University of Minnesota
ECON 8581 Prof. Fatih Guvenen
Homework 2 November 17, 2022
Angelo Mendes

Interpolation Methods

The codes in this section were all ran in MATLAB using the function *interp1*. For this function, the default method is already the linear interpolation. For us to implement cubic splines it is enough to set *method='spline'*[1][2]. This function allow me to both evaluate and store the coefficients of the new spline function. The first lesson learned here is that with enough knot points interpolated versions of the utility function will always (or at least almost always) seem good. Since we aim to use the least amount of points as possible, I started with few knot-points, lets say 10, and plot differences between linear and spline interpolations for $\theta \in \{1, 1.5, 3, 4\}$. As expected, since we are taking too few gridpoints, some of the interpolations seems bad.

Finally, I construct a function that takes the parameters of the problem and which of the utility specifications I want to evaluate and give me the number of knot-points, n, such that $\|u(x) - \hat{u}(x)\|_\infty < \epsilon$. The code starts with a guess n_0, makes the interpolation using n_0 points and evaluates the interpolation in a finer grid (10,000 gridpoints).

[1] The cubic spline implemented by Matlab requires **second** derivatives to be continuous. An alternative method function *"pchip"* requires **only first** derivatives to be continuous, which makes this interpolation method less smooth than the cubic spline. In cases where we have an accentuated curvature (as in the CRRA with $\alpha = 10$) this may be a drawback of the smoothness of the cubic spline. Even though we implemented just one of those, it is important to know potential drawbacks, and at least know there exists an alternative (actually, there are probably many of them). In this specific case, I think we can overcome spline's "oversmoothness" since the curvature is clustered in a specific region of the grid. Then, concentrating points in that region can help a lot.

[2] Check the boundary conditions used by those functions. We also have *"csape"* that allow us to choose boundary conditions.

	$u(c) = \log(c)$		$u(c) = \sqrt{c}$	
	Linear	Spline	Linear	Spline
$\theta = 1.0$	1172	118	55	21
$\theta = 1.5$	990	67	69	33
$\theta = 3.0$	331	46	49	28
$\theta = 4.0$	194	36	48	23

Table 1: Number of knot-points needed for the error to be at most 1% of the function value (10,000 gridpoints)

	$u(c) = \frac{c^{1-\alpha}}{1-\alpha}$					
	Linear			Spline		
	$\alpha = 2$	$\alpha = 5$	$\alpha = 10$	$\alpha = 2$	$\alpha = 5$	$\alpha = 10$
$\theta = 1.0$	186	626	1348	73	218	451
$\theta = 1.5$	215	678	1418	94	255	511
$\theta = 3.0$	141	430	868	53	106	197
$\theta = 4.0$	122	379	824	44	89	162

Table 2: Same exercise for different values of α in a CRRA utility function

Notice that even in the worst scenarios above we need less than 15% of the finer grid to get a really good approximation. Also, it is clear that splines perform better than the linear interpolation, since they need less knot-points to deliver a good approximation. Of course, as α increases we need more gridpoints. Finally, it is interesting to note that in some functions, like $\sqrt{c}$, we don't really need high values of θ[3]

For the Chebyshev polynomials, I started from the 5th order polynomial just to generate some plots, and then I created a function that updates the order p of the polynomial and evaluates the approximation until it reaches the tolerance ϵ (pretty similar to what I have done for the linear interpolation and splines). In this case here, unsurprisingly

[3] Not a big difference between $\theta = 3$ and $\theta = 4$. The idea, I think, is that the curvature of this function is not clustered in a specific region of the grid, as in the CRRA case.

(as we saw in the Runge example in class) the polynomials really struggle to get a good approximation. I reduced the amount of points in the consumption grid from 10,000 to 1,000 and got $p_1 = 18$ and $p_2 = 11$, where p_i is the optimal order of the polynomial that approximates the utility function u_i. For the CRRA function the error does not converge even for less gridpoints. [4]

The Perils of Extrapolation

We now want to see how the functions generated by the extrapolation methods perform outside the boundaries of the grid. More formally, we are going to evaluate $\hat{u}(c^*)$, for $c^* \in \{0.05, 5.1, 5.5, 6\}$ and compare with the real values $u(c)$ in those same values.

