

Legends	
Colors	Links
Added	(f)irst change
Changed	(n)ext change
Deleted	(t)op

american_option.jl

is unchanged - Python file exists

ar1_spec_rad.jl

is unchanged - Python file exists

bellman_envelope.jl

is unchanged - Python file bellman_envelope.py does not exist

binom_stoch_dom.jl

is unchanged - Python file binom_stoch_dom.py does not exist

compute_spec_rad.jl

is changed - Python file exists

old public code	new private (jstac) code
<pre> 1 using LinearAlgebra 2 f(A) = maximum(abs(λ) for λ in eigvals(A)) # Spectral radius 3 A = [0.4 0.1; # Test with arbitrary A 4 0.7 0.2] 5 Sprint(f(A)) </pre>	<pre> 1 using LinearAlgebra 2 f(A) = maximum(abs(λ) for λ in eigvals(A)) # Spectral radius 3 A = [0.4 0.1; # Test with arbitrary A 4 0.7 0.2] 5 Sprint(f(A)) </pre>

concave_map_fp.jl

is unchanged - Python file concave_map_fp.py does not exist

cont_time_js.jl

new - Python file cont_time_js.py does not exist

data

new - Python file data does not exist

expo_curve.py

new - Python file expo_curve.py does not exist

ez_dp_code.jl

new - Python file ez_dp_code.py does not exist

ez_f_shapes.jl

is unchanged - Python file ez_f_shapes.py does not exist

ez_model.jl

new - Python file ez_model.py does not exist

ez_noncontraction.jl

is changed - Python file ez_noncontraction.py does not exist

old public code	new private (jstac) code
<pre> 18 f(w) = F(w; θ=-10) 19 ax.plot(w_grid, w_grid, "k--", alpha=0.6, label=L"45") 20 ax.plot(w_grid, f.(w_grid), label=L"U = F") 21 ax.set_xticks((0, 1, 2)) 22 ax.set_yticks((0, 1, 2)) </pre>	<pre> 18 f(w) = F(w; θ=-10) 19 ax.plot(w_grid, w_grid, "k--", alpha=0.6, label=L"45") 20 ax.plot(w_grid, f.(w_grid), label=L"\hat K = F") 21 ax.set_xticks((0, 1, 2)) 22 ax.set_yticks((0, 1, 2)) </pre>

ez_plot_functions.jl

new - Python file ez_plot_functions.py does not exist

ez_policy_plot.jl

new - Python file ez_policy_plot.py does not exist

ez_sub_test.jl

new - Python file ez_sub_test.py does not exist

ez_timings.jl

new - Python file ez_timings.py does not exist

ez_utility.jl

is unchanged - Python file exists

finite_lq.jl

is unchanged - Python file exists

finite_opt_saving_0.jl

is changed - Python file exists

old public code	new private (jstac) code
<pre>2function create_savings_model(; R=1.01, β=0.98, γ=2.5, 3 w_min=0.01, w_max=5.0, w_size=200, 4 ρ=0.9, v=0.1, y_size=5) 5 w_grid = LinRange(w_min, w_max, w_size) 6end 10 11"Get the value v_σ of policy σ." 12function B(i, j, k, v, model) 13 (; β, R, γ, w_grid, y_grid, Q) = model 14end</pre>	<pre>2function create_savings_model(; R=1.01, β=0.98, γ=2.5, 3 w_min=0.01, w_max=20.0, w_size=200, 4 ρ=0.9, v=0.1, y_size=5) 5 w_grid = LinRange(w_min, w_max, w_size) 6end 10 11"Get the value v_σ of policy σ." 12function B(w, y, w', v) = u(R*w + y - w') + β Σ_y' v(w', y') Q(y, y'). 13function B(i, j, k, v, model) 14 (; β, R, γ, w_grid, y_grid, Q) = model 15end</pre>

finite_opt_saving_1.jl

is changed - Python file exists

old public code	new private (jstac) code
<pre>11end 12 13"Get the value v_σ of policy σ." 14function get_value(σ, model) 15 # Unpack and set up 16 (; β, R, γ, w_grid, y_grid, Q) = model 17 18 wn, yn = length(w_grid), length(y_grid) 19 n = wn * yn 20 u(c) = c^(1-γ) / (1-γ) 21 22 # Function to extract (i, j) from m = i + (j-1)*wn 23 single_to_multi(m) = (m-1)%wn + 1, div(m-1, wn) + 1 24 # Allocate and create single index versions of P_σ and r_σ 25 P_σ = zeros(n, n) 26 r_σ = zeros(n) 27 for m in 1:n 28 i, j = single_to_multi(m) 29 w, y, w' = w_grid[i], y_grid[j], w_grid[σ[i, j]] 30 r_σ[m] = u(w + y - w'/R) 31 for m' in 1:n 32 i', j' = single_to_multi(m') 33 if i' == σ[i, j] 34 P_σ[m, m'] = Q[j, j'] 35 end 36 end 37 end 38 39 # Solve for the value of σ 40 41 v_σ = (I - β * P_σ) \ r_σ 42 # Return as multi-index array</pre>	<pre>11end 12 13"Get the value v_σ of policy σ." 14function get_value(σ, model) 15 # Unpack and set up 16 (; β, R, γ, w_grid, y_grid, Q) = model 17 w_idx, y_idx = (eachindex(g) for g in (w_grid, y_grid)) 18 wn, yn = length(w_idx), length(y_idx) 19 n = wn * yn 20 u(c) = c^(1-γ) / (1-γ) 21 22 # Build P_σ and r_σ as multi-index arrays 23 P_σ = zeros(wn, yn, wn, yn) 24 25 r_σ = zeros(wn, yn) 26 for (i, j) in product(w_idx, y_idx) 27 28 w, y, w' = w_grid[i], y_grid[j], w_grid[σ[i, j]] 29 r_σ[i, j] = u(w + y - w'/R) 30 for (i', j') in product(w_idx, y_idx) 31 if i' == σ[i, j] 32 P_σ[i, j, i', j'] = Q[j, j'] 33 end 34 end 35 end 36 37 # Reshape for matrix algebra 38 P_σ = reshape(P_σ, n, n) 39 r_σ = reshape(r_σ, n) 40 # Apply matrix operations --- solve for the value of σ 41 v_σ = (I - β * P_σ) \ r_σ 42 # Return as multi-index array</pre>

