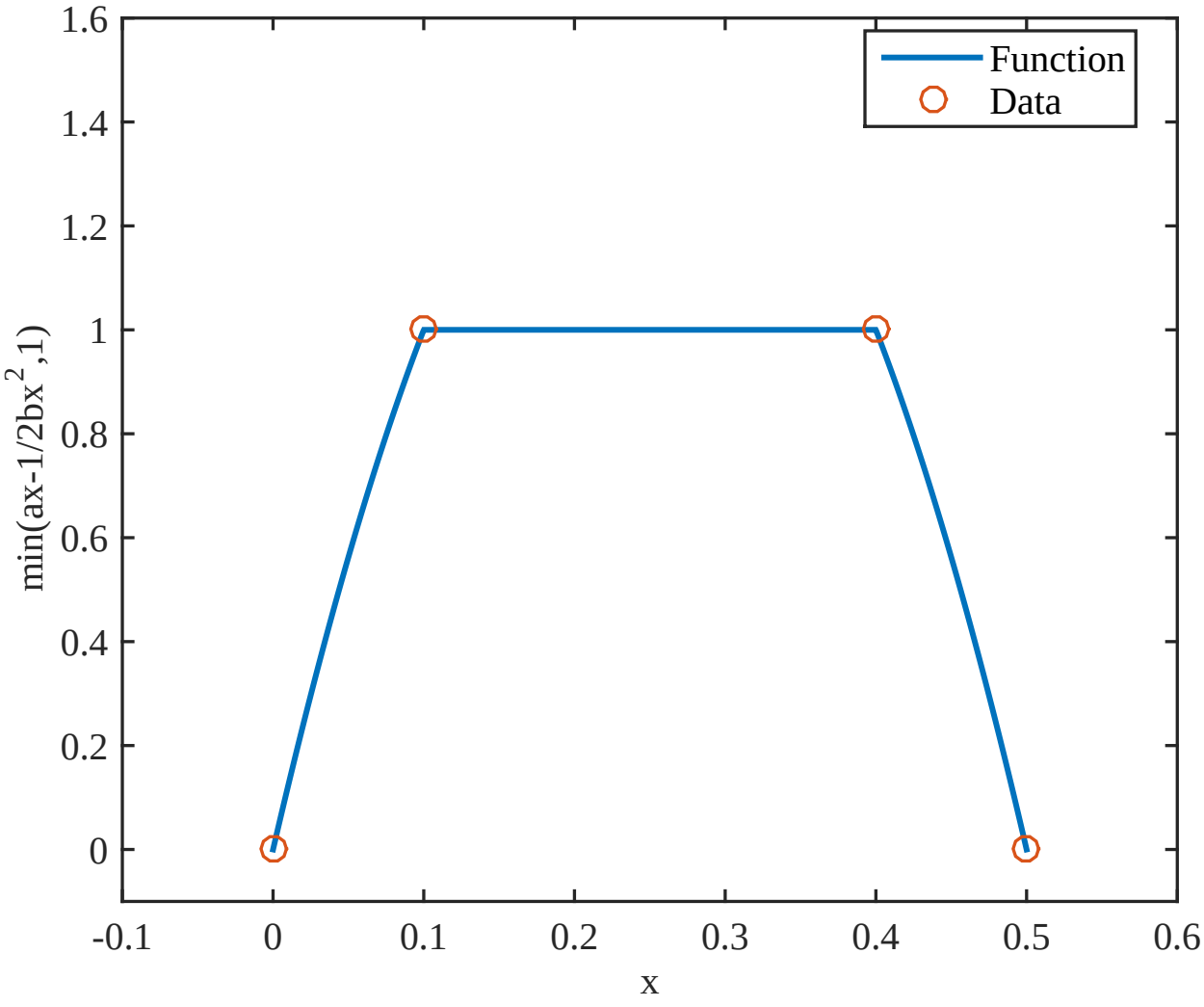
Functional approximations: Spline vs. linear



