Advanced Macroeconomics II

Handout 3 - Interpolation

Sergio Ocampo

Western University

September 30, 2020

Short recap

Prototypical DP problem:

$$V(k,z) = \max_{\{c,k'\}} u(c) + \beta E\left[V\left(k^{'},z^{'}\right)|z\right]$$
$$\text{s.t.} c + k^{'} = f(k,z)$$
$$z^{'} = h(z,\eta);\eta \text{ stochastic}$$

- We are looking for functions $\mathbf{V}, \mathbf{g^c}, \mathbf{g^k}$: We cannot solve this

We need to solve an approximate problem:

1. Discretize state space (functions are now vectors)
2. Approximate continuous function: **Interpolation**
   - Requires "exact" solution of maximization problem: **Optimization**

Interpolation: The problem

- We want to know function V...
  - But we only know $\{V(x_1), \ldots, V(x_N)\}$
- When working with V we will often need $V(x)$ for $x \notin \{x_1, \ldots, x_N\}$
- We want a function $\tilde{V}$ that we can evaluate at any x
  - It must be that $\tilde{V}(x_i) = V(x_i)$ for all $x \in \{x_1, \ldots, x_N\}$
- The problem now is how to find this function $\tilde{V}$

Interpolation: Two approaches

1. "Global" approximation
   - Approximate with a known function and evaluate that!
   - But functions are infinite dimensional...
   - Choose functions from some vector space! Basis is finite dimensional
   - Problem is to find coefficients for linear combination
   - Ex: Polynomial approximation

2. Local approximation
   - Match the function locally (between two nodes)
   - The local function is called a **Spline**
   - Splines can be as flexible as you need them to be
   - Ex: Cubic splines, shape preserving splines

Polynomial Approximation

Polynomial approximation

1. Set a family of polynomials with basis for the vector space

$$\{\phi_0(x), \phi_1(x), \ldots, \phi_M(x)\}$$

2. The objective is to write the interpolated function as

$$\tilde{V}(x) = \sum_{m=0}^{M} a_m \phi_m(x)$$

3. We are looking for $\{a_0, \ldots, a_M\}$ such that

$$y_i = V(x_i) = \tilde{V}(x_i) = \sum_{m=0}^{M} a_m \phi_m(x_i) \qquad x_i \in \{x_1, \ldots, x_N\}$$

Polynomial approximation

Then what we have is a linear problem:

$$y = Aa$$

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \qquad a = \begin{bmatrix} a_0 \\ \vdots \\ a_M \end{bmatrix} \qquad A = \begin{bmatrix} \phi_0(x_1) & \dots & \phi_M(x_1) \\ \vdots & \ddots & \vdots \\ \phi_0(x_N) & \dots & \phi_M(x_N) \end{bmatrix}$$

- We need to set $M = N - 1$ to fit the values
- We need to choose a basis for our polynomial
  - Monomial basis ($\phi_m(x) = x^m$)
  - Newton basis $\left(\phi_m(x) = \Pi_{j=0}^{m-1}(x - x_j)\right)$

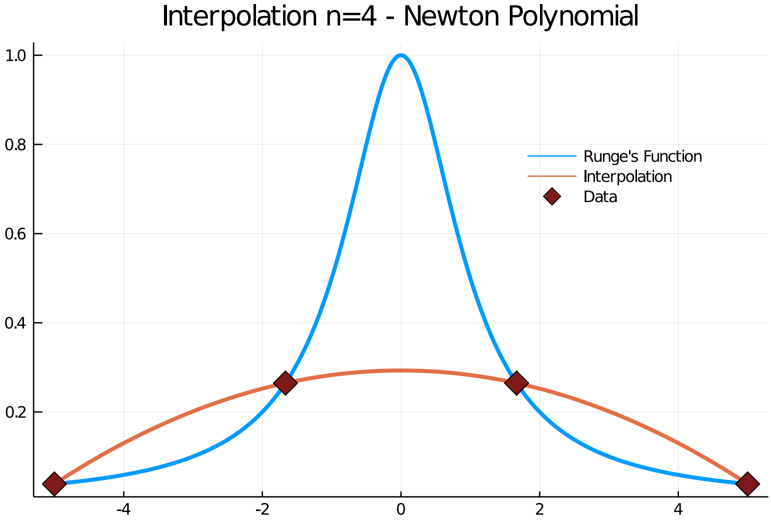
Weierstrass Approximation Theorem

Theorem

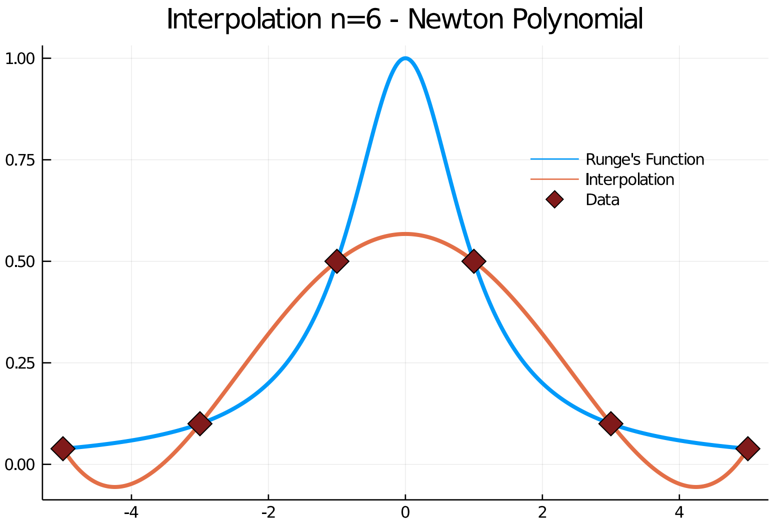
Let $f : [a, b] \to \mathbb{R}$ be a continuous function.
For all $\epsilon > 0$, there exists a polynomial of order n, $P_M(x)$, such that for all $x \in [a, b]$, we have $\|f(x) - P_M(x)\|_\infty < \epsilon$.
Further, $\lim_{M\to\infty} \|f(x) - P_M(x)\|_\infty = 0$.

- *It looks like using polynomials is a great idea!*
- *With enough nodes $\{x_i\}$ we can approximate any continuous function*
- *Success comes at a cost: Higher order polynomials*
    - *Polynomials start to oscillate dramatically at higher orders*

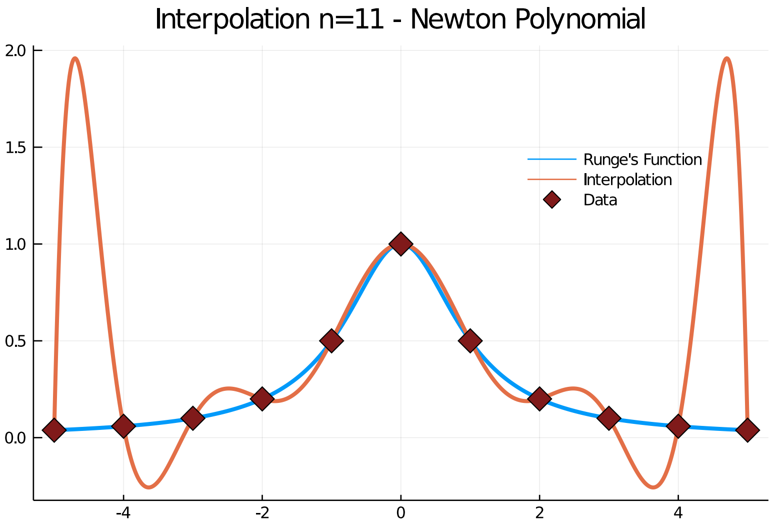
Runge example: $f(x) = {}^{1}/_{1+x^2}$



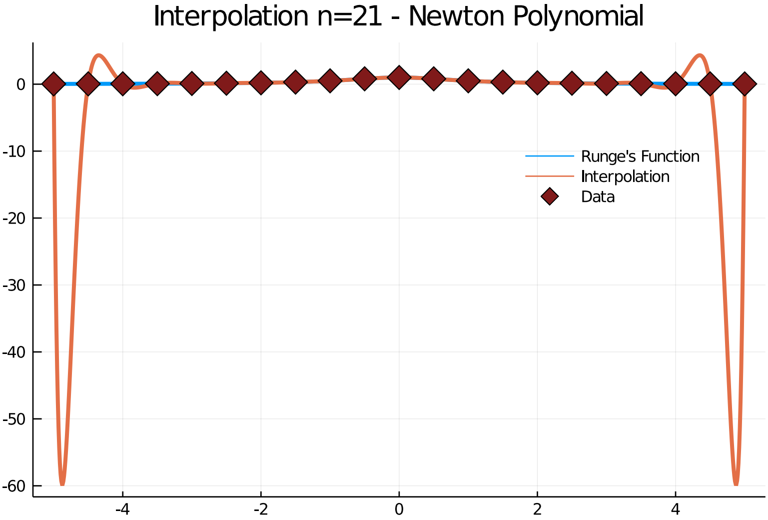
Runge example: $f(x) = {}^{1}\!/_{1+x^2}$



Runge example: $f(x) = {}^1\!/_{1+x^2}$



Runge example: $f(x) = 1/_{1+x^2}$



Two options

1. Find a better location for nodes
   - Hard to know which node placement works for your particular problem
   - We will come back to this at the end
2. Avoid "global" approximation (one size does not fit all)
   - Lets talk about splines

Splines

Splines

Spline function: A function that consists of polynomial pieces joined together with some smoothness conditions.