finite_opt_saving_2.jl

is changed - Python file exists

old public code	new private (jstac) code
<pre>40end 41 42# Plots 43end</pre>	<pre>40end 41 42# == Simulations and inequality measures == # 43 44function simulate_wealth(m) 45 model = create_savings_model() 46 σ_star = optimistic_policy_iteration(model) 47 (; β, R, γ, w_grid, y_grid, Q) = model 48 49 # Simulate labor income (indices rather than grid values) 50 mc = MarkovChain(Q) 51 y_idx_series = simulate(mc, m) 52 53 # Compute corresponding wealth time series 54 w_idx_series = similar(y_idx_series) 55 w_idx_series[1] = 1 # initial condition 56 for t in 1:(m-1) 57 i, j = w_idx_series[t], y_idx_series[t] 58 w_idx_series[t+1] = σ_star[i, j] 59 end 60 w_series = w_grid[w_idx_series]</pre>

old public code		new private (jstac) code	
		62	
		63	return w_series
		64	end
		65	
		66	function lorenz(v) # assumed sorted vector
		67	S = cumsum(v) # cumulative sums: [v[1], v[1] + v[2], ...]
		68	F = (1:length(v)) / length(v)
		69	L = S ./ S[end]
		70	return (; F, L) # returns named tuple
		71	end
		72	
		73	gini(v) = (2 * sum(i * y for (i,y) in enumerate(v))/sum(v)
		74	- (length(v) + 1))/length(v)
		75	
		76	
		77	
		78	== Plots ==
		79	
44		80	using PyPlot
45	using PyPlot		
99	plt.show()	134	plt.show()
100	end	135	end
		136	
		137	
		138	function plot time series(; m=2 000,
		139	savefig=false,
		140	figname="./figures/finite_opt_saving_ts.pdf")
		141	
		142	w_series = simulate wealth(m)
		143	fig, ax = plt.subplots(figsize=(9, 5.2))
		144	ax.plot(w_series, label=L"w t")
		145	ax.set_xlabel("time", fontsize=fontsize)
		146	ax.legend(fontsize=fontsize)
		147	plt.show()
		148	if savefig
		149	fig.savefig(figname)
		150	end
		151	end
		152	
		153	function plot histogram(; m=1 000 000,
		154	savefig=false,
		155	figname="./figures/finite_opt_saving_hist.pdf")
		156	
		157	w_series = simulate wealth(m)
		158	g = round(gini(sort(w_series)), digits=2)
		159	fig, ax = plt.subplots(figsize=(9, 5.2))
		160	ax.hist(w_series, bins=40, density=true)
		161	ax.set_xlabel("wealth", fontsize=fontsize)
		162	ax.text(15, 0.4, "Gini = \$g", fontsize=fontsize)
		163	plt.show()
		164	
		165	if savefig
		166	fig.savefig(figname)
		167	end
		168	end
		169	
		170	function plot lorenz(; m=1 000 000,
		171	savefig=false,
		172	figname="./figures/finite_opt_saving_lorenz.pdf")
		173	
		174	w_series = simulate wealth(m)
		175	(; F, L) = lorenz(sort(w_series))
		176	
		177	fig, ax = plt.subplots(figsize=(9, 5.2))
		178	ax.plot(F, F, label="Lorenz curve, equality")
		179	ax.plot(F, L, label="Lorenz curve, wealth distribution")
		180	ax.legend()
		181	plt.show()
		182	
		183	if savefig
		184	fig.savefig(figname)
		185	end
		186	end

firm_exit.jl

is unchanged - Python file exists

firm_hiring.jl

is unchanged - Python file exists

fosd_tauchen.jl

is unchanged - Python file fosd_tauchen.py does not exist

howard_newton.jl

is changed - Python file howard_newton.py does not exist

old public code		new private (jstac) code	
24	ax.plot(xgrid, Tsp.(xgrid), lw=2, alpha=0.6, label=lb_Tsp)	24	ax.plot(xgrid, Tsp.(xgrid), lw=2, alpha=0.6, label=lb_Tsp)
25		25	
26	ax.plot(xgrid, xgrid, "k--", lw=1, alpha=0.7, label=L"45")	26	ax.plot(xgrid, xgrid, "k--", lw=1, alpha=0.7, label=L"45 \circ ")
27		27	
28	fpl = (v1,)	28	fpl = (v1,)

