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
f	1 using LinearAlgebra	f	1 using LinearAlgebra
n	2 r(A) = maximum(abs(λ) for λ in eigvals(A)) # Spectral radius	n	2 ρ(A) = maximum(abs(λ) for λ in eigvals(A)) # Spectral radius
	3 A = [0.4 0.1; # Test with arbitrary A		3 A = [0.4 0.1; # Test with arbitrary A
	4 0.7 0.2]		4 0.7 0.2]
t	5 print(r(A))	t	5 print(ρ(A))

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
18	`    f(w) = F(w; θ=-10)`	18	`    f(w) = F(w; θ=-10)`
19	`    ax.plot(w_grid, w_grid, "k--", alpha=0.6, label=L"45")`	19	`    ax.plot(w_grid, w_grid, "k--", alpha=0.6, label=L"45")`
t 20	`    ax.plot(w_grid, f.(w_grid), label=L"U = F")`	t 20	`    ax.plot(w_grid, f.(w_grid), label=L"\hat K = F")`
21	`    ax.set_xticks((0, 1, 2))`	21	`    ax.set_xticks((0, 1, 2))`
22	`    ax.set_yticks((0, 1, 2))`	22	`    ax.set_yticks((0, 1, 2))`

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

```
function create_savings_model(; R=1.01, β=0.98, γ=2.5,
                                w_min=0.01, w_max=5.0, w_size=200,
                                ρ=0.9, ν=0.1, y_size=5)
    w_grid = LinRange(w_min, w_max, w_size)
end

"B(w, y, w') = u(R*w + y - w') + β Σ_y' v(w', y') Q(y, y')."
function B(i, j, k, v, model)
    (; β, R, γ, w_grid, y_grid, Q) = model
```

new private (jstac) code

```
function create_savings_model(; R=1.01, β=0.98, γ=2.5,
                                w_min=0.01, w_max=20.0, w_size=200,
                                ρ=0.9, ν=0.1, y_size=5)
    w_grid = LinRange(w_min, w_max, w_size)
end

"B(w, y, w', v) = u(R*w + y - w') + β Σ_y' v(w', y') Q(y, y')."
function B(i, j, k, v, model)
    (; β, R, γ, w_grid, y_grid, Q) = model
```

finite_opt_saving_1.jl

is changed - Python file exists

old public code

```
end


"Get the value v_σ of policy σ."
function get_value(σ, model)
    # Unpack and set up
    (; β, R, γ, w_grid, y_grid, Q) = model

    wn, yn = length(w_grid), length(y_grid)
    n = wn * yn
    u(c) = c^(1-γ) / (1-γ)
    # Function to extract (i, j) from m = i + (j-1)*wn"
    single_to_multi(m) = (m-1)%wn + 1, div(m-1, wn) + 1
```

new private (jstac) code

```
end


"Get the value v_σ of policy σ."
function get_value(σ, model)
    # Unpack and set up
    (; β, R, γ, w_grid, y_grid, Q) = model
    w_idx, y_idx = (eachindex(g) for g in (w_grid, y_grid))
    wn, yn = length(w_idx), length(y_idx)
    n = wn * yn
    u(c) = c^(1-γ) / (1-γ)
    # Build P σ and r σ as multi-index arrays
    P_σ = zeros(wn, yn, wn, yn)
```

old public code	new private (jstac) code
23 # Allocate and create single index versions of P_σ and r_σ	
24 P_σ = zeros(n, n)	23 r_σ = zeros(wn, yn)
25 r_σ = zeros(n)	24 for (i, j) in product(w_idx, y_idx)
26 for m in 1:n	
27 i, j = single_to_multi(m)	
28 w, y, w′ = w_grid[i], y_grid[j], w_grid[σ[i, j]]	25 w, y, w′ = w_grid[i], y_grid[j], w_grid[σ[i, j]]
29 r_σ[m] = u(w + y - w′/R)	26 r_σ[i, j] = u(w + y - w′/R)
30 for m′ in 1:n	27 for (i′, j′) in product(w_idx, y_idx)
31 i′, j′ = single_to_multi(m′)	
32 if i′ == σ[i, j]	28 if i′ == σ[i, j]
n 33 P_σ[m, m′] = Q[j, j′]	n 29 P_σ[i, j, i′, j′] = Q[j, j′]
34 end	30 end
35 end	31 end
36 end	32 end
t 37 # Solve for the value of σ	t 33 # Reshape for matrix algebra
	34 P_σ = reshape(P_σ, n, n)
	35 r_σ = reshape(r_σ, n)
	36 # Apply matrix operations --- solve for the value of σ
38 v_σ = (I - β * P_σ) \ r_σ	37 v_σ = (I - β * P_σ) \ r_σ
39 # Return as multi-index array	38 # Return as multi-index array