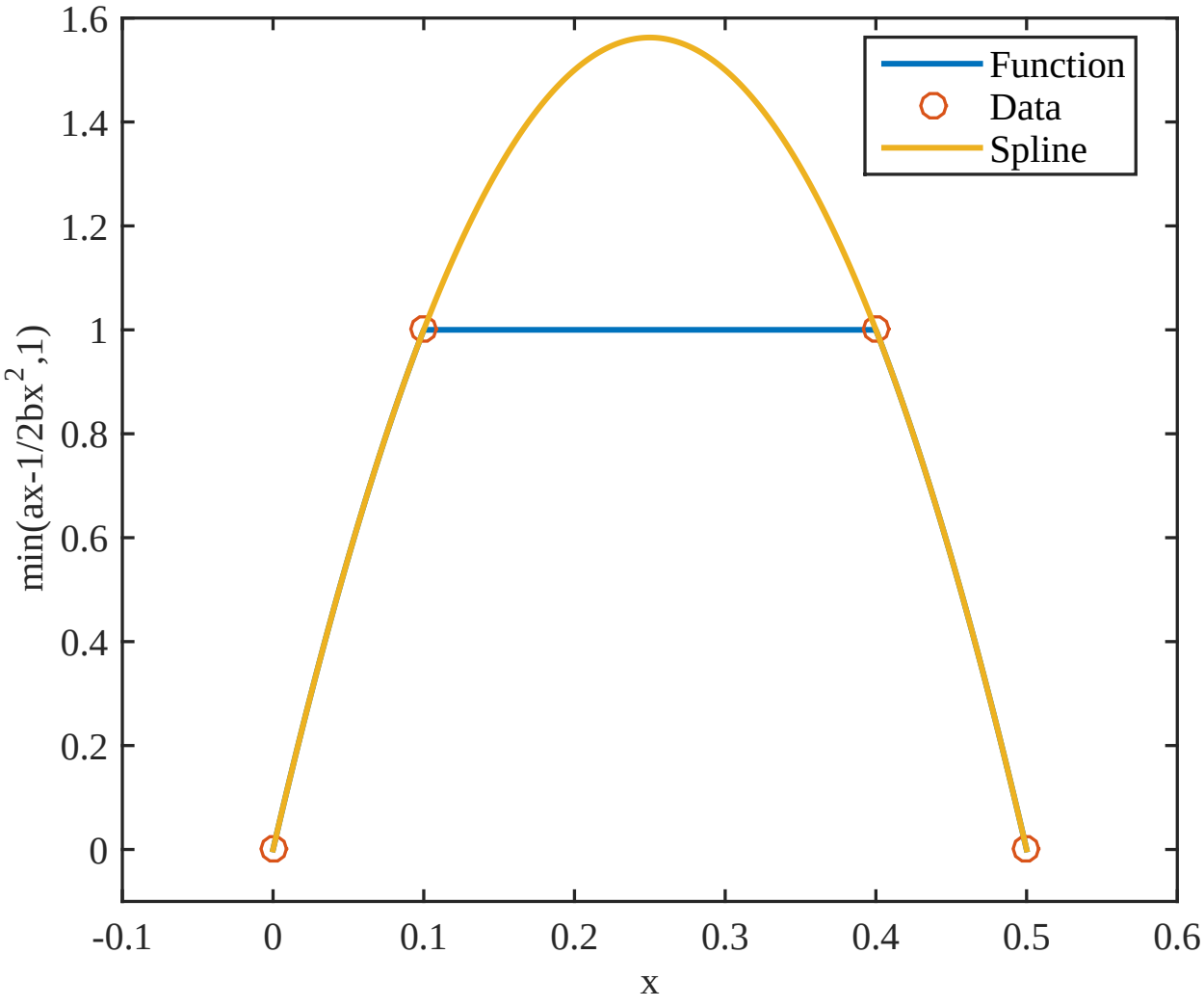
Functional approximations: Spline vs. linear



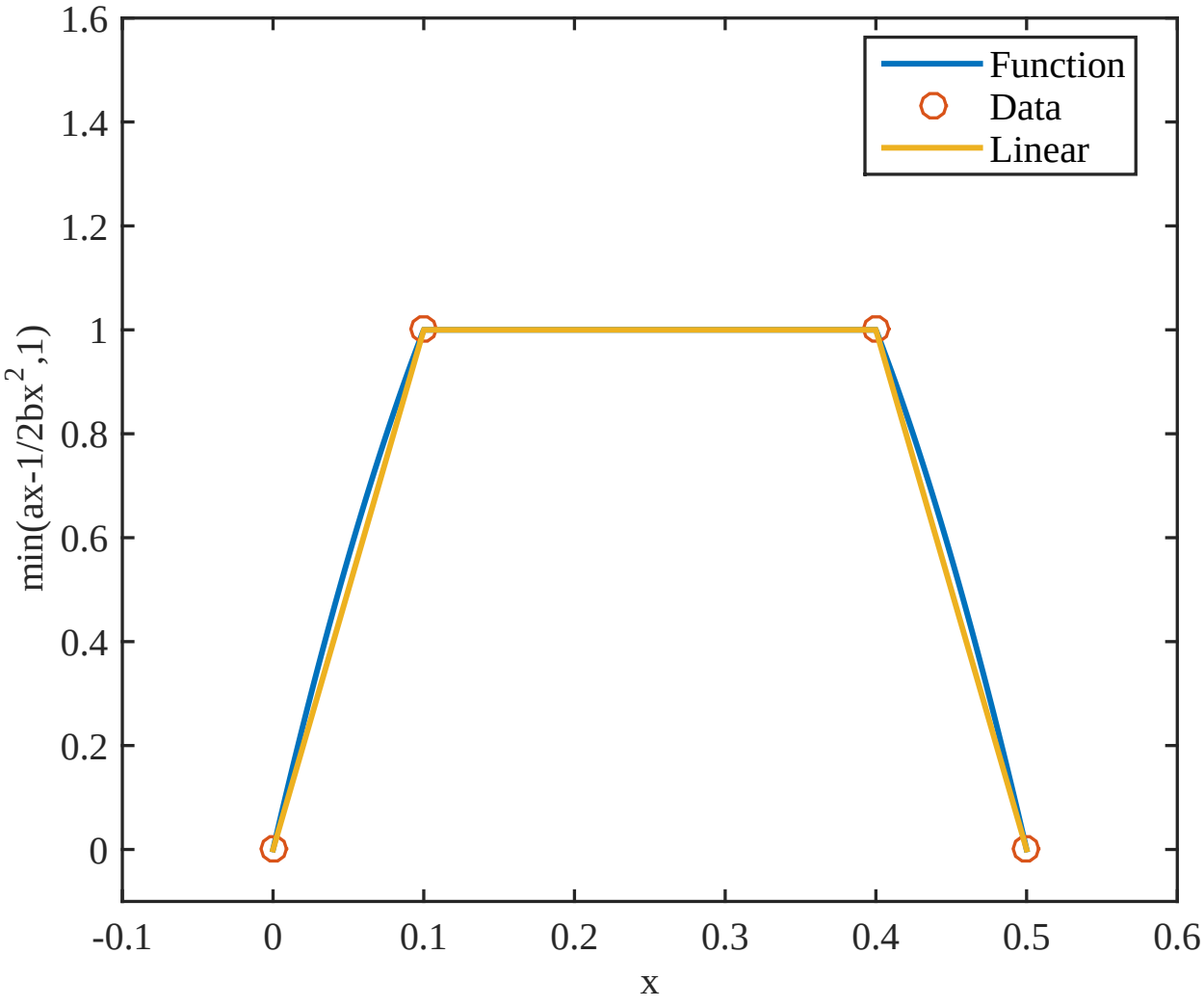
Functional approximations: Spline vs. linear



