# Macroeconomía Internacional

Francisco Roldán
IMF

October 2021

# Big $K$, little $k$

- Problema del agente, tomando como dadas $p_C(A,z), y(A,z), \Phi(A,z)$

$$v(a,A,z) = \max_{a'} u(c) + \beta \mathbb{E}\left[v(a',A',z') \mid z\right]$$

$$\text{sujeto a } \; p_C(A,z)c + \frac{a'}{1+r} = y(A,z) + a$$

$$A' = \Phi(A,z)$$

- Al mismo tiempo, deducir $p_C(A,z), y(A,z), \Phi(A,z)$ de las decisiones del agente

# Dos observaciones

1. Buscamos el equilibrio de un juego entre un agente y los demás

2. Comparar con la ecuación de Euler

$$u'(c)\frac{1}{p_C(A,z)} = \beta(1+r)\mathbb{E}\left[u'(c')\frac{1}{p_C(\Phi(A,z),z')} \mid z\right]$$

# Fórmulas de CES

- Indice de precios

$$p_C(A,z) = \left[\varpi_N^{\frac{1}{1+\eta}} p_N^{\frac{\eta}{1+\eta}} + \varpi_T^{\frac{1}{1+\eta}} p_T^{\frac{\eta}{1+\eta}}\right]^{\frac{1+\eta}{\eta}}$$

- Demanda de los componentes

$$c_T = \varpi_T \left(\frac{p_T}{p_C}\right)^{-\eta} c$$

# Agregados en equilibrio

1. Ahorros del agente representativo

$$\Phi(A, z) = a'(A, A, z)$$

2. Consumo total de transables

$$c_T(A, z) = \varpi_T \left( \frac{1}{p_C(A, z)} \right)^{-\eta} c(A, A, z)$$

3. Demanda de trabajo $H(A, z)$ y por lo tanto el producto $h_N^{\alpha} p_N + z h_T^{\alpha}$

$$\begin{cases} h_N & = \left( \frac{\alpha}{w} \frac{\varpi_N}{\varpi_T} \right)^{\frac{1}{1+\alpha\eta}} c_T^{1+\eta} \\ h_T & = \left( \frac{z\alpha}{w} \right)^{\frac{1}{1-\alpha}} \\ h_N + h_T & = 1, w > \bar{w} \end{cases} \quad o \quad \begin{cases} h_N & = \left( \frac{\alpha}{\bar{w}} \frac{\varpi_N}{\varpi_T} \right)^{\frac{1}{1+\alpha\eta}} c_T^{1+\eta} \\ h_T & = \left( \frac{z\alpha}{\bar{w}} \right)^{\frac{1}{1-\alpha}} \\ h_N + h_T & < 1 \end{cases}$$