- **Linear splines:** Use linear polynomials (straight lines) to join nodes
  - Easy to calculate. For $x \in [x_i, x_{i+1}]$ we just have:

$$\tilde{V}(x) = A(x) \cdot V(x_i) + B(x) \cdot V(x_{i+1})$$

    where: $A(x) = \frac{x_{i+1}-x}{x_{i+1}-x_i} \qquad B(x) = 1 - A(x) = \frac{x-x_i}{x_{i+1}-x_i}$

  - Resulting function is continuous but not smooth
    - Curse of dimensionality applies if looking for good approximation
  - First derivatives do not exist at nodes $\{x_1, \ldots, x_N\}$ (FOC)
  - However: Fast, robust method

Splines - Linear splines

Algorithm 1: Linear Splines

Result: Interpolated value y_hat at point x

Define grids ;

x_grid = $(x_1, \ldots, x_N)$;

y_grid = $(y_1, \ldots, y_N)$;

Locate closest indeces to x on grid ;

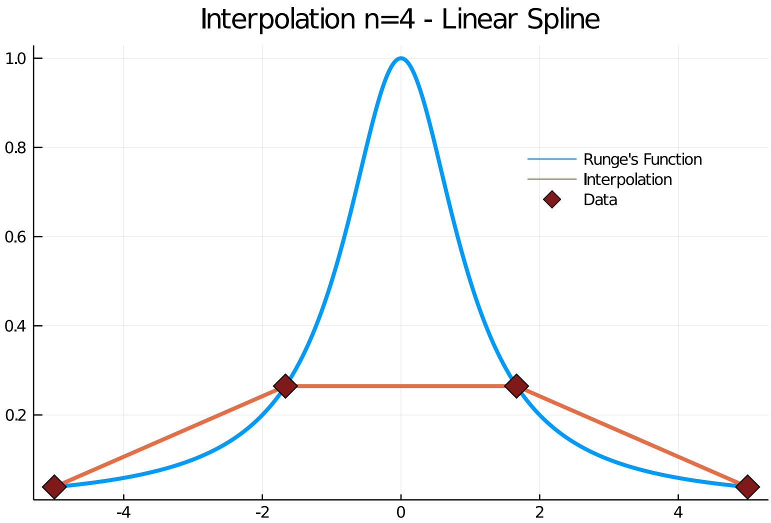
ind = findmax(sign.(x_grid .- x))[2] - 1 ;

Compute interpolation ;

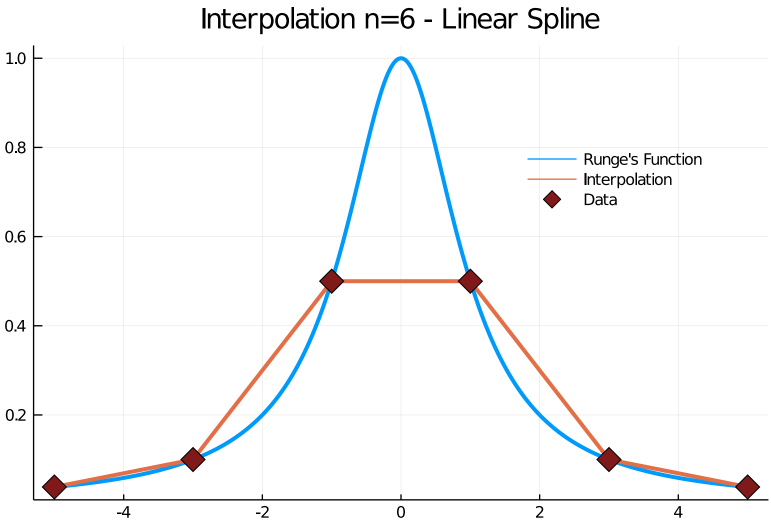
A_x = (x_grid[ind+1] - x)/(x_grid[ind+1]-x_grid[ind]) ;

y_hat = A_x*y_grid[ind] + (1-A_x)*y_grid[ind+1] ;

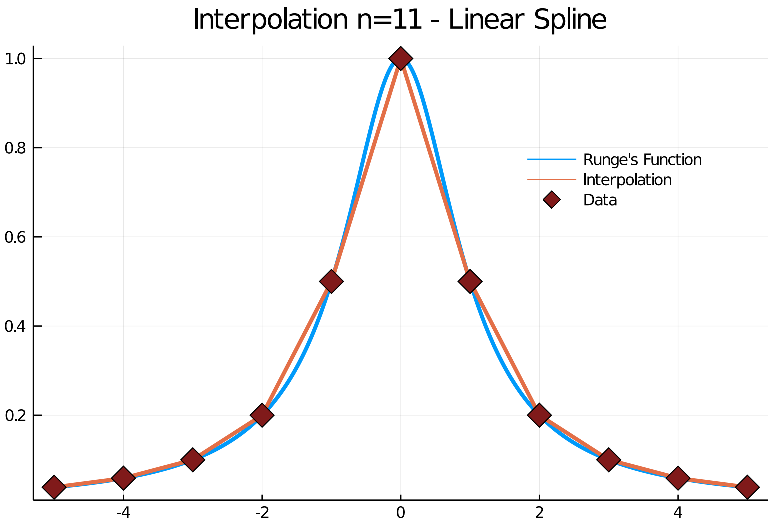
Runge example: $f(x) = 1/_{1+x^2}$ - Linear Splines



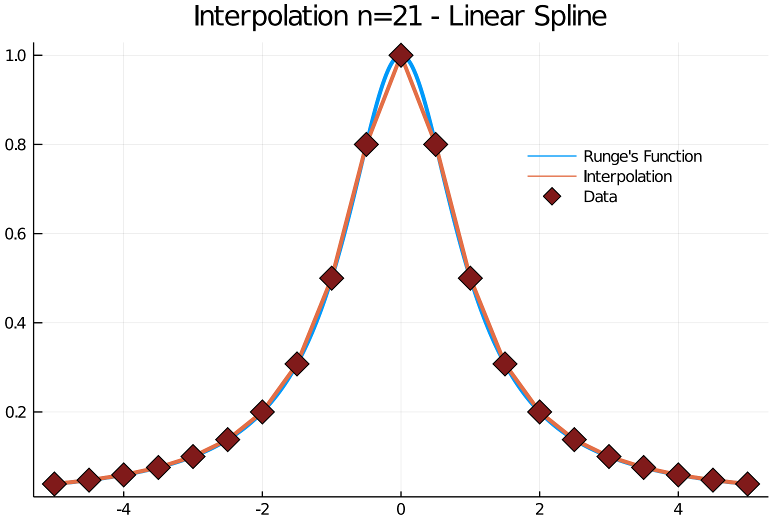
Runge example: $f(x) = {}^{1}/_{1+x^2}$ - Linear Splines



Runge example: $f(x) = 1/_{1+x^2}$ - Linear Splines



Runge example: $f(x) = 1/1+x^2$ - Linear Splines



Spline - Cubic splines

Use cubic polynomials to join nodes so that:

1. Match nodes exactly $\tilde{V}(x_i) = V(x_i)$ for $x_i \in \{x_1, \ldots, x_N\}$
2. First derivatives are continuously differentiable everywhere
3. Second derivatives are continuous everywhere

Cubic splines - Importance of derivatives

This is a preferred method for economic applications:

- First derivatives obtained as a byproduct of interpolation
- Easy to compute (invert a tri-diagonal system)

Derivatives are key:

- Many optimization algorithms are gradient-based
- First order conditions (Euler equation) depend on V'
- EGM depends on V' to avoid solving the Euler equation

(How to) Cubic splines

We are using cubic polynomials, so the second derivative is linear!

We want our approximation to satisfy:

$$\tilde{V}''(x) = A(x)\, V''(x_i) + B(x)\, V''(x_{i+1})$$

- This guarantees that $\tilde{V}''(x_i) = V''(x_i)$ and that second derivatives are continuous
- Then $\tilde{V}$ is twice continuously differentiable

Note: For easy of exposition I will change from $V(x)$ to y notation

(How to) Cubic splines

Twice continuous differentiability implies that:

$$\tilde{y}\left(x\right)=A\left(x\right)\cdot y_{i}+B\left(x\right)\cdot y_{i+1}+C\left(x\right)\cdot y_{i}^{''}+D\left(x\right)\cdot y_{i+1}^{''}$$

where: $C\left(x\right)=\frac{1}{6}\left(A^{3}\left(x\right)-A\left(x\right)\right)\left(x_{i+1}-x_{i}\right)^{2}$ and $D\left(x\right)=\frac{1}{6}\left(B^{3}\left(x\right)-B\left(x\right)\right)\left(x_{i+1}-x_{i}\right)^{2}$

- **Good news:** You only need to compute A and B to get all coefficients
- **Bad news:** You need to find out values for $\left\{y_{i}^{''}\right\}$...

