Homework 1

Alexander Sawyer

November 1, 2021

1 Problem 2

The computed integral and clock speed for each of the methods is below.

	Computed Integral	Clock speed (seconds)
Midpoint	-18.209525399958356	0.0229733
Trapazoid	-18.209525400030852	0.0222532
Simpson	-18.20997886913046	0.0224115
Monte Carlo	-18.19962351935149	0.0500674

Each of the quadrature methods use 1,000,000 points, and the Monte Carlo method uses 10 sampling strata and 100,000 points per stratum. The quadrature methods produce more similar integrals compared to the Monte Carlo method, with the midpoint and trapezoid methods computing the same integral up to 6 decimal points. The Monte Carlo method takes twice as long as the quadrature methods, all of which take a comparable amount of time.

2 Problem 3

The table below shows the computed solution to

$$\min_{(x,y)} 100(y-x^2)^2 + (1-x^2),$$

the L^2 distance from the actual solution $(1,1)$, and the clock speed for each method. The convergence criterion is that L^2 norm between the computed objective function in subsequent iterations is less than 10^{-8}.

There is a clear speed ranking with about an order of magnitude separation between each method and with Newton's method as the fastest and BFGS as the slowest. However, the conjugate descent is the most accurate method by far.

	Computed x^*	Computed y^*	L^2 Error	Clock Speed
Newton	0.9999476470720947	0.9998935139862254	0.00011865959796772592	$7.24e-5$
BFGS	0.9998511832682779	0.9996991191657029	0.0003356720067085991	1.0636255
Steepest Descent	0.9965968196734682	0.9931956113220144	0.007607978812798539	0.7594367
Conjugate Descent	1.0000000001987526	1.0000000003992382	$4.45975024919865e-10$	0.0194991

3 Problem 4

I implement the steepest descent algorithm to solve the planner's problem. For the $n = m = 3$ case, I first set the total endowment of each good to 1 and all parameters $\boldsymbol{\alpha}$, $\boldsymbol{\omega}$, and $\boldsymbol{\lambda}$ equal to test that the program delivers the result that each agent receives $\frac{1}{3}$ of each good; the program delivers this result with a clock speed of 1.2199 seconds. Next, I set $\boldsymbol{\alpha} = (4, 1, 1)$ and $\boldsymbol{\lambda} = (3, 1, 1)$, both separately and together. The resulting allocations, with each row pertaining to a household and each column pertaining to a good, are below. The initial guess in each case is an equal allocation.

1) $\alpha = (1, 1, 1)$; $\lambda = (3, 1, 1)$

$$\text{Allocation:} \quad \begin{bmatrix} 0.40630117126668186 & 0.40630117126668186 & 0.4063011712599973 \\ 0.2968511777821215 & 0.2968511777821215 & 0.2968511777859386 \\ 0.2968476509511966 & 0.2968476509511966 & 0.29684765095406407 \end{bmatrix}$$

Time: 0.8148

2) $\alpha = (4, 1, 1)$; $\lambda = (1, 1, 1)$

$$\text{Allocation:} \quad \begin{bmatrix} 0.33333313333333336 & 0.3333333333333333 & 0.3333333333333333 \\ 0.33333313333333336 & 0.3333333333333333 & 0.3333333333333333 \\ 0.33333373333333327 & 0.33333333333333337 & 0.33333333333333337 \end{bmatrix}$$

Time: 0.0770

3) $\alpha = (4, 1, 1)$; $\lambda = (3, 1, 1)$

$$\text{Allocation:} \quad \begin{bmatrix} 0.4063057236747447 & 0.40629755799893413 & 0.4062975580031113 \\ 0.2968468784098328 & 0.2968543373140233 & 0.29685433731124955 \\ 0.2968473979154225 & 0.2968481046870426 & 0.29684810468563916 \end{bmatrix}$$

Time: 1.0104

Predictably, increasing the Pareto weight to the first household increases its allocations relative to the other households. However, changing α, the multiplicative weights on the different goods in the utility function, does not change how the goods are allocated for either of the Pareto-weight settings.

Next, I set $\boldsymbol{\alpha}$ back to $(1, 1, 1)$ and $\boldsymbol{\omega}$ to

$$\omega_j^i = \begin{cases} -3 & i = j \\ -1.5 & \text{Else.} \end{cases}$$

The result is

$$\text{Allocation:} \quad \begin{bmatrix} 0.48313405604067117 & 0.3537323019690853 & 0.35372233233817246 \\ 0.2584328141694655 & 0.38783258138385895 & 0.2584422319998127 \\ 0.2584331297898633 & 0.25843511664705576 & 0.38783543566201484 \end{bmatrix}$$

Time: 1.4802

Thus, altering $\boldsymbol{\omega}$ introduces heterogeneity among household allocations across goods, with a higher value of ω_j^i shifting consumption to good j for household i.

Finally, as a check to see if the program can handle higher dimensions, $n = m = 10$, I compute $n = m = 10$ with the same parameter-setting pattern as above. I do not report the resulting allocation for brevity, but the program computes the solution in 22.0045 seconds.

4 Problem 5

I compute the market-clearing prices using two nested Newton-descent-direction backtracking algorithms (from section 9.7 in the nonlinear equation chapter posted on canvas). In particular, the solution algorithm nests a nonlinear solver for the household's first order conditions (given prices) inside of a nonlinear solver that computes the prices that clear all markets. I.e, the a solution algorithm to solve the system

$$\boldsymbol{F}_j(\boldsymbol{x}_j) = \left[x_j^1 - \left(\frac{\alpha_1}{\alpha_1} p_1 \left[e_j^1 + \sum_{l=2}^{m} p_l (e_j^l - x_j^l) \right]^{\omega_j^1} \right)^{\frac{1}{\omega_j^1}}, ..., x_j^n - \left(\frac{\alpha_1}{\alpha_n} p_n \left[e_j^1 + \sum_{l=2}^{m} p_l (e_j^l - x_j^l) \right]^{\omega_j^n} \right)^{\frac{1}{\omega_j^n}} \right] = \mathbf{0}$$

for $j = 1, ..., m$ and given p is nested within a nonlinear solver that solves

$$\boldsymbol{h}(\boldsymbol{p}) = \left[\sum_{j=1}^{n} e_j^1 - x_j^1(\boldsymbol{p}), ..., \sum_{j=1}^{n} e_j^m - x_j^m(\boldsymbol{p}) \right] = \mathbf{0}$$

I first set $n = m = 3$, and $e_j^i = 1$ for all i and j, and as a baseline test $\boldsymbol{\alpha} = (1, 1, 1)$ and $\omega_j^i = -1.5$ for all i, j. The program solves for $p = (1, 1, 1)$ in 0.1976 seconds. The table below reports the solution for other parameter settings:

	Price Vector	Clock Speed
$\alpha = (1, 2, 3)$	(0.9999916553877417 1.999993716066203; 2.999988615508454)	0.2472185
$\omega_i^i = -3 \ \forall i$	(1,1,1)	0.1994142
$e_3^i = 2 \ \forall i$	(0.9999981726993488 0.999997959512597 0.3535520094709583)	0.2569091
$\alpha = (1, 2, 3)$ and $e_3^i = 2$	(0.9999375684476575 1.9999038780157838 1.060604066610689)	0.2393908