finite_opt_saving_2.jl

is changed - Python file exists

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

old public code		new private (jstac) code	
		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 == #
44		79	
45	using PyPlot	80	using PyPlot
99	plt.show()	134	plt.show()
100	end	135	end
t		t 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()

old public code	new private (jstac) code
	181 plt.show() 182 183 if savefig 184 fig.savefig(figname) 185 end 186end

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
t 26 ax.plot(xgrid, xgrid, "k--", lw=1, alpha=0.7, label=L"45")	t 26 ax.plot(xgrid, xgrid, "k--", lw=1, alpha=0.7, label=L"45^{\circ}")
27	27
28 fp1 = (v1,)	28 fp1 = (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
include("s_approx.jl")	include("s_approx.jl")
using Distributions, OffsetArrays	using Distributions
m(x) = max(x, 0) # Convenience function	m(x) = max(x, 0) # Convenience function
function create_inventory_model(; β=0.98, # discount factor	function create_inventory_model(; β=0.98, # discount factor
K=40, # maximum inventory	K=40, # maximum inventory
c=0.2, κ=2, # cost parameters	c=0.2, κ=2, # cost paramters
p=0.6) # demand parameter	p=0.6) # demand parameter
ϕ(d) = (1 - p)^d * p # demand pdf	ϕ(d) = (1 - p)^d * p # demand pdf
	x_vals = collect(0:K) # set of inventory levels
return (; β, K, c, κ, p, ϕ)	return (; β, K, c, κ, p, ϕ, x_vals)
end	end
"The function B(x, a, v) = r(x, a) + β Σ_x′ v(x′) P(x, a, x′)."	"The function B(x, a, v) = r(x, a) + β Σ_x′ v(x′) P(x, a, x′)."
function B(x, a, v, model; d_max=100)	function B(x, a, v, model; d_max=100)
(; β, K, c, κ, p, ϕ) = model	(; β, K, c, κ, p, ϕ, x_vals) = model
reward = sum(min(x, d)*ϕ(d) for d in 0:d_max) - c * a - κ * (a > 0)	revenue = sum(min(x, d) * ϕ(d) for d in 0:d_max)
	current_profit = revenue - c * a - κ * (a > 0)
continuation_value = β * sum(v[m(x - d) + a] * ϕ(d) for d in 0:d_max)	next_value = sum(v[m(x - d) + a + 1] * ϕ(d) for d in 0:d_max)
return reward + continuation_value	return current_profit + β * next_value
end	end
"The Bellman operator."	"The Bellman operator."
function T(v, model)	function T(v, model)
(; β, K, c, κ, p, ϕ) = model	(; β, K, c, κ, p, ϕ, x_vals) = model
new_v = similar(v)	new_v = similar(v)
for x in 0:K	for (x_idx, x) in enumerate(x_vals)
Γx = 0:(K - x)	Γx = 0:(K - x)
new_v[x], _ = findmax(B(x, a, v, model) for a in Γx)	new_v[x_idx], _ = findmax(B(x, a, v, model) for a in Γx)
end	end
return new_v	return new_v
"Get a v-greedy policy. Returns a zero-based array."	"Get a v-greedy policy. Returns a zero-based array."
function get_greedy(v, model)	function get_greedy(v, model)
(; β, K, c, κ, p, ϕ) = model	(; β, K, c, κ, p, ϕ, x_vals) = model
σ_star = OffsetArray(zeros(Int32, K+1), 0:K)	σ_star = zero(x_vals)
for x in 0:K	for (x_idx, x) in enumerate(x_vals)
Γx = 0:(K - x)	Γx = 0:(K - x)
_, a_idx = findmax(B(x, a, v, model) for a in Γx)	_, a_idx = findmax(B(x, a, v, model) for a in Γx)
σ_star[x] = Γx[a_idx]	σ_star[x_idx] = Γx[a_idx]
end	end
return σ_star	return σ_star
"Use successive_approx to get v_star and then compute greedy."	"Use successive_approx to get v_star and then compute greedy."
function solve_inventory_model(v_init, model)	function solve_inventory_model(v_init, model)
(; β, K, c, κ, p, ϕ) = model	(; β, K, c, κ, p, ϕ, x_vals) = model
v_star = successive_approx(v -> T(v, model), v_init)	v_star = successive_approx(v -> T(v, model), v_init)
σ_star = get_greedy(v_star, model)	σ_star = get_greedy(v_star, model)
# Create an instance of the model and solve it	# Create an instance of the model and solve it
model = create_inventory_model()	model = create_inventory_model()
(; β, K, c, κ, p, ϕ) = model	(; β, K, c, κ, p, ϕ, x_vals) = model
v_init = OffsetArray(zeros(K+1), 0:K)	v_init = zeros(length(x_vals))
v_star, σ_star = solve_inventory_model(v_init, model)	v_star, σ_star = solve_inventory_model(v_init, model)
for t in 1:(ts_length-1)	for t in 1:(ts_length-1)
D = rand(G)	D = rand(G)
X[t+1] = m(X[t] - D) + σ_star[X[t]]	X[t+1] = m(X[t] - D) + σ_star[X[t] + 1]