# 7 diferencias

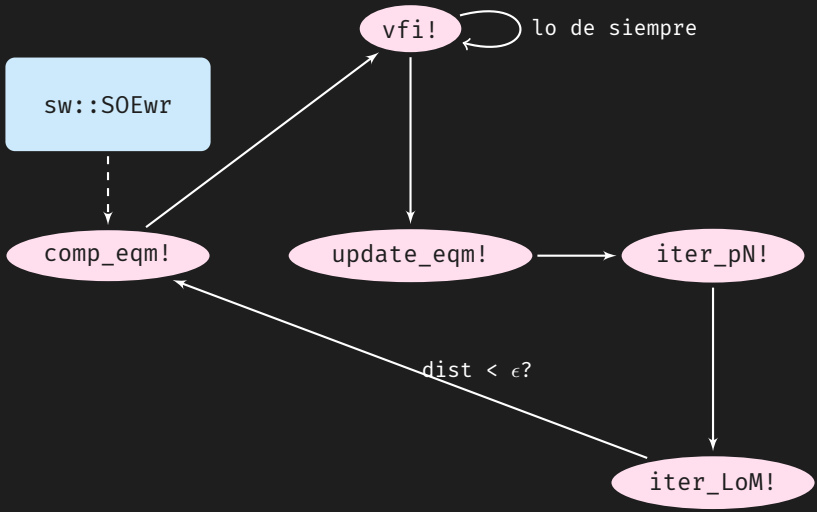
- Restricción de presupuesto del agente

$$p_C(A,z)c + \frac{a'}{1+r} = y(A,z) + a$$

- Restricción externa de la economía 

$$\frac{A'}{1+r} - A = \underbrace{y(A,z) - p_C(A,z)c}_{=CA}$$

# Pseudo-código



# Otro problema de fluctuación de ingresos?

# Códigos

# Constructor

```
mutable struct SOEwr <: SOE
  β::Float64
  γ::Float64
  r::Float64
  \varpiN::Float64
  \varpiT::Float64
  \eta::Float64
  \alpha::Float64
  wbar::Float64

  agrid::Vector{Float64}
  zgrid::Vector{Float64}
  Pz::Matrix{Float64}

  v::Dict{Symbol, Array{Float64, 3}}

  pN::Array{Float64, 2}
  w::Array{Float64, 2}
  Ap::Array{Float64, 2}
  Y::Array{Float64, 2}
end
```

- Parámetros, como siempre
  - ... pero podríamos ponerlos en un `Dict`, no?
- Grillas para $a$, $A$, $z$, probabilidades para $z$
  - ... pero podríamos ponerlas en un `Dict`, no?
- El `Dict` para las funciones de valor, ahorro, consumo
- Variables endógenas agregadas
  - ... pero podríamos ponerlas en un `Dict`, no?

# Constructor

```
function SOEwr(; β = 0.96, γ = 2, r = 1.02, \varpiN = 0.55, \eta = 1/0.83-1, \alpha = 0.67, wbar = 0.7
    \rhoz = 0.945, \sigmaz = 0.025, Na = 40, Nz = 21, amin = -0.5, amax = 10)

  \varpiT = 1 - \varpiN

  agrid = range(amin, amax, length = Na)
  zchain = tauchen(Nz, \rhoz, \sigmaz, 0, 3)
  zgrid, Pz = exp.(zchain.state_values), zchain.p

  v = Dict(key => ones(Na, Na, Nz) for key in [:v, :c, :a])

  pN, w = ones(Na, Nz), ones(Na, Nz)
  Ap = [av for av in agrid, zv in zgrid]
  Y  = [exp(zv) for av in agrid, zv in zgrid]

  return SOEwr(β, γ, r, \varpiN, \varpiT, \eta, \alpha, wbar, agrid, zgrid, Pz, v, pN, w, Ap, Y)
end
```

# Funciones básicas con trucos

```
function utility(c, sw::SOE)
  γ = sw.γ
  cmin = 1e-3
  if c < cmin
    return utility(cmin,sw) + (c-cmin) * (cmin)^-γ
  else
    γ == 1 && return log(c)
    return c^(1-γ)/(1-γ)
  end
end

function price_index(pN, pT, sw::SOE)
  \varpiN, \varpiT, \eta = sw.\varpiN, sw.\varpiT, sw.\eta
  return ( \varpiN^(1/(1+\eta))*pN^(\eta/(1+\eta)) + \varpiT^(1/(1+\eta))*pT^(\eta/(1+\eta)) )^((1+\et
end
price_index(pN, sw::SOE) = price_index(pN, 1, sw)
```

# Evaluar la $v$ dado $a'$

```
function expect_v(apv, Apv, pz, itp_v, sw::SOE)
  Ev = 0.0
  for (jzp, zpv) in enumerate(sw.zgrid)
    prob = pz[jzp]
    Ev += prob * itp_v[:v](apv, Apv, zpv)
  end
  return Ev
end

budget_constraint(apv, av, yv, r, pCv) = ( yv + av - apv/(1+r) ) / pCv

function eval_value(apv, av, yv, Apv, pz, pCv, itp_v, sw::SOE)
  c = budget_constraint(apv, av, yv, sw.r, pCv)
  u = utility(c, sw)
  Ev = expect_v(apv, Apv, pz, itp_v, sw)
  return u + sw.β * Ev
end
```