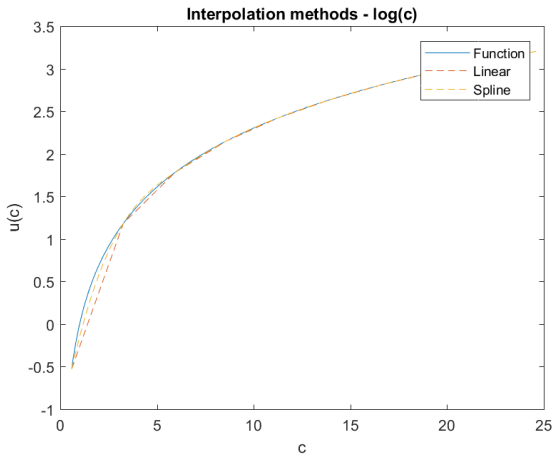
	$\hat{u}_1(c^*)$			$\hat{u}_2(c^*)$		
	linear	spline	$\log(c^*)$	linear	spline	$\sqrt{c^*}$
$c^* = 0.05$	-2.4737	-2.5800	-2.9957	0.2715	0.2558	0.2236
$c^* = 5.10$	1.6306	1.6293	1.6292	2.2591	2.2583	2.2583
$c^* = 5.50$	1.7153	1.7058	1.7047	2.3511	2.3455	2.3452
$c^* = 6.00$	1.8212	1.7964	1.7918	2.4661	2.4509	2.4495

	CRRA ($\alpha = 2$)		
	linear	spline	real
$c^* = 0.05$	-10.7759	-11.4834	-20.0000
$c^* = 5.10$	-0.1955	-0.1959	-0.1961
$c^* = 5.50$	-0.1776	-0.1792	-0.1818
$c^* = 6.00$	-0.1551	-0.156	-0.1667

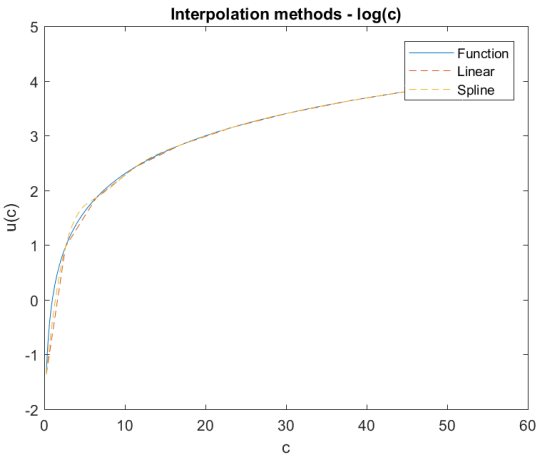
[4] **Obs:** The Chebyshev polynomials are polynomials from $[-1, 1]$ to $[-1, 1]$ scaled by some coefficients in order to fit a function. We can guarantee that we can make an arbitrarily good approximation of a smooth by polynomials since the Chebyshev polynomials are the basis for the space of smooth functions. Finally, even though they do not perform well to interpolate utility functions, those polynomials can be useful, **since its roots can provide us a good strategy on how to optimally locate gridpoints.**