old public code		new private (jstac) code	
73	end	75	end
74	return X	76	return X

inventory_sdd.jl

is changed - **Python file exists**

old public code	new private (jstac) code
"""	"""
Inventory management model with state-dependent discounting. The discount factor takes the form β_t = Z_t, where (Z_t) is a discretization of a Gaussian AR(1) process	Inventory management model with state-dependent discounting. The discount factor takes the form β t = Z t, where (Z_t) is a discretization of the Gaussian AR(1) process
X t = ρ X {t-1} + b + ν W t.	X t = ρ X {t-1} + b + ν W t.
include("s_approx.jl")	include("s_approx.jl")
using LinearAlgebra, Distributions, OffsetArrays, QuantEcon	using LinearAlgebra, Distributions, QuantEcon
function create_sdd_inventory_model(;	function create_sdd_inventory_model(;
ρ=0.98, ν=0.002, n_z=20, b=0.97, # Z state parameters	ρ=0.98, ν=0.002, n_z=20, b=0.97, # Z state parameters
K=40, c=0.2, κ=0.8, p=0.6) # firm and demand parameters	K=40, c=0.2, κ=0.8, p=0.6) # firm and demand parameters
ϕ(d) = (1 - p)^d * p # demand pdf	ϕ(d) = (1 - p)^d * p # demand pdf
	y_vals = collect(0:K) # inventory levels
mc = tauchen(n_z, ρ, ν)	mc = tauchen(n_z, ρ, ν)
z_vals, Q = mc.state_values .+ b, mc.p	z_vals, Q = mc.state_values .+ b, mc.p
rL = maximum(abs.(eigvals(z_vals .* Q)))	ρL = maximum(abs.(eigvals(z_vals .* Q)))
@assert rL < 1 "Error: r(L) ≥ 1." # check r(L) < 1	@assert ρL < 1 "Error: ρ(L) ≥ 1." # check ρ(L) < 1
return (; K, c, κ, p, ϕ, z_vals, Q)	return (; K, c, κ, p, ϕ, y_vals, z_vals, Q)
end	end
m(x) = max(x, 0) # Convenience function	m(y) = max(y, 0) # Convenience function
"The function B(x, z, a, v) = r(x, a) + β(z) Σ_x′ v(x′) P(x, a, x′)."	"The function B(x, a, v) = r(x, a) + β(x) Σ_x′ v(x′) P(x, a, x′)."
function B(x, i_z, a, v, model; d_max=100)	function B(x, i_z, a, v, model; d_max=100)
(; K, c, κ, p, ϕ, z_vals, Q) = model	(; K, c, κ, p, ϕ, y_vals, z_vals, Q) = model
z = z vals[i z]	z = z vals[i z]
reward = sum(min(x, d)*ϕ(d) for d in 0:d_max) - c * a - κ * (a > 0)	revenue = sum(min(x, d)*ϕ(d) for d in 0:d_max)
	current_profit = revenue - c * a - κ * (a > 0)
cv = 0.0	cv = 0.0
for (i_z′, z′) in enumerate(z_vals)	for i z′ in eachindex(z_vals)
	for d in 0:d_max
cv += sum(v[m(x - d) + a, i_z′] * ϕ(d) for d in 0:d_max) * Q[i_z, i_z′]	cv += v[m(x - d) + a + 1, i_z′] * ϕ(d) * Q[i_z, i_z′]
	end
end	end
return reward + z * cv	return current_profit + z * cv
end	end
"The Bellman operator."	"The Bellman operator."
function T(v, model)	function T(v, model)
(; K, c, κ, p, ϕ, z_vals, Q) = model	(; K, c, κ, p, ϕ, y_vals, z_vals, Q) = model
new_v = similar(v)	new_v = similar(v)
for (i z, z) in enumerate(z_vals)	for (i z, z) in enumerate(z vals)
for x in 0:K	for (i_y, y) in enumerate(y_vals)
Γx = 0:(K - x)	Γy = 0:(K - y)
new_v[x, i_z], _ = findmax(B(x, i_z, a, v, model) for a in Γx)	new_v[i_y, i_z], _ = findmax(B(y, i_z, a, v, model) for a in Γy)

