Tópicos en Macroeconomía Internacional con Aplicaciones Cuantitativas

Francisco Roldán
IMF

October 2020

Plan

Tres modelos

1. RBC con default (Arellano, 2008)
   ... La deuda se paga con el valor presente del superávit externo, pero **cuándo** se paga la deuda?
2. Modelos con rigideces nominales (Schmitt-Grohé y Uribe, 2016)
   ... Tipo de cambio, externalidades de demanda
3. Deflación fisheriana y sudden stops (Bianchi, 2011)
   ... Cómo el precio del colateral amplifica la salida de capitales

Plan

Tres modelos

1. RBC con default (Arellano, 2008)

   . . . La deuda se paga con el valor presente del superávit externo, pero **cuándo** se paga la deuda?

2. Modelos con rigideces nominales (Schmitt-Grohé y Uribe, 2016)

   . . . Tipo de cambio, externalidades de demanda

3. Deflación fisheriana y sudden stops (Bianchi, 2011)

   . . . Cómo el precio del colateral amplifica la salida de capitales

Plan

Tres modelos

1. RBC con default (Arellano, 2008)

   . . . La deuda se paga con el valor presente del superávit externo, pero **cuándo** se paga la deuda?

2. Modelos con rigideces nominales (Schmitt-Grohé y Uribe, 2016)

   . . . Tipo de cambio, externalidades de demanda

3. Deflación fisheriana y sudden stops (Bianchi, 2011)

   . . . Cómo el precio del colateral amplifica la salida de capitales

Objetivos

1. Discusión no exhaustiva de la mecánica de los modelos

2. Foco en aplicación cuantitativa
   - ... Códigos para resolver, simular, calibrar, graficar

Por qué?

3. Julia
   - Iteración en la función de valor
   - Iteración en la ecuación de Euler
   - Método de grillas endógenas

Objetivos

1. Discusión no exhaustiva de la mecánica de los modelos

2. Foco en aplicación cuantitativa

   . . . Códigos para resolver, simular, calibrar, graficar

Por qué?

3. Julia
   - Iteración en la función de valor
   - Iteración en la ecuación de Euler
   - Método de grillas endógenas

Objetivos

1. Discusión no exhaustiva de la mecánica de los modelos
2. Foco en aplicación cuantitativa
   ... Códigos para resolver, simular, calibrar, graficar

Por qué?

3. Julia
   - Iteración en la función de valor
   - Iteración en la ecuación de Euler
   - Método de grillas endógenas

Objetivos

1. Discusión no exhaustiva de la mecánica de los modelos

2. Foco en aplicación cuantitativa

   ... Códigos para resolver, simular, calibrar, graficar

   Por qué?

3. Julia

   - Iteración en la función de valor
   - Iteración en la ecuación de Euler
   - Método de grillas endógenas

Hoy

1. Repaso de programación dinámica
   ... McCall (1970)

2. Julia
   ... Multiple dispatch, `structs`, estabilidad de tipos

3. Códigos para hacer `vfi` en McCall (1970)

Programación Dinámica

McCall (1970)

- Un agente busca trabajo.
- Preferencias standard: utilidad u, descuento β.
- Los trabajos son heterogéneos y sólo difieren en el salario que pagan.
- Cada período llega una oferta de trabajo $w \overset{iid}{\sim} F(\cdot)$
- Sólo se puede aceptar un trabajo. El agente recibe b mientras busca

- Cómo decide el agente qué trabajo aceptar?

McCall (1970)

- Un agente busca trabajo.
- Preferencias standard: utilidad u, descuento β.
- Los trabajos son heterogéneos y sólo difieren en el salario que pagan.
- Cada período llega una oferta de trabajo $w \overset{iid}{\sim} F(\cdot)$
- Sólo se puede aceptar un trabajo. El agente recibe b mientras busca

- **Cómo decide el agente qué trabajo aceptar?**

McCall (1970) escrito difícil

Problema del agente:

$$V = \max_{T} \mathbb{E}_0 \left[\sum_{t=0}^{T-1} \beta^t u(b) + \sum_{t=T}^{\infty} \beta^t u(w_T) \right]$$

$$\text{sujeto a } w_t \overset{iid}{\sim} F(\cdot)$$

$$T \text{ debe ser adaptado a } \mathcal{F}(\{w_t\})$$

- T es una función de los salarios w_t sacados antes de T.
- Cómo elijo T? En qué conjunto vive T? Cuál es la CPO?