Plots

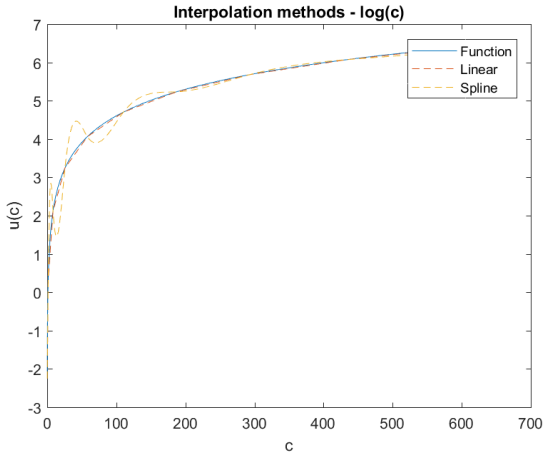
$$u(c) = \log(c)$$



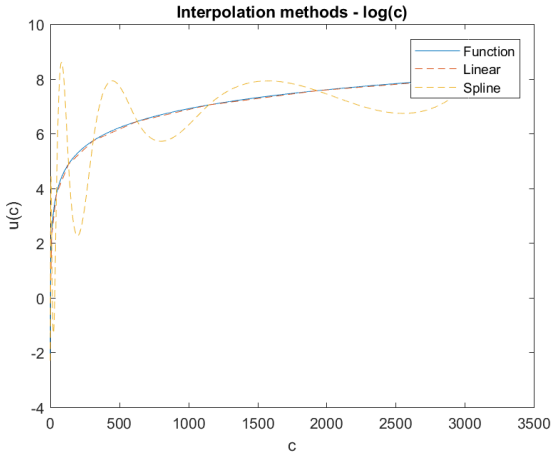
(a) $\theta = 1$



(b) $\theta = 1.5$

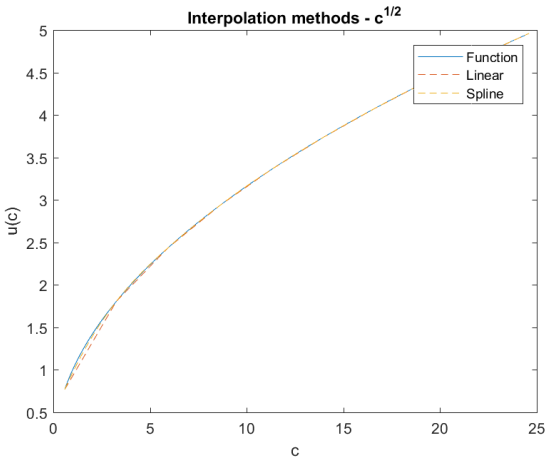


(c) $\theta = 3$

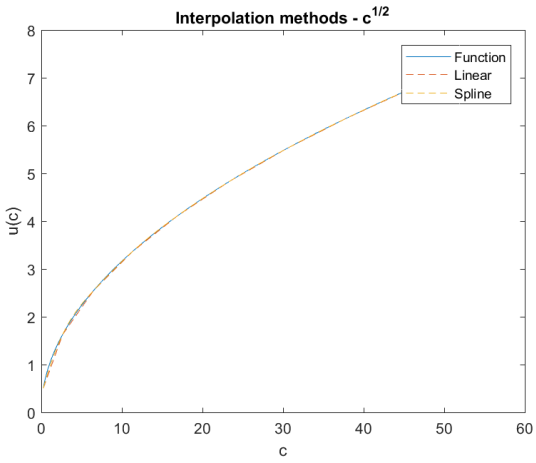


(d) $\theta = 4$

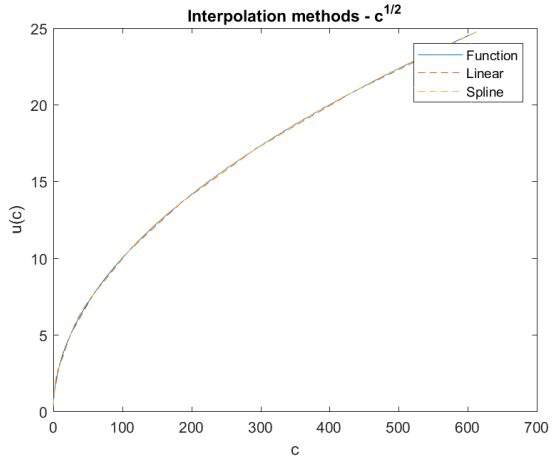
$u(c) = \sqrt{c}$



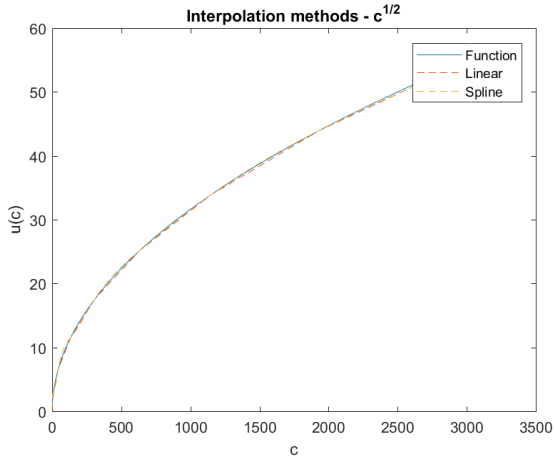
(e) $\theta = 1$



(f) $\theta = 1.5$

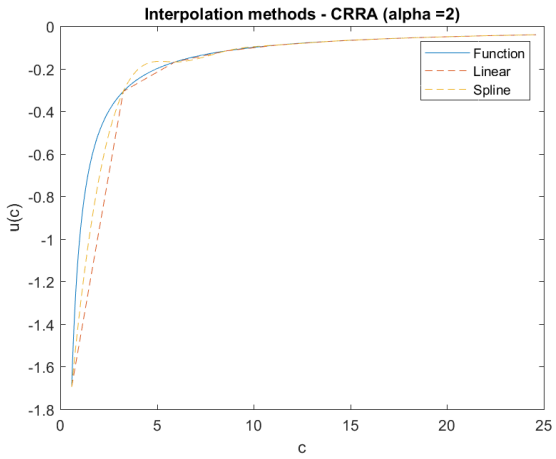


(g) $\theta = 3$