old public code	new private (jstac) code
47 end	50 end
48 end	51 end
52 "Get a v-greedy policy. Returns a zero-based array."	55 "Get a v-greedy policy. Returns a zero-based array."
53 function get_greedy(v, model)	56 function get_greedy(v, model)
n 54 (; K, c, κ, p, ϕ, z_vals, Q) = model	n 57 (; K, c, κ, p, ϕ, y_vals, z_vals, Q) = model
55 n_z = length(z_vals)	58 n_z = length(z_vals)
n 56 σ_star = OffsetArray(zeros(Int32, K+1, n_z), 0:K, 1:n_z)	n 59 σ_star = zeros(Int32, K+1, n_z)
57 for (i_z, z) in enumerate(z_vals)	60 for (i_z, z) in enumerate(z_vals)
n 58 for x in 0:K	n 61 for (i_y, y) in enumerate(y_vals)
59 Γx = 0:(K - x)	62 Γy = 0:(K - y)
60 _, a_idx = findmax(B(x, i_z, a, v, model) for a in Γx)	63 _, i_a = findmax(B(y, i_z, a, v, model) for a in Γy)
61 σ_star[x, i_z] = Γx[a_idx]	64 σ_star[i_y, i_z] = Γy[i_a]
62 end	65 end
63 end	66 end
68 "Use successive_approx to get v_star and then compute greedy."	71 "Use successive_approx to get v_star and then compute greedy."
69 function solve_inventory_model(v_init, model)	72 function solve_inventory_model(v_init, model)
n 70 (; K, c, κ, p, ϕ, z_vals, Q) = model	n 73 (; K, c, κ, p, ϕ, y_vals, z_vals, Q) = model
71 v_star = successive_approx(v -> T(v, model), v_init)	74 v_star = successive_approx(v -> T(v, model), v_init)
72 σ_star = get_greedy(v_star, model)	75 σ_star = get_greedy(v_star, model)
84 # Create an instance of the model and solve it	87 # Create an instance of the model and solve it
85 model = create_sdd_inventory_model()	88 model = create_sdd_inventory_model()
n 86 (; K, c, κ, p, ϕ, z_vals, Q) = model	n 89 (; K, c, κ, p, ϕ, y_vals, z_vals, Q) = model
87 n_z = length(z_vals)	90 n_z = length(z_vals)
n 88 v_init = OffsetArray(zeros(Float64, K+1, n_z), 0:K, 1:n_z)	n 91 v_init = zeros(Float64, K+1, n_z)
89 println("Solving model.")	92 println("Solving model.")
90 v_star, σ_star = solve_inventory_model(v_init, model)	93 v_star, σ_star = solve_inventory_model(v_init, model)
99 for t in 1:(ts_length-1)	102 for t in 1:(ts_length-1)
100 D′ = rand(G)	103 D′ = rand(G)
n 101 X[t+1] = m(X[t] - D′) + σ_star[X[t], i_z[t]]	n 104 X[t+1] = m(X[t] - D′) + σ_star[X[t] + 1, i_z[t]]
102 end	105 end
103 return X, z_vals[i_z]	106 return X, z_vals[i_z]
112	115
113 ax = axes[1]	116 ax = axes[1]
n 114 ax.plot(X, label=L"X_t", alpha=0.7)	n 117 ax.plot(X, label="inventory", alpha=0.7)
115 ax.set_xlabel(L"t", fontsize=fontsize)	118 ax.set_xlabel(L"t", fontsize=fontsize)
n 116 ax.set_ylabel("inventory", fontsize=fontsize)	n
117 ax.legend(fontsize=fontsize, frameon=false)	119 ax.legend(fontsize=fontsize, frameon=false)
118 ax.set_ylim(0, maximum(X)+3)	120 ax.set_ylim(0, maximum(X)+3)
124 ax.plot(r, label=L"r_t", alpha=0.7)	126 ax.plot(r, label=L"r_t", alpha=0.7)
125 ax.set_xlabel(L"t", fontsize=fontsize)	127 ax.set_xlabel(L"t", fontsize=fontsize)
t 126 ax.set_ylabel("interest rate", fontsize=fontsize)	t
127 ax.legend(fontsize=fontsize, frameon=false)	128 ax.legend(fontsize=fontsize, frameon=false)
128 #ax.set_ylim(0, maximum(X)+8)	129 #ax.set_ylim(0, maximum(X)+8)