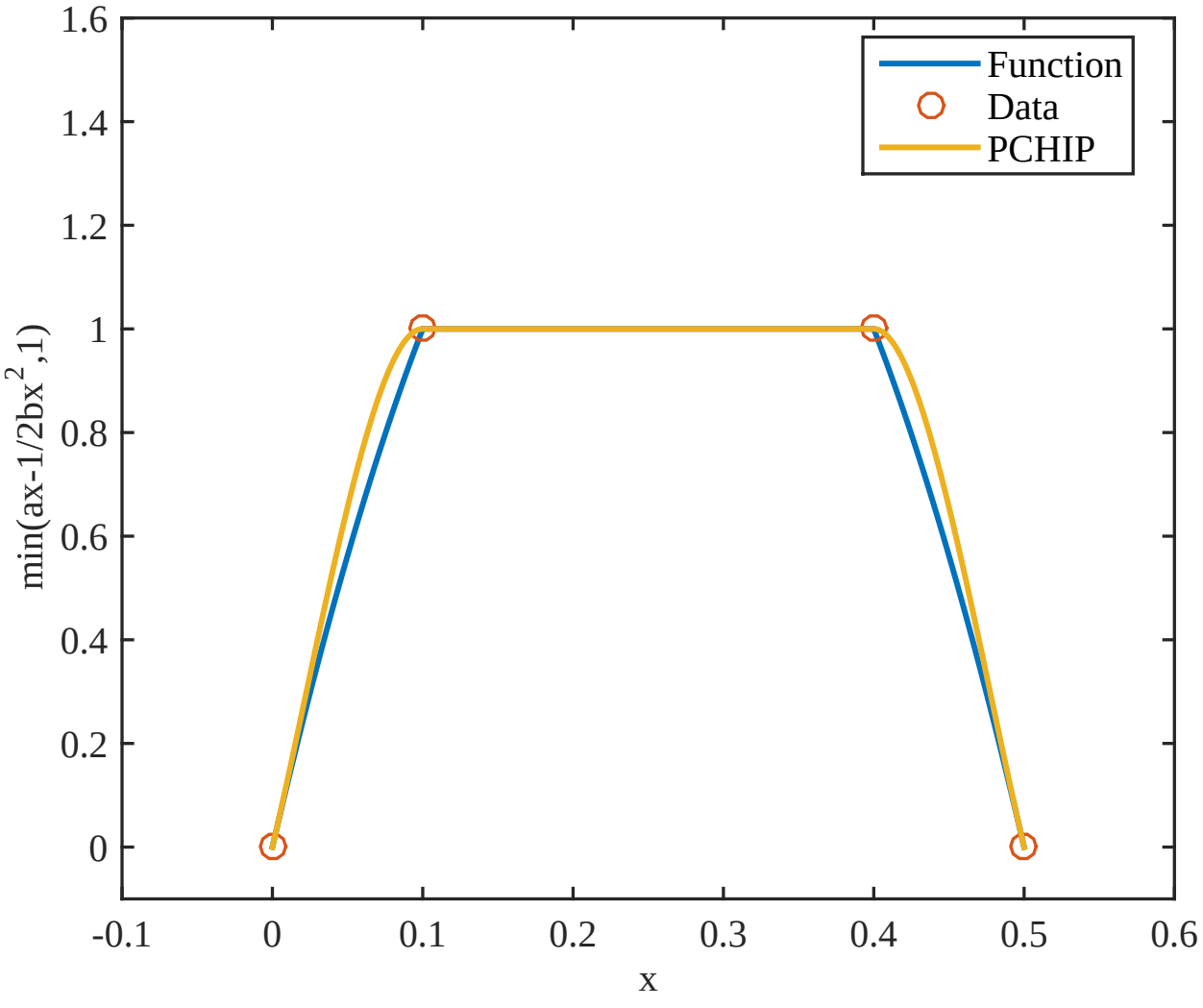
Functional approximations: Spline vs. linear



Functional approximations: Spline vs. linear



Functional approximations: Spline vs. linear



Implementation

- The implementation of these methods in Matlab are easy
- Given two one-dimensional vectors x and y, you can just type
- `interp1(x,y,z,'linear')` for linear interpolation
- `interp1(x,y,z,'spline')` for spline interpolation
- `interp1(x,y,z,'pchip')` for pchip interpolation

Implementation

- In recent versions of Matlab you can also use the `griddedInterpolant` command.
- In particular you type `g = griddedInterpolant(x,y)`, and then type `g(z)` to evaluate the approximation.
- Play around with it!

Functional approximations: Spectral methods

- What about spectral methods?
- Splines are comprised by several “local” polynomial approximations
- 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
  - Tremendous speed!
- Costs: Can display oscillating and very weird behavior.
- Extrapolation is usually out of the question.

Chebyshev Polynomials

- Given data (x, y), we could in principle used least squares to find the best fitting polynomial

$$y \approx a_0 + a_1 x + a_2 x^2 + a_3 x^3 \ldots$$

- Just create the matrix

$$X = \begin{pmatrix} 1 & x_1 & x_1^2 \ldots \\ 1 & x_2 & x_2^2 \ldots \\ \vdots & & \ddots \end{pmatrix} \tag{1}$$

- And find the coefficients using least squares

$$\mathbf{a} = X \backslash y$$

Chebyshev Polynomials

- The “problem” is that the monomials x, x^2, x^3 etc, are very collinear.
- And there are much more efficient ways of approximating a function than using monomials.
- Chebyshev Polynomials are such methods.