iid_job_search.jl

is unchanged - Python file exists

iid_job_search_cv.jl

is unchanged - Python file iid_job_search_cv.py does not exist

inventory_cont_time.jl

new - Python file inventory_cont_time.py does not exist

inventory_dp.jl

is changed - Python file exists

old public code	new private (jstac) code
<pre>1include("s_approx.jl") 2using Distributions, OffsetArrays 3m(x) = max(x, 0) # Convenience function 4 5function create_inventory_model(; β=0.98, # discount factor 6 K=40, # maximum inventory 7 c=0.2, κ=2, # cost parameters 8 p=0.6) # demand parameter 9 φ(d) = (1 - p)^d * p # demand pdf 10 return (; β, K, c, κ, p, φ) 11end 12 13"The function B(x, a, v) = r(x, a) + β ∑_x' v(x') P(x, a, x')." 14function B(x, a, v, model; d_max=100) 15 (; β, K, c, κ, p, φ) = model 16 reward = sum(min(x, d)*φ(d) for d in 0:d_max) - c * a - κ * (a > 0) 17 continuation_value = β * sum(v[m(x - d) + a] * φ(d) for d in 0:d_max) 18 return reward + continuation_value 19end 20 21"The Bellman operator." 22function T(v, model) 23 (; β, K, c, κ, p, φ) = model 24 new_v = similar(v) 25 for x in 0:K 26 Ix = 0:(K - x) 27 new_v[x], _ = findmax(B(x, a, v, model) for a in Ix) 28 end 29 return new_v 30end 31 32"Get a v-greedy policy. Returns a zero-based array." 33function get_greedy(v, model) 34 (; β, K, c, κ, p, φ) = model 35 σ_star = OffsetArray(zeros{Int32, K+1}, 0:K) 36 for x in 0:K 37 Ix = 0:(K - x) 38 _, a_idx = findmax(B(x, a, v, model) for a in Ix) 39 σ_star[x] = Ix[a_idx] 40 end 41 return σ_star 42end 43 44"Use successive approx to get v_star and then compute greedy." 45function solve_inventory_model(v_init, model) 46 (; β, K, c, κ, p, φ) = model 47 v_star = successive_approx(v -> T(v, model), v_init) 48 σ_star = get_greedy(v_star, model) 49end 50 51# Create an instance of the model and solve it 52model = create_inventory_model() 53(; β, K, c, κ, p, φ) = model 54v_init = OffsetArray(zeros{K+1}, 0:K) 55v_star, σ_star = solve_inventory_model(v_init, model) 56 57for t in 1:(ts_length-1) 58 D = rand(G) 59 X[t+1] = m(X[t] - D) + σ_star[X[t]] 60end 61return X</pre>	<pre>1include("s_approx.jl") 2using Distributions 3m(x) = max(x, 0) # Convenience function 4 5function create_inventory_model(; β=0.98, # discount factor 6 K=40, # maximum inventory 7 c=0.2, κ=2, # cost parameters 8 p=0.6) # demand parameter 9 φ(d) = (1 - p)^d * p # demand pdf 10 x_vals = collect(0:K) # set of inventory levels 11 return (; β, K, c, κ, p, φ, x_vals) 12end 13 14"The function B(x, a, v) = r(x, a) + β ∑_x' v(x') P(x, a, x')." 15function B(x, a, v, model; d_max=100) 16 (; β, K, c, κ, p, φ, x_vals) = model 17 revenue = sum(min(x, d) * φ(d) for d in 0:d_max) 18 current_profit = revenue - c * a - κ * (a > 0) 19 next_value = sum(v[m(x - d) + a + 1] * φ(d) for d in 0:d_max) 20 return current_profit + β * next_value 21end 22 23"The Bellman operator." 24function T(v, model) 25 (; β, K, c, κ, p, φ, x_vals) = model 26 new_v = similar(v) 27 for (x_idx, x) in enumerate(x_vals) 28 Ix = 0:(K - x) 29 new_v[x_idx], _ = findmax(B(x, a, v, model) for a in Ix) 30 end 31 return new_v 32end 33 34"Get a v-greedy policy. Returns a zero-based array." 35function get_greedy(v, model) 36 (; β, K, c, κ, p, φ, x_vals) = model 37 σ_star = zeros{Int32, length(x_vals)} 38 for (x_idx, x) in enumerate(x_vals) 39 Ix = 0:(K - x) 40 _, a_idx = findmax(B(x, a, v, model) for a in Ix) 41 σ_star[x_idx] = Ix[a_idx] 42 end 43 return σ_star 44end 45 46"Use successive approx to get v_star and then compute greedy." 47function solve_inventory_model(v_init, model) 48 (; β, K, c, κ, p, φ, x_vals) = model 49 v_star = successive_approx(v -> T(v, model), v_init) 50 σ_star = get_greedy(v_star, model) 51end 52 53# Create an instance of the model and solve it 54model = create_inventory_model() 55(; β, K, c, κ, p, φ, x_vals) = model 56v_init = zeros{length(x_vals)} 57v_star, σ_star = solve_inventory_model(v_init, model) 58 59for t in 1:(ts_length-1) 60 D = rand(G) 61 X[t+1] = m(X[t] - D) + σ_star[X[t] + 1] 62end 63return X</pre>

inventory_sdd.jl

is changed - Python file exists

old public code	new private (jstac) code
<pre>1""" 2 3Inventory management model with state-dependent discounting. The discount 4factor takes the form β_t = Z_t, where (Z_t) is a discretization of a 5Gaussian AR(1) process 6 7 X_t = ρ X_{t-1} + b + v W_t. 8 9 10include("s_approx.jl") 11using LinearAlgebra, Distributions, OffsetArrays, QuantEcon 12 13function create_sdd_inventory_model(; 14 ρ=0.98, v=0.002, n_z=20, b=0.97, # Z state parameters 15 K=40, c=0.2, κ=0.8, p=0.6) # firm and demand parameters 16 φ(d) = (1 - p)^d * p # demand pdf 17 mc = tauchen(n_z, ρ, v) 18 z_vals, Q = mc.state_values .+ b, mc.p 19 rL = maximum(abs.(eigvals(z_vals .* Q))) 20 @assert rL < 1 "Error: r(L) ≥ 1." # check r(L) < 1 21 return (; K, c, κ, p, φ, z_vals, Q) 22end 23 24m(x) = max(x, 0) # Convenience function 25 26"The function B(x, z, a, v) = r(x, a) + β(z) ∑_x' v(x') P(x, a, x')." 27function B(x, i_z, a, v, model; d_max=100) 28 (; K, c, κ, p, φ, z_vals, Q) = model</pre>	<pre>1""" 2 3Inventory management model with state-dependent discounting. 4The discount factor takes the form β_t = Z_t, where (Z_t) is 5a discretization of the Gaussian AR(1) process 6 7 X_t = ρ X_{t-1} + b + v W_t. 8 9 10include("s_approx.jl") 11using LinearAlgebra, Distributions, QuantEcon 12 13function create_sdd_inventory_model(; 14 ρ=0.98, v=0.002, n_z=20, b=0.97, # Z state parameters 15 K=40, c=0.2, κ=0.8, p=0.6) # firm and demand parameters 16 φ(d) = (1 - p)^d * p # demand pdf 17 y_vals = collect(0:K) # inventory levels 18 mc = tauchen(n_z, ρ, v) 19 z_vals, Q = mc.state_values .+ b, mc.p 20 pL = maximum(abs.(eigvals(z_vals .* Q))) 21 @assert pL < 1 "Error: p(L) ≥ 1." # check p(L) < 1 22 return (; K, c, κ, p, φ, y_vals, z_vals, Q) 23end 24 25m(y) = max(y, 0) # Convenience function 26 27"The function B(x, a, v) = r(x, a) + β(x) ∑_x' v(x') P(x, a, x')." 28function B(x, i_z, a, v, model; d_max=100) 29 (; K, c, κ, p, φ, y_vals, z_vals, Q) = model</pre>