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
f	1 using QuantEcon, LinearAlgebra, IterTools	f	1 using QuantEcon, LinearAlgebra, IterTools
	2		2
n	3 function create_savings_model(; R=1.01, β=0.98, γ=2.5,	n	3 function create_savings_model(; β=0.98, γ=2.5,
	4 w_min=0.01, w_max=10.0, w_size=100,		4 w_min=0.01, w_max=20.0, w_size=100,

old public code

```
                                ρ=0.9, ν=0.1, z_size=20,
                                ϵ_min=-0.25, ϵ_max=0.25, ϵ_size=30)
    ϵ_grid = LinRange(ϵ_min, ϵ_max, ϵ_size)
    ϕ = ones(ϵ_size) * (1 / ϵ_size)  # Uniform distribution
    w_grid = LinRange(w_min, w_max, w_size)
    mc = tauchen(z_size, ρ, ν)
    z_grid, Q = exp.(mc.state_values), mc.p
    return (; β, R, γ, ϵ_grid, ϕ, w_grid, z_grid, Q)
end



"""
The function

B(w, z, ϵ, w') =
    u(w + z + ϵ - w'/R) + β Σ v(w', z', ϵ') Q(z, z') ϕ(ϵ')

"""
function B(i, j, k, l, v, model)
    (; β, R, γ, ϵ_grid, ϕ, w_grid, z_grid, Q) = model
    w, z, ϵ, w' = w_grid[i], z_grid[j], ϵ_grid[k], w_grid[l]

    c = w + z + ϵ - (w'/ R)
    exp_value = 0.0
    for m in eachindex(z_grid)
        for n in eachindex(ϵ_grid)
            exp_value += v[l, m, n] * Q[j, m] * ϕ[n]
        end
    end
    return c > 0 ? c^(1-γ)/(1-γ) + β * exp_value : -Inf
end

"The policy operator."
function T_σ(v, σ, model)
    (; β, R, γ, ϵ_grid, ϕ, w_grid, z_grid, Q) = model

    w_idx, z_idx, ϵ_idx = (eachindex(g) for g in (w_grid, z_grid, ϵ_grid))
    v_new = similar(v)
    for (i, j, k) in product(w_idx, z_idx, ϵ_idx)
        v_new[i, j, k] = B(i, j, k, σ[i, j, k], v, model)
    end

"Compute a v-greedy policy."
function get_greedy(v, model)
    (; β, R, γ, ϵ_grid, ϕ, w_grid, z_grid, Q) = model
    w_idx, z_idx, ϵ_idx = (eachindex(g) for g in (w_grid, z_grid, ϵ_grid))
    σ = Array{Int32}(undef, length(w_idx), length(z_idx), length(ϵ_idx))
    for (i, j, k) in product(w_idx, z_idx, ϵ_idx)
        _, σ[i, j, k] = findmax(B(i, j, k, l, v, model) for l in w_idx)
    end

"Optimistic policy iteration routine."
function optimistic_policy_iteration(model; tolerance=1e-5, m=100)
    (; β, R, γ, ϵ_grid, ϕ, w_grid, z_grid, Q) = model
    v = zeros(length(w_grid), length(z_grid), length(ϵ_grid))
    error = tolerance + 1
    while error > tolerance