(How to) Cubic splines

How to solve for the unknown second derivatives? With first derivatives!

- First derivative of $\tilde{V}$ satisfies:

$$\frac{\partial y}{\partial x}(x) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{x_{i+1} - x_i}{6}\left[\left(3A(x)^2 - 1\right) y_i^{''} - \left(3B(x)^2 - 1\right) y_{i+1}^{''}\right]$$

  Note that once we know $y^{''}$ we get first derivative for free!

(How to) Cubic splines

How to solve for the unknown second derivatives? With first derivatives!

- First derivative of $\tilde{V}$ satisfies:

$$\frac{\partial y}{\partial x}(x) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{x_{i+1} - x_i}{6}\left[\left(3A(x)^2 - 1\right)y_i^{''} - \left(3B(x)^2 - 1\right)y_{i+1}^{''}\right]$$

  Note that once we know $y^{''}$ we get first derivative for free!

- But we want these derivatives to be continuous (at the grid nodes):

$$\underbrace{\frac{y_i - y_{i-1}}{x_i - x_{i-1}} - \frac{x_i - x_{i-1}}{6}\left[-y_{i-1}^{''} - 2y_i^{''}\right]}_{\lim_{x \to x_i^-} \frac{\partial y}{\partial x}} = \underbrace{\frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{x_{i+1} - x_i}{6}\left[2y_i^{''} + y_{i+1}^{''}\right]}_{\lim_{x \to x_i^+} \frac{\partial y}{\partial x}}$$

(How to) Cubic splines

How to solve for the unknown second derivatives? With first derivatives!

- First derivative of $\tilde{V}$ satisfies:

$$\frac{\partial y}{\partial x}(x) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{x_{i+1} - x_i}{6}\left[\left(3A(x)^2 - 1\right) y_i^{''} - \left(3B(x)^2 - 1\right) y_{i+1}^{''}\right]$$

  Note that once we know $y^{''}$ we get first derivative for free!

- Rearrange:

$$\underbrace{\underbrace{\frac{x_i - x_{i-1}}{6}}_{c_{i-1}} y_{i-1}^{''} + \underbrace{\frac{x_{i+1} - x_{i-1}}{3}}_{d_i} y_i^{''} + \underbrace{\frac{x_{i+1} - x_i}{6}}_{c_i} y_{i+1}^{''}}_{\text{Linear system on } y^{''}} = \underbrace{\underbrace{\frac{y_{i+1} - y_i}{x_{i+1} - x_i}}_{s_i} - \underbrace{\frac{y_i - y_{i-1}}{x_i - x_{i-1}}}_{s_{i-1}}}_{\text{Diff. of Slopes (Known)}}$$

(How to) Cubic splines

- The last equation holds in all interior nodes of the grid
  - We have $N-2$ equations but N unknowns...
- To solve this we need to impose boundary conditions:
  - **Natural Spline:** Spline is linear at boundaries $y_1'' = y_N'' = 0$
    - This is the normal assumption
    - Helps for extrapolation (more on this at the end)
  - **Flat Spline:** Spline is flat at boundaries $y_1' = y_N' = 0$

(How to) Cubic splines

To find cubic splines solve this (tri-diagonal) linear system:

$$\begin{bmatrix} 2c_1 & -c_1 & & & & & & \\ c_1 & d_i & c_2 & & & & & \\ & & \ddots & & & & & \\ & & c_{i-1} & d_i & c_i & & & \\ & & & & \ddots & & & \\ & & & & c_{N-2} & d_{N-1} & c_{N-1} & \\ & & & & & & -c_N & 2c_N \end{bmatrix} \cdot \begin{bmatrix} y_1^{''} \\ y_2^{''} \\ \vdots \\ y_i^{''} \\ \vdots \\ y_{N-1}^{''} \\ y_N^{''} \end{bmatrix} = \begin{bmatrix} s_1 - b_1 \\ s_2 - s_1 \\ \vdots \\ s_i - s_{i-1} \\ \vdots \\ s_{N-1} - s_{N-2} \\ s_N - b_N \end{bmatrix}$$

$$Cy^{''} = S$$

(How to) Cubic splines

Algorithm 2: Cubic Splines

Result: Interpolated value y_hat at point x

Define grids and boudnary conditions (either on y' or y'') ;

x_grid = $(x_1, \dots, x_N)$ y_grid = $(y_1, \dots, y_N)$;

Solve tri-diagonal system for vector of y'': y_pp = C\S ;

Locate closest indeces to x on grid ;

ind = findmax(sign.(x_grid .- x))[2] - 1 ;

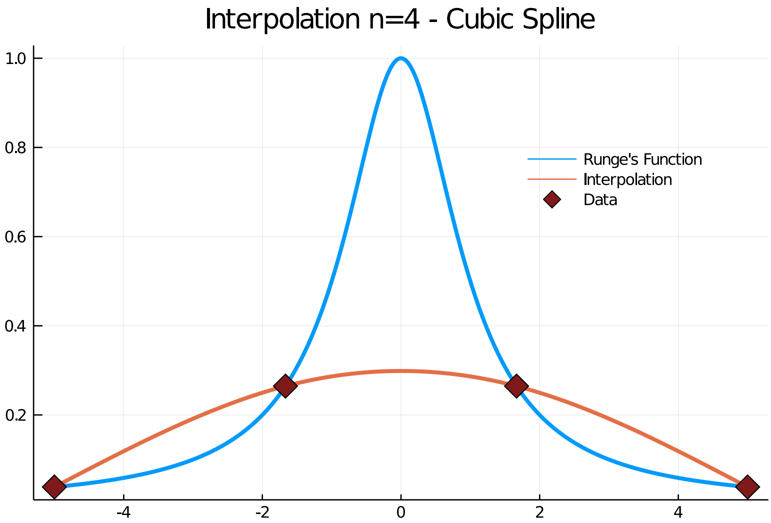
Compute interpolation ;

A_x = (x_grid[ind+1] - x)/(x_grid[ind+1]-x_grid[ind]) ;

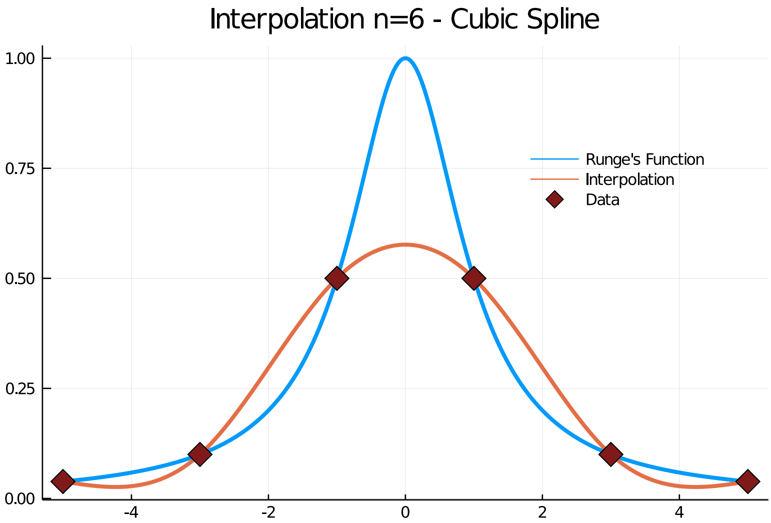
Compute B_x, C_x, D_x accordingly ;

y_hat = A_x*y_grid[ind] + (1-A_x)*y_grid[ind+1] +
C_x*y_pp[ind] + D_x*y_pp[ind+1] ;

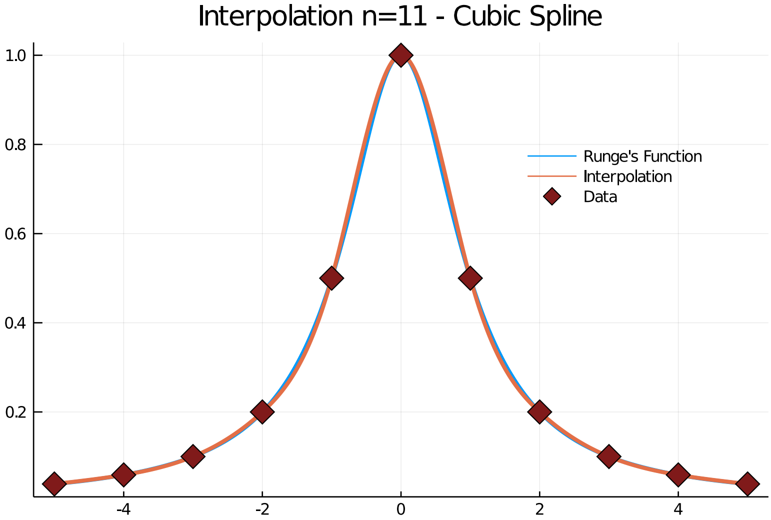
Runge example: $f(x) = 1/_{1+x^2}$ - Cubic Splines