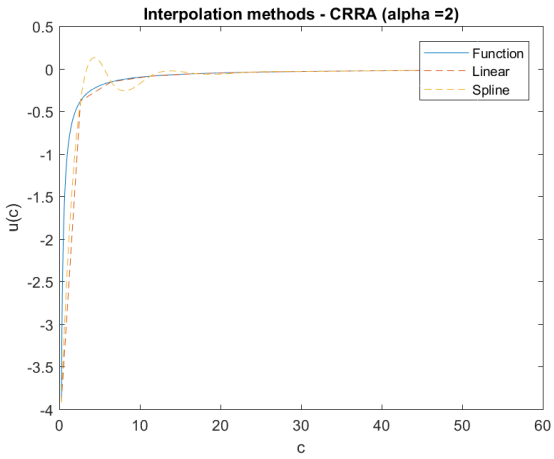


(h) $\theta = 4$

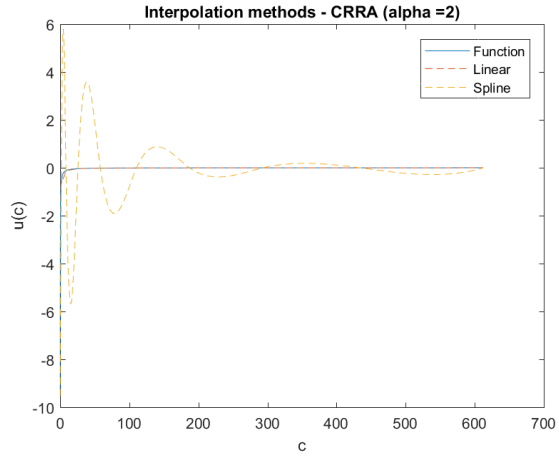
$$u(c) = \frac{c^{1-\alpha}}{1-\alpha}$$



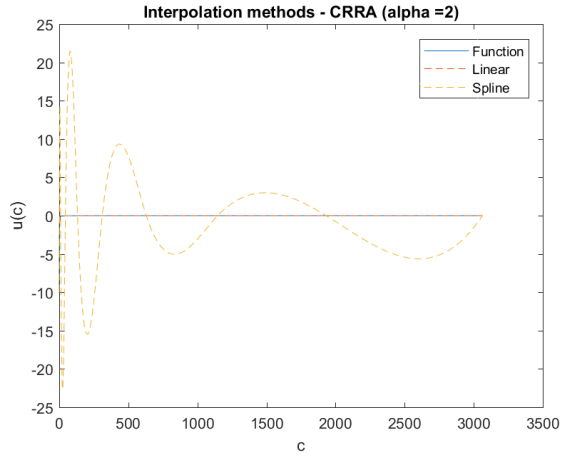
(i) $\theta = 1$



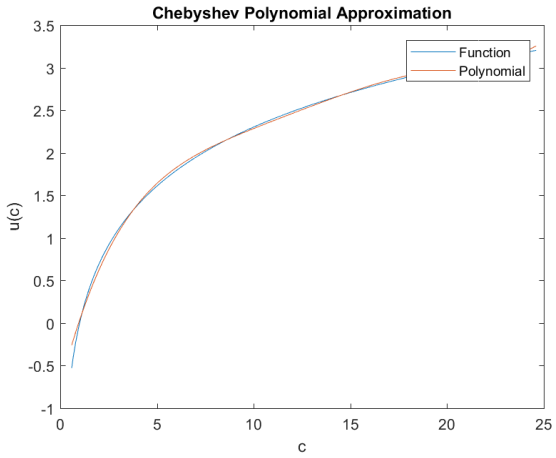
(j) $\theta = 1.5$



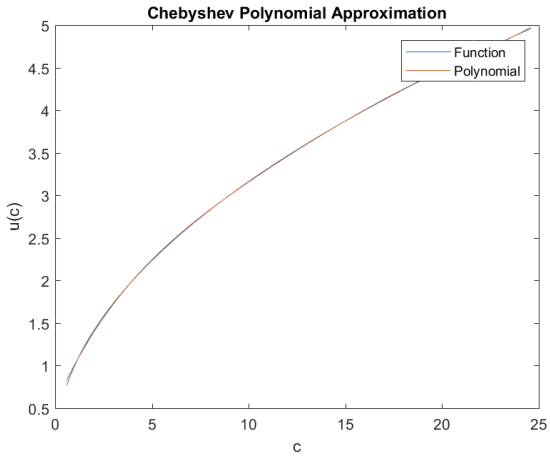
(k) $\theta = 3$



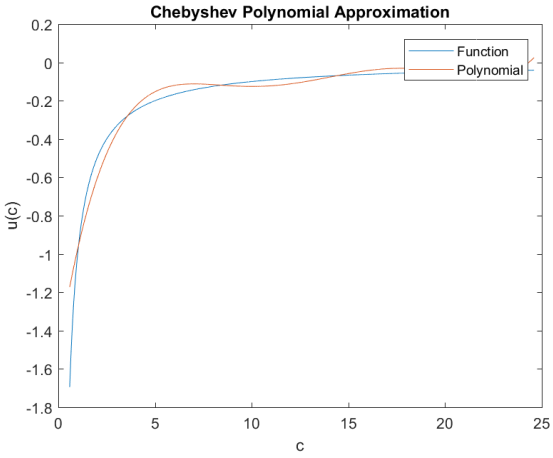
(l) $\theta = 4$



(m) $u(c) = \log(c)$



(n) $u(c) = \sqrt{c}$



(o) CRRA ($\alpha = 2$)