McCall (1970) escrito difícil

Problema del agente:

$$V = \max_{T} \mathbb{E}_0 \left[\sum_{t=0}^{T-1} \beta^t u(b) + \sum_{t=T}^{\infty} \beta^t u(w_T) \right]$$

$$\text{sujeto a } w_t \overset{iid}{\sim} F(\cdot)$$

$$T \text{ debe ser adaptado a } \mathcal{F}(\{w_t\})$$

- T es una función de los salarios w_t sacados antes de T.
- Cómo elijo T? En qué conjunto vive T? Cuál es la CPO?

McCall (1970) escrito difícil

Problema del agente:

$$V = \max_{T} \mathbb{E}_0 \left[\sum_{t=0}^{T-1} \beta^t u(b) + \sum_{t=T}^{\infty} \beta^t u(w_T) \right]$$

$$\text{sujeto a } w_t \overset{iid}{\sim} F(\cdot)$$

$$T \text{ debe ser adaptado a } \mathcal{F}(\{w_t\})$$

- T es una función de los salarios w_t sacados antes de T.
- Cómo elijo T? En qué conjunto vive T? Cuál es la CPO?


COME TO THE
DARK
SIDE
WE HAVE COOKIES

El Lado Oscuro

- En t, si todavía no acepté una oferta, después de ver w_t

$$V_t = \begin{cases} \sum_{j=0}^{\infty} \beta^j u(w_t) & \text{si acepto} \\ u(b) + \beta \max_T \mathbb{E}\left[\sum_{j=0}^{T} \beta^j u(b) + \sum_{j=T}^{\infty} \beta^j u(w_T)\right] & \text{si rechazo} \end{cases}$$

- Así que

$$V_t = \max \left\{ u(b) + \beta \mathbb{E}\left[V_{t+1}\right], R(w_t) \right\}$$

- MAGIA: V_t no depende de t dado w_t

El Lado Oscuro

- En t, si todavía no acepté una oferta, después de ver w_t

$$V_t = \begin{cases} \sum_{j=0}^{\infty} \beta^j u(w_t) = R(w_t) & \text{si acepto} \\ u(b) + \beta \max_T \mathbb{E}\left[\sum_{j=0}^{T} \beta^j u(b) + \sum_{j=T}^{\infty} \beta^j u(w_T)\right] & \text{si rechazo} \end{cases}$$

- Así que

$$V_t = \max\left\{u(b) + \beta \mathbb{E}\left[V_{t+1}\right], R(w_t)\right\}$$

- MAGIA: V_t no depende de t dado w_t

El Lado Oscuro

- En t, si todavía no acepté una oferta, después de ver w_t

$$V_t = \begin{cases} \sum_{j=0}^{\infty} \beta^j u(w_t) = R(w_t) & \text{si acepto} \\ u(b) + \beta \max_T \mathbb{E}\left[\sum_{j=0}^{T} \beta^j u(b) + \sum_{j=T}^{\infty} \beta^j u(w_T)\right] & \text{si rechazo} \end{cases}$$

- Así que

$$V_t = \max\left\{u(b) + \beta\mathbb{E}\left[V_{t+1}\right], R(w_t)\right\}$$

- MAGIA: V_t no depende de t dado w_t

El Lado Oscuro

- En t, si todavía no acepté una oferta, después de ver w_t

$$V_t = \begin{cases} \sum_{j=0}^{\infty} \beta^j u(w_t) = R(w_t) & \text{si acepto} \\ u(b) + \beta \max_T \mathbb{E}\left[\sum_{j=0}^{T} \beta^j u(b) + \sum_{j=T}^{\infty} \beta^j u(w_T)\right] & \text{si rechazo} \end{cases}$$