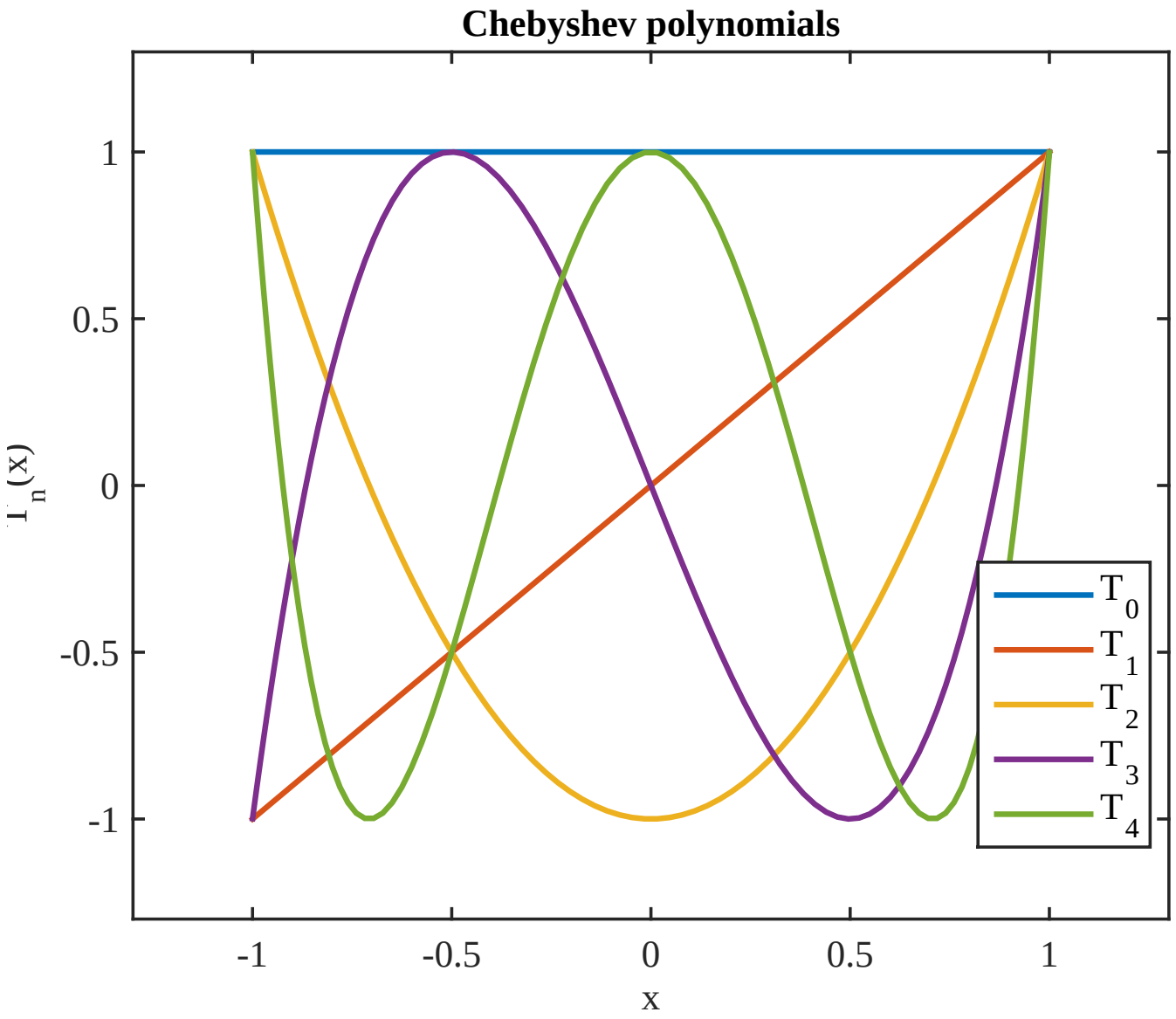
Chebyshev Polynomials

- Chebyshev Polynomials are denoted $T_n : [-1, 1] \to [-1, 1]$.
- They are recursively defined as

$$T_0 = 1, T_1 = x,$$
$$T_n = 2xT_{n-1} - T_{n-2}$$

- And they look kinda pretty.

Chebyshev Polynomials



Chebyshev Polynomials: How to implement

- Convert your x data to the interval $[-1, 1]$. Call this new data X.
- That's easy to do. Define $b = 2/(x_{max} - x_{min})$, and $a = 1 - bx_{max}$. Then $X = a + bx$ is in $[-1, 1]$.
- Recursive calculate your polynomials and put them in a matrix, M

$$M = \begin{pmatrix} T_0(x_1) & T_1(x_1) & T_2(x_1)\ldots \\ T_0(x_2) & T_1(x_2) & T_2(x_2)\ldots \\ \vdots & & \ddots \end{pmatrix} \tag{2}$$