```

new private (jstac) code

```
                                ρ=0.9, ν=0.1, y_size=20,
                                η_min=0.75, η_max=1.25, η_size=2)
    η_grid = LinRange(η_min, η_max, η_size)
    ϕ = ones(η_size) * (1 / η_size)  # Uniform distributoin
    w_grid = LinRange(w_min, w_max, w_size)
    mc = tauchen(y_size, ρ, ν)
    y_grid, Q = exp.(mc.state_values), mc.p
    return (; β, γ, η_grid, ϕ, w_grid, y_grid, Q)
end


"""
B(w, y, η, w') = u(w + y - w'/η)) + β Σ v(w', y', η') Q(y, y') ϕ(η')

"""
function B(i, j, k, l, v, model)
    (; β, γ, η_grid, ϕ, w_grid, y_grid, Q) = model
    w, y, η, w' = w_grid[i], y_grid[j], η_grid[k], w_grid[l]
    u(c) = c^(1-γ)/(1-γ)
    c = w + y - (w'/ η)
    exp_value = 0.0
    for m in eachindex(y_grid)
        for n in eachindex(η_grid)
            exp_value += v[l, m, n] * Q[j, m] * ϕ[n]
        end
    end
    return c > 0 ? u(c) + β * exp_value : -Inf
end

"The policy operator."
function T_σ(v, σ, model)
    (; β, γ, η_grid, ϕ, w_grid, y_grid, Q) = model
    grids = w_grid, y_grid, η_grid
    w_idx, y_idx, η_idx = (eachindex(g) for g in grids)
    v_new = similar(v)
    for (i, j, k) in product(w_idx, y_idx, η_idx)
        v_new[i, j, k] = B(i, j, k, σ[i, j, k], v, model)
    end

"Compute a v-greedy policy."
function get_greedy(v, model)
    (; β, γ, η_grid, ϕ, w_grid, y_grid, Q) = model
    w_idx, y_idx, η_idx = (eachindex(g) for g in (w_grid, y_grid, η_grid))
    σ = Array{Int32}(undef, length(w_idx), length(y_idx), length(η_idx))
    for (i, j, k) in product(w_idx, y_idx, η_idx)
        _, σ[i, j, k] = findmax(B(i, j, k, l, v, model) for l in w_idx)
    end

"Optimistic policy iteration routine."
function optimistic_policy_iteration(model; tolerance=1e-5, m=100)
    (; β, γ, η_grid, ϕ, w_grid, y_grid, Q) = model
    v = zeros(length(w_grid), length(y_grid), length(η_grid))
    error = tolerance + 1
    while error > tolerance

