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
`using LinearAlgebra`	`using LinearAlgebra`
`r(A) = maximum(abs(λ) for λ in eigvals(A))  # Spectral radius`	`ρ(A) = maximum(abs(λ) for λ in eigvals(A))  # Spectral radius`
`A = [0.4 0.1;                               # Test with arbitrary A`	`A = [0.4 0.1;                               # Test with arbitrary A`
`     0.7 0.2]`	`     0.7 0.2]`
`print(r(A))`	`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
`    f(w) = F(w; θ=-10)`	`    f(w) = F(w; θ=-10)`
`    ax.plot(w_grid, w_grid, "k--", alpha=0.6, label=L"45")`	`    ax.plot(w_grid, w_grid, "k--", alpha=0.6, label=L"45")`
`    ax.plot(w_grid, f.(w_grid), label=L"U = F")`	`    ax.plot(w_grid, f.(w_grid), label=L"\hat K = F")`
`    ax.set_xticks((0, 1, 2))`	`    ax.set_xticks((0, 1, 2))`
`    ax.set_yticks((0, 1, 2))`	`    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	new private (jstac) code
function create_savings_model(; R=1.01, β=0.98, γ=2.5,	function create_savings_model(; R=1.01, β=0.98, γ=2.5,
w_min=0.01, w_max=5.0, w_size=200,	w_min=0.01, w_max=20.0, w_size=200,
ρ=0.9, ν=0.1, y_size=5)	ρ=0.9, ν=0.1, y_size=5)
w_grid = LinRange(w_min, w_max, w_size)	w_grid = LinRange(w_min, w_max, w_size)
end	end
"B(w, y, w′) = u(R*w + y - w′) + β Σ_y′ v(w′, y′) Q(y, y′)."	"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)	function B(i, j, k, v, model)
(; β, R, γ, w_grid, y_grid, Q) = model	(; β, R, γ, w_grid, y_grid, Q) = model

finite_opt_saving_1.jl

is changed - Python file exists

old public code	new private (jstac) code
end	end
"Get the value v_σ of policy σ."	"Get the value v_σ of policy σ."
function get_value(σ, model)	function get_value(σ, model)
# Unpack and set up	# Unpack and set up
(; β, R, γ, w_grid, y_grid, Q) = model	(; 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_grid), length(y_grid)	wn, yn = length(w_idx), length(y_idx)
n = wn * yn	n = wn * yn
u(c) = c^(1-γ) / (1-γ)	u(c) = c^(1-γ) / (1-γ)
# Function to extract (i, j) from m = i + (j-1)*wn"	# Build P σ and r σ as multi-index arrays
single_to_multi(m) = (m-1)%wn + 1, div(m-1, wn) + 1	P_σ = zeros(wn, yn, wn, yn)
# Allocate and create single index versions of P_σ and r_σ	
P_σ = zeros(n, n)	r_σ = zeros(wn, yn)
r_σ = zeros(n)	for (i, j) in product(w_idx, y_idx)
for m in 1:n	
i, j = single_to_multi(m)	
w, y, w′ = w_grid[i], y_grid[j], w_grid[σ[i, j]]	w, y, w′ = w_grid[i], y_grid[j], w_grid[σ[i, j]]
r_σ[m] = u(w + y - w′/R)	r_σ[i, j] = u(w + y - w′/R)
for m′ in 1:n	for (i′, j′) in product(w_idx, y_idx)
i′, j′ = single_to_multi(m′)	
if i′ == σ[i, j]	if i′ == σ[i, j]
P_σ[m, m′] = Q[j, j′]	P_σ[i, j, i′, j′] = Q[j, j′]
end	end
end	end
end	end
# Solve for the value of σ	# Reshape for matrix algebra
	P_σ = reshape(P_σ, n, n)
	r_σ = reshape(r_σ, n)
	# Apply matrix operations --- solve for the value of σ
v_σ = (I - β * P_σ) \ r_σ	v_σ = (I - β * P_σ) \ r_σ
# Return as multi-index array	# Return as multi-index array

