UNIVERSIDAD NACIONAL DEL ALTIPLANO

Ingeniería Estadística e Informática

INVENTARIOS CON JULIA

Modelos de Gestión Óptima Computacional

EOQ • Descuentos • Probabilísticos • Optimización

Implementación Computacional con Lenguaje Julia

AUTOR

Etzel Yuliza Peralta López

Material de Estudio Computacional con Julia

CONTENIDO DEL CAPÍTULO

Índice general

Capítulo 1

Inventarios con Julia

INTRODUCCIÓN

Los modelos de inventarios constituyen una herramienta fundamental en la gestión empresarial moderna. En este capítulo implementaremos estos modelos utilizando el lenguaje de programación Julia, que ofrece excelente rendimiento para cálculos matemáticos y optimización.
Julia combina la facilidad de uso de Python con la velocidad de C++, siendo ideal para resolver problemas complejos de optimización en inventarios.

1.1. Configuración del Entorno Julia

PAQUETES NECESARIOS

```julia
# Instalacion de paquetes necesarios
using Pkg
Pkg.add(["Optim", "Distributions", "Plots",
"DataFrames", "Printf"])

# Importar librerias
using Optim
using Distributions
using Plots
using DataFrames
using Printf
using Statistics
```

1.2. Modelo EOQ Básico

IMPLEMENTACIÓN EOQ

```
"""
Modelo EOQ (Economic Order Quantity) basico
Parametros:
- D: Demanda anual
- S: Costo de pedido por orden
- H: Costo de almacenamiento por unidad por ñao
"""
function eoq_basic(D, S, H)
Q_star = sqrt(2 * D * S / H)
total_cost = sqrt(2 * D * S * H)
orders_per_year = D / Q_star
time_between_orders = Q_star / D * 365 # dias

return (
Q_star = Q_star,
total_cost = total_cost,
orders_per_year = orders_per_year,
time_between_orders = time_between_orders
)
end

# Funcion para mostrar resultados
function print_eoq_results(result, title="Resultados EOQ")
println("="^50)
println(title)
println("="^50)
@printf("Cantidad optima (Q*): %.2f unidades\n",
result.Q_star)
@printf("Costo total anual: \$%.2f\n",
result.total_cost)
@printf("Pedidos por ñao: %.2f\n",
result.orders_per_year)
@printf("Tiempo entre pedidos: %.2f dias\n",
result.time_between_orders)
println("="^50)
end
```

1.2.1. Ejercicio Resuelto: EOQ con Pedidos Retrasados

PROBLEMA

Clínica de Optometría: Una clínica vende 10,000 monturas anuales. El proveedor cobra \$15 por unidad, con costo de pedido de \$50. El costo de déficit es \$15 por montura/año. El costo de retención anual es 30 % del costo de compra.

SOLUCIÓN EN JULIA

```
"""
EOQ con pedidos retrasados (backorders permitidos)
Parametros adicionales:
- B: Costo de deficit por unidad por ñao
"""
function eoq_backorders(D, S, H, B)
# Cantidad optima con backorders
Q_star = sqrt(2 * D * S / H) * sqrt((H + B) / B)

# Inventario maximo
I_max = Q_star * (B / (H + B))

# Deficit maximo
deficit_max = Q_star - I_max

# Costo total
total_cost = sqrt(2 * D * S * H * B / (H + B))

# Pedidos por ñao
orders_per_year = D / Q_star

return (
Q_star = Q_star,
I_max = I_max,
deficit_max = deficit_max,
total_cost = total_cost,
orders_per_year = orders_per_year
)
end

# Datos del problema de la clinica
D = 10000 # monturas/ñao
S = 50    # $ por pedido
unit_cost = 15 # $ por unidad
H = 0.30 * unit_cost # $ por unidad/ñao
B = 15    # $ por unidad/ñao (costo de deficit)

# Resolver el problema
resultado_clinica = eoq_backorders(D, S, H, B)

println("CLINICA DE OPTOMETRIA - EOQ CON BACKORDERS")
println("="^60)
@printf("Cantidad optima (Q*): %.2f monturas\n",
resultado_clinica.Q_star)
@printf("Inventario maximo: %.2f monturas\n",
resultado_clinica.I_max)
@printf("Deficit maximo: %.2f monturas\n",
resultado_clinica.deficit_max)
@printf("Costo total anual: \$%.2f\n",
resultado_clinica.total_cost)
@printf("Pedidos por ñao: %.2f\n",
resultado_clinica.orders_per_year)
```

1.3. Modelo de Lote de Producción

MODELO DE PRODUCCIÓN

```
"""
Modelo de lote de produccion (Production Lot Size)
Parametros:
- D: Demanda anual
- S: Costo de preparacion
- H: Costo de almacenamiento por unidad por ñao
- P: Tasa de produccion anual
"""
function production_lot_model(D, S, H, P)
if P <= D
error("La tasa de produccion debe ser mayor que la demanda")
end

# Cantidad optima de produccion
Q_star = sqrt(2 * D * S / H) * sqrt(P / (P - D))

# Inventario maximo
I_max = Q_star * (1 - D/P)

# Tiempo de produccion
t_production = Q_star / P * 365 # dias

# Tiempo de ciclo
t_cycle = Q_star / D * 365 # dias

# Costo total
total_cost = sqrt(2 * D * S * H * (P - D) / P)

# Corridas por ñao
runs_per_year = D / Q_star

return (
Q_star = Q_star,
I_max = I_max,
t_production = t_production,
t_cycle = t_cycle,
total_cost = total_cost,
runs_per_year = runs_per_year
)
end

# Ejemplo de aplicacion: Fabrica de componentes electronicos
D_prod = 8000  # unidades/ñao
S_prod = 100   # $ por preparacion
H_prod = 2.5   # $ por unidad/ñao
P_prod = 12000 # unidades/ñao

resultado_prod = production_lot_model(D_prod, S_prod, H_prod, P_prod)

println("MODELO DE LOTE DE PRODUCCION")
println("="^50)
@printf("Cantidad optima (Q*): %.2f unidades\n",
resultado_prod.Q_star)
@printf("Inventario maximo: %.2f unidades\n",
resultado_prod.I_max)
@printf("Tiempo de produccion: %.2f dias\n",
resultado_prod.t_production)
@printf("Tiempo de ciclo: %.2f dias\n",
resultado_prod.t_cycle)
@printf("Costo total anual: \$%.2f\n",
resultado_prod.total_cost)
@printf("Corridas por ñao: %.2f\n",
```

```
    resultado_prod.runs_per_year)

    # Analisis del ciclo de produccion
    println("\nANALISIS DEL CICLO:")
    @printf("Tiempo produciendo: %.1f%% del ciclo\n",
    resultado_prod.t_production/resultado_prod.t_cycle * 100)
    @printf("Tiempo sin producir: %.1f%% del ciclo\n",
    (1 - resultado_prod.t_production/resultado_prod.t_cycle) * 100)
```