```

old public code	new private (jstac) code
## == Functions for modified OPI == ##	## == Functions for modified OPI == ##
"D(w, z, ϵ, w′, g) = u(w + z + ϵ - w′/R) + β g(z, w′)."	"D(w, y, η, w′, g) = u(w + y - w′/η) + β g(y, w′)."
@inline function D(i, j, k, l, g, model)	@inline function D(i, j, k, l, g, model)
(; β, R, γ, ϵ_grid, ϕ, w_grid, z_grid, Q) = model	(; β, γ, η_grid, ϕ, w_grid, y_grid, Q) = model
w, z, ϵ, w′ = w_grid[i], z_grid[j], ϵ_grid[k], w_grid[l]	w, y, η, w′ = w_grid[i], y_grid[j], η_grid[k], w_grid[l]
	u(c) = c^(1-γ)/(1-γ)
c = w + z + ϵ - (w′ / R)	c = w + y - (w′/η)
return c > 0 ? c^(1-γ)/(1-γ) + β * g[j, l] : -Inf	return c > 0 ? u(c) + β * g[j, l] : -Inf
end	end
"Compute a g-greedy policy."	"Compute a g-greedy policy."
function get_g_greedy(g, model)	function get_g_greedy(g, model)
(; β, R, γ, ϵ_grid, ϕ, w_grid, z_grid, Q) = model	(; β, γ, η_grid, ϕ, w_grid, y_grid, Q) = model
w_idx, z_idx, ϵ_idx = (eachindex(g) for g in (w_grid, z_grid, ϵ_grid))	w_idx, y_idx, η_idx = (eachindex(g) for g in (w_grid, y_grid, η_grid))
σ = Array{Int32}(undef, length(w_idx), length(z_idx), length(ϵ_idx))	σ = Array{Int32}(undef, length(w_idx), length(y_idx), length(η_idx))
for (i, j, k) in product(w_idx, z_idx, ϵ_idx)	for (i, j, k) in product(w_idx, y_idx, η_idx)
_, σ[i, j, k] = findmax(D(i, j, k, l, g, model) for l in w_idx)	_, σ[i, j, k] = findmax(D(i, j, k, l, g, model) for l in w_idx)
end	end
"The modified policy operator."	"The modified policy operator."
function R_σ(g, σ, model)	function R_σ(g, σ, model)
(; β, R, γ, ϵ_grid, ϕ, w_grid, z_grid, Q) = model	(; β, γ, η_grid, ϕ, w_grid, y_grid, Q) = model
w_idx, z_idx, ϵ_idx = (eachindex(g) for g in (w_grid, z_grid, ϵ_grid))	w_idx, y_idx, η_idx = (eachindex(g) for g in (w_grid, y_grid, η_grid))
g_new = similar(g)	g_new = similar(g)
for (j, i′) in product(z_idx, w_idx) # j -> z, i′ -> w′	for (j, i′) in product(y_idx, w_idx) # j indexes y, i′ indexes w′
out = 0.0	out = 0.0
for j′ in z_idx # j′ -> z′	for j′ in y_idx # j′ indexes y′
for k′ in ϵ_idx # k′ -> ϵ′	for k′ in η_idx # k′ indexes η′
# D(w′, z′, ϵ′, σ(w′, z′, ϵ′), g)	
out += D(i′, j′, k′, σ[i′, j′, k′], g, model) *	out += D(i′, j′, k′, σ[i′, j′, k′], g, model) *
Q[j, j′] * ϕ[k′]	Q[j, j′] * ϕ[k′]
"Modified optimistic policy iteration routine."	"Modified optimistic policy iteration routine."
function mod_opi(model; tolerance=1e-5, m=100)	function mod_opi(model; tolerance=1e-5, m=100)
(; β, R, γ, ϵ_grid, ϕ, w_grid, z_grid, Q) = model	(; β, γ, η_grid, ϕ, w_grid, y_grid, Q) = model
g = zeros(length(z_grid), length(w_grid))	g = zeros(length(y_grid), length(w_grid))
error = tolerance + 1	error = tolerance + 1
while error > tolerance	while error > tolerance
# Plots	# == Simulations and inequality measures == #
	function simulate_wealth(m)
	model = create_savings_model()
	(; β, γ, η_grid, ϕ, w_grid, y_grid, Q) = model
	σ_star = mod_opi(model)
	# Simulate labor income
	mc = MarkovChain(Q)
	y_idx_series = simulate(mc, m)
	# IID Markov chain with uniform draws
	l = length(η_grid)
	mc = MarkovChain(ones(l, l) * (1/l))