finite_opt_saving_2.jl

is changed - Python file exists

old public code	new private (jstac) code
end	end
	# == Simulations and inequality measures == #
# Plots	function simulate_wealth(m)
	model = create_savings_model()
	σ_star = optimistic_policy_iteration(model)
	(; β, R, γ, w_grid, y_grid, Q) = model
	# Simulate labor income (indices rather than grid values)
	mc = MarkovChain(Q)
	y_idx_series = simulate(mc, m)
	# Compute corresponding wealth time series
	w_idx_series = similar(y_idx_series)
	w_idx_series[1] = 1 # initial condition
	for t in 1:(m-1)
	i, j = w_idx_series[t], y_idx_series[t]
	w_idx_series[t+1] = σ_star[i, j]
	end
	w_series = w_grid[w_idx_series]

old public code	new private (jstac) code
	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
plt.show()	plt.show()
end	end
t	t
	function plot time series(; m=2 000,
	savefig=false,
	figname="../figures/finite_opt_saving_ts.pdf")
	w series = simulate wealth(m)
	fig, ax = plt.subplots(figsize=(9, 5.2))
	ax.plot(w series, label=L"w t")
	ax.set xlabel("time", fontsize=fontsize)
	ax.legend(fontsize=fontsize)
	plt.show()
	if savefig
	fig.savefig(figname)
	end
	end
	function plot histogram(; m=1 000 000,
	savefig=false,
	figname="../figures/finite_opt_saving_hist.pdf")
	w series = simulate wealth(m)
	g = round(gini(sort(w series)), digits=2)
	fig, ax = plt.subplots(figsize=(9, 5.2))
	ax.hist(w series, bins=40, density=true)
	ax.set xlabel("wealth", fontsize=fontsize)
	ax.text(15, 0.4, "Gini = $g", fontsize=fontsize)
	plt.show()
	if savefig
	fig.savefig(figname)
	end
	end
	function plot lorenz(; m=1 000 000,
	savefig=false,
	figname="../figures/finite_opt_saving_lorenz.pdf")
	w series = simulate wealth(m)
	(; F, L) = lorenz(sort(w_series))
	fig, ax = plt.subplots(figsize=(9, 5.2))
	ax.plot(F, F, label="Lorenz curve, equality")
	ax.plot(F, L, label="Lorenz curve, wealth distribution")
	ax.legend()
	plt.show()
	if savefig
	fig.savefig(figname)
	end
	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
ax.plot(xgrid, Tsp.(xgrid), lw=2, alpha=0.6, label=lb_Tsp)	ax.plot(xgrid, Tsp.(xgrid), lw=2, alpha=0.6, label=lb_Tsp)
t ax.plot(xgrid, xgrid, "k--", lw=1, alpha=0.7, label=L"45")	t ax.plot(xgrid, xgrid, "k--", lw=1, alpha=0.7, label=L"45^{\circ}")
fp1 = (v1,)	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

```
include("s_approx.jl")
using Distributions, OffsetArrays
m(x) = max(x, 0)  # Convenience function

function create_inventory_model(; β=0.98,      # discount factor
                                  K=40,        # maximum inventory
                                  c=0.2, κ=2,  # cost parameters
                                  p=0.6)       # demand parameter
    ϕ(d) = (1 - p)^d * p  # demand pdf

    return (; β, K, c, κ, p, ϕ)
end

"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)
    (; β, K, c, κ, p, ϕ) = model
    reward = sum(min(x, d)*ϕ(d) for d in 0:d_max) - c * a - κ * (a > 0)

    continuation_value = β * sum(v[m(x - d) + a] * ϕ(d) for d in 0:d_max)
    return reward + continuation_value
end

"The Bellman operator."
function T(v, model)
    (; β, K, c, κ, p, ϕ) = model
    new_v = similar(v)
    for x in 0:K
        Γx = 0:(K - x)
        new_v[x], _ = findmax(B(x, a, v, model) for a in Γx)
    end
    return new_v
```

```
"Get a v-greedy policy.  Returns a zero-based array."
function get_greedy(v, model)
    (; β, K, c, κ, p, ϕ) = model
    σ_star = OffsetArray(zeros(Int32, K+1), 0:K)
    for x in 0:K
        Γx = 0:(K - x)
        _, a_idx = findmax(B(x, a, v, model) for a in Γx)
        σ_star[x] = Γx[a_idx]
    end
    return σ_star
```

```
"Use successive_approx to get v_star and then compute greedy."
function solve_inventory_model(v_init, model)
    (; β, K, c, κ, p, ϕ) = model
    v_star = successive_approx(v -> T(v, model), v_init)
    σ_star = get_greedy(v_star, model)
```

```
# Create an instance of the model and solve it
model = create_inventory_model()
(; β, K, c, κ, p, ϕ) = model
v_init = OffsetArray(zeros(K+1), 0:K)
v_star, σ_star = solve_inventory_model(v_init, model)