1.3.1. Comparación: EOQ vs Lote de Producción

⚖ COMPARACIÓN DE MODELOS

```
# Comparacion entre EOQ clasico y modelo de produccion
function compare_eoq_production(D, S, H, P)
# EOQ clasico (como si fuera compra instantanea)
Q_eoq = sqrt(2 * D * S / H)
cost_eoq = sqrt(2 * D * S * H)

# Modelo de produccion
result_prod = production_lot_model(D, S, H, P)

# Calcular ahorros
savings = cost_eoq - result_prod.total_cost
savings_percent = (savings / cost_eoq) * 100

println("COMPARACION: EOQ vs LOTE DE PRODUCCION")
println("="^60)
@printf("EOQ Clasico (Q): %.2f unidades\n", Q_eoq)
@printf("Lote Produccion (Q*): %.2f unidades\n", result_prod.Q_star)
@printf("Diferencia en cantidad: %.2f unidades\n",
result_prod.Q_star - Q_eoq)
println("-"^60)
@printf("Costo EOQ: \$%.2f\n", cost_eoq)
@printf("Costo Produccion: \$%.2f\n", result_prod.total_cost)
@printf("Ahorro anual: \$%.2f (%.1f%%)\n", savings, savings_percent)

return (
eoq = Q_eoq,
production = result_prod.Q_star,
savings = savings,
savings_percent = savings_percent
)
end

# Ejecutar comparacion con datos del ejemplo
comparacion = compare_eoq_production(D_prod, S_prod, H_prod, P_prod)
```

1.4. Modelo con Tiempo de Entrega

MODELO CON LEAD TIME

```
"""
EOQ con tiempo de entrega
Parametros adicionales:
- L: Tiempo de entrega en dias
- working_days: Dias laborables por ñao
"""
function eoq_lead_time(D, S, H, L, working_days=365)
# EOQ basico
Q_star = sqrt(2 * D * S / H)

# Punto de reorden
daily_demand = D / working_days
R = daily_demand * L

# Tiempo entre pedidos
time_between_orders = Q_star / daily_demand

# Numero de pedidos por ñao
orders_per_year = D / Q_star

# Costo total
total_cost = sqrt(2 * D * S * H)

return (
Q_star = Q_star,
R = R,
daily_demand = daily_demand,
time_between_orders = time_between_orders,
orders_per_year = orders_per_year,
total_cost = total_cost
)
end

# Ejemplo: ICR LLC
D_icr = 19500  # componentes/ñao
S_icr = 50     # $ por pedido
unit_cost_icr = 22.00
H_icr = 0.02 * 12 * unit_cost_icr # 5.28 $/componente/ñao
L_icr = 4      # dias
working_days_icr = 307

resultado_icr = eoq_lead_time(D_icr, S_icr, H_icr, L_icr,
    ↪ working_days_icr)

println("ICR LLC - EOQ CON TIEMPO DE ENTREGA")
println("="^60)
@printf("Cantidad optima (Q*): %.0f componentes\n",
resultado_icr.Q_star)
@printf("Punto de reorden (R): %.0f componentes\n",
```

```
    resultado_icr.R)
    @printf("Demanda diaria: %.2f componentes/dia\n",
    resultado_icr.daily_demand)
    @printf("Tiempo entre pedidos: %.2f dias\n",
    resultado_icr.time_between_orders)
    @printf("Pedidos por ñao: %.1f\n",
    resultado_icr.orders_per_year)
    @printf("Costo total anual: \$%.2f\n",
    resultado_icr.total_cost)
```

1.5. Modelo con Descuentos por Cantidad

DESCUENTOS POR CANTIDAD

```
    """
    Modelo EOQ con descuentos por cantidad
    """
    struct DiscountTier
    min_qty::Float64
    max_qty::Float64
    unit_price::Float64
    end

    function eoq_quantity_discount(D, S, i,
        ↪ discount_tiers::Vector{DiscountTier})
    results = []

    for tier in discount_tiers
    H = i * tier.unit_price # Costo de almacenamiento

    # EOQ para este nivel de precio
    Q_eoq = sqrt(2 * D * S / H)

    # Verificar si Q_eoq esta en el rango valido
    if tier.min_qty <= Q_eoq <= tier.max_qty
    Q_feasible = Q_eoq
    else
    # Usar el punto de quiebre mas cercano
    Q_feasible = tier.min_qty
    end

    # Calcular costo total
    annual_purchase_cost = D * tier.unit_price
    annual_ordering_cost = (D / Q_feasible) * S
    annual_holding_cost = (Q_feasible / 2) * H
    total_cost = annual_purchase_cost + annual_ordering_cost +
        ↪ annual_holding_cost

    push!(results, (
```

```
    tier = tier,
    Q_eoq = Q_eoq,
    Q_feasible = Q_feasible,
    total_cost = total_cost,
    purchase_cost = annual_purchase_cost,
    ordering_cost = annual_ordering_cost,
    holding_cost = annual_holding_cost
    ))
    end

    # Encontrar la opcion de menor costo
    min_cost_idx = argmin([r.total_cost for r in results])
    optimal_result = results[min_cost_idx]

    return (
    optimal = optimal_result,
    all_options = results
    )
    end

    # Datos del problema MBI Computadoras
    D_mbi = 5200 # discos/ñao
    S_mbi = 50   # $ por pedido
    i_mbi = 0.20 # 20% tasa de costo de mantener

    # Definir estructura de descuentos
    descuentos_mbi = [
    DiscountTier(1, 99, 100.0),
    DiscountTier(100, 499, 95.0),
    DiscountTier(500, Inf, 90.0)
    ]

    resultado_mbi = eoq_quantity_discount(D_mbi, S_mbi, i_mbi, descuentos_mbi)

    println("MBI COMPUTADORAS - DESCUENTOS POR CANTIDAD")
    println("="^70)

    println("ANALISIS DE TODAS LAS OPCIONES:")
    for (i, option) in enumerate(resultado_mbi.all_options)
    tier = option.tier
    println("\nCategoria $i: $(tier.min_qty) - $(tier.max_qty == Inf ?
        ↪ "Infinito" : tier.max_qty)")
    @printf(" Precio unitario: \$%.2f\n", tier.unit_price)
    @printf(" EOQ calculado: %.2f\n", option.Q_eoq)
    @printf(" Cantidad factible: %.0f\n", option.Q_feasible)
    @printf(" Costo total: \$%.2f\n", option.total_cost)
    end

    println("\n" * "="^70)
    println("SOLUCION OPTIMA:")
    optimal = resultado_mbi.optimal
    @printf("Cantidad optima: %.0f unidades\n", optimal.Q_feasible)
    @printf("Precio unitario: \$%.2f\n", optimal.tier.unit_price)
    @printf("Costo total anual: \$%.2f\n", optimal.total_cost)
```