old public code	new private (jstac) code
<pre> 30 z = z_vals[i z] 31 reward = sum(min(x, d)*φ(d) for d in 0:d_max) - c * a - κ * (a > 0) 32 cv = 0.0 33 for (i_z', z') in enumerate(z_vals) 34 cv += sum(v[m(x - d) + a, i_z'] * φ(d) for d in 0:d_max) * Q[i_z, i_z'] 35 end 36 return reward + z * cv 37 end 38 39 "The Bellman operator." 40 function T(v, model) 41 (; K, c, κ, p, φ, z_vals, Q) = model 42 new_v = similar(v) 43 for (i z, z) in enumerate(z_vals) 44 for x in 0:K 45 Ix = 0:(K - x) 46 new_v[x, i_z], _ = findmax(B(x, i_z, a, v, model) for a in Ix) 47 end 48 end </pre>	<pre> 30 z = z_vals[i z] 31 revenue = sum(min(x, d)*φ(d) for d in 0:d_max) 32 current_profit = revenue - c * a - κ * (a > 0) 33 cv = 0.0 34 for i z' in eachindex(z_vals) 35 for d in 0:d_max 36 cv += v[m(x - d) + a + 1, i_z'] * φ(d) * Q[i_z, i_z'] 37 end 38 end 39 return current_profit + z * cv 40 end 41 42 "The Bellman operator." 43 function T(v, model) 44 (; K, c, κ, p, φ, y_vals, z_vals, Q) = model 45 new_v = similar(v) 46 for (i z, z) in enumerate(z_vals) 47 for (i y, y) in enumerate(y_vals) 48 Iy = 0:(K - y) 49 new_v[i y, i_z], _ = findmax(B(y, i_z, a, v, model) for a in Iy) 50 end 51 end </pre>
<pre> 52 "Get a v-greedy policy. Returns a zero-based array." 53 function get_greedy(v, model) 54 (; K, c, κ, p, φ, z_vals, Q) = model 55 n z = length(z_vals) 56 σ_star = OffsetArray(zeros{Int32, K+1, n z}, 0:K, 1:n z) 57 for (i z, z) in enumerate(z_vals) 58 for x in 0:K 59 Ix = 0:(K - x) 60 a_idx = findmax(B(x, i z, a, v, model) for a in Ix) 61 σ_star[x, i_z] = Ix[a_idx] 62 end 63 end </pre>	<pre> 53 "Get a v-greedy policy. Returns a zero-based array." 54 function get_greedy(v, model) 55 (; K, c, κ, p, φ, y_vals, z_vals, Q) = model 56 n z = length(z_vals) 57 σ_star = zeros{Int32, K+1, n z} 58 for (i z, z) in enumerate(z_vals) 59 for (i y, y) in enumerate(y_vals) 60 Iy = 0:(K - y) 61 i_a = findmax(B(y, i z, a, v, model) for a in Iy) 62 σ_star[i y, i_z] = Iy[i_a] 63 end 64 end </pre>
<pre> 68 "Use successive_approx to get v_star and then compute greedy." 69 function solve_inventory_model(v_init, model) 70 (; K, c, κ, p, φ, z_vals, Q) = model 71 v_star = successive_approx(v -> T(v, model), v_init) 72 σ_star = get_greedy(v_star, model) </pre>	<pre> 71 "Use successive_approx to get v_star and then compute greedy." 72 function solve_inventory_model(v_init, model) 73 (; K, c, κ, p, φ, y_vals, z_vals, Q) = model 74 v_star = successive_approx(v -> T(v, model), v_init) 75 σ_star = get_greedy(v_star, model) </pre>
<pre> 84 # Create an instance of the model and solve it 85 model = create_sdd_inventory_model() 86 (; K, c, κ, p, φ, z_vals, Q) = model 87 n z = length(z_vals) 88 v_init = OffsetArray(zeros{Float64, K+1, n z}, 0:K, 1:n z) 89 println("Solving model.") 90 v_star, σ_star = solve_inventory_model(v_init, model) </pre>	<pre> 87 # Create an instance of the model and solve it 88 model = create_sdd_inventory_model() 89 (; K, c, κ, p, φ, y_vals, z_vals, Q) = model 90 n z = length(z_vals) 91 v_init = zeros{Float64, K+1, n z} 92 println("Solving model.") 93 v_star, σ_star = solve_inventory_model(v_init, model) </pre>
<pre> 99 for t in 1:(ts_length-1) 100 D' = rand(G) 101 X[t+1] = m(X[t] - D') + σ_star[X[t], i_z[t]] 102 end 103 return X, z_vals[i z] </pre>	<pre> 102 for t in 1:(ts_length-1) 103 D' = rand(G) 104 X[t+1] = m(X[t] - D') + σ_star[X[t] + 1, i_z[t]] 105 end 106 return X, z_vals[i z] </pre>
<pre> 112 ax = axes[1] 113 ax.plot(X, label=L"X t", alpha=0.7) 114 ax.set_xlabel(L"t", fontsize=fontsize) 115 ax.set_ylabel("inventory", fontsize=fontsize) 116 ax.legend(fontsize=fontsize, frameon=false) 117 ax.set_ylim(0, maximum(X)+3) </pre>	<pre> 115 ax = axes[1] 116 ax.plot(X, label="inventory", alpha=0.7) 117 ax.set_xlabel(L"t", fontsize=fontsize) 118 ax.legend(fontsize=fontsize, frameon=false) 119 ax.set_ylim(0, maximum(X)+3) </pre>
<pre> 124 ax.plot(r, label=L"r t", alpha=0.7) 125 ax.set_xlabel(L"t", fontsize=fontsize) 126 ax.set_ylabel("interest rate", fontsize=fontsize) 127 ax.legend(fontsize=fontsize, frameon=false) 128 #ax.set_ylim(0, maximum(X)+8) </pre>	<pre> 126 ax.plot(r, label=L"r t", alpha=0.7) 127 ax.set_xlabel(L"t", fontsize=fontsize) 128 ax.legend(fontsize=fontsize, frameon=false) 129 #ax.set_ylim(0, maximum(X)+8) </pre>