```

```
    for t in 1:(ts_length-1)
        D = rand(G)
        X[t+1] = m(X[t] - D) + σ_star[X[t]]
    end
    return X
```

new private (jstac) code

```
include("s_approx.jl")
using Distributions
m(x) = max(x, 0)  # Convenience function

function create_inventory_model(; β=0.98,      # discount factor
                                  K=40,        # maximum inventory
                                  c=0.2, κ=2,  # cost paramters
                                  p=0.6)       # demand parameter
    ϕ(d) = (1 - p)^d * p         # demand pdf
    x_vals = collect(0:K)        # set of inventory levels
    return (; β, K, c, κ, p, ϕ, x_vals)
end

"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)
    (; β, K, c, κ, p, ϕ, x_vals) = model
    revenue = sum(min(x, d) * ϕ(d) for d in 0:d_max)
    current_profit = revenue - c * a - κ * (a > 0)
    next_value = sum(v[m(x - d) + a + 1] * ϕ(d) for d in 0:d_max)
    return current_profit + β * next_value
end

"The Bellman operator."
function T(v, model)
    (; β, K, c, κ, p, ϕ, x_vals) = model
    new_v = similar(v)
    for (x_idx, x) in enumerate(x_vals)
        Γx = 0:(K - x)
        new_v[x_idx], _ = findmax(B(x, a, v, model) for a in Γx)
    end
    return new_v
```

```
"Get a v-greedy policy.  Returns a zero-based array."
function get_greedy(v, model)
    (; β, K, c, κ, p, ϕ, x_vals) = model
    σ_star = zero(x_vals)
    for (x_idx, x) in enumerate(x_vals)
        Γx = 0:(K - x)
        _, a_idx = findmax(B(x, a, v, model) for a in Γx)
        σ_star[x_idx] = Γx[a_idx]
    end
    return σ_star
```

```
"Use successive_approx to get v_star and then compute greedy."
function solve_inventory_model(v_init, model)
    (; β, K, c, κ, p, ϕ, x_vals) = model
    v_star = successive_approx(v -> T(v, model), v_init)
    σ_star = get_greedy(v_star, model)
```

```
# Create an instance of the model and solve it
model = create_inventory_model()
(; β, K, c, κ, p, ϕ, x_vals) = model
v_init = zeros(length(x_vals))
v_star, σ_star = solve_inventory_model(v_init, model)

```

```
    for t in 1:(ts_length-1)
        D = rand(G)
        X[t+1] = m(X[t] - D) + σ_star[X[t] + 1]
    end
    return X
```

inventory_sdd.jl

is changed - Python file exists

old public 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

    X_t = ρ X_{t-1} + b + ν W_t.
```

```

include("s_approx.jl")
using LinearAlgebra, Distributions, OffsetArrays, QuantEcon

function create_sdd_inventory_model(;
        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
    ϕ(d) = (1 - p)^d * p                  # demand pdf

    mc = tauchen(n_z, ρ, ν)
    z_vals, Q = mc.state_values .+ b, mc.p
    rL = maximum(abs.(eigvals(z_vals .* Q)))
    @assert  rL < 1 "Error: r(L) ≥ 1."    # check r(L) < 1
    return (; K, c, κ, p, ϕ, z_vals, Q)
end