1.6. Modelos Probabilísticos

MODELOS PROBABILÍSTICOS

```
"""
Modelo de demanda discreta (newsvendor problem)
"""
function newsvendor_discrete(demand_probs::Dict, c, h, p)
# c: costo de compra
# h: costo de almacenamiento
# p: costo de escasez (perdida por faltante)

# Calcular proporcion critica
critical_ratio = (p - c) / (p + h)

# Ordenar demandas
demands = sort(collect(keys(demand_probs)))

# Calcular probabilidades acumuladas
cum_prob = 0.0
optimal_q = demands[1]

println("Analisis del modelo newsvendor:")
println("Proporcion critica: $(round(critical_ratio, digits=4))")
println("\nTabla de probabilidades:")
println("Demanda\tProb\tProb_Acum")

for d in demands
cum_prob += demand_probs[d]
println("$d\t$(demand_probs[d])\t$(round(cum_prob, digits=3))")

if cum_prob >= critical_ratio && optimal_q == demands[1]
optimal_q = d
end
end

return (
optimal_quantity = optimal_q,
critical_ratio = critical_ratio,
demands = demands,
probabilities = demand_probs
)
end

# Ejemplo: Producto con demanda incierta
demanda_probs = Dict(
0 => 0.05, 1 => 0.10, 2 => 0.10, 3 => 0.20, 4 => 0.25,
5 => 0.15, 6 => 0.05, 7 => 0.05, 8 => 0.05
```

```
    )

    c_producto = 10 # costo de compra
    h_producto = 1  # costo de almacenamiento
    p_producto = 15 # costo de escasez

    resultado_newsvendor = newsvendor_discrete(demanda_probs, c_producto,
    h_producto, p_producto)

    println("\nPRODUCTO CON DEMANDA INCIERTA")
    println("="^50)
    @printf("Cantidad optima a ordenar: %d unidades\n",
    resultado_newsvendor.optimal_quantity)
```

1.6.1. Modelo Probabilístico de Chicago Cheese

</> CHICAGO CHEESE - JULIA

```
    """
    Modelo probabilistico de un periodo - Chicago Cheese
    """
    function chicago_cheese_model()
    # Datos del problema
    selling_price = 100 # precio de venta por caja
    production_cost = 75 # costo de produccion por caja
    salvage_value = 50 # valor de salvamento por caja

    # Costos marginales
    Co = selling_price - production_cost # costo de oportunidad (falta)
    Cu = production_cost - salvage_value # costo de exceso

    # Proporcion critica
    critical_ratio = Co / (Co + Cu)

    # Distribucion de demanda
    demands = [10, 11, 12, 13, 14]
    probabilities = [0.2, 0.3, 0.2, 0.2, 0.1]
    cum_probabilities = cumsum(probabilities)

    println("CHICAGO CHEESE - ANALISIS DE PRODUCCION")
    println("="^60)
    @printf("Precio de venta: \$%d por caja\n", selling_price)
    @printf("Costo de produccion: \$%d por caja\n", production_cost)
    @printf("Valor de salvamento: \$%d por caja\n", salvage_value)
    println("-"^60)
    @printf("Costo de oportunidad (Co): \$%d\n", Co)
    @printf("Costo de exceso (Cu): \$%d\n", Cu)
    @printf("Proporcion critica: %.3f\n", critical_ratio)

    println("\nDistribucion de demanda:")
```

```
    println("Demanda\tProb\tProb_Acum")

    optimal_production = demands[1]
    for (i, d) in enumerate(demands)
    println("$d\t$(probabilities[i])\t$(cum_probabilities[i])")

    if cum_probabilities[i] >= critical_ratio && optimal_production ==
        ↪ demands[1]
    optimal_production = d
    end
    end

    println("\n" * "="^60)
    println("RECOMENDACION:")
    println("Producir 11 o 12 cajas de queso")
    println("Como la proporcion critica es 0.5 y P(D <= 11) = 0.5")
    println("La decision optima puede ser 11 o 12 cajas")

    return (
    optimal_production = optimal_production,
    critical_ratio = critical_ratio,
    Co = Co,
    Cu = Cu
    )
    end

    # Ejecutar analisis
    resultado_chicago = chicago_cheese_model()
```

1.7. Modelo de Producción con Pedidos Atrasados

PRODUCCIÓN CON BACKORDERS

```
    """
    Modelo de produccion con pedidos atrasados permitidos
    Combina el modelo de lote de produccion con la posibilidad de
    tener demanda pendiente (backorders)

    Parametros:
    - D: Demanda anual
    - S: Costo de preparacion por corrida
    - H: Costo de almacenamiento por unidad por ñao
    - P: Tasa de produccion anual
    - B: Costo de deficit por unidad por ñao
    """
    function production_backorder_model(D, S, H, P, B)
    if P <= D
    error("La tasa de produccion debe ser mayor que la demanda")
    end
```

```
    # PASO 1: Calcular la cantidad optima de produccion
    # Formula: Q* = sqrt(2DS/H) * sqrt((H+B)/B) * sqrt(P/(P-D))
    Q_base = sqrt(2 * D * S / H) # EOQ basico
    factor_backorder = sqrt((H + B) / B) # Factor por backorders
    factor_production = sqrt(P / (P - D)) # Factor por produccion

    Q_star = Q_base * factor_backorder * factor_production

    # PASO 2: Calcular el inventario maximo
    # Durante la produccion, el inventario maximo sera menor que Q*
    # porque parte de la produccion satisface la demanda inmediata
    I_max = Q_star * (B / (H + B)) * (1 - D/P)

    # PASO 3: Calcular el deficit maximo
    # Es la cantidad de demanda pendiente antes de iniciar produccion
    deficit_max = Q_star * (H / (H + B))

    # PASO 4: Tiempos del ciclo
    # Tiempo total del ciclo
    t_cycle = Q_star / D * 365 # dias

    # Tiempo de produccion
    t_production = Q_star / P * 365 # dias

    # Tiempo sin producir
    t_no_production = t_cycle - t_production

    # PASO 5: Costo total anual
    # Formula: CT = sqrt(2*D*S*H*B*(P-D)/(P*(H+B)))
    total_cost = sqrt(2 * D * S * H * B * (P - D) / (P * (H + B)))

    # PASO 6: Numero de corridas por ñao
    runs_per_year = D / Q_star

    return (
    Q_star = Q_star,
    I_max = I_max,
    deficit_max = deficit_max,
    t_cycle = t_cycle,
    t_production = t_production,
    t_no_production = t_no_production,
    total_cost = total_cost,
    runs_per_year = runs_per_year,
    # Informacion adicional para analisis
    Q_base = Q_base,
    factor_backorder = factor_backorder,
    factor_production = factor_production
    )
    end
```