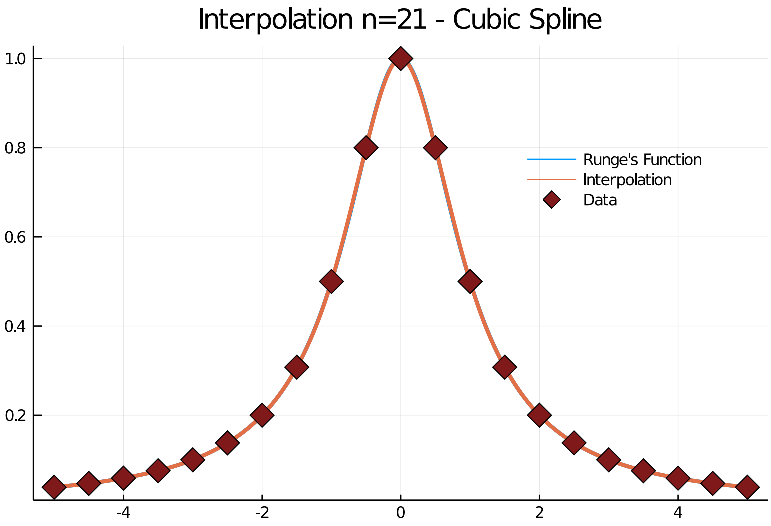
Runge example: $f(x) = {}^{1}/_{1+x^2}$ - Cubic Splines



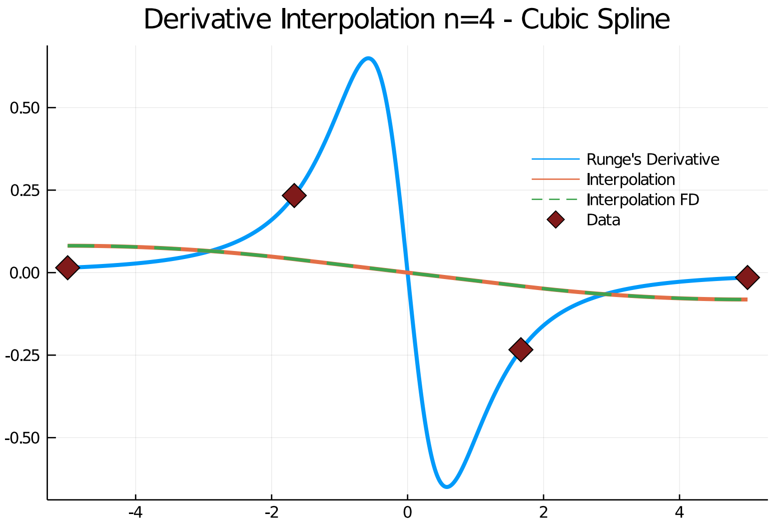
Runge example: $f(x) = 1/_{1+x^2}$ - Cubic Splines



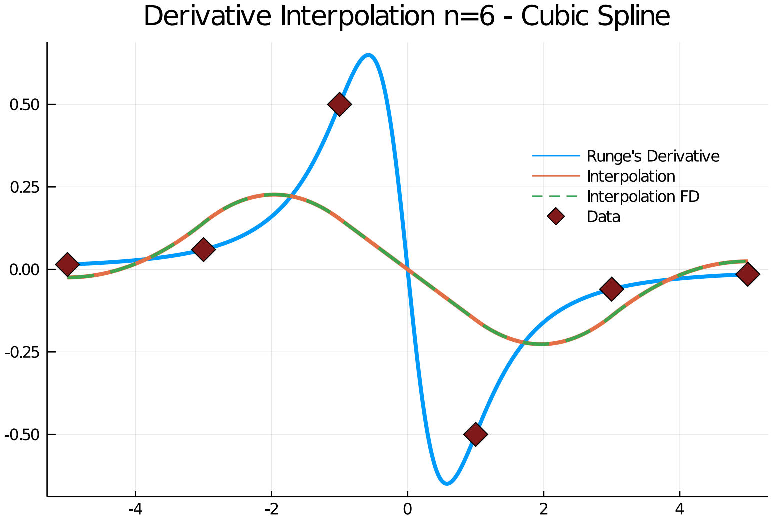
Runge example: $f(x) = 1/1+x^2$ - Cubic Splines



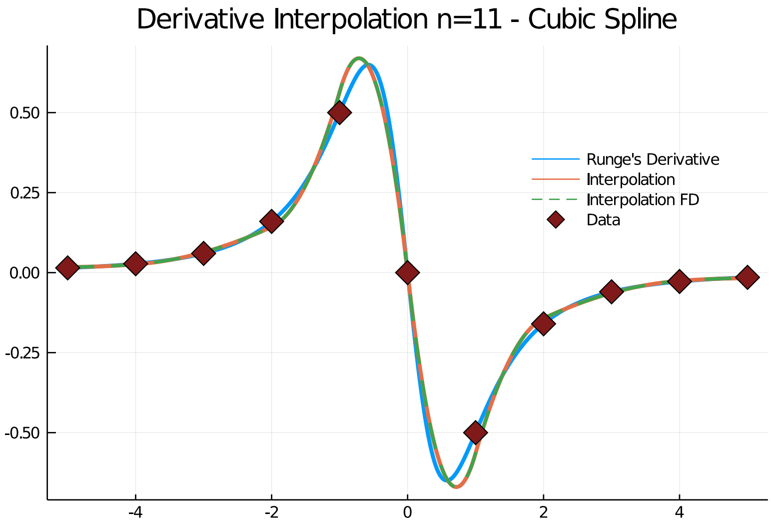
Runge example: $f(x) = {}^{1}/_{1+x^2}$ - Derivative



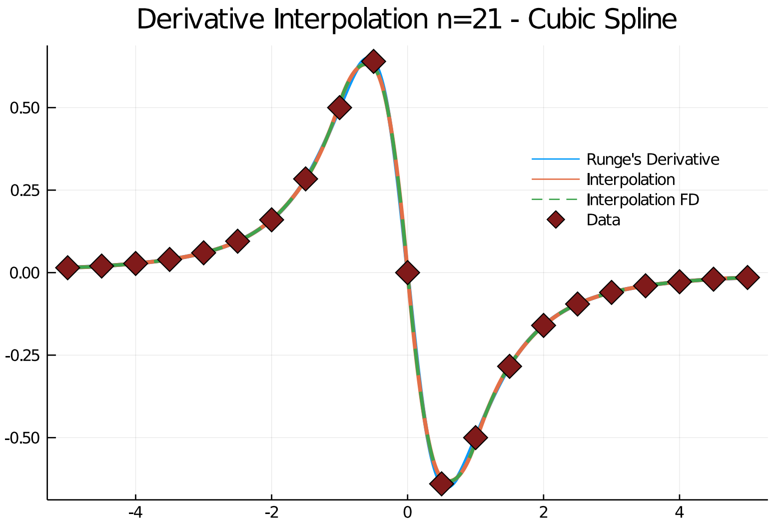
Runge example: $f(x) = {}^{1}/_{1+x^2}$ - Derivative



Runge example: $f(x) = {}^{1}\!/_{1+x^2}$ - Derivative



Runge example: $f(x) = 1/_{1+x^2}$ - Derivative

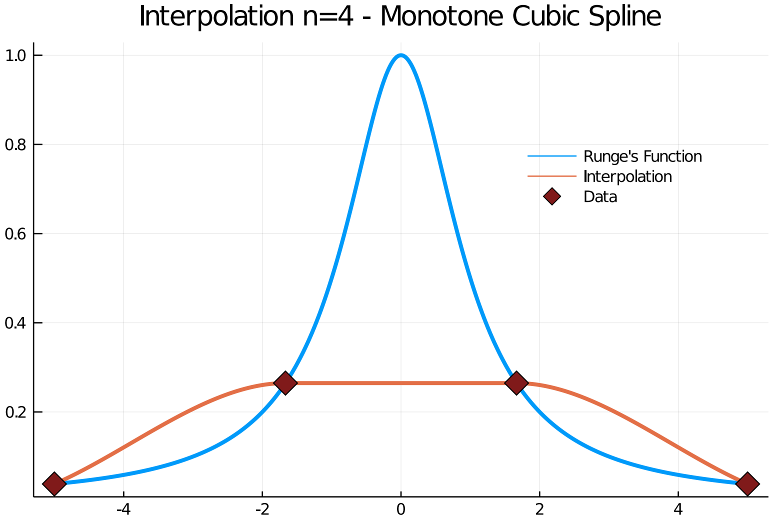


Spline - Shape preserving splines

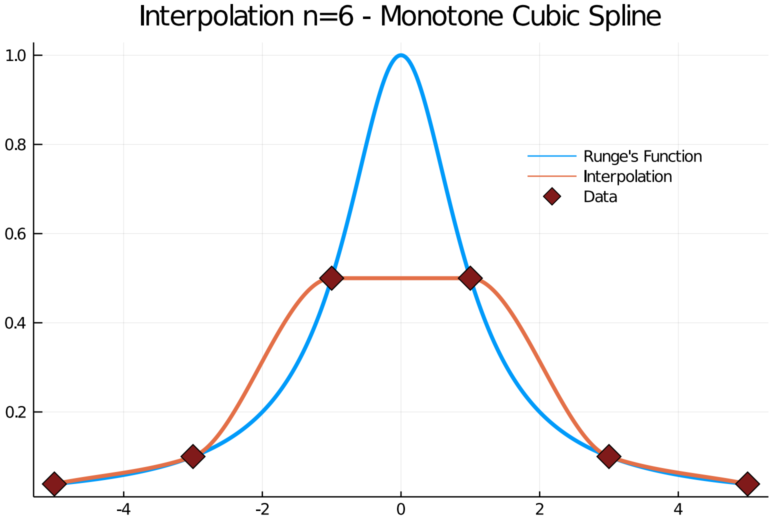
There are other types of splines (of course!)