Chebyshev Polynomials: How to implement

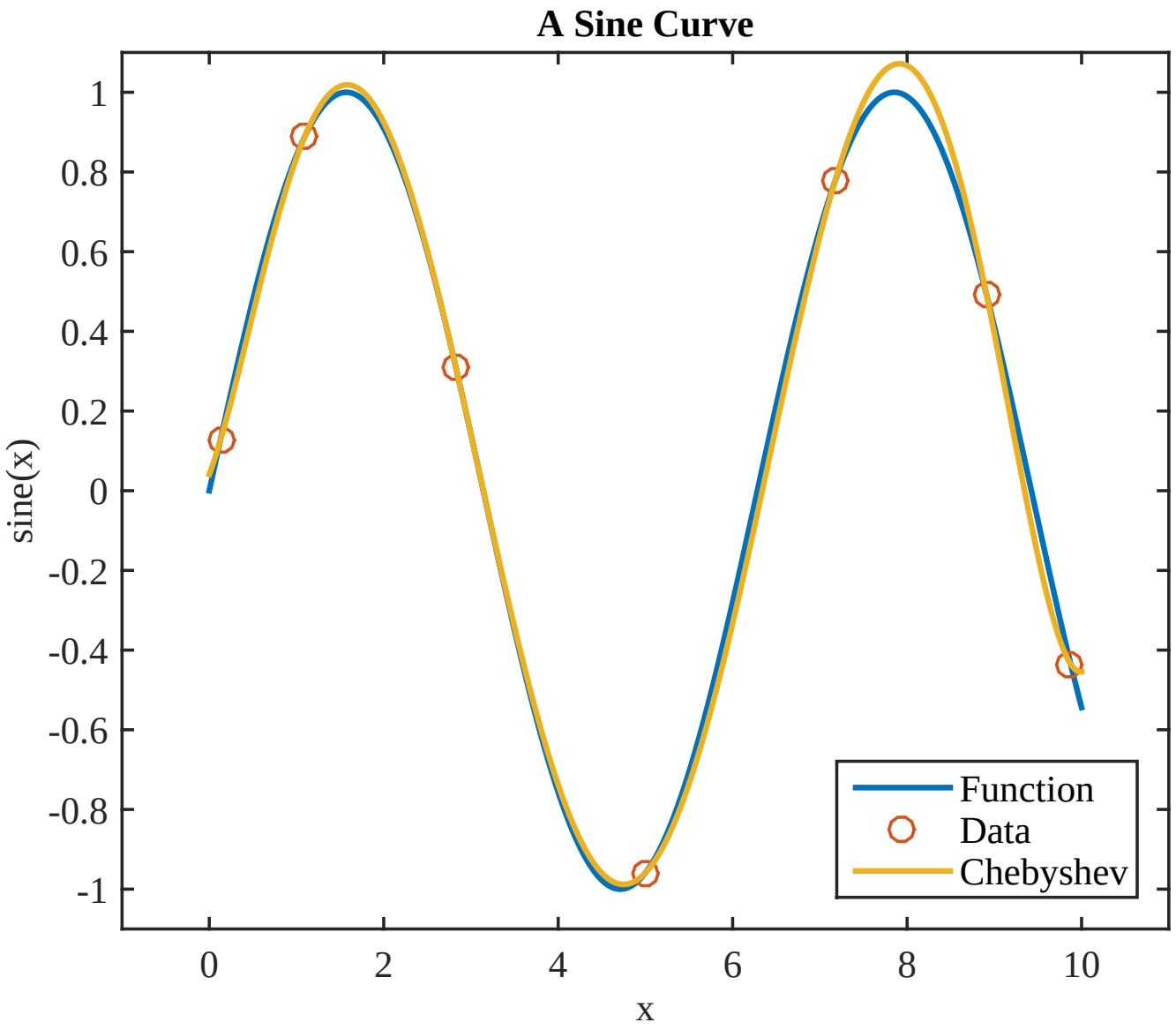
- Use least squares to find the coefficients