1.7.1. Ejercicio Resuelto: Empresa Ladrillera

PROBLEMA EMPRESA LADRILLERA

Empresa Ladrillera: Una empresa tiene demanda anual de 210,000 ladrillos. Los produce a ritmo mensual de 37,500 ladrillos. Costo de preparación $450 por corrida. Costo anual de almacenamiento 1.2 $/unidad. Costo anual por demanda pendiente 0.5 $/unidad. Considerar 360 días/año.

SOLUCIÓN PASO A PASO

```
# DATOS DEL PROBLEMA
D_ladrillos = 210000  # ladrillos/ñao
P_ladrillos = 37500 * 12 # 37,500/mes = 450,000/ñao
S_ladrillos = 450     # $ por corrida
H_ladrillos = 1.2     # $ por ladrillo/ñao
B_ladrillos = 0.5     # $ por ladrillo/ñao (deficit)

println("EMPRESA LADRILLERA - PRODUCCION CON BACKORDERS")
println("="^70)
println("DATOS DEL PROBLEMA:")
@printf("Demanda anual (D): %d ladrillos\n", D_ladrillos)
@printf("Tasa de produccion (P): %d ladrillos/ñao\n", P_ladrillos)
@printf("Costo de preparacion (S): \$%.0f\n", S_ladrillos)
@printf("Costo de almacenamiento (H): \$%.1f/ladrillo/ñao\n", H_ladrillos)
@printf("Costo de deficit (B): \$%.1f/ladrillo/ñao\n", B_ladrillos)

# RESOLVER EL PROBLEMA
resultado_ladrillos = production_backorder_model(
D_ladrillos, S_ladrillos, H_ladrillos, P_ladrillos, B_ladrillos)

println("\n" * "="^70)
println("ANALISIS PASO A PASO:")
println("-"^70)

println("PASO 1: Factores de calculo")
@printf("EOQ basico: %.2f ladrillos\n", resultado_ladrillos.Q_base)
@printf("Factor backorder: %.4f\n", resultado_ladrillos.factor_backorder)
@printf("Factor produccion: %.4f\n",
    ↪ resultado_ladrillos.factor_production)

println("\nPASO 2: Resultados principales")
@printf("Cantidad optima (Q*): %.2f ladrillos\n",
    ↪ resultado_ladrillos.Q_star)
@printf("Inventario maximo: %.2f ladrillos\n", resultado_ladrillos.I_max)
@printf("Deficit maximo: %.2f ladrillos\n",
    ↪ resultado_ladrillos.deficit_max)

println("\nPASO 3: Analisis temporal")
@printf("Tiempo de ciclo: %.2f dias\n", resultado_ladrillos.t_cycle)
@printf("Tiempo de produccion: %.2f dias\n",
    ↪ resultado_ladrillos.t_production)
```

```
    @printf("Tiempo sin producir: %.2f dias\n",
        ↪ resultado_ladrillos.t_no_production)

    println("\nPASO 4: Analisis economico")
    @printf("Costo total anual: \$%.2f\n", resultado_ladrillos.total_cost)
    @printf("Corridas por ñao: %.2f\n", resultado_ladrillos.runs_per_year)

    # COMPARACION CON MODELO SIN BACKORDERS
    resultado_sin_backorders = production_lot_model(
    D_ladrillos, S_ladrillos, H_ladrillos, P_ladrillos)

    ahorro = resultado_sin_backorders.total_cost -
        ↪ resultado_ladrillos.total_cost
    ahorro_porcentaje = (ahorro / resultado_sin_backorders.total_cost) * 100

    println("\nPASO 5: Comparacion con modelo sin backorders")
    @printf("Costo sin backorders: \$%.2f\n",
        ↪ resultado_sin_backorders.total_cost)
    @printf("Costo con backorders: \$%.2f\n", resultado_ladrillos.total_cost)
    @printf("Ahorro anual: \$%.2f (%.2f%%)\n", ahorro, ahorro_porcentaje)
```