- **Monotone Splines:**
  - Cubic polynomials between nodes
  - Continuous first derivatives, but not necessarily second derivatives
  - Choose the slopes at $\{x_i\}$ so that interpolation respects monotonicity
    - On intervals where the data is monotonic, so is the spline, and at points where the data has a local extremum, so does the spline
- **Schumaker Splines:**
  - Quadratic splines preserving monotonicity or concavity
  - Faster to compute, oscillates less, worth checking out
  - Shape restrictions already mess up second derivatives

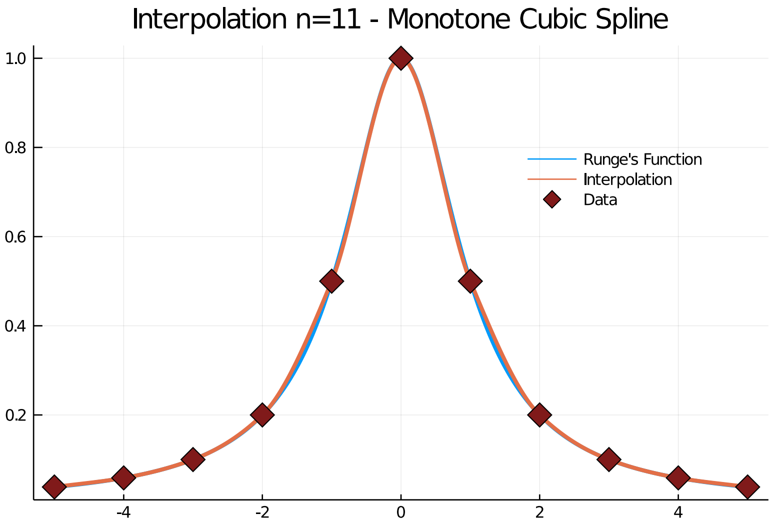
Runge example: $f(x) = 1/1+x^2$ - Montone Splines



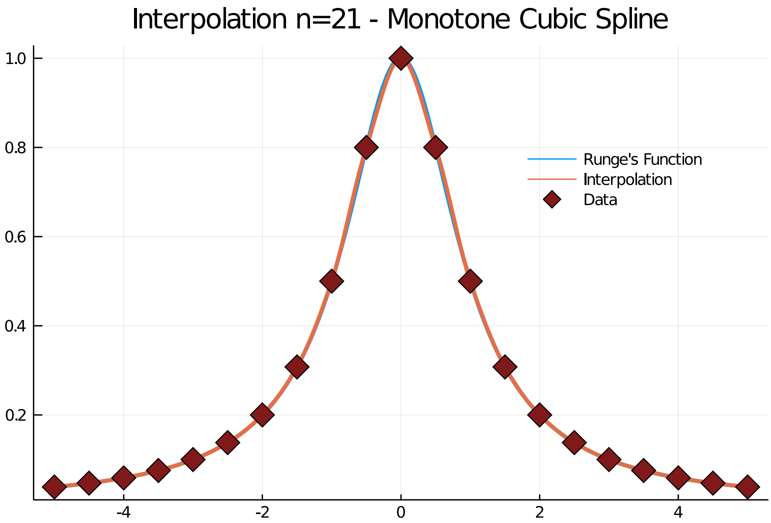
Runge example: $f(x) = {}^{1}/_{1+x^2}$ - Montone Splines



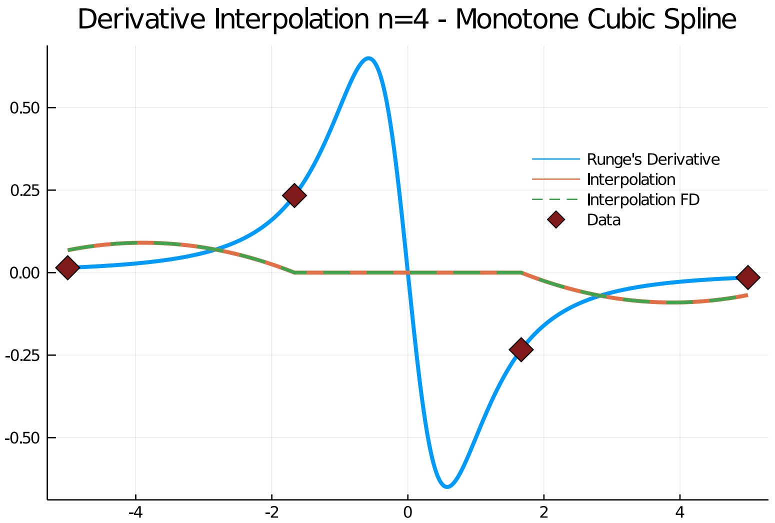
Runge example: $f(x) = {}^{1}/_{1+x^2}$ - Montone Splines



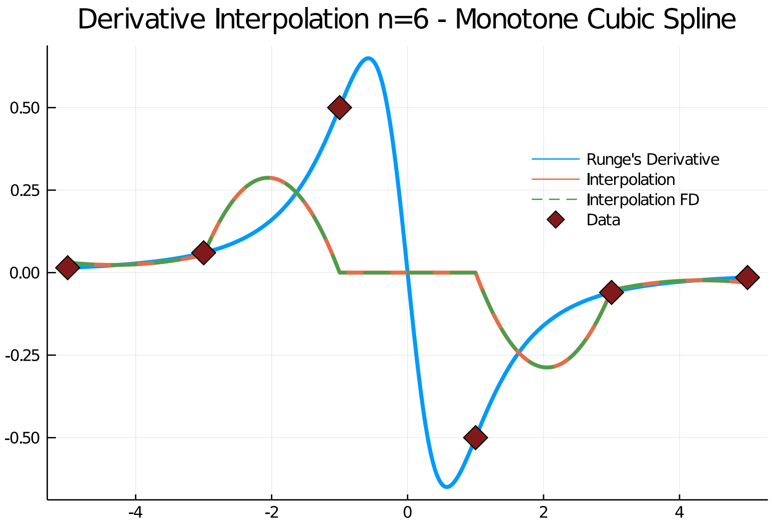
Runge example: $f(x) = {}^{1}/_{1+x^2}$ - Montone Splines



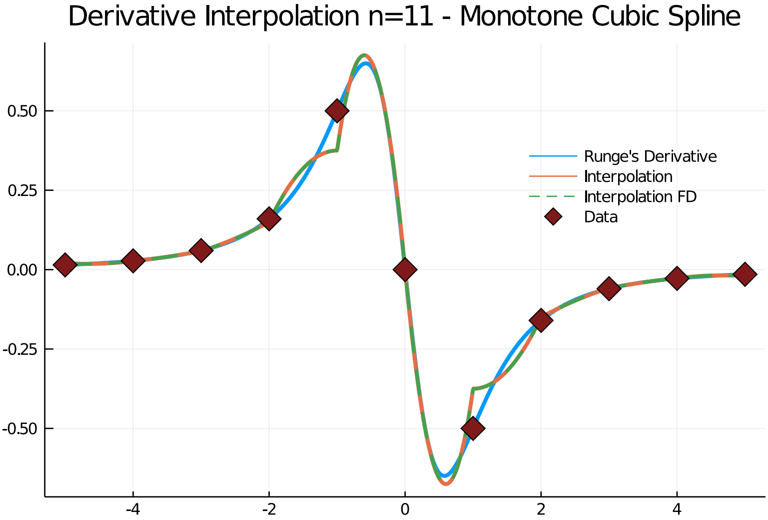
Runge example: $f(x) = {}^{1}/_{1+x^2}$ - Derivative



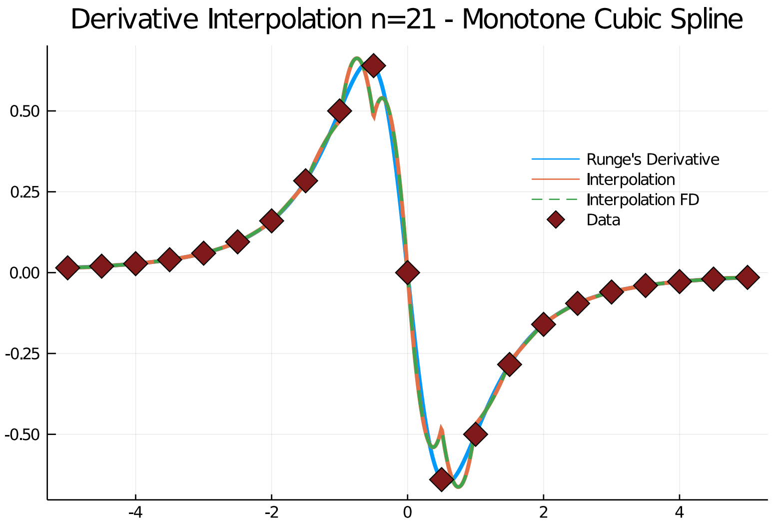
Runge example: $f(x) = 1/1+x^2$ - Derivative