$$y = a_0 T_0(x) + a_1 T_1(x) + a_2 T_2(x) \ldots$$

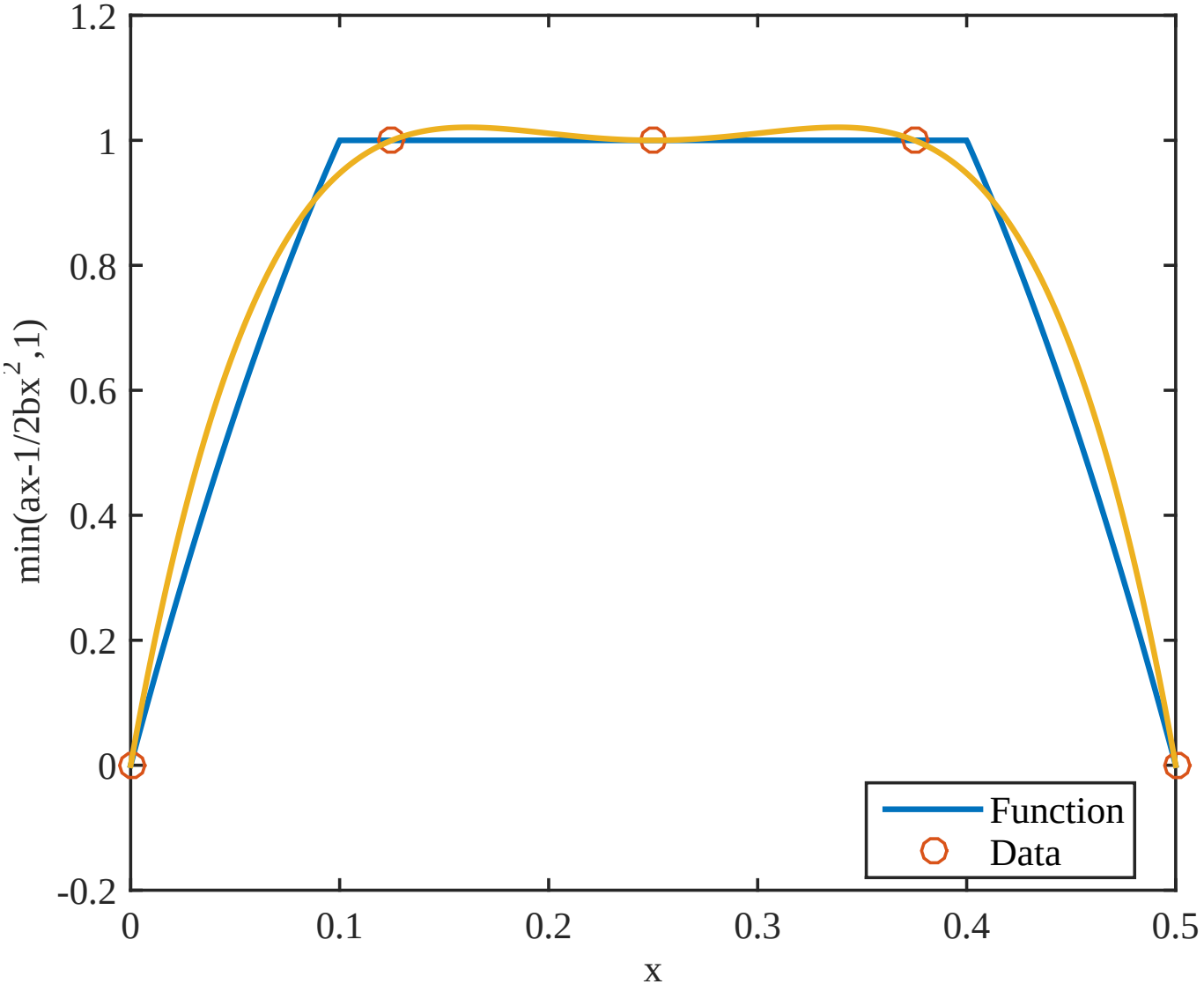
- And then evaluate at point z

$$y = a_0 T_0(z) + a_1 T_1(z) + a_2 T_2(z) \ldots$$

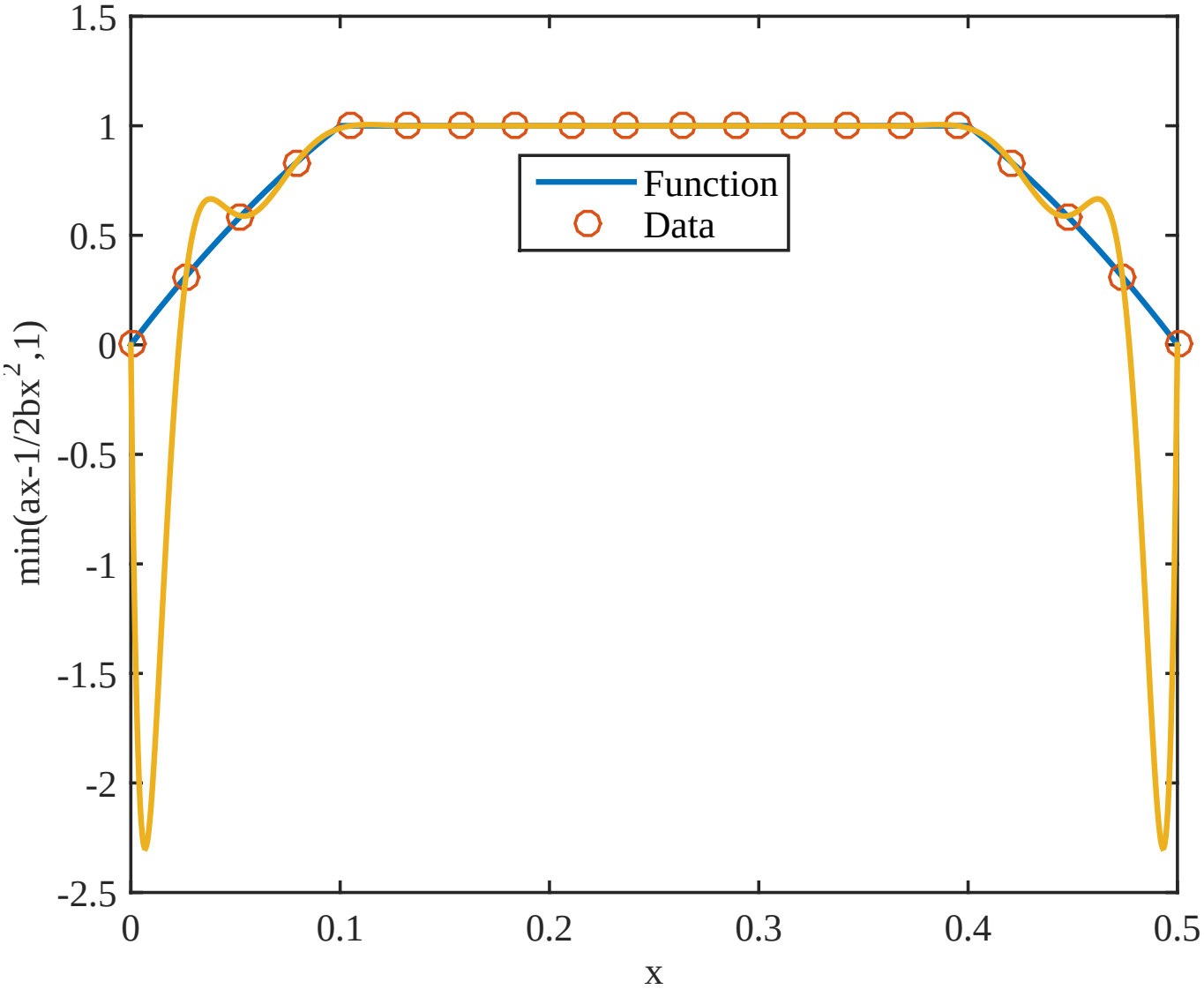
Chebyshev Polynomials



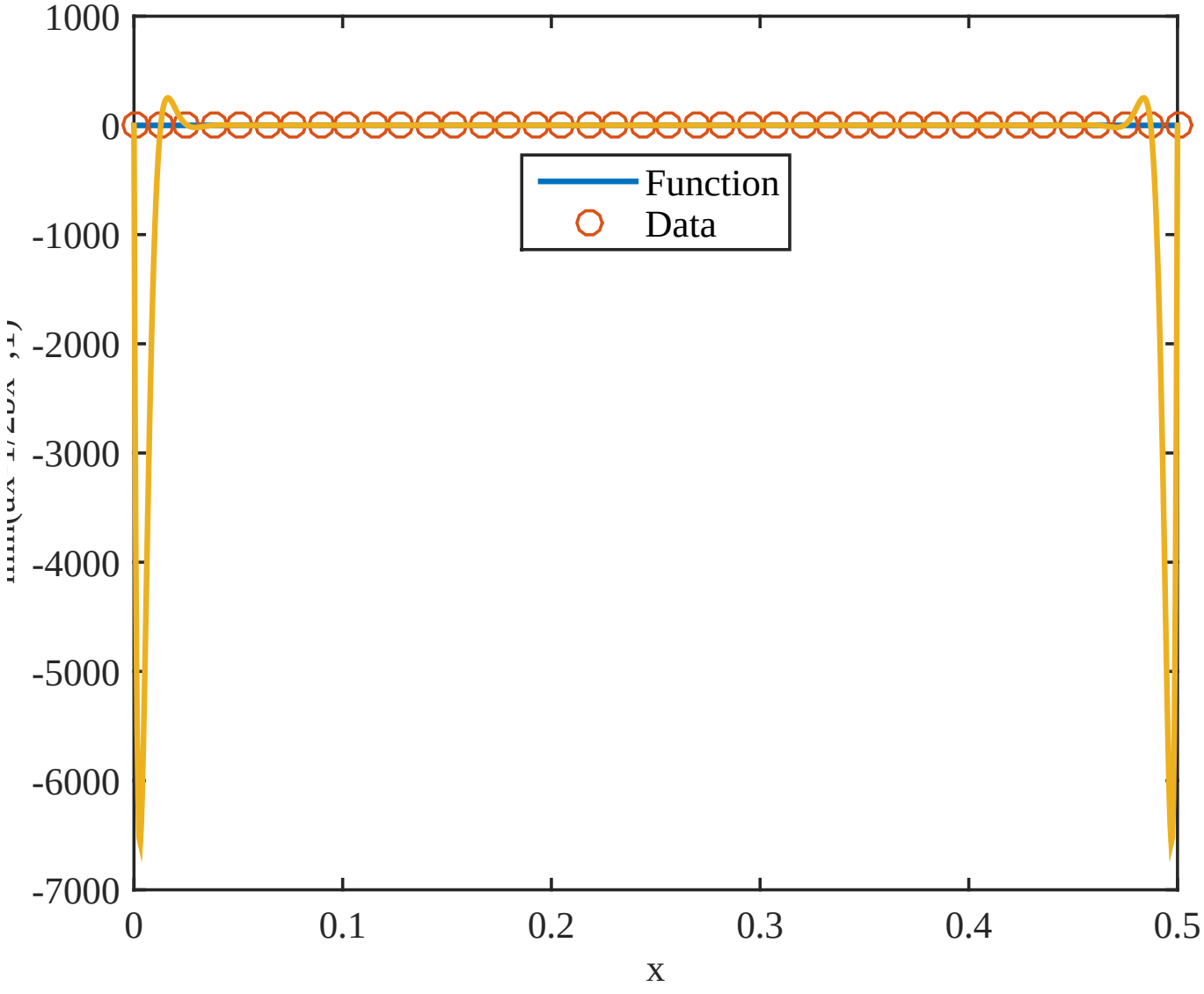
Chebyshev Polynomials



Chebyshev Polynomials



Chebyshev Polynomials



Putting things to use

- So how are we going to use these ideas?
- Many economic models can be collapsed into the condition

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

  where $f(\cdot)$ is a known function,
- Thus, for each x there exist a (hopefully unique) y such that the above condition is satisfied.
- That is, there exist a mapping from x to y, $y = g(x)$ such that

$$f(x, g(x)) = 0 \quad \forall\ x \in X$$

- And for any $\tilde{y} \neq g(x)$

$$f(x, \tilde{y}) \neq 0$$

Putting things to use

- So how do we find $g(x)$ now?
- Pick a grid $(x_i)_{i=1}^N$, and use some nonlinear solver to find $(y_i)_{i=1}^N$ such that

$$f(x_i, y_i) = 0, \quad i = 1, \ldots, N$$

- Then use your data $(x_i, y_i)_{i=1}^N$ together with some functional approximation to find $\tilde{g}(x)$

Putting things to use

- As an example take a standard two period consumption savings problem

$$\max_{k'}\{u(m-a_1)+\beta u(a_1(1+r)+w)\}$$

- With $m = a_0(1+r)+w$.
- First order conditions are

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

- We wish to find $a_1 = g(m)$, or $c_1 = h(m) = m - g(m)$.