1.8. Modelo EOQ con Desabastecimientos Planeados

⚠ EOQ CON ESCASEZ PLANEADA

```
    """
    Modelo EOQ con desabastecimientos planeados
    Permite escasez planificada para reducir costos totales

    Parametros:
    - D: Demanda anual
    - S: Costo de pedido
    - H: Costo de almacenamiento por unidad por ñao
    - B: Costo de desabastecimiento por unidad por ñao
    - L: Tiempo de entrega (opcional)
    """
    function eoq_planned_shortage(D, S, H, B, L=0)

    # PASO 1: Verificar que el modelo tenga sentido economico
    if B <= H
    println("ADVERTENCIA: El costo de escasez (B=$B) debe ser mayor")
    println("que el costo de almacenamiento (H=$H) para que tenga")
    println("sentido economico permitir desabastecimientos.")
    end

    # PASO 2: Calcular cantidad optima de pedido
    # Formula: Q* = sqrt(2*D*S/H) * sqrt((H+B)/B)
    Q_eoq_basico = sqrt(2 * D * S / H)
    factor_escasez = sqrt((H + B) / B)
```

```
    Q_star = Q_eoq_basico * factor_escasez

    # PASO 3: Calcular inventario maximo
    # Es la cantidad maxima en stock antes de agotar
    I_max = Q_star * (B / (H + B))

    # PASO 4: Calcular desabastecimiento maximo
    # Es la cantidad maxima de demanda pendiente
    S_max = Q_star * (H / (H + B))

    # PASO 5: Calcular tiempos del ciclo
    tiempo_ciclo = Q_star / D * 365 # dias
    tiempo_con_inventario = I_max / D * 365 # dias
    tiempo_desabastecido = S_max / D * 365 # dias

    # PASO 6: Calcular costos
    # Costo total: CT = sqrt(2*D*S*H*B/(H+B))
    costo_total = sqrt(2 * D * S * H * B / (H + B))

    # Costo de pedidos
    costo_pedidos = (D / Q_star) * S

    # Costo de almacenamiento
    costo_almacenamiento = (I_max^2 / (2 * Q_star)) * H

    # Costo de desabastecimiento
    costo_desabastecimiento = (S_max^2 / (2 * Q_star)) * B

    # PASO 7: Punto de reorden (si hay tiempo de entrega)
    if L > 0
    demanda_diaria = D / 365
    R = demanda_diaria * L - S_max # Punto de reorden negativo!
    else
    R = -S_max # Sin tiempo de entrega
    end

    # PASO 8: Numero de pedidos por ñaoñ
    pedidos_por_ao = D / Q_star

    return (
    # Resultados principales
    Q_star = Q_star,
    I_max = I_max,
    S_max = S_max,
    R = R,

    # Analisis temporal
    tiempo_ciclo = tiempo_ciclo,
    tiempo_con_inventario = tiempo_con_inventario,
    tiempo_desabastecido = tiempo_desabastecido,

    # Analisis de costos
    costo_total = costo_total,
    costo_pedidos = costo_pedidos,
```

```
    costo_almacenamiento = costo_almacenamiento,
    costo_desabastecimiento = costo_desabastecimiento,

    # Metricas operativasñ
    pedidos_por_ao = ñpedidos_por_ao,

    # Para comparaciones
    Q_eoq_basico = Q_eoq_basico,
    factor_escasez = factor_escasez
    )
    end
```

1.8.1. Ejercicio Resuelto: Producto con Escasez Planeada

PROBLEMA CON DESABASTECIMIENTO

Producto con Escasez: Un administrador considera un producto con demanda de 800 unidades anuales. Costo de mantener 0.25€/unidad/mes. Costo de pedido 150€. Costo de no tener unidad 20€/unidad/año. Tiempo de entrega 1 mes.

SOLUCIÓN DETALLADA

```
# DATOS DEL PROBLEMA
D_producto = 800        # unidades/ñao
S_producto = 150        # € por pedido
H_producto = 0.25 * 12  # €0.25/mes = €3/ñao por unidad
B_producto = 20         # € por unidad/ñao
L_producto = 1/12       # 1 mes = 1/12 ñaos

println("PRODUCTO CON ESCASEZ PLANEADA")
println("="^60)
println("ANALISIS PASO A PASO:")
println("-"^60)

# PASO 1: Mostrar datos y verificar viabilidad
println("PASO 1: Datos del problema")
@printf("Demanda anual: %d unidades\n", D_producto)
@printf("Costo de pedido: %.0f €\n", S_producto)
@printf("Costo de almacenamiento: %.2f €/unidad/ñao\n", H_producto)
@printf("Costo de desabastecimiento: %.0f €/unidad/ñao\n", B_producto)
@printf("Tiempo de entrega: %.3f ñaos (1 mes)\n", L_producto)

if B_producto > H_producto
println("VIABLE: Costo escasez > Costo almacenamiento")
else
println("REVISAR: Costo escasez <= Costo almacenamiento")
end

# PASO 2: Resolver con el modelo
```

```
    resultado_escasez = eoq_planned_shortage(
    D_producto, S_producto, H_producto, B_producto, L_producto)

    println("\nPASO 2: Calculos principales")
    @printf("EOQ basico (sin escasez): %.2f unidades\n",
    resultado_escasez.Q_eoq_basico)
    @printf("Factor de escasez: %.4f\n", resultado_escasez.factor_escasez)
    @printf("Cantidad optima (Q*): %.2f unidades\n", resultado_escasez.Q_star)

    println("\nPASO 3: Niveles de inventario")
    @printf("Inventario maximo: %.2f unidades\n", resultado_escasez.I_max)
    @printf("Desabastecimiento maximo: %.2f unidades\n",
        ↪ resultado_escasez.S_max)
    @printf("Punto de reorden: %.2f unidades\n", resultado_escasez.R)

    println("\nPASO 4: Analisis temporal (dias)")
    @printf("Tiempo de ciclo: %.2f dias\n", resultado_escasez.tiempo_ciclo)
    @printf("Tiempo con inventario: %.2f dias\n",
    resultado_escasez.tiempo_con_inventario)
    @printf("Tiempo desabastecido: %.2f dias\n",
    resultado_escasez.tiempo_desabastecido)

    println("\nPASO 5: Analisis de costos €()")
    @printf("Costo de pedidos: %.2f\n", resultado_escasez.costo_pedidos)
    @printf("Costo de almacenamiento: %.2f\n",
    resultado_escasez.costo_almacenamiento)
    @printf("Costo de desabastecimiento: %.2f\n",
    resultado_escasez.costo_desabastecimiento)
    @printf("COSTO TOTAL: %.2f €\n", resultado_escasez.costo_total)

    # PASO 6: Comparacion con EOQ clasico
    costo_eoq_clasico = sqrt(2 * D_producto * S_producto * H_producto)
    ahorro_anual = costo_eoq_clasico - resultado_escasez.costo_total
    porcentaje_ahorro = (ahorro_anual / costo_eoq_clasico) * 100

    println("\nPASO 6: Comparacion con EOQ clasico")
    @printf("Costo EOQ clasico: %.2f €\n", costo_eoq_clasico)
    @printf("Costo con escasez: %.2f €\n", resultado_escasez.costo_total)
    @printf("Ahorro anual: %.2f €(%.2f%%)\n", ahorro_anual,
        ↪ porcentaje_ahorro)

    println("\nPASO 7: Interpretacion gerencial")
    println("- El modelo permite desabastecimientos controlados")
    println("- Se ahorra en costos de almacenamiento")
    @printf("  - %.1f%% del tiempo habra inventario disponible\n",
    (resultado_escasez.tiempo_con_inventario/resultado_escasez.tiempo_ciclo)*100)
    @printf("  - %.1f%% del tiempo habra desabastecimiento\n",
    (resultado_escasez.tiempo_desabastecido/resultado_escasez.tiempo_ciclo)*100)
```