Runge example: $f(x) = 1/_{1+x^2}$ - Derivative



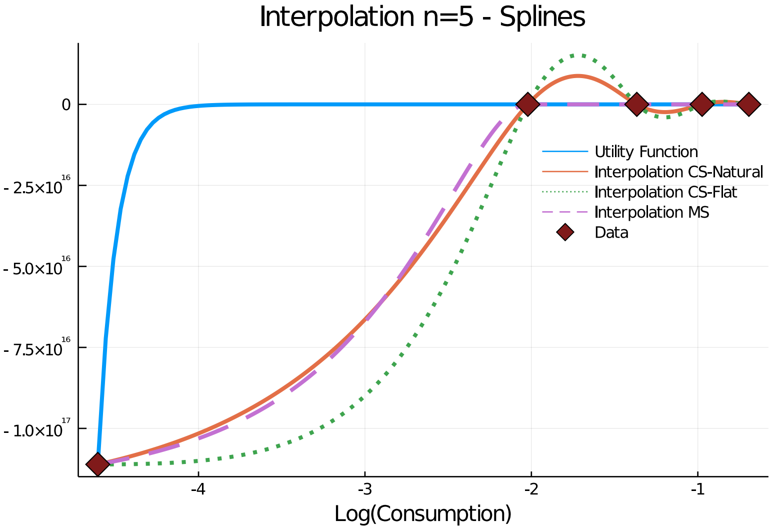
Runge example: $f(x) = {}^{1}/_{1+x^2}$ - Derivative



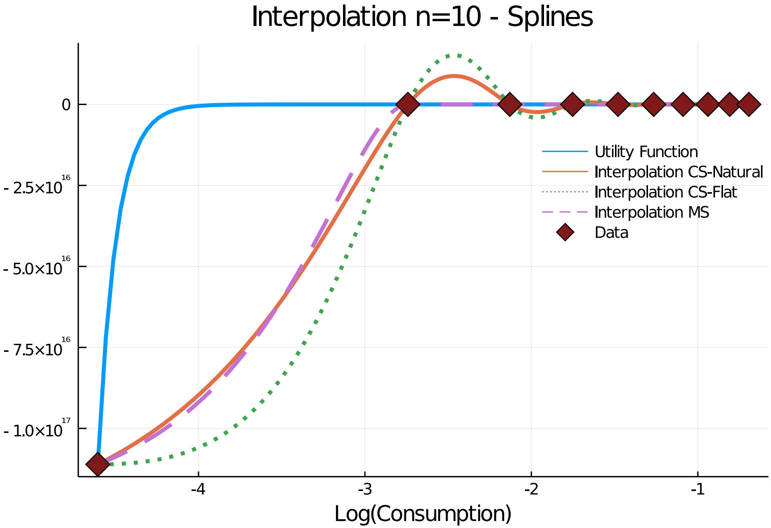
Spline - Monotone splines

- A good idea when cubic splines are too wavy or jumpy
  - Important functions with a lot of curvature
- You pay the price with potentially funky first derivatives
- Important to test your interpolation on the type of functions you use
  - Hard to know ex-ante what will work

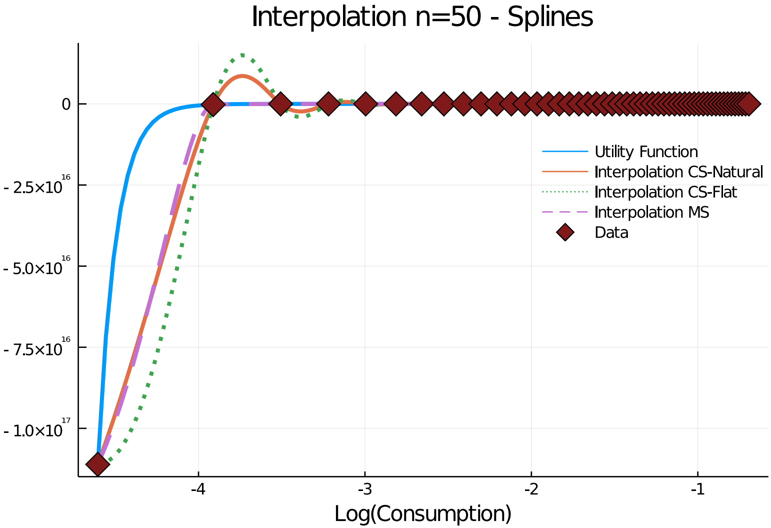
CRRA $u(c) = \frac{c^{1-\sigma}}{1-\sigma}$; $\sigma = 10$



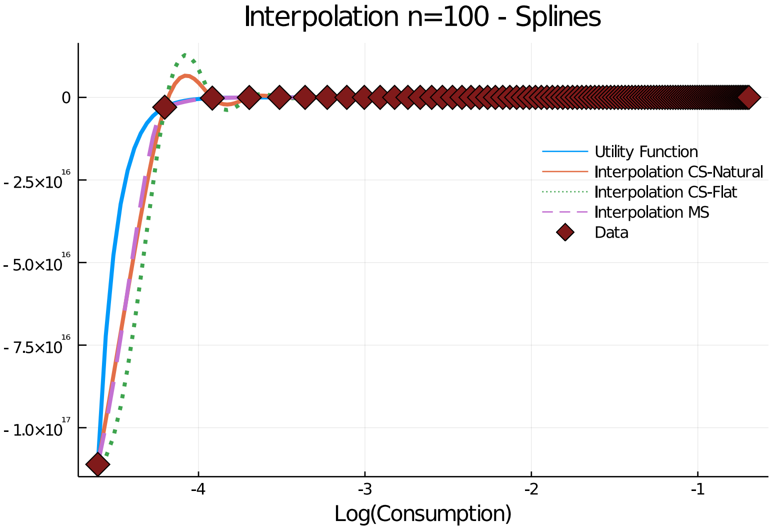
CRRA $u(c) = \frac{c^{1-\sigma}}{1-\sigma}$; $\sigma = 10$



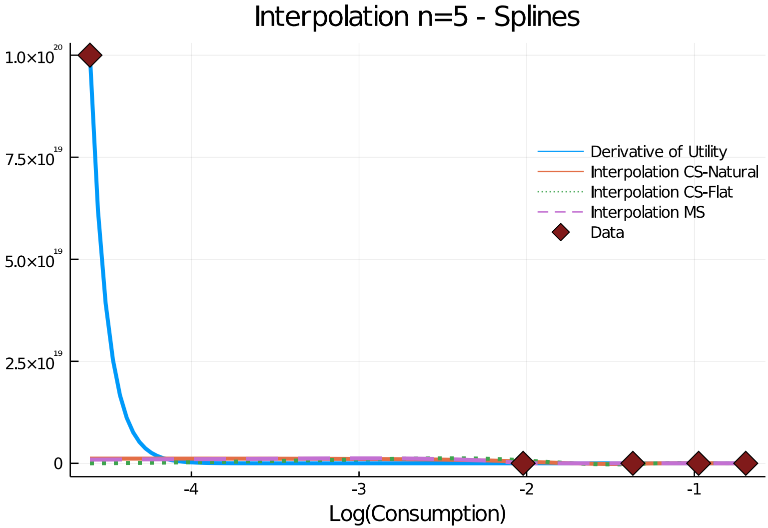
CRRA $u(c) = \frac{c^{1-\sigma}}{1-\sigma}$; $\sigma = 10$



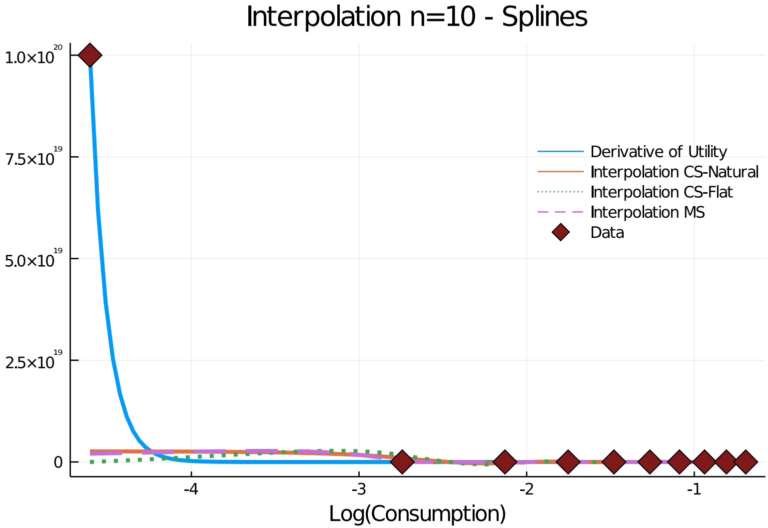
CRRA $u(c) = \frac{c^{1-\sigma}}{1-\sigma}$; $\sigma = 10$