m(x) = max(x, 0)  # Convenience function

"The function B(x, z, a, v) = r(x, a) + β(z) Σ_x′ v(x′) P(x, a, x′)."
function B(x, i_z, a, v, model; d_max=100)
    (; K, c, κ, p, ϕ, z_vals, Q) = model
```

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 the Gaussian AR(1) process

    X_t = ρ X_{t-1} + b + ν W_t.
```

```

include("s_approx.jl")
using LinearAlgebra, Distributions, QuantEcon

function create_sdd_inventory_model(;
            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
    ϕ(d) = (1 - p)^d * p                      # demand pdf
    y_vals = collect(0:K)                     # inventory levels
    mc = tauchen(n_z, ρ, ν)
    z_vals, Q = mc.state_values .+ b, mc.p
    ρL = maximum(abs.(eigvals(z_vals .* Q)))
    @assert  ρL < 1 "Error: ρ(L) ≥ 1."    # check ρ(L) < 1
    return (; K, c, κ, p, ϕ, y_vals, z_vals, Q)
end

m(y) = max(y, 0)  # Convenience function

"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)
    (; K, c, κ, p, ϕ, y_vals, z_vals, Q) = model
```

old public code	new private (jstac) code

old public code

```
    z = z_vals[i_z]
    reward = sum(min(x, d)*ϕ(d) for d in 0:d_max) - c * a - κ * (a > 0)

    cv = 0.0
    for (i_z′, z′) in enumerate(z_vals)

        cv += sum(v[m(x - d) + a, i_z′] * ϕ(d) for d in 0:d_max) * Q[i_z, i_z′]

    end
    return reward + z * cv
end

"The Bellman operator."
function T(v, model)
    (; K, c, κ, p, ϕ, z_vals, Q) = model
    new_v = similar(v)
    for (i_z, z) in enumerate(z_vals)
        for x in 0:K
            Γx = 0:(K - x)
            new_v[x, i_z], _ = findmax(B(x, i_z, a, v, model) for a in Γx)
        end
    end
```

```
"Get a v-greedy policy.  Returns a zero-based array."
function get_greedy(v, model)
    (; K, c, κ, p, ϕ, z_vals, Q) = model
    n_z = length(z_vals)
    σ_star = OffsetArray(zeros(Int32, K+1, n_z), 0:K, 1:n_z)
    for (i_z, z) in enumerate(z_vals)
        for x in 0:K
            Γx = 0:(K - x)
            _, a_idx = findmax(B(x, i_z, a, v, model) for a in Γx)
            σ_star[x, i_z] = Γx[a_idx]
        end
    end
```

```
"Use successive_approx to get v_star and then compute greedy."
function solve_inventory_model(v_init, model)
    (; K, c, κ, p, ϕ, z_vals, Q) = model
    v_star = successive_approx(v -> T(v, model), v_init)
    σ_star = get_greedy(v_star, model)
```

```
# Create an instance of the model and solve it
model = create_sdd_inventory_model()
(; K, c, κ, p, ϕ, z_vals, Q) = model
n_z = length(z_vals)
v_init = OffsetArray(zeros(Float64, K+1, n_z), 0:K, 1:n_z)
println("Solving model.")
v_star, σ_star = solve_inventory_model(v_init, model)
```

```
    for t in 1:(ts_length-1)
        D′ = rand(G)
        X[t+1] = m(X[t] - D′) + σ_star[X[t], i_z[t]]
    end
    return X, z_vals[i_z]
```

```

    ax = axes[1]
    ax.plot(X, label=L"X_t", alpha=0.7)
    ax.set_xlabel(L"t", fontsize=fontsize)
    ax.set_ylabel("inventory", fontsize=fontsize)
    ax.legend(fontsize=fontsize, frameon=false)
    ax.set_ylim(0, maximum(X)+3)
```

```
    ax.plot(r, label=L"r_t", alpha=0.7)
    ax.set_xlabel(L"t", fontsize=fontsize)
    ax.set_ylabel("interest rate", fontsize=fontsize)
    ax.legend(fontsize=fontsize, frameon=false)
    #ax.set_ylim(0, maximum(X)+8)
```