inventory_sim.jl

is unchanged - Python file exists

is_irreducible.jl

is unchanged - Python file exists

js_with_sep_sim.jl

is unchanged - Python file js_with_sep_sim.py does not exist

laborer_sim.jl

is unchanged - Python file exists

lake.jl

is unchanged - Python file lake.py does not exist

linear_iter.jl

is unchanged - Python file exists

linear_iter_fig.jl

is unchanged - Python file exists

lqcontrol.py

is unchanged - Python file lqcontrol.py does not exist

markov_js.jl

is unchanged - Python file exists

markov_js_with_sep.jl

is unchanged - Python file exists

modified_opt_savings.jl

is changed - Python file exists

old public code	new private (jstac) code
<pre>1 using QuantEcon, LinearAlgebra, IterTools 2 3 function create_savings_model(; R=1.01, β=0.98, γ=2.5, 4 w_min=0.01, w_max=10.0, w_size=100, 5 c=0.9, v=0.1, z_size=20, 6 ε_min=-0.25, ε_max=0.25, ε_size=30) 7 ε_grid = LinRange(ε_min, ε_max, ε_size) 8 φ = ones(ε_size) * (1 / ε_size) # Uniform distribution 9 w_grid = LinRange(w_min, w_max, w_size) 10 mc = tauchen(z_size, ρ, v) 11 z_grid, Q = exp.(mc.state_values), mc.p 12 return (; β, R, γ, ε_grid, φ, w_grid, z_grid, Q) 13 end 14 15</pre>	<pre>1 using QuantEcon, LinearAlgebra, IterTools 2 3 function create_savings_model(; β=0.98, γ=2.5, 4 w_min=0.01, w_max=20.0, w_size=100, 5 c=0.9, v=0.1, y_size=20, 6 η_min=0.75, η_max=1.25, η_size=2) 7 η_grid = LinRange(η_min, η_max, η_size) 8 φ = ones(η_size) * (1 / η_size) # Uniform distribution 9 w_grid = LinRange(w_min, w_max, w_size) 10 mc = tauchen(y_size, ρ, v) 11 y_grid, Q = exp.(mc.state_values), mc.p 12 return (; β, γ, η_grid, φ, w_grid, y_grid, Q) 13 end 14 15 16</pre>
<pre>17 18 """ 19 The function 20 21 B(w, z, ε, w') = 22 u(w + z + ε - w'/R) + β Σ v(w', z', ε') Q(z, z') φ(ε') 23 24 """ 25 function B(i, j, k, l, v, model) 26 (; β, R, γ, ε_grid, φ, w_grid, z_grid, Q) = model 27 w, z, ε, w' = w_grid[i], z_grid[j], ε_grid[k], w_grid[l] 28 c = w + z + ε - (w' / R) 29 exp_value = 0.0 30 for m in eachindex(z_grid) 31 for n in eachindex(ε_grid) 32 exp_value += v[l, m, n] * Q[j, m] * φ[n] 33 end 34 end 35 return c > 0 ? c^(1-γ) / (1-γ) + β * exp_value : -Inf 36 end 37 38 "The policy operator." 39 function T_σ(v, σ, model) 40 (; β, R, γ, ε_grid, φ, w_grid, z_grid, Q) = model 41 w_idx, z_idx, ε_idx = (eachindex(g) for g in (w_grid, z_grid, ε_grid)) 42 v_new = similar(v) 43 for (i, j, k) in product(w_idx, z_idx, ε_idx) 44 v_new[i, j, k] = B(i, j, k, σ[i, j, k], v, model) 45 end 46 end 47 48 "Compute a v-greedy policy." 49 function get_greedy(v, model) 50 (; β, R, γ, ε_grid, φ, w_grid, z_grid, Q) = model 51 w_idx, z_idx, ε_idx = (eachindex(g) for g in (w_grid, z_grid, ε_grid)) 52 σ = Array{Int32}(undef, length(w_idx), length(z_idx), length(ε_idx)) 53 for (i, j, k) in product(w_idx, z_idx, ε_idx) 54 σ[i, j, k] = findmax(B(i, j, k, l, v, model) for l in w_idx) 55 end 56 end 57 58 "Optimistic policy iteration routine." 59 function optimistic_policy_iteration(model; tolerance=1e-5, m=100) 60 (; β, R, γ, ε_grid, φ, w_grid, z_grid, Q) = model 61 v = zeros(length(w_grid), length(z_grid), length(ε_grid)) 62 error = tolerance + 1 63 while error > tolerance 64 65</pre>	<pre>18 19 """ 20 B(w, y, η, w') = u(w + y - w'/η) + β Σ v(w', y', η') Q(y, y') φ(η') 21 22 """ 23 function B(i, j, k, l, v, model) 24 (; β, γ, η_grid, φ, w_grid, y_grid, Q) = model 25 w, y, η, w' = w_grid[i], y_grid[j], η_grid[k], w_grid[l] 26 u(c) = c^(1-γ) / (1-γ) 27 c = w + y - (w' / η) 28 exp_value = 0.0 29 for m in eachindex(y_grid) 30 for n in eachindex(η_grid) 31 exp_value += v[l, m, n] * Q[j, m] * φ[n] 32 end 33 end 34 return c > 0 ? u(c) + β * exp_value : -Inf 35 end 36 37 "The policy operator." 