1.9. Modelo de Revisión Periódica (R,S)

MODELO (R,S) DE REVISION PERIODICA

```
"""
Modelo de revision periodica (R,S)
Sistema donde se revisa el inventario cada R periodos
y se ordena hasta llegar al nivel S

Parametros:
- demand_probs: Diccionario con probabilidades de demanda
- h: Costo de almacenamiento por unidad por periodo
- p: Costo de ruptura/faltante por unidad
- review_period: Periodo de revision R
"""
function periodic_review_model(demand_probs::Dict, h, p, review_period=1)

# PASO 1: Calcular la proporcion critica
# Formula: p/(h+p) donde p es costo de ruptura, h costo almacenamiento
critical_ratio = p / (h + p)

println("MODELO DE REVISION PERIODICA (R,S)")
println("="^60)
println("PASO 1: Calculo de proporcion critica")
println("Costo de almacenamiento (h): $(h)")
println("Costo de ruptura (p): $(p)")
println("Proporcion critica: p/(h+p) = $(p)/($(p)+$(h)) =
    ↪ $(critical_ratio)")

# PASO 2: Crear tabla de probabilidades acumuladas
demandas = sort(collect(keys(demand_probs)))

println("\nPASO 2: Tabla de analisis de demanda")
println("d\tP(d)\tP(D<=d)\tp(d)/d\tSuma q*p(d)/d\tM(D<=d)")
println("-"^60)

prob_acum = 0.0
suma_q_prob_d = 0.0
tabla_resultados = []

for d in demandas
prob_d = demand_probs[d]
prob_acum += prob_d

# Calcular p(d)/d (costo marginal por unidad)
if d > 0
marginal_cost = prob_d / d
suma_q_prob_d += d * marginal_cost
else
marginal_cost = Inf # Evitar division por cero
suma_q_prob_d = 0.0
end
```

```
    # Funcion M(D<=d) para toma de decisiones
    M_d = prob_acum + (1/2) * suma_q_prob_d

    push!(tabla_resultados, (
    demanda = d,
    prob = prob_d,
    prob_acum = prob_acum,
    marginal_cost = marginal_cost,
    suma_marginal = suma_q_prob_d,
    M_value = M_d
    ))

    println("$d\t$(prob_d)\t$(prob_acum)\t", end="")
    if marginal_cost == Inf
    println("Infinito\t$(suma_q_prob_d)\t$(M_d)")
    else
    println("$(marginal_cost)\t$(suma_q_prob_d)\t$(M_d)")
    end
    end

    # PASO 3: Encontrar el nivel optimo S
    optimal_S = demandas[1] # Valor por defecto

    for resultado in tabla_resultados
    if resultado.prob_acum >= critical_ratio
    optimal_S = resultado.demanda
    break
    end
    end

    println("\nPASO 3: Determinacion del nivel optimo S")
    println("Buscamos el primer d donde P(D<=d) >= $(critical_ratio)")
    println("Nivel optimo S* = $(optimal_S) unidades")

    # PASO 4: Calcular metricas de ñdesempeo
    # Calcular la demanda esperada
    demanda_esperada = sum(d * demand_probs[d] for d in demandas)

    # Calcular nivel de servicio
    prob_no_ruptura = sum(demand_probs[d] for d in demandas if d <= optimal_S)
    nivel_servicio = prob_no_ruptura * 100

    # Inventario promedio
    inventario_promedio = optimal_S - demanda_esperada

    println("\nPASO 4: Metricas de ñdesempeo")
    println("Demanda esperada: $(demanda_esperada) unidades")
    println("Inventario promedio: $(inventario_promedio) unidades")
    println("Nivel de servicio: $(nivel_servicio)%")
    println("Probabilidad de ruptura: $(100 - nivel_servicio)%")

    return (
    optimal_S = optimal_S,
```

```
    critical_ratio = critical_ratio,
    demand_expected = demanda_esperada,
    average_inventory = inventario_promedio,
    service_level = nivel_servicio,
    tabla_analisis = tabla_resultados,
    review_period = review_period
    )
    end
```

1.9.1. Ejercicio Resuelto: Producto con Revisión Periódica

PROBLEMA DE REVISIÓN PERIÓDICA

Producto con Revisión: Un producto cuesta 60€, pero si no se dispone cuando se necesita, ocasiona pérdida de 800€. Si se ha comprado y no se utiliza, debe pagarse almacenamiento de 10€ por período. La demanda es discreta con probabilidades dadas en tabla.

SOLUCIÓN COMPLETA

```
# DATOS DEL PROBLEMA (basados en Tabla 8.4 del PDF)
# Probabilidades de demanda discreta
demanda_revision = Dict(
0 => 0.1,  # P(D=0) = 0.1
1 => 0.2,  # P(D=1) = 0.2
2 => 0.2,  # P(D=2) = 0.2
3 => 0.3,  # P(D=3) = 0.3
4 => 0.1,  # P(D=4) = 0.1
5 => 0.1   # P(D=5) = 0.1
)

# Costos del problema
costo_ruptura = 800.0    # € por unidad faltante
costo_almacenamiento = 10.0 # €por unidad en inventario por periodo

println("PRODUCTO CON REVISION PERIODICA - ANALISIS COMPLETO")
println("="^70)

# RESOLVER EL MODELO
resultado_revision = periodic_review_model(
demanda_revision, costo_almacenamiento, costo_ruptura)

println("\nRESUMEN EJECUTIVO:")
println("="^50)
println("NIVEL OPTIMO de inventario (S*): $(resultado_revision.optimal_S)
    ↪ unidades")
println("PROPORCION critica: $(resultado_revision.critical_ratio)")
println("DEMANDA esperada: $(resultado_revision.demand_expected) unidades")
println("INVENTARIO promedio: $(resultado_revision.average_inventory)
```

```
        ↪ unidades")
println("NIVEL de servicio: $(resultado_revision.service_level)%")

println("\nINTERPRETACION GERENCIAL:")
println("-"^50)
println("1. POLITICA OPTIMA:")
@printf("  - Revisar inventario cada periodo\n")
@printf("  - Ordenar hasta tener %d unidades\n",
    ↪ resultado_revision.optimal_S)

println("\n2. IMPLICACIONES OPERATIVAS:")
println("  - Costo muy alto de ruptura €($(costo_ruptura)) justifica
    ↪ inventario alto")
println("  - Se mantendra inventario promedio de
    ↪ $(resultado_revision.average_inventory) unidades")
println("  - $(resultado_revision.service_level)% de las veces se
    ↪ satisfara la demanda completamente")

println("\n3. ANALISIS DE SENSIBILIDAD:")
# Probar diferentes niveles S para mostrar el impacto
println("  Impacto de diferentes niveles S:")
for S_test in 2:6
prob_satisfaccion = sum(demanda_revision[d] for d in
    ↪ keys(demanda_revision) if d <= S_test)
inventario_prom = S_test - resultado_revision.demand_expected
println("  S=$(S_test):
    ↪ Servicio=$(round(prob_satisfaccion*100,digits=1))%,
    ↪ Inv.Prom=$(round(inventario_prom,digits=1))")
end

println("\n4. COMPARACION DE COSTOS ESPERADOS:")
# Calcular costo esperado para el nivel optimo
S_opt = resultado_revision.optimal_S
costo_almacen_esp = resultado_revision.average_inventory *
    ↪ costo_almacenamiento

# Calcular costo de ruptura esperado
costo_ruptura_esp = 0.0
for d in keys(demanda_revision)
if d > S_opt
faltante = d - S_opt
costo_ruptura_esp += faltante * costo_ruptura * demanda_revision[d]
end
end

costo_total_esp = costo_almacen_esp + costo_ruptura_esp

println("  Costo almacenamiento esperado:
    ↪ €$(round(costo_almacen_esp,digits=2))")
println("  Costo ruptura esperado: €$(round(costo_ruptura_esp,digits=2))")
println("  COSTO TOTAL ESPERADO: €$(round(costo_total_esp,digits=2)) por
    ↪ periodo")
```