new private (jstac) code

```
    z = z_vals[i_z]
    revenue = sum(min(x, d)*ϕ(d) for d in 0:d_max)
    current_profit =  revenue - c * a - κ * (a > 0)
    cv = 0.0
    for i_z′ in eachindex(z_vals)
        for d in 0:d_max
            cv += v[m(x - d) + a + 1, i_z′] * ϕ(d) * Q[i_z, i_z′]
        end
    end
    return current_profit + z * cv
end

"The Bellman operator."
function T(v, model)
    (; K, c, κ, p, ϕ, y_vals, z_vals, Q) = model
    new_v = similar(v)
    for (i_z, z) in enumerate(z_vals)
        for (i_y, y) in enumerate(y_vals)
            Γy = 0:(K - y)
            new_v[i_y, i_z], _ = findmax(B(y, i_z, a, v, model) for a in Γy)
        end
    end
```

```
"Get a v-greedy policy.  Returns a zero-based array."
function get_greedy(v, model)
    (; K, c, κ, p, ϕ, y_vals, z_vals, Q) = model
    n_z = length(z_vals)
    σ_star = zeros(Int32, K+1, n_z)
    for (i_z, z) in enumerate(z_vals)
        for (i_y, y) in enumerate(y_vals)
            Γy = 0:(K - y)
            _, i_a = findmax(B(y, i_z, a, v, model) for a in Γy)
            σ_star[i_y, i_z] = Γy[i_a]
        end
    end
```

```
"Use successive_approx to get v_star and then compute greedy."
function solve_inventory_model(v_init, model)
    (; K, c, κ, p, ϕ, y_vals, z_vals, Q) = model
    v_star = successive_approx(v -> T(v, model), v_init)
    σ_star = get_greedy(v_star, model)
```

```
# Create an instance of the model and solve it
model = create_sdd_inventory_model()
(; K, c, κ, p, ϕ, y_vals, z_vals, Q) = model
n_z = length(z_vals)
v_init = zeros(Float64, K+1, n_z)
println("Solving model.")
v_star, σ_star = solve_inventory_model(v_init, model)
```

```
    for t in 1:(ts_length-1)
        D′ = rand(G)
        X[t+1] = m(X[t] - D′) + σ_star[X[t] + 1, i_z[t]]
    end
    return X, z_vals[i_z]
```

```

    ax = axes[1]
    ax.plot(X, label="inventory", alpha=0.7)
    ax.set_xlabel(L"t", fontsize=fontsize)

    ax.legend(fontsize=fontsize, frameon=false)
    ax.set_ylim(0, maximum(X)+3)
```

```
    ax.plot(r, label=L"r_t", alpha=0.7)
    ax.set_xlabel(L"t", fontsize=fontsize)

    ax.legend(fontsize=fontsize, frameon=false)
    #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

```julia
using QuantEcon, LinearAlgebra, IterTools

function create_savings_model(; R=1.01, β=0.98, γ=2.5,
                                w_min=0.01, w_max=10.0, w_size=100,
                                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




## == Functions for modified OPI == ##

"D(w, z, ϵ, w′, g) = u(w + z + ϵ - w′/R) + β g(z, w′)."
@inline function D(i, j, k, l, g, 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)
    return c > 0 ? c^(1-γ)/(1-γ) + β * g[j, l] : -Inf
end

"Compute a g-greedy policy."
function get_g_greedy(g, 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(D(i, j, k, l, g, model) for l in w_idx)
    end
"The modified policy operator."
function R_σ(g, 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))
    g_new = similar(g)
```

new private (jstac) code

```julia
using QuantEcon, LinearAlgebra, IterTools

function create_savings_model(; β=0.98, γ=2.5,
                                w_min=0.01, w_max=20.0, w_size=100,
                                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



## == Functions for modified OPI == ##

"D(w, y, η, w′, g) = u(w + y - w′/η) + β g(y, w′)."
@inline function D(i, j, k, l, g, 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)
    return c > 0 ? u(c) + β * g[j, l] : -Inf
end