38 function T_σ(v, σ, model) 39 (; β, γ, η_grid, φ, w_grid, y_grid, Q) = model 40 grids = w_grid, y_grid, η_grid 41 w_idx, y_idx, η_idx = (eachindex(g) for g in grids) 42 v_new = similar(v) 43 for (i, j, k) in product(w_idx, y_idx, η_idx) 44 v_new[i, j, k] = B(i, j, k, σ[i, j, k], v, model) 45 end 46 end 47 48 "Compute a v-greedy policy." 49 function get_greedy(v, model) 50 (; β, γ, η_grid, φ, w_grid, y_grid, Q) = model 51 w_idx, y_idx, η_idx = (eachindex(g) for g in (w_grid, y_grid, η_grid)) 52 σ = Array{Int32}(undef, length(w_idx), length(y_idx), length(η_idx)) 53 for (i, j, k) in product(w_idx, y_idx, η_idx) 54 σ[i, j, k] = findmax(B(i, j, k, l, v, model) for l in w_idx) 55 end 56 end 57 58 "Optimistic policy iteration routine." 59 function optimistic_policy_iteration(model; tolerance=1e-5, m=100) 60 (; β, γ, η_grid, φ, w_grid, y_grid, Q) = model 61 v = zeros(length(w_grid), length(y_grid), length(η_grid)) 62 error = tolerance + 1 63 while error > tolerance 64 65</pre>
<pre>77 78 79 80 81 ## == Functions for modified OPI == ## 82 83 "D(w, z, ε, w', g) = u(w + z + ε - w'/R) + β g(z, w')." 84 @inline function D(i, j, k, l, g, model) 85 (; β, R, γ, ε_grid, φ, w_grid, z_grid, Q) = model 86 w, z, ε, w' = w_grid[i], z_grid[j], ε_grid[k], w_grid[l] 87 c = w + z + ε - (w' / R) 88 return c > 0 ? c^(1-γ) / (1-γ) + β * g[j, l] : -Inf 89 end 90 91 "Compute a g-greedy policy." 92 function get_g_greedy(g, model) 93 (; β, R, γ, ε_grid, φ, w_grid, z_grid, Q) = model 94 w_idx, z_idx, ε_idx = (eachindex(g) for g in (w_grid, z_grid, ε_grid)) 95 σ = Array{Int32}(undef, length(w_idx), length(z_idx), length(ε_idx)) 96 for (i, j, k) in product(w_idx, z_idx, ε_idx) 97 σ[i, j, k] = findmax(D(i, j, k, l, g, model) for l in w_idx) 98 end 99 end 100 101 "The modified policy operator." 102 function R_σ(g, σ, model) 103 (; β, R, γ, ε_grid, φ, w_grid, z_grid, Q) = model 104 w_idx, z_idx, ε_idx = (eachindex(g) for g in (w_grid, z_grid, ε_grid)) 105 g_new = similar(g) 106</pre>	<pre>76 77 78 79 80 ## == Functions for modified OPI == ## 81 82 "D(w, y, η, w', g) = u(w + y - w'/η) + β g(y, w')." 83 @inline function D(i, j, k, l, g, model) 84 (; β, γ, η_grid, φ, w_grid, y_grid, Q) = model 85 w, y, η, w' = w_grid[i], y_grid[j], η_grid[k], w_grid[l] 86 u(c) = c^(1-γ) / (1-γ) 87 c = w + y - (w' / η) 88 return c > 0 ? u(c) + β * g[j, l] : -Inf 89 end 90 91 "Compute a g-greedy policy." 92 function get_g_greedy(g, model) 93 (; β, γ, η_grid, φ, w_grid, y_grid, Q) = model 94 w_idx, y_idx, η_idx = (eachindex(g) for g in (w_grid, y_grid, η_grid)) 95 σ = Array{Int32}(undef, length(w_idx), length(y_idx), length(η_idx)) 96 for (i, j, k) in product(w_idx, y_idx, η_idx) 97 σ[i, j, k] = findmax(D(i, j, k, l, g, model) for l in w_idx) 98 end 99 end 100 101 "The modified policy operator." 102 function R_σ(g, σ, model) 103 (; β, γ, η_grid, φ, w_grid, y_grid, Q) = model 104 w_idx, y_idx, η_idx = (eachindex(g) for g in (w_grid, y_grid, η_grid)) 105 g_new = similar(g) 106</pre>