- Así que

$$V_t = \max\left\{u(b) + \beta \mathbb{E}\left[V_{t+1}\right], R(w_t)\right\}$$

- MAGIA: V_t no depende de t dado w_t

El Lado Oscuro

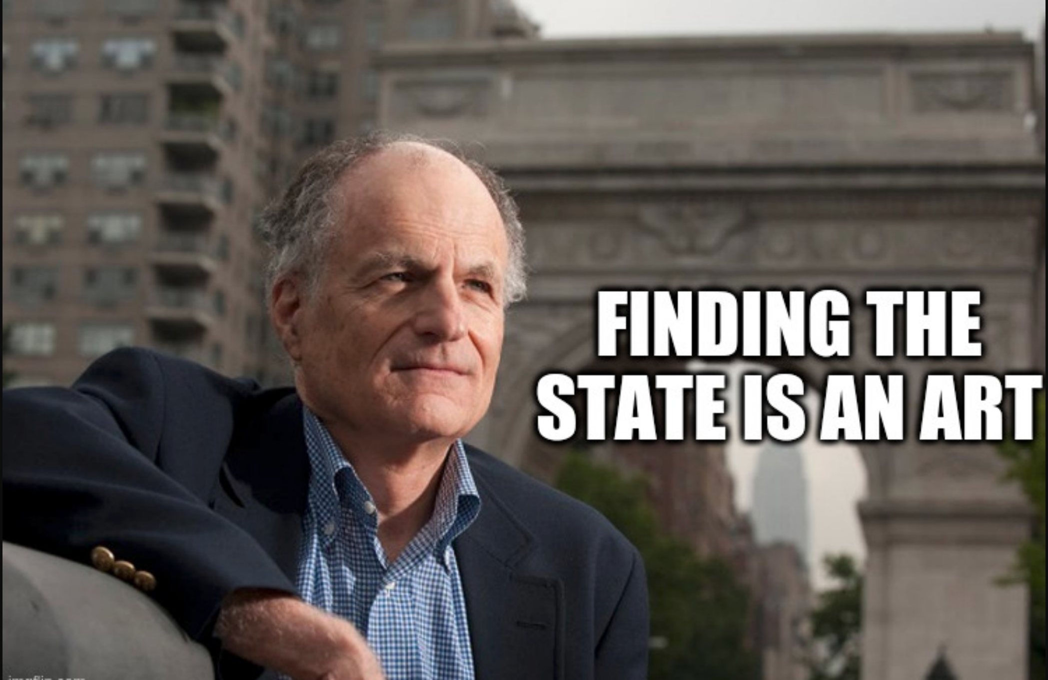
- En t, si todavía no acepté una oferta, después de ver w_t

$$V_t = \begin{cases} \sum_{j=0}^{\infty} \beta^j u(w_t) = R(w_t) & \text{si acepto} \\ u(b) + \beta \max_T \mathbb{E}\left[\sum_{j=0}^{T} \beta^j u(b) + \sum_{j=T}^{\infty} \beta^j u(w_T)\right] & \text{si rechazo} \end{cases}$$

- Así que

$$V_t = \max \left\{ u(b) + \beta \mathbb{E}\left[V_{t+1}\right], R(w_t) \right\}$$

- MAGIA: V_t no depende de t dado w_t


FINDING THE
STATE IS AN ART

McCall (1970) escrito recursivo

$$V(w_t) = \max \left\{ u(b) + \beta \mathbb{E}\left[V(w_{t+1})\right], R(w_t) \right\}$$

- Programación dinámica a mano

$$R(w) = \sum_{j=0}^{\infty} \beta^j u(w)$$

McCall (1970) escrito recursivo

$$V(w_t) = \max \left\{ u(b) + \beta \mathbb{E} \left[V(w_{t+1}) \right], R(w_t) \right\}$$

- Programación dinámica a mano

$$R(w) = \sum_{j=0}^{\infty} \beta^j u(w)$$

McCall (1970) escrito recursivo

$$V(w_t) = \max \left\{ u(b) + \beta \mathbb{E}\left[V(w_{t+1})\right], R(w_t) \right\}$$

- Programación dinámica a mano

$$R(w) = \sum_{j=0}^{\infty} \beta^j u(w) = u(w) + \sum_{j=1}^{\infty} \beta^j u(w)$$

McCall (1970) escrito recursivo

$$V(w_t) = \max\left\{u(b) + \beta\mathbb{E}\left[V(w_{t+1})\right], R(w_t)\right\}$$

- Programación dinámica a mano

$$R(w) = \sum_{j=0}^{\infty} \beta^j u(w) = u(w) + \sum_{j=1}^{\infty} \beta^j u(w)$$

$$= u(w) + \beta \sum_{j=0}^{\infty} \beta^j u(w)$$

McCall (1970) escrito recursivo

$$V(w_t) = \max \left\{ u(b) + \beta \mathbb{E}\left[V(w_{t+1})\right], R(w_t) \right\}$$