"Compute a g-greedy policy."
function get_g_greedy(g, 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(D(i, j, k, l, g, model) for l in w_idx)
    end
"The modified policy operator."
function R_σ(g, σ, 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))
    g_new = similar(g)
```

old public code	new private (jstac) code

old public code

```
    for (j, i′) in product(z_idx, w_idx)  # j -> z, i′ -> w′
        out = 0.0
        for j′ in z_idx # j′ -> z′
            for k′ in ϵ_idx # k′ -> ϵ′
                # D(w′, z′, ϵ′, σ(w′, z′, ϵ′), g)
                out += D(i′, j′, k′, σ[i′, j′, k′], g, model) *
                        Q[j, j′] * ϕ[k′]
```

new private (jstac) code

```
    for (j, i′) in product(y_idx, w_idx)  # j indexes y, i′ indexes w′
        out = 0.0
        for j′ in y_idx                    # j′ indexes y′
            for k′ in η_idx                # k′ indexes η′

                out += D(i′, j′, k′, σ[i′, j′, k′], g, model) *
                        Q[j, j′] * ϕ[k′]
```

old public code

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

new private (jstac) code

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

old public code

```


# Plots


using PyPlot
```

new private (jstac) code

```


# == 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))
    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
```

old public code

```


function plot_contours(; savefig=false,
                         figname="../figures/modified_opt_savings_1.pdf")

    model = create_savings_model()
    (; β, R, γ, ϵ_grid, ϕ, w_grid, z_grid, Q) = model
    σ_star = mod_opi(model)

    fig, axes = plt.subplots(2, 1, figsize=(10, 8))
    z_idx, ϵ_idx = eachindex(z_grid), eachindex(ϵ_grid)
    H = zeros(length(z_grid), length(ϵ_grid))

    w_indices = (1, length(w_grid))
```

new private (jstac) code

```


function plot_contours(; savefig=false,
                         figname="../figures/modified_opt_savings_1.pdf")

    model = create_savings_model()
    (; β, γ, η_grid, ϕ, w_grid, y_grid, Q) = model
    σ_star = optimistic_policy_iteration(model)

    fig, axes = plt.subplots(2, 1, figsize=(10, 8))
    y_idx, η_idx = eachindex(y_grid), eachindex(η_grid)
    H = zeros(length(y_grid), length(η_grid))

    w_indices = (1, length(w_grid))
```

old public code

```
    for (ax, w_idx, title) in zip(axes, w_indices, titles)

        for (i_z, i_ϵ) in product(z_idx, ϵ_idx)
            w, z, ϵ = w_grid[w_idx], z_grid[i_z], ϵ_grid[i_ϵ]
            H[i_z, i_ϵ] = w_grid[σ_star[w_idx, i_z, i_ϵ]]
        end

        cs1 = ax.contourf(z_grid, ϵ_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")

        ax.set_title(title, fontsize=fontsize)
        ax.set_xlabel(L"z", fontsize=fontsize)
        ax.set_ylabel(L"\varepsilon", fontsize=fontsize)
    end
```

new private (jstac) code

```
    for (ax, w_idx, title) in zip(axes, w_indices, titles)

        for (i_y, i_ϵ) in product(y_idx, η_idx)
            w, y, η = w_grid[w_idx], y_grid[i_y], η_grid[i_ϵ]
            H[i_y, i_ϵ] = w_grid[σ_star[w_idx, i_y, i_ϵ]] / (w+y)
        end

        cs1 = ax.contourf(y_grid, η_grid, transpose(H), alpha=0.5)


        plt.colorbar(cs1, ax=ax) #, format="%.6f")

        ax.set_title(title, fontsize=fontsize)
        ax.set_xlabel(L"y", fontsize=fontsize)
        ax.set_ylabel(L"\varepsilon", fontsize=fontsize)
    end
```

old public code