old public code	new private (jstac) code
<pre> 109 for (j, i') in product(z_idx, w_idx) # j -> z, i' -> w' 110 out = 0.0 111 for j' in z_idx # j' -> z' 112 for k' in e_idx # k' -> e' 113 # D(w', z', e', o(w', z', e'), g) 114 out += D(i', j', k', o[i', j', k'], g, model) * 115 Q[j, j'] * phi[k'] </pre>	<pre> 107 for (j, i') in product(y_idx, w_idx) # j indexes y, i' indexes w' 108 out = 0.0 109 for j' in y_idx # j' indexes y' 110 for k' in eta_idx # k' indexes eta' 111 out += D(i', j', k', o[i', j', k'], g, model) * 112 Q[j, j'] * phi[k'] </pre>
<pre> 124 "Modified optimistic policy iteration routine." 125 function mod_opi(model; tolerance=1e-5, m=100) 126 (; beta, R, gamma, e_grid, phi, w_grid, z_grid, Q) = model 127 g = zeros(length(z_grid), length(w_grid)) 128 error = tolerance + 1 129 while error > tolerance </pre>	<pre> 121 "Modified optimistic policy iteration routine." 122 function mod_opi(model; tolerance=1e-5, m=100) 123 (; beta, gamma, eta_grid, phi, w_grid, y_grid, Q) = model 124 g = zeros(length(y_grid), length(w_grid)) 125 error = tolerance + 1 126 while error > tolerance </pre>
<pre> 140 141 142 # Plots </pre>	<pre> 137 138 139 == Simulations and inequality measures == 140 141 function simulate_wealth(m) 142 143 model = create_savings_model() 144 (; beta, gamma, eta_grid, phi, w_grid, y_grid, Q) = model 145 sigma_star = mod_opi(model) 146 147 # Simulate labor income 148 mc = MarkovChain(Q) 149 y_idx_series = simulate(mc, m) 150 151 # IID Markov chain with uniform draws 152 l = length(eta_grid) 153 mc = MarkovChain(ones(l, l) * (1/l)) 154 eta_idx_series = simulate(mc, m) 155 156 w_idx_series = similar(y_idx_series) 157 w_idx_series[1] = 1 158 for t in 1:(m-1) 159 i, j, k = w_idx_series[t], y_idx_series[t], eta_idx_series[t] 160 w_idx_series[t+1] = sigma_star[i, j, k] 161 end 162 163 w_series = w_grid[w_idx_series] 164 return w_series 165 end 166 167 168 function lorenz(v) # assumed sorted vector 169 S = cumsum(v) # cumulative sums: [v[1], v[1] + v[2], ...] 170 F = (1:length(v)) / length(v) 171 L = S ./ S[end] 172 return (; F, L) # returns named tuple 173 end 174 175 176 gini(v) = (2 * sum(i * y for (i,y) in enumerate(v)) / sum(v) 177 - (length(v) + 1)) / length(v) 178 179 180 == Plots == 181 </pre>
<pre> 143 144 using PyPlot </pre>	<pre> 182 using PyPlot </pre>
<pre> 148 149 150 function plot_contours(; savefig=false, 151 filename="./figures/modified_opt_savings_1.pdf") 152 153 model = create_savings_model() 154 (; beta, R, gamma, e_grid, phi, w_grid, z_grid, Q) = model 155 sigma_star = mod_opi(model) 156 157 fig, axes = plt.subplots(2, 1, figsize=(10, 8)) 158 z_idx, e_idx = eachindex(z_grid), eachindex(e_grid) 159 H = zeros(length(z_grid), length(e_grid)) 160 161 w_indices = (1, length(w_grid)) </pre>	<pre> 186 187 188 189 function plot_contours(; savefig=false, 190 filename="./figures/modified_opt_savings_1.pdf") 191 192 model = create_savings_model() 193 (; beta, gamma, eta_grid, phi, w_grid, y_grid, Q) = model 194 sigma_star = optimistic_policy_iteration(model) 195 196 fig, axes = plt.subplots(2, 1, figsize=(10, 8)) 197 y_idx, eta_idx = eachindex(y_grid), eachindex(eta_grid) 198 H = zeros(length(y_grid), length(eta_grid)) 199 200 w_indices = (1, length(w_grid)) </pre>
<pre> 163 for (ax, w_idx, title) in zip(axes, w_indices, titles) 164 165 for (i_z, i_e) in product(z_idx, e_idx) 166 w, z, e = w_grid[w_idx], z_grid[i_z], e_grid[i_e] 167 H[i_z, i_e] = w_grid[sigma_star[w_idx, i_z, i_e]] 168 end 169 170 cs1 = ax.contourf(z_grid, e_grid, transpose(H), alpha=0.5) 171 #ctrl = ax.contour(w_vals, z_vals, transpose(H), levels=[0.0]) 172 #plt.clabel(ctrl, inline=1, fontsize=13) 173 plt.colorbar(cs1, ax=ax) #, format="%0.6f") 174 175 ax.set_title(title, fontsize=fontsize) 176 ax.set_xlabel(L"z", fontsize=fontsize) 177 ax.set_ylabel(L"\varepsilon", fontsize=fontsize) 178 end </pre>	<pre> 202 for (ax, w_idx, title) in zip(axes, w_indices, titles) 203 204 for (i_y, i_eta) in product(y_idx, eta_idx) 205 w, y, eta = w_grid[w_idx], y_grid[i_y], eta_grid[i_eta] 206 H[i_y, i_eta] = w_grid[sigma_star[w_idx, i_y, i_eta]] / (w+y) 207 end 208 209 cs1 = ax.contourf(y_grid, eta_grid, transpose(H), alpha=0.5) 210 211 plt.colorbar(cs1, ax=ax) #, format="%0.6f") 212 213 ax.set_title(title, fontsize=fontsize) 214 ax.set_xlabel(L"y", fontsize=fontsize) 215 ax.set_ylabel(L"\eta", fontsize=fontsize) 216 end </pre>
<pre> 186 187 </pre>	<pre> 223 224 225 function plot_policies(; savefig=false, 226 filename="./figures/modified_opt_savings_2.pdf") 227 228 model = create_savings_model() 229 (; beta, gamma, eta_grid, phi, w_grid, y_grid, Q) = model 230 sigma_star = mod_opi(model) 231 y_bar = floor(Int, length(y_grid) / 2) # Index of mid-point of y_grid 232 233 fig, ax = plt.subplots(figsize=(9, 5.2)) 234 ax.plot(w_grid, w_grid, "k--", label=L"45") 235 236 for (i, eta) in enumerate(eta_grid) 237 label = L"\sigma^*" * " " at " * L"\eta = " * "\\$eta" 238 ax.plot(w_grid, w_grid[sigma_star[:, y_bar, i]], label=label) 239 end 240 ax.legend(fontsize=fontsize) 241 plt.show() 242 </pre>