# Elegir $a'$

```
function optim_value(av, yv, Apv, pz, pCv, itp_v, sw::SOE)

  obj_f(x) = -eval_value(x, av, yv, Apv, pz, pCv, itp_v, sw)
  amin, amax = extrema(sw.agrid)

  res = Optim.optimize(obj_f, amin, amax)

  apv = res.minimizer
  v   = -res.minimum

  c = budget_constraint(apv, av, yv, sw.r, pCv)

  return v, apv, c
end
```

# Actualizar la v

```
function vf_iter!(new_v, sw::SOE)
  itp_v = Dict(key => interpolate((sw.agrid, sw.agrid, sw.zgrid), sw.v[key],
     Gridded(Linear())) for key in keys(sw.v))

  for (jA, Av) in enumerate(sw.agrid), (jz, zv) in enumerate(sw.zgrid)
    pNv = sw.pN[jA, jz]
    pCv = price_index(pNv, sw)

    Apv, yv, pz = sw.Ap[jA, jz], sw.Y[jA, jz], sw.Pz[jz, :]
    for (ja, av) in enumerate(sw.agrid)
      v, apv, c = optim_value(av, yv, Apv, pz, pCv, itp_v, sw)

      new_v[:v][ja, jA, jz] = v
      new_v[:a][ja, jA, jz] = apv
      new_v[:c][ja, jA, jz] = c
    end
  end
end
```

# Value function iteration

```
function update_v!(new_v, sw::SOE; upd_\eta = 1)
  for key in keys(new_v)
    sw.v[key] = sw.v[key] + upd_\eta * (new_v[key] - sw.v[key])
  end
end

function vfi!(sw::SOE; tol=1e-4, maxiter = 2000)
  iter, dist = 0, 1+tol
  new_v = Dict(key => similar(val) for (key, val) in sw.v)
  while iter < maxiter && dist > tol
    iter += 1
    vf_iter!(new_v, sw)
    dist = maximum([ norm(new_v[key] - sw.v[key]) / (1+norm(sw.v[key])) for key in keys(sw.v) ])
    norm_v = 1+maximum([norm(sw.v[key]) for key in keys(sw.v)])
    print("Iteration $iter: dist = $(@sprintf("%0.3g", dist)) at |v| =
      $(@sprintf("%0.3g", norm_v))\n")
    update_v!(new_v, sw)
  end
  return dist
end
```

# Respirar

- Nuestro `vfi!` de siempre encuentra el consumo y el ahorro de un agente dados
  - La ley de movimiento de $A$
  - El precio relativo $p_N$, aka el tipo de cambio real
  - Los niveles de producto $y_N$, $y_T$

- Nos falta encontrar esas cosas
  - Agregar las decisiones de los agentes
  - Encontrar los precios que igualen demanda y oferta (de qué?)

- Big $K$, little $k$
  - El agente representativo piensa que $a$ y $A$ son cosas distintas
  - El consumo agregado en $(A, z)$ es $c(A, A, z)$

# Respirar

- Nuestro `vfi!` de siempre encuentra el consumo y el ahorro de un agente dados
  - La ley de movimiento de $A$
  - El precio relativo $p_N$, aka el tipo de cambio real
  - Los niveles de producto $y_N$, $y_T$

- Nos falta encontrar esas cosas
  - Agregar las decisiones de los agentes
  - Encontrar los precios que igualen demanda y oferta (de qué?)