- Programación dinámica a mano

$$R(w) = \sum_{j=0}^{\infty} \beta^j u(w) = u(w) + \sum_{j=1}^{\infty} \beta^j u(w)$$

$$= u(w) + \beta \sum_{j=0}^{\infty} \beta^j u(w)$$

McCall (1970) escrito recursivo

$$V(w_t) = \max \left\{ u(b) + \beta \mathbb{E}\left[V(w_{t+1})\right], R(w_t) \right\}$$

- Programación dinámica a mano

$$\begin{aligned} R(w) &= \sum_{j=0}^{\infty} \beta^j u(w) = u(w) + \sum_{j=1}^{\infty} \beta^j u(w) \\ &= u(w) + \beta \sum_{j=0}^{\infty} \beta^j u(w) = u(w) + \beta R(w) \end{aligned}$$

McCall (1970) escrito recursivo

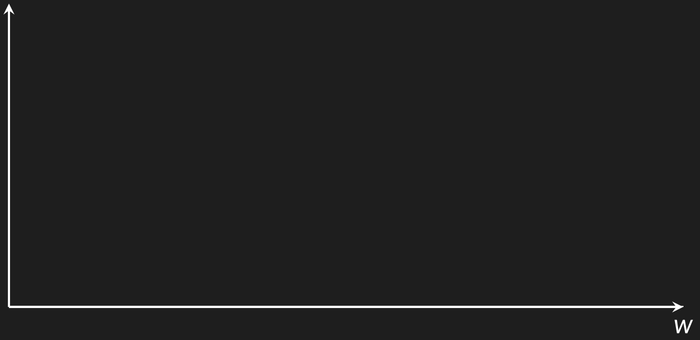
$$V(w_t) = \max \left\{ u(b) + \beta \mathbb{E}\left[V(w_{t+1})\right], R(w_t) \right\}$$

- Programación dinámica a mano

$$\begin{aligned} R(w) &= \sum_{j=0}^{\infty} \beta^j u(w) = u(w) + \sum_{j=1}^{\infty} \beta^j u(w) \\ &= u(w) + \beta \sum_{j=0}^{\infty} \beta^j u(w) = u(w) + \beta R(w) \\ &= \frac{u(w)}{1-\beta} \end{aligned}$$

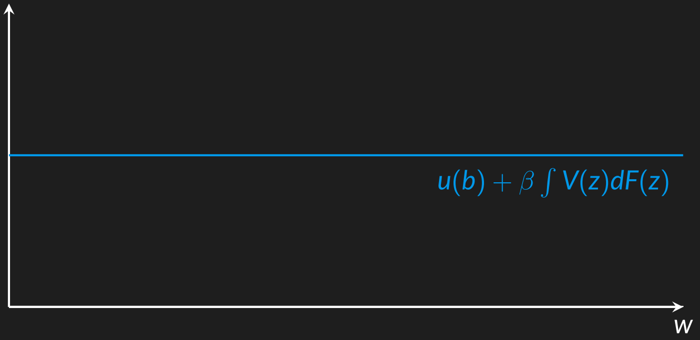
McCall (1970) escrito recursivo

$$V(w) = \max\left\{u(b) + \beta \int V(z)dF(z), \frac{u(w)}{1-\beta}\right\}$$



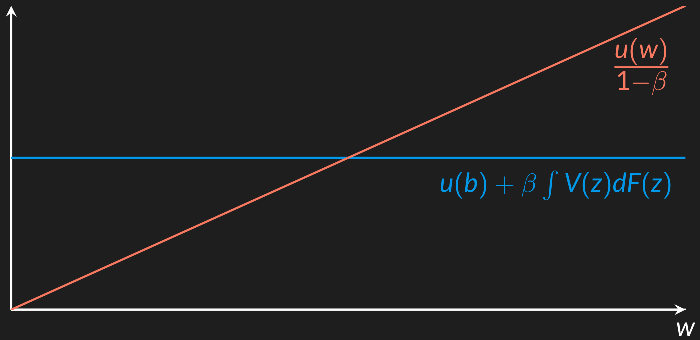
McCall (1970) escrito recursivo

$$V(w) = \max \left\{ u(b) + \beta \int V(z)dF(z), \frac{u(w)}{