old public code	new private (jstac) code
	<pre>243 plt.tight_layout() 244 if savefig 245 fig.savefig(filename) 246 end 247 plt.show() 248 end 249 250 251 function plot time series(; m=2 000, 252 savefig=false, 253 filename="../figures/modified_opt_savings_ts.pdf") 254 255 w series = simulate wealth(m) 256 fig, ax = plt.subplots(figsize=(9, 5.2)) 257 ax.plot(w series, label=L"w t") 258 ax.legend(fontsize=fontsize) 259 ax.set xlabel("time", fontsize=fontsize) 260 plt.show() 261 if savefig 262 fig.savefig(filename) 263 end 264 265 end 266 267 function plot histogram(; m=1 000 000, 268 savefig=false, 269 filename="../figures/modified_opt_savings_hist.pdf") 270 271 w series = simulate wealth(m) 272 g = round(gini(sort(w series)), digits=2) 273 fig, ax = plt.subplots(figsize=(9, 5.2)) 274 ax.hist(w series, bins=40, density=true) 275 ax.set xlabel("wealth", fontsize=fontsize) 276 ax.text(15, 0.7, "Gini = \$g", fontsize=fontsize) 277 plt.show() 278 if savefig 279 fig.savefig(filename) 280 end 281 282 end 283 284 285 function plot lorenz(; m=1 000 000, 286 savefig=false, 287 filename="../figures/modified_opt_savings_lorenz.pdf") 288 289 w series = simulate wealth(m) 290 (; F, L) = lorenz(sort(w series)) 291 292 fig, ax = plt.subplots(figsize=(9, 5.2)) 293 ax.plot(F, F, label="Lorenz curve, equality") 294 ax.plot(F, L, label="Lorenz curve, wealth distribution") 295 ax.legend() 296 plt.show() 297 if savefig 298 fig.savefig(filename) 299 end 300 301 end</pre>

monopolist_adj_costs.py

is unchanged - Python file monopolist_adj_costs.py does not exist

newton.jl

is unchanged - Python file newton.py does not exist

newton_solow.jl

is unchanged - Python file newton_solow.py does not exist

optimality_illustration.jl

is changed - Python file optimality_illustration.py does not exist

old public code	new private (jstac) code
<pre>31 end 32 33 ax.plot(xgrid, xgrid, "k--", lw=1, alpha=0.7, label=L"45") 34 35 ax.plot(xgrid, T1, "k-", lw=1)</pre>	<pre>31 end 32 33 ax.plot(xgrid, xgrid, "k--", lw=1, alpha=0.7, label=L"45^{circ}") 34 35 ax.plot(xgrid, T1, "k-", lw=1)</pre>

parallel_in_julia.ipynb

is unchanged - Python file parallel_in_julia.ipynb does not exist

pd_ratio.jl

is unchanged - Python file exists

plot_interest_rates.jl

is unchanged - Python file plot_interest_rates.py does not exist

power_series.jl

is unchanged - Python file exists

quantile_function.jl

new - Python file quantile_function.py does not exist

quantile_js.jl

new - Python file quantile_js.py does not exist

random_walk.jl

is unchanged - Python file random_walk.py does not exist

risk_sensitive_js.jl

is unchanged - Python file exists

rs_utility.jl

is unchanged - Python file exists

solow_fp.jl

is unchanged - Python file exists

solow_fp_adjust.jl

is unchanged - Python file solow_fp_adjust.py does not exist

stoch_dom_finite.jl

is unchanged - Python file stoch_dom_finite.py does not exist

s_approx.jl

is changed - Python file exists

old public code		new private (jstac) code	
1	"""	1	"""
2	Computes the approximate fixed point of T via successive approximation.	2	Computes an approximate fixed point of a given operator T
3		3	via successive approximation.
4	"""	4	"""
5	function successive_approx(T,	5	function successive_approx(T,
6	u_0;	6	u_0;
7	tolerance=1e-6,	7	tolerance=1e-6,
8	max_iter=10_000,	8	max_iter=10_000,
9	print_step=25)	9	print_step=25)
10	u = u_0	10	u = u_0
11	error = Inf	11	error = Inf
12		12	
13		13	
14	while (error > tolerance) & (k <= max_iter)	14	while (error > tolerance) & (k <= max_iter)
15	u_new = T(u)	15	u_new = T(u)
16	error = maximum(abs.(u_new - u))	16	error = maximum(abs.(u_new - u))
17		17	
18	if k % print_step == 0	18	if k % print_step == 0
19	println("Completed iteration \$k with error \$error.")	19	println("Completed iteration \$k with error \$error.")
20	end	20	end
21		21	
22	u = u_new	22	u = u_new
23	k += 1	23	k += 1
24		24	
25	println("Terminated successfully in \$k iterations.")	25	println("Terminated successfully in \$k iterations.")
26	else	26	else
27	println("Termination Warning: Error is greater than tolerance.")	27	println("Warning: hit iteration bound.")
28	end	28	end
29		29	

tauchen.jl

is unchanged - Python file tauchen.py does not exist

three_fixed_points.jl

new - Python file three_fixed_points.py does not exist

two_period_job_search.jl

is unchanged - Python file exists

up_down_stable.jl

new - Python file up_down_stable.py does not exist

val_consumption.jl

is unchanged - Python file val_consumption.py does not exist

v_star_illus.jl

is unchanged - Python file v_star_illus.py does not exist