- Big $K$, little $k$
  - El agente representativo piensa que $a$ y $A$ son cosas distintas
  - El consumo agregado en $(A, z)$ es $c(A, A, z)$

# Respirar

- Nuestro `vfi!` de siempre encuentra el consumo y el ahorro de un agente dados
  - La ley de movimiento de $A$
  - El precio relativo $p_N$, aka el tipo de cambio real
  - Los niveles de producto $y_N$, $y_T$
- Nos falta encontrar esas cosas
  - Agregar las decisiones de los agentes
  - Encontrar los precios que igualen demanda y oferta (de qué?)
- Big $K$, little $k$
  - El agente representativo piensa que $a$ y $A$ son cosas distintas
  - El consumo agregado en $(A, z)$ es $c(A, A, z)$

# Agregados (de atrás para adelante)

```
function comp_eqm!(sw::SOE; tol = 1e-3, maxiter = 2000)
  iter, dist = 0, 1+tol
  new_p = similar(sw.pN)
  tol_vfi = 1e-2
  while dist > tol && iter < maxiter
    iter += 1
    print("Outer Iteration $iter (tol = $(@sprintf("%0.3g",tol_vfi)))\n")
    dist_v = vfi!(sw, tol = tol_vfi)

    norm_p = norm(sw.pN)
    dist_p = update_eqm!(new_p, sw) / (1+norm_p)

    dist = max(dist_p, 10*dist_v)

    print("After $iter iterations, dist = $(@sprintf("%0.3g",dist_p)) at |pN| =
       $(@sprintf("%0.3g", norm_p))\n\n")

    tol_vfi = max(1e-4, tol_vfi * 0.9)
  end
end
```

# Iteración de la ley de movimiento

```
function iter_LoM!(sw::SOE; upd_\eta = 1)
  for jA in eachindex(sw.agrid), jz in eachindex(sw.zgrid)
    sw.Ap[jA, jz] = (1-upd_\eta) * sw.Ap[jA, jz] + upd_\eta * sw.v[:a][jA, jA, jz]
  end
end

function update_eqm!(new_p, sw::SOE; upd_\eta = 1)
  iter_pN!(new_p, sw)
  iter_LoM!(sw)
  dist = norm(new_p - sw.pN)
  sw.pN = sw.pN + upd_\eta * (new_p - sw.pN)
  return dist
end
```

# Iteración sobre los precios

```
function iter_pN!(new_p, sw::SOE; upd_\eta = 1)
  minp = 0.9 * minimum(sw.pN)
  maxp = 1.1 * maximum(sw.pN)
  for (jA, Av) in enumerate(sw.agrid), (jz, zv) in enumerate(sw.zgrid)
    pNg = sw.pN[jA, jz]
    pcC = sw.v[:c][jA, jA, jz] * price_index(pNg, sw)

    obj_f(x) = diff_pN(x, pcC, zv, sw).F

    res = Optim.optimize(obj_f, minp, maxp)

    p = sw.pN[jA, jz] * (1-upd_\eta) + res.minimizer * upd_\eta
    new_p[jA, jz] = p

    others = diff_pN(p, pcC, zv, sw)
    sw.Y[jA, jz] = others.y
    sw.w[jA, jz] = others.w
  end
end
```