Nonlinear equation solver

- Let's first figure out how we can solve a nonlinear equation (such as the Euler equation above).
- A general (one-dimensional) problem would be

$$f(x) = 0$$

with $x \in \mathbb{R}$

Nonlinear equation solver

- Suppose we have a guess, x^{old}
- The idea is to update this guess until we solve the equation
- Take a first order Taylor approximation of f around x^{old}

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

Nonlinear equation solver

- Suppose we have a guess, x^{old}
- The idea is to update this guess until we solve the equation
- Take a first order Taylor approximation of f around x^{old}

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

- Set to zero and solve for x

$$x^{new} = x^{old} - \frac{f(x^{old})}{f'(x^{old})}$$

Nonlinear equation solver

$$x^{new} = x^{old} - \frac{f(x^{old})}{f'(x^{old})}$$

- Pretty easy.
- In our case

$$f(x) = -u'(m - a_1) + \beta(1 + r)u'(a_1(1 + r) + w)$$
$$f'(x) = u''(m - a_1) + \beta(1 + r)^2 u''(a_1(1 + r) + w)$$

- Straightforward interpretation

Nonlinear equation solver

- Suppose that

$$f(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{pmatrix}$$

  and $x \in \mathbb{R}^n$
- That is we're looking for a solution to a system of equations.

Nonlinear equation solver

- Not a problem. Taylor expansion again,

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

with

$$J_x = \begin{pmatrix} \partial f_1(x)/\partial x_1 & \partial f_1(x)/\partial x_2 & \cdots & \partial f_1(x)/\partial x_n \\ \partial f_2(x)/\partial x_1 & \partial f_2(x)/\partial x_2 & \cdots & \partial f_2(x)/\partial x_n \\ \vdots & \vdots & \ddots & \vdots \\ \partial f_n(x)/\partial x_1 & \partial f_n(x)/\partial x_2 & \cdots & \partial f_n(x)/\partial x_n \end{pmatrix}$$

Nonlinear equation solver

- Solution

$$x^{new} = x^{old} - J^{-1}_{x^{old}} f(x^{old})$$

- This is called Newton's method (or Newton-Raphson)
- It generally works well and it converges quite fast
- But it is important to have a good initial guess

Nonlinear equation solver

- Let's look at some code to see how this is done.

Nonlinear equation solver

- While Newton's method is handy, sometimes it's comforting to know that there are more reliable methods available too.
- In particular, suppose that $f(x)$ is continuous and that $f(a)$ and $f(b)$ have opposite sign.
- Then by the intermediate value theorem there exist a $x^* \in [a, b]$ such that $f(x^*) = 0$.
- This insight will lead us to the bisection method.

Nonlinear equation solver

Bisection method

- pick x' as

$$x' = \frac{a+b}{2}$$

- If $f(x')$ and $f(b)$ have opposite sign, set $a = x'$.
- If $f(x')$ and $f(a)$ have opposite sign, set $b = x'$.
- Repeat until $|a - b| < \varepsilon$

Nonlinear equation solver

- There are many other methods available, but many of the them just exploit the above properties.
- The secant method uses a line-secant between two guesses to provide an approximate derivative (saves computation time)
- Broyden's method generalises the secant method to many dimensions
- Brent's method combines stuff like the bisection with the secant method to behave optimally

Numerical integration

- Numerical integration is normally known as “quadrature”.
- For most simple problems you can just use the commands `quad` or `integral`.
- These are very accurate and robust.

Numerical integration

- But they are also slow.
- Most of the time when we need to integrate something, we are evaluating an expectation.
- But often the shocks we analyse are normal, or log-normal.
- The best quadrature technique is then "Gauss-Hermite" quadrature.

Numerical integration

- I have attached a program called `hermquad` which you can use in this case
- For instance, if you type `[X,W] = hermquad(5)` you'll find two vectors

$$X = \begin{pmatrix} 2.0202 \\ 0.9586 \\ 0 \\ -0.9586 \\ -2.0202 \end{pmatrix} \quad \text{and} \quad W = \begin{pmatrix} 0.0200 \\ 0.3936 \\ 0.9453 \\ 0.3936 \\ 0.0200 \end{pmatrix}$$

- The X matrix contains the abscissae and W the weights.

Numerical integration

- These need to be normalised. In particular, given some standard deviation σ, define

$$\hat{X} = X\sqrt{2}\sigma, \quad \hat{W} = \pi^{-1/2}W.$$

For $\sigma = 0.01$ you'll see that

$$\hat{X} = \begin{pmatrix} 0.0286 \\ 0.0136 \\ 0 \\ -0.0136 \\ -0.0286 \end{pmatrix} \quad \text{and} \quad \hat{W} = \begin{pmatrix} 0.0113 \\ 0.2221 \\ 0.5333 \\ 0.2221 \\ 0.0113 \end{pmatrix}$$

Numerical integration

- Then suppose I would like to calculate

$$\int h(x)f(x)dx$$

where $f(x)$ is a normal pdf and h(x) is some nonlinear function.

- A good approximation is then

$$\sum_{i}^{N} h(x_i)w_i$$

where x_i and w_i are the elements from $\hat{X}$ and $\hat{W}$

Numerical integration

- Suppose $h(x) = x^2$.
- Then with N equal to 5

$$\sum_{i}^{N} h(x_i) w_i = 0.0001$$

and $\sqrt{0.001} = 0.01$. Which was the standard deviation we provided.

- Lesson: With normal shocks, $N = 5$ and Gauss-Hermite quadrature goes a long way.

Numerical integration

- Lastly, we often encounter stochastic processes like

$$z_t = \rho z_{t-1} + \varepsilon_t$$

- If these are normal, we can use our quadrature techniques from above to deal with this.
- But it's much easier, and faster, to deal with stochastic shocks that follow a transition matrix instead.
- Luckily, there are methods to convert VAR processes to transition matrices.

Numerical integration

- The most used approach was developed by Tauchen (1986)
- I will not describe it here, but Martin Floden (Stockholm University) has the code on his webpage.
- You provide a ρ and a variance for ε, specify how many states you wish to have, and specify the boundaries.
- Out comes a transition matrix which approximates your underlying VAR!