1.10. Modelo con Demanda Normal

</> DEMANDA NORMAL - TIENDA DE ABARROTES

```
"""
Modelo de inventario con demanda normal y backorders
"""
function inventory_normal_demand(mu0, sigma0, k, L, h, Cu,
    ↪ weeks_per_year=52)
# Parametros de demanda durante lead time
muL = mu0 * L / weeks_per_year
sigmaL = sigma0 * sqrt(L / weeks_per_year)

# Para encontrar Q* y R* optimos, usamos el metodo iterativo
# Inicializacion
Q = sqrt(2 * mu0 * k / h) # EOQ inicial

# Funcion para calcular el punto de reorden optimo
function optimal_reorder_point(Q, muL, sigmaL, h, Cu)
# Factor de seguridad optimo
z_star = quantile(Normal(0,1), Cu / (Cu + h))
R_star = muL + z_star * sigmaL
return R_star, z_star
end

# Iteracion para encontrar Q* y R* simultaneamente
for iteration in 1:10
R, z = optimal_reorder_point(Q, muL, sigmaL, h, Cu)

# Funcion de perdida unitaria esperada
L_z = pdf(Normal(0,1), z) - z * (1 - cdf(Normal(0,1), z))

# Actualizar Q
Q_new = sqrt(2 * mu0 * (k + Cu * sigmaL * L_z) / h)

if abs(Q_new - Q) < 0.01
Q = Q_new
break
end
Q = Q_new
end

R, z = optimal_reorder_point(Q, muL, sigmaL, h, Cu)

return (
Q_star = Q,
R_star = R,
muL = muL,
sigmaL = sigmaL,
z_star = z,
orders_per_year = mu0 / Q
)
end

```

```julia
# Datos del problema: Tienda de abarrotes
mu0_tienda = 1000 # cajas/ñao (promedio)
sigma0_tienda = 40.8 # cajas/ñao (desviacion estandar)
k_tienda = 50    # $ por pedido
L_tienda = 2     # semanas de lead time
h_tienda = 10    # $ por caja/ñao
Cu_tienda = 20   # $ por caja (costo de agotamiento)

resultado_tienda = inventory_normal_demand(mu0_tienda, sigma0_tienda,
    ↪ k_tienda,
L_tienda, h_tienda, Cu_tienda)

println("TIENDA DE ABARROTES - DEMANDA NORMAL")
println("="^60)
@printf("Demanda promedio anual: %.0f cajas\n", mu0_tienda)
@printf("Desviacion estandar anual: %.1f cajas\n", sigma0_tienda)
@printf("Lead time: %d semanas\n", L_tienda)
println("-"^60)
@printf("Demanda promedio durante lead time: %.1f cajas\n",
    ↪ resultado_tienda.muL)
@printf("Desviacion estandar durante lead time: %.3f cajas\n",
    ↪ resultado_tienda.sigmaL)
println("-"^60)
@printf("Cantidad optima (Q*): %.2f cajas\n", resultado_tienda.Q_star)
@printf("Punto de reorden (R*): %.2f cajas\n", resultado_tienda.R_star)
@printf("Factor de seguridad (z*): %.3f\n", resultado_tienda.z_star)
@printf("Pedidos por ñao: %.2f\n", resultado_tienda.orders_per_year)
```