# Encontrar un precio (y un salario!)

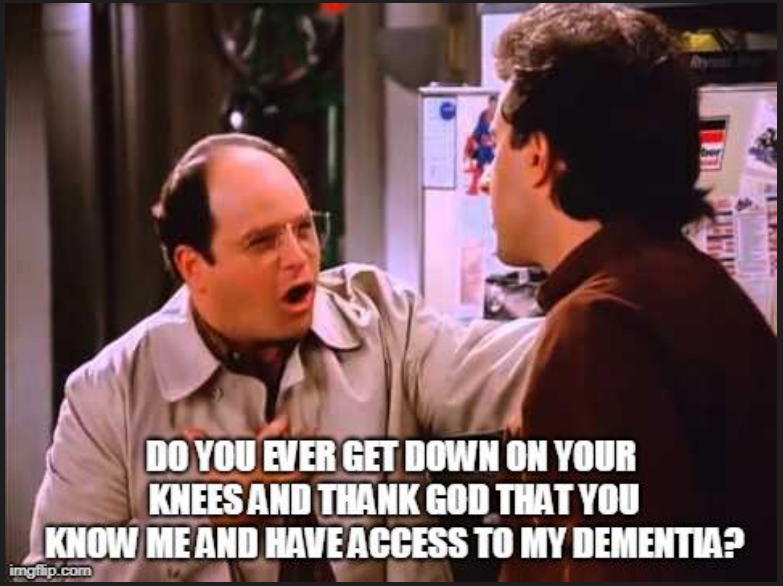
```
function diff_pN(pNv, pcC, zv, sw::SOE)
  \alpha, \varpiN, \varpiT, \eta, wbar = sw.\alpha, sw.\varpiN, sw.\varpiT, sw.\eta, sw.wbar

  pCv = price_index(pNv, sw)
  C = pcC / pCv

  cT = C * \varpiT * pCv^\eta      # c_i = \varpi_i (p_i/p)^(-\eta) C

  hN, hT, wopt = find_w(zv, cT, wbar, sw)

  yN = hN^\alpha
  yT = zv * hT^\alpha

  pN_new = \varpiN / \varpiT * (cT/yN)^(1+\eta)

  output = pN_new * yN + yT

  return (F = (pN_new-pNv)^2, y = output, w = wopt)
end
```

# Encontrar el salario (si hace falta)

```
function labor_demand(zv, cT, w, sw::SOE)
  \alpha, \varpiN, \varpiT, \eta = sw.\alpha, sw.\varpiN, sw.\varpiT, sw.\eta
  hN = (\alpha/w * \varpiN / \varpiT)^(1/(1+\alpha*\eta)) * cT^(1+\eta)
  hT = (zv*\alpha/w)^(1/(1-\alpha))
  return (h = hN+hT, hN = hN, hT = hT)
end

function find_w(zv, cT, wbar, sw::SOE)
  hN = labor_demand(zv, cT, wbar, sw).hN
  hT = labor_demand(zv, cT, wbar, sw).hT
  H = hN + hT
  if H < 1
    wopt = wbar
  else
    f(w) = (labor_demand(zv, cT, w, sw).h - 1)^2
    res = Optim.optimize(f, wbar, 2*wbar)
    wopt = res.minimizer
    hN = labor_demand(zv, cT, wopt, sw).hN
    hT = labor_demand(zv, cT, wopt, sw).hT
  end
  return hN, hT, wopt
end
```