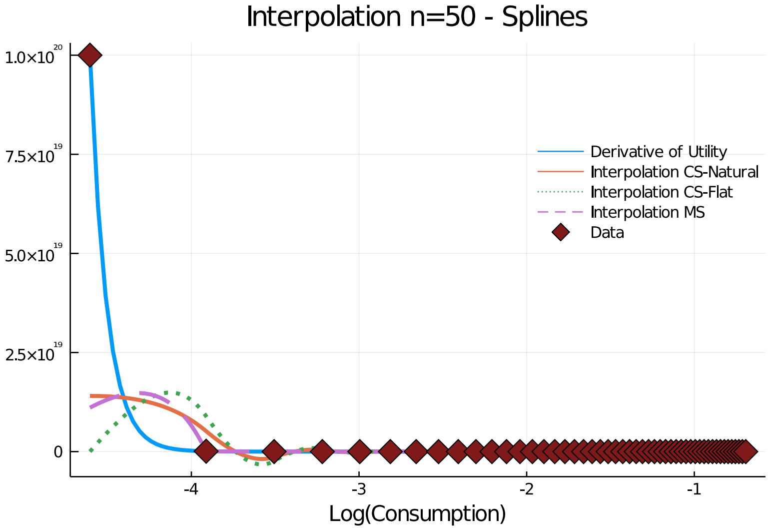
CRRA $u(c) = \frac{c^{1-\sigma}}{1-\sigma}$; $\sigma = 10$ - Derivatives



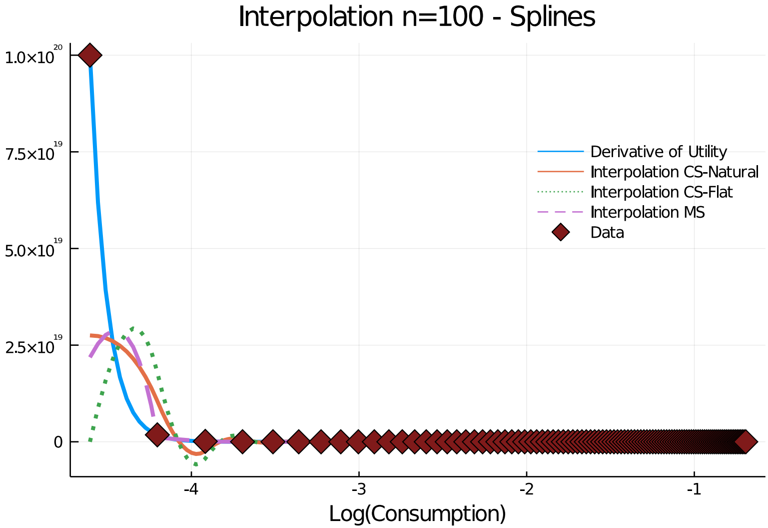
CRRA $u(c) = \frac{c^{1-\sigma}}{1-\sigma}$; $\sigma = 10$ - Derivatives



CRRA $u(c) = \frac{c^{1-\sigma}}{1-\sigma}$; $\sigma = 10$ - Derivatives



CRRA $u(c) = \frac{c^{1-\sigma}}{1-\sigma}$; $\sigma = 10$ - Derivatives



Boundary conditions

- No good approximation at the bottom...
- Reason: Bad boundary conditions
  - Natural spline has $u'' = 0$, flat spline is worse with $u' = 0$
- Monotone spline performs better in level... but can't capture lower end

Solution: Supply your own first order conditions

- You have to write your own function for this

Grid Spacing

Grid spacing

- Part of the problem of interpolating is that we are wasting information
- Too many nodes in uninteresting parts of the function
- How to better allocate grid space?
  1. Put more grid nodes where there is more curvature!
  2. More Put more grid nodes where it matters (say around k_{ss})
- This also affects kinks
  - Kinks (coming from a discrete choice) change curvature
  - Better to deal with them with linear interpolation
  - You need more points there

Grid spacing - Algorithm

Algorithm 3: Curved Grid: Polynomial or Exponential Scaling

Function `Curved_Grid(`n,a,b,θ,$Type$`)`:

grid = range(0,1,length=n)

if *Type==Polynomial* **then**

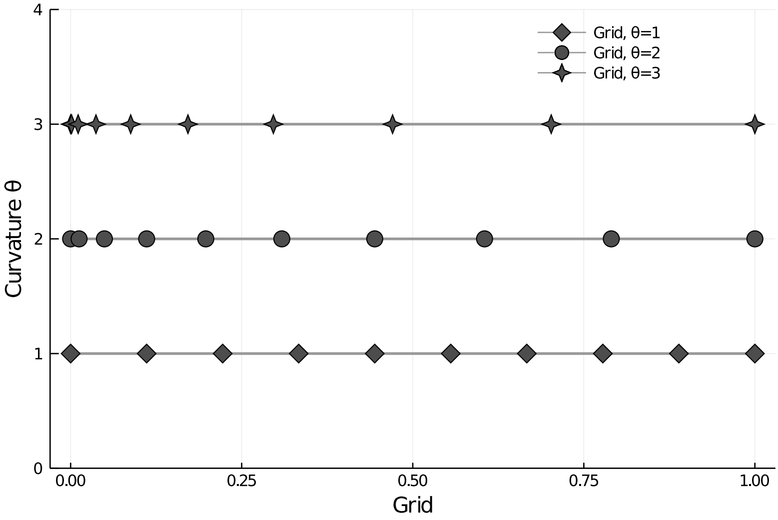
grid = a + (b-a)*$grid^{\theta}$

if *Type==Exponential* **then**

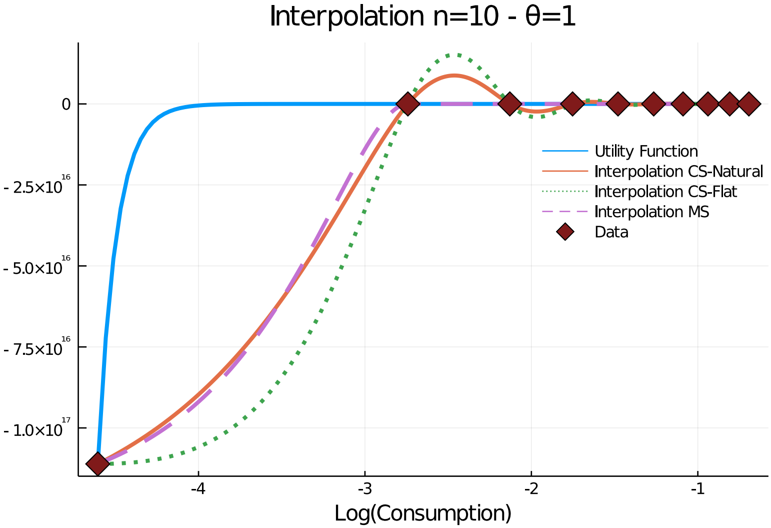
grid = a + (b-a)*$\frac{exp(\theta * grid) - 1)}{exp(\theta) - 1}$