1.11. Visualización de Resultados

GRÁFICOS Y VISUALIZACIÓN

```julia
"""
Funciones para visualizar los modelos de inventario
"""
using Plots
plotly() # Backend interactivo

function plot_eoq_costs(D, S, H, Q_range=nothing)
if Q_range === nothing
Q_opt = sqrt(2*D*S/H)
Q_range = range(Q_opt*0.1, Q_opt*3, length=100)
end

ordering_costs = [D*S/Q for Q in Q_range]
holding_costs = [Q*H/2 for Q in Q_range]
total_costs = ordering_costs .+ holding_costs

Q_opt = sqrt(2*D*S/H)
min_cost = sqrt(2*D*S*H)
```

```
    p = plot(Q_range, [ordering_costs holding_costs total_costs],
    labels=["Costo de Pedido" "Costo de Almacen" "Costo Total"],
    linewidth=2,
    xlabel="Cantidad de Pedido (Q)",
    ylabel="Costo Anual (\$)",
    title="Analisis de Costos EOQ",
    legend=:topright)

    # Marcar el punto optimo
    scatter!(p, [Q_opt], [min_cost],
    markersize=8,
    markercolor=:red,
    label="Q* Optimo ($(round(Int,Q_opt)))")

    return p
    end

    function plot_inventory_level(Q, D, L=0, periods=4)
    """Graficar el nivel de inventario a traves del tiempo"""

    days_per_cycle = Q / D * 365
    total_days = periods * days_per_cycle

    time_points = Float64[]
    inventory_levels = Float64[]

    for period in 0:(periods-1)
    cycle_start = period * days_per_cycle

    # Inicio del ciclo con Q unidades
    push!(time_points, cycle_start)
    push!(inventory_levels, Q)

    # Final del ciclo con 0 unidades
    push!(time_points, cycle_start + days_per_cycle - 0.01)
    push!(inventory_levels, 0.01)
    end

    p = plot(time_points, inventory_levels,
    linewidth=2,
    xlabel="Tiempo (dias)",
    ylabel="Nivel de Inventario",
    title="Patron de Inventario EOQ",
    legend=false,
    color=:blue)

    # Agregar lineas de reorden si hay lead time
    if L > 0
    reorder_level = D * L / 365
    hline!(p, [reorder_level],
    linestyle=:dash,
    color=:red,
    label="Punto de Reorden")
```

```
    end

    return p
    end

    function compare_models_table()
    """Crear tabla comparativa de los diferentes modelos"""

    modelos = ["EOQ Basico", "EOQ con Backorders", "Lote de Produccion",
    "Con Lead Time", "Descuentos", "Probabilistico"]

    caracteristicas = ["Demanda constante", "Permite faltantes",
    "Produccion gradual", "Tiempo de entrega",
    "Precios variables", "Demanda incierta"]

    aplicaciones = ["Compras simples", "Servicio flexible",
    "Manufactura", "Compras con demora",
    "Compras a granel", "Productos perecederos"]

    df_comparacion = DataFrame(
    Modelo = modelos,
    Caracteristica = caracteristicas,
    Aplicacion = aplicaciones
    )

    return df_comparacion
    end

    # Generar ejemplo de graficos
    println("Generando visualizaciones...")

    # Ejemplo con datos de la clinica
    D = 10000; S = 50; H = 4.5
    plot_clinica = plot_eoq_costs(D, S, H)

    # Patron de inventario
    Q_opt = sqrt(2*D*S/H)
    plot_pattern = plot_inventory_level(Q_opt, D)

    # Tabla comparativa
    tabla_comparacion = compare_models_table()
    println("\nCOMPARACION DE MODELOS DE INVENTARIO:")
    println(tabla_comparacion)
```

1.12. Funciones de Utilidad y Herramientas

HERRAMIENTAS ADICIONALES

```
"""
Modulo completo para analisis de inventarios
"""
module InventoryModels

using Optim, Distributions, DataFrames, Printf

export EOQResult, solve_eoq, solve_eoq_backorders,
solve_production_lot, analyze_quantity_discounts

struct EOQResult
Q_star::Float64
total_cost::Float64
orders_per_year::Float64
time_between_orders::Float64
additional_info::Dict{String, Any}
end

function sensitivity_analysis(D, S, H, param_range=0.8:0.1:1.2)
"""Analisis de sensibilidad para parametros EOQ"""

results = DataFrame(
D_factor = Float64[],
S_factor = Float64[],
H_factor = Float64[],
Q_star = Float64[],
Total_Cost = Float64[]
)

base_Q = sqrt(2*D*S/H)
base_cost = sqrt(2*D*S*H)

for d_factor in param_range
for s_factor in param_range
for h_factor in param_range
new_D = D * d_factor
new_S = S * s_factor
new_H = H * h_factor

new_Q = sqrt(2*new_D*new_S/new_H)
new_cost = sqrt(2*new_D*new_S*new_H)

push!(results, (d_factor, s_factor, h_factor, new_Q, new_cost))
end
end
end

return results
end
```

```
    function monte_carlo_inventory(D_dist, S_dist, H_dist, n_simulations=1000)
    """Simulacion Monte Carlo para analisis de incertidumbre"""

    results = Float64[]

    for i in 1:n_simulations
    D_sample = rand(D_dist)
    S_sample = rand(S_dist)
    H_sample = rand(H_dist)

    Q_sample = sqrt(2 * D_sample * S_sample / H_sample)
    push!(results, Q_sample)
    end

    return (
    mean_Q = mean(results),
    std_Q = std(results),
    quantiles = quantile(results, [0.05, 0.25, 0.5, 0.75, 0.95]),
    all_results = results
    )
    end

    end  # fin del modulo

    # Ejemplo de analisis Monte Carlo
    println("ANALISIS MONTE CARLO - INCERTIDUMBRE EN PARAMETROS")
    println("="^60)

    # Definir distribuciones de parametros (con incertidumbre)
    D_distribution = Normal(10000, 500) # Demanda con incertidumbre
    S_distribution = Normal(50, 5)     # Costo de pedido con incertidumbre
    H_distribution = Normal(4.5, 0.5) # Costo de almacen con incertidumbre

    mc_result = InventoryModels.monte_carlo_inventory(D_distribution,
        ↪ S_distribution,
    H_distribution, 5000)

    @printf("EOQ promedio: %.2f unidades\n", mc_result.mean_Q)
    @printf("Desviacion estandar: %.2f unidades\n", mc_result.std_Q)
    println("\nCuantiles:")
    println("5%: $(round(mc_result.quantiles[1], digits=2))")
    println("25%: $(round(mc_result.quantiles[2], digits=2))")
    println("50% (mediana): $(round(mc_result.quantiles[3], digits=2))")
    println("75%: $(round(mc_result.quantiles[4], digits=2))")
    println("95%: $(round(mc_result.quantiles[5], digits=2))")
```

1.13. Conclusiones y Mejores Prácticas

SÍNTESIS Y RECOMENDACIONES

Ventajas de usar Julia para Inventarios:

1. **Rendimiento Superior:** Julia ejecuta cálculos matemáticos a velocidad cercana a C
2. **Sintaxis Clara:** Código fácil de leer y mantener
3. **Ecosistema Científico:** Paquetes especializados para optimización
4. **Interactividad:** Ideal para análisis exploratorio y prototipado
5. **Escalabilidad:** Desde problemas simples hasta sistemas complejos

Mejores Prácticas Implementadas:

- ✔ Funciones modulares y reutilizables
- ✔ Documentación clara con docstrings
- ✔ Manejo de errores y validación de entrada
- ✔ Análisis de sensibilidad incorporado
- ✔ Visualización de resultados
- ✔ Simulación Monte Carlo para incertidumbre

EXTENSIONES FUTURAS

Desarrollos Avanzados con Julia:

- → Optimización multiobjetivo con JuMP.jl
- → Modelos de inventario dinámicos con DifferentialEquations.jl
- → Machine Learning para predicción de demanda con MLJ.jl
- → Interfaces web interactivas con PlutoUI.jl
- → Optimización distribuida para problemas grandes

Julia: El futuro de la optimización computacional en inventarios