# Graficar (finalmente)

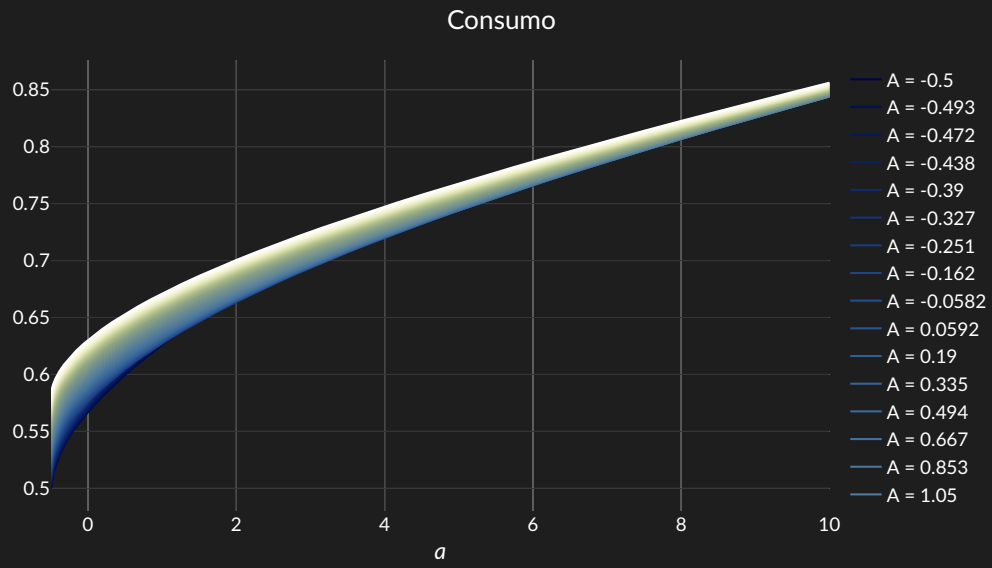
# Gráficos y TOC



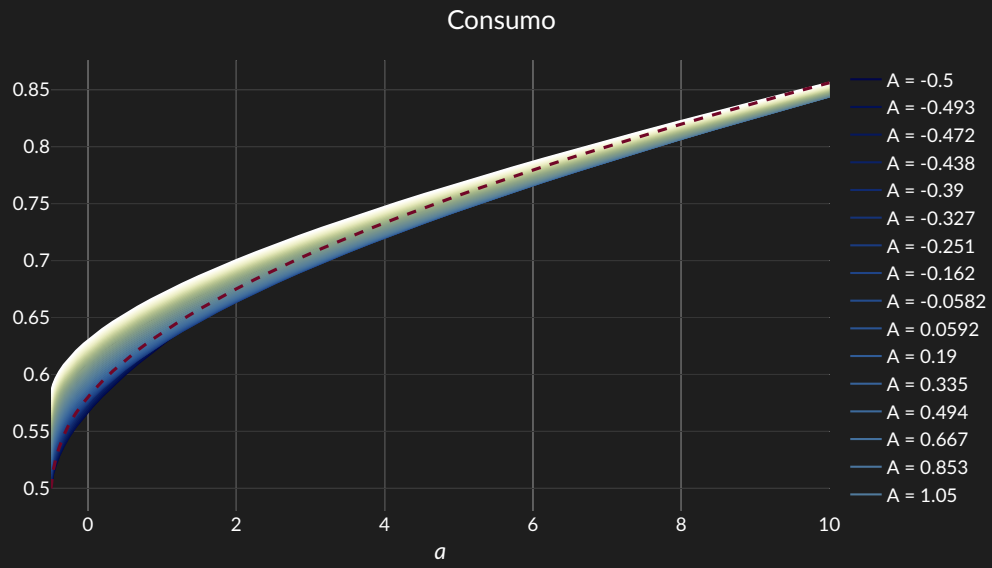
# Funciones de valor/comportamiento

```
function plot_cons(sw::SOE; indiv=false)
  jA, jz, Na = 5, 5, length(sw.agrid)

  cons_mat = [sw.v[:c][ja, jA, jz] for ja in eachindex(sw.agrid), jA in eachindex(sw.agrid)]
  cons_agg = [sw.v[:c][ja, ja, jz] for ja in eachindex(sw.agrid)]

  colvec = [get(ColorSchemes.davos, (jA-1)/(Na-1)) for jA in eachindex(sw.agrid)]

  scats = [scatter(x=sw.agrid, y=cons_mat[:, jA], marker_color=colvec[jA], name = "A =
     $(@sprintf("%0.3g",Av))") for (jA, Av) in enumerate(sw.agrid)]

  indiv || push!(scats, scatter(x=sw.agrid, y=cons_agg, line_dash="dash", line_width=3,
     name= "Agregado", line_color="#710627"))

  layout = Layout(title="Consumo",
    font_family = "Lato", font_size = 18, width = 1920*0.5, height=1080*0.5,
    paper_bgcolor="#1e1e1e", plot_bgcolor="#1e1e1e", font_color="white",
    xaxis = attr(zeroline = false, gridcolor="#353535", title="<i>a"),
    yaxis = attr(zeroline = false, gridcolor="#353535"),
    )
  plot(scats, layout)
end
```