old public code	new private (jstac) code
	η_idx_series = simulate(mc, m)
	w_idx_series = similar(y_idx_series)
	w_idx_series[1] = 1
	for t in 1:(m-1)
	i, j, k = w_idx_series[t], y_idx_series[t], η_idx_series[t]
	w_idx_series[t+1] = σ_star[i, j, k]
	end
	w_series = w_grid[w_idx_series]
	return w_series
	end
	function lorenz(v) # assumed sorted vector
	S = cumsum(v) # cumulative sums: [v[1], v[1] + v[2], ...]
	F = (1:length(v)) / length(v)
	L = S ./ S[end]
	return (; F, L) # returns named tuple
	end
	gini(v) = (2 * sum(i * y for (i,y) in enumerate(v))/sum(v)
	- (length(v) + 1))/length(v)
	# == Plots == #
using PyPlot	using PyPlot
function plot_contours(; savefig=false,	function plot_contours(; savefig=false,
figname="../figures/modified_opt_savings_1.pdf")	figname="../figures/modified_opt_savings_1.pdf")
model = create_savings_model()	model = create_savings_model()
(; β, R, γ, ϵ_grid, ϕ, w_grid, z_grid, Q) = model	(; β, γ, η_grid, ϕ, w_grid, y_grid, Q) = model
σ_star = mod_opi(model)	σ_star = optimistic_policy_iteration(model)
fig, axes = plt.subplots(2, 1, figsize=(10, 8))	fig, axes = plt.subplots(2, 1, figsize=(10, 8))
z_idx, ϵ_idx = eachindex(z_grid), eachindex(ϵ_grid)	y_idx, η_idx = eachindex(y_grid), eachindex(η_grid)
H = zeros(length(z_grid), length(ϵ_grid))	H = zeros(length(y_grid), length(η_grid))
w_indices = (1, length(w_grid))	w_indices = (1, length(w_grid))
for (ax, w_idx, title) in zip(axes, w_indices, titles)	for (ax, w_idx, title) in zip(axes, w_indices, titles)
for (i_z, i_ϵ) in product(z_idx, ϵ_idx)	for (i_y, i_ϵ) in product(y_idx, η_idx)
w, z, ϵ = w_grid[w_idx], z_grid[i_z], ϵ_grid[i_ϵ]	w, y, η = w_grid[w_idx], y_grid[i_y], η_grid[i ϵ]
H[i_z, i_ϵ] = w_grid[σ_star[w_idx, i_z, i_ϵ]]	H[i_y, i_ϵ] = w_grid[σ_star[w_idx, i_y, i_ϵ]] / (w+y)
end	end
cs1 = ax.contourf(z_grid, ϵ_grid, transpose(H), alpha=0.5)	cs1 = ax.contourf(y_grid, η_grid, transpose(H), alpha=0.5)
#ctr1 = ax.contour(w_vals, z_vals, transpose(H), levels=[0.0])	
#plt.clabel(ctr1, inline=1, fontsize=13)	
plt.colorbar(cs1, ax=ax) #, format="%.6f")	plt.colorbar(cs1, ax=ax) #, format="%.6f")
ax.set_title(title, fontsize=fontsize)	ax.set_title(title, fontsize=fontsize)
ax.set_xlabel(L"z", fontsize=fontsize)	ax.set_xlabel(L"y", fontsize=fontsize)

old public code	new private (jstac) code
177 ax.set_ylabel(L"\varepsilon", fontsize=fontsize)	214 ax.set_ylabel(L"\varepsilon", fontsize=fontsize)
178 end	215 end
186	223
187	224
t	225 function plot policies(; savefig=false,
	226 figname="../figures/modified_opt_savings_2.pdf")
	227
	228 model = create savings model()
	229 (; β, γ, η grid, ϕ, w grid, y grid, Q) = model
	230 σ 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, η) in enumerate(η grid)
	237 label = L"\sigma^*" * " at " * L"\eta = " * "$η"
	238 ax.plot(w_grid, w_grid[σ_star[:, y_bar, i]], label=label)
	239 end
	240 ax.legend(fontsize=fontsize)
	241 plt.show()
	242
	243 plt.tight layout()
	244 if savefig
	245 fig.savefig(figname)
	246 end
	247 plt.show()
	248 end
	249
	250
	251 function plot time series(; m=2 000,
	252 savefig=false,
	253 figname="../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(figname)
	263 end
	264
	265 end
	266
	267 function plot histogram(; m=1 000 000,
	268 savefig=false,
	269 figname="../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()

old public code	new private (jstac) code
	278 if savefig 279 fig.savefig(figname) 280 end 281 282 end 283 284 285 function plot_lorenz(; m=1_000_000, 286 savefig=false, 287 figname="../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(figname) 299 end 300 301 end

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
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)	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)

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
"""	"""
Computes the approximate fixed point of T via successive approximation.	Computes an approximate fixed point of a given operator T
	via successive approximation.
"""	"""
function successive_approx(T, # Operator (callable)	function successive_approx(T, # operator (callable)
u_0; # Initial condition	u_0; # initial condition
tolerance=1e-6, # Error tolerance	tolerance=1e-6, # error tolerance
max_iter=10_000, # Max iteration bound	max_iter=10_000, # max iteration bound
print_step=25) # Print at multiples	print_step=25) # print at multiples
u = u_0	u = u_0
error = Inf	error = Inf
while (error > tolerance) & (k <= max_iter)	while (error > tolerance) & (k <= max_iter)
u_new = T(u)	u_new = T(u)
error = maximum(abs.(u_new - u))	error = maximum(abs.(u_new - u))
if k % print_step == 0	if k % print_step == 0
println("Completed iteration $k with error $error.")	println("Completed iteration $k with error $error.")
end	end
u = u_new	u = u_new
k += 1	k += 1
println("Terminated successfully in $k iterations.")	println("Terminated successfully in $k iterations.")
else	else
println("Termination Warning: Error is greater than tolerance.")	println("Warning: hit iteration bound.")
end	end

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