```


```

new private (jstac) code

```


function plot_policies(; savefig=false,
                         figname="../figures/modified_opt_savings_2.pdf")

    model = create_savings_model()
    (; β, γ, η_grid, ϕ, w_grid, y_grid, Q) = model
    σ_star = mod_opi(model)
    y_bar = floor(Int, length(y_grid) / 2)  # Index of mid-point of y_grid

    fig, ax = plt.subplots(figsize=(9, 5.2))
    ax.plot(w_grid, w_grid, "k--", label=L"45")

    for (i, η) in enumerate(η_grid)
        label = L"\sigma^*" * " at " * L"\eta = " * "$η"
        ax.plot(w_grid, w_grid[σ_star[:, y_bar, i]], label=label)
    end
    ax.legend(fontsize=fontsize)
    plt.show()

```

old public code	new private (jstac) code

```
    plt.tight_layout()
    if savefig
        fig.savefig(figname)
    end
    plt.show()
end


function plot_time_series(; m=2_000,
                            savefig=false,
                            figname="../figures/modified_opt_savings_ts.pdf")

    w_series = simulate_wealth(m)
    fig, ax = plt.subplots(figsize=(9, 5.2))
    ax.plot(w_series, label=L"w_t")
    ax.legend(fontsize=fontsize)
    ax.set_xlabel("time", fontsize=fontsize)
    plt.show()
    if savefig
        fig.savefig(figname)
    end

end

function plot_histogram(; m=1_000_000,
                            savefig=false,
                            figname="../figures/modified_opt_savings_hist.pdf")

    w_series = simulate_wealth(m)
    g = round(gini(sort(w_series)), digits=2)
    fig, ax = plt.subplots(figsize=(9, 5.2))
    ax.hist(w_series, bins=40, density=true)
    ax.set_xlabel("wealth", fontsize=fontsize)
    ax.text(15, 0.7, "Gini = $g", fontsize=fontsize)
    plt.show()
    if savefig
        fig.savefig(figname)
    end

end


function plot_lorenz(; m=1_000_000,
                            savefig=false,
                            figname="../figures/modified_opt_savings_lorenz.pdf")

    w_series = simulate_wealth(m)
    (; F, L) = lorenz(sort(w_series))

    fig, ax = plt.subplots(figsize=(9, 5.2))
    ax.plot(F, F, label="Lorenz curve, equality")
    ax.plot(F, L, label="Lorenz curve, wealth distribution")
    ax.legend()
    plt.show()
    if savefig
        fig.savefig(figname)
    end

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
	`end`		`end`
t	`ax.plot(xgrid, xgrid, "k--", lw=1, alpha=0.7, label=L"45")`	t	`ax.plot(xgrid, xgrid, "k--", lw=1, alpha=0.7, label=L"45^{\circ}")`
	`ax.plot(xgrid, T1, "k-", lw=1)`		`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

```
"""
Computes the approximate fixed point of T via successive approximation.


"""
function successive_approx(T,                       # Operator (callable)
                           u_0;                     # Initial condition
                           tolerance=1e-6,          # Error tolerance
                           max_iter=10_000,         # Max iteration bound
                           print_step=25)           # Print at multiples
    u = u_0
    error = Inf

    while (error > tolerance) & (k <= max_iter)

        u_new = T(u)
        error = maximum(abs.(u_new - u))

        if k % print_step == 0
            println("Completed iteration $k with error $error.")
        end

        u = u_new
        k += 1
        println("Terminated successfully in $k iterations.")
    else
        println("Termination Warning: Error is greater than tolerance.")
    end

```

new private (jstac) code

```
"""
Computes an approximate fixed point of a given operator T
via successive approximation.

"""
function successive_approx(T,                  # operator (callable)
                           u_0;                # initial condition
                           tolerance=1e-6,     # error tolerance
                           max_iter=10_000,    # max iteration bound
                           print_step=25)      # print at multiples
    u = u_0
    error = Inf

    while (error > tolerance) & (k <= max_iter)

        u_new = T(u)
        error = maximum(abs.(u_new - u))

        if k % print_step == 0
            println("Completed iteration $k with error $error.")
        end

        u = u_new
        k += 1
        println("Terminated successfully in $k iterations.")
    else
        println("Warning: hit iteration bound.")
    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