return grid

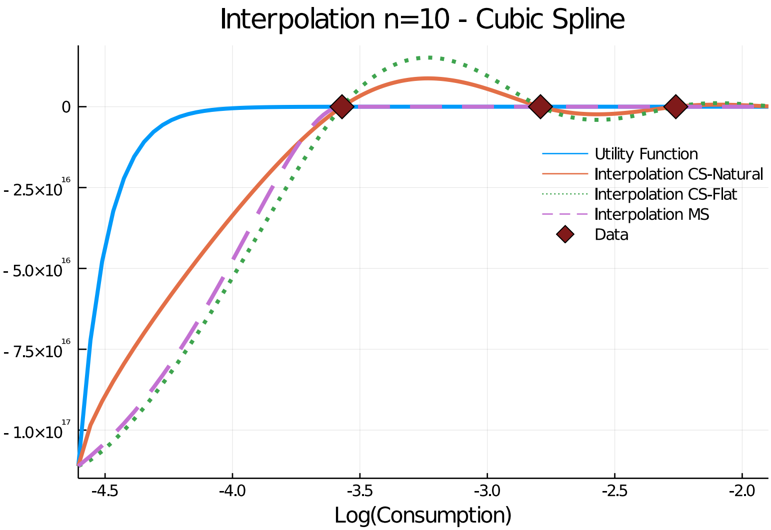
Grid spacing - Polynomial grid example



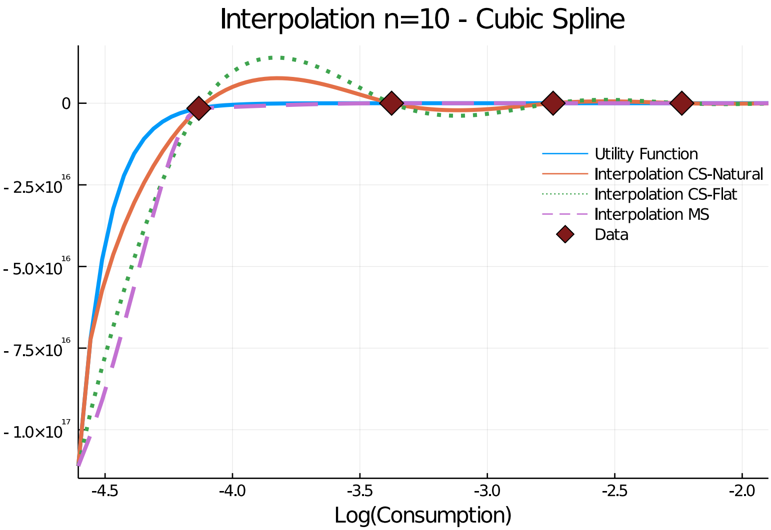
Grid spacing - Back to CRRA



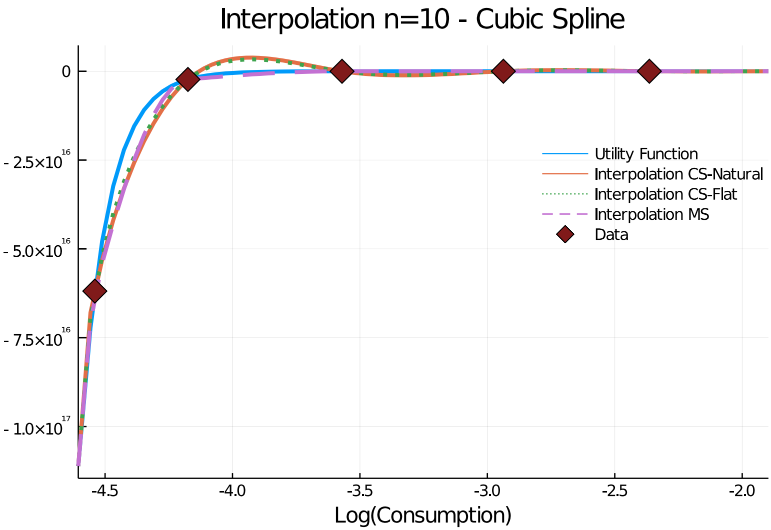
Grid spacing - Back to CRRA



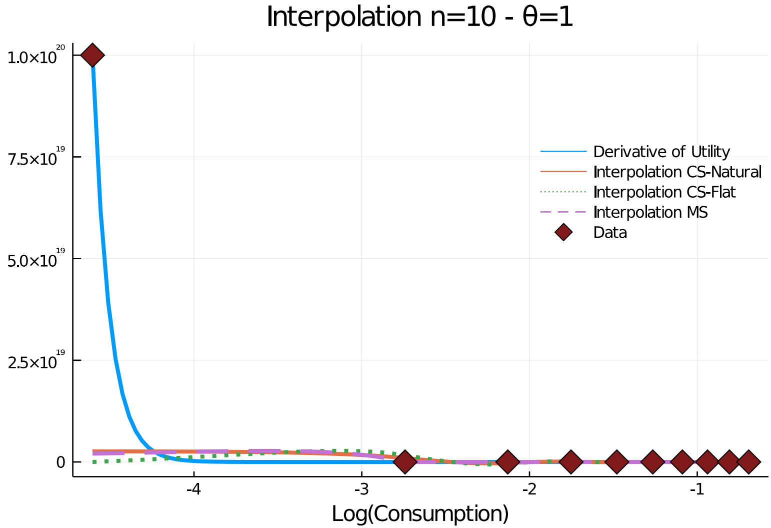
Grid spacing - Back to CRRA



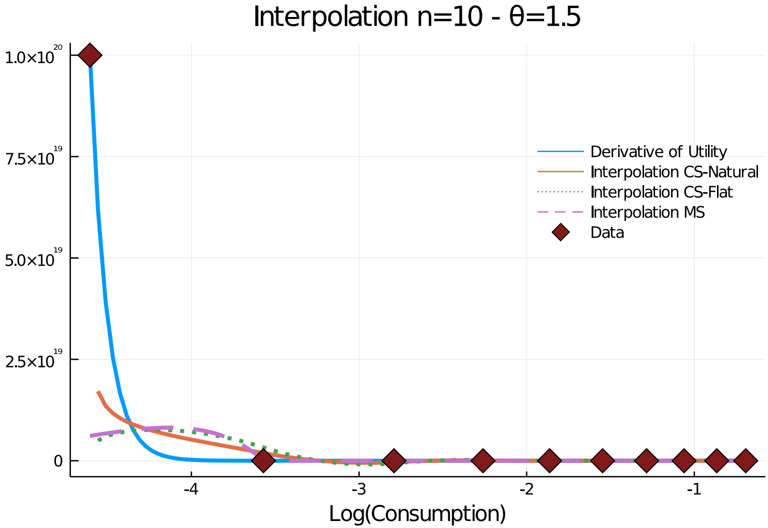
Grid spacing - Back to CRRA



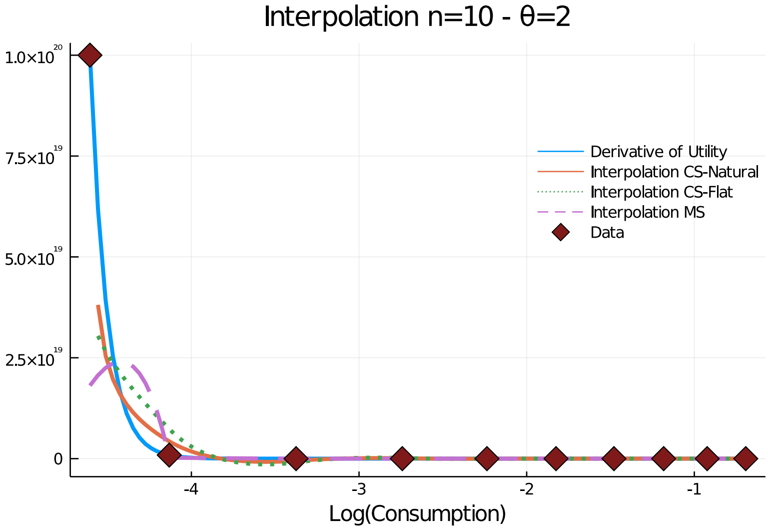
CRRA $u(c) = \frac{c^{1-\sigma}}{1-\sigma}$; $\sigma = 10$ - Derivatives



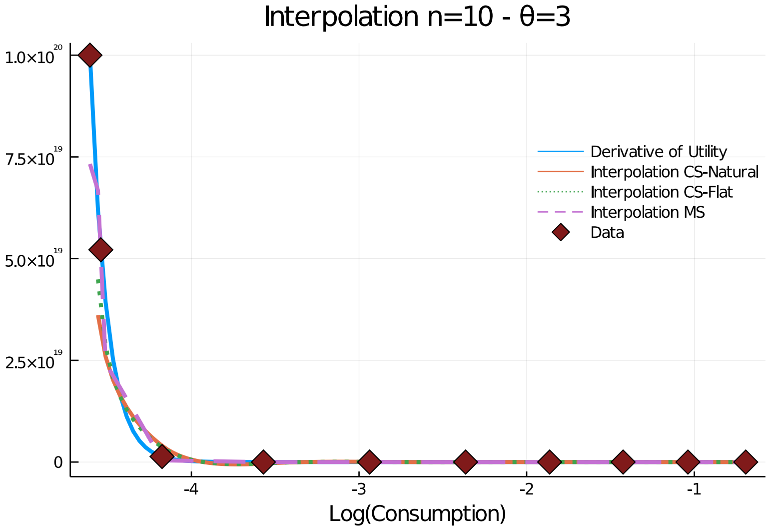
CRRA $u(c) = \frac{c^{1-\sigma}}{1-\sigma}$; $\sigma = 10$ - Derivatives



CRRA $u(c) = \frac{c^{1-\sigma}}{1-\sigma}$; $\sigma = 10$ - Derivatives



CRRA $u(c) = \frac{c^{1-\sigma}}{1-\sigma}$; $\sigma = 10$ - Derivatives



Final Words

Extrapolation - Just don't

- Extrapolating is dangerous
  - Extrapolating is lethal if you use high degree polynomials
- Abstain at all costs from extrapolating
- If you must extrapolate use linear extrapolation
- Unless you have some theory on your side
  - Theory is great because it tells you what to do!
  - Ex: Pareto Extrapolation:
    An Analytical Framework for Studying Tail Inequality by Akira-Toda & Gouin-Bonenfant

Coda: Practical advice

- Always re-solve your models on a much finer grid and confirm that your main results are dependent on grid size
  - Only practical way to check impact of approximation errors coming from interpolations
- Don't go for the bazooka! Often times simpler methods work best
  - You will be surprised to find that some bad-looking interpolations actually yield the same results as much more accurate (and more costly to compute) interpolations.
  - Value robustness of the method over fancy tools
- All rules have exceptions... Sometimes you cannot make approximation errors, you will need specialized algorithms tailored to your problem