# Consumo


Consumo
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0
2
4
6
8
10
a
A = -0.5
A = -0.493
A = -0.472
A = -0.438
A = -0.39
A = -0.327
A = -0.251
A = -0.162
A = -0.0582
A = 0.0592
A = 0.19
A = 0.335
A = 0.494
A = 0.667
A = 0.853
A = 1.05

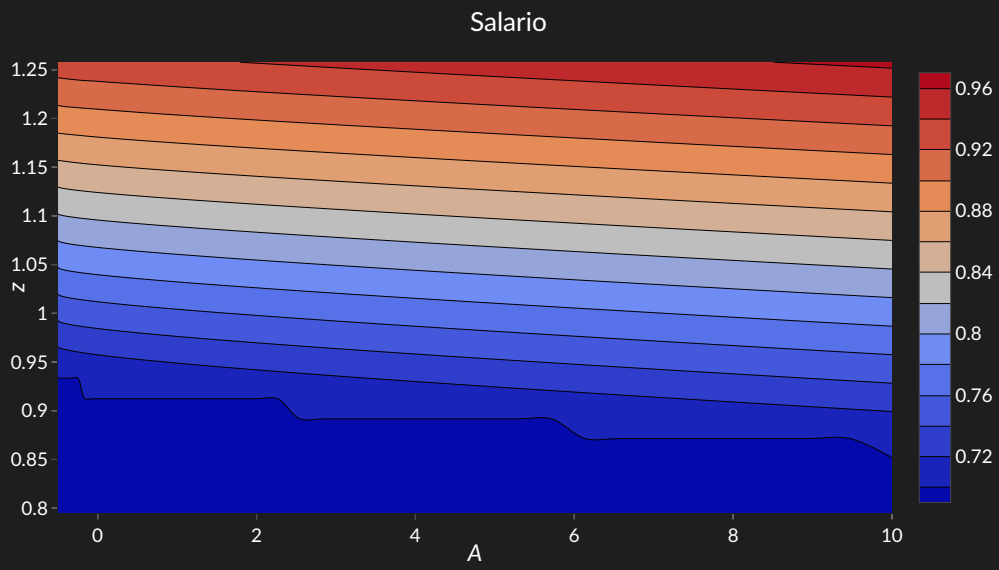
# Consumo


Consumo
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0
2
4
6
8
10
a
A = -0.5
A = -0.493
A = -0.472
A = -0.438
A = -0.39
A = -0.327
A = -0.251
A = -0.162
A = -0.0582
A = 0.0592
A = 0.19
A = 0.335
A = 0.494
A = 0.667
A = 0.853
A = 1.05

# Salarios de equilibrio

```
function plot_wage(sw::SOE)

  con = contour(x=sw.agrid, y=sw.zgrid,
    z = sw.w)

  layout = Layout(title="Salario",
    font_family = "Lato", font_size = 18, width = 1920*0.5, height=1080*0.5,
    paper_bgcolor="#1e1e1e", plot_bgcolor="#1e1e1e", font_color="white",
    xaxis = attr(zeroline = false, gridcolor="#353535", title="<i>A"),
    yaxis = attr(zeroline = false, gridcolor="#353535", title="<i>z"),
    )

  plot(con, layout)
end
```

# Salarios


Salario
1.25
1.2
1.15
1.1
1.05
z
1
0.95
0.9
0.85
0.8
0
2
4
6
8
10
A
0.96
0.92
0.88
0.84
0.8
0.76
0.72

# Cierre

# Conclusión

## Para seguir

- Los invito a
  - Resolver con y sin rigidez de salarios
  - Análisis "empírico" con los datos del simulador
    - `DataFrames`!
  - Resolver el problema del planner en esta economía. Ayudas:
    1. El planner entiende que $a = A$ (un solo estado!)
    2. El planner entiende que la restricción es $h \leq \mathcal{H}(c_T, \bar{w})$
  - Combinar Schmitt-Grohé y Uribe (planner) con Arellano.
    - Tesis de doctorado de Anzoategui (2020)
    - Bianchi, Ottonello, y Presno (2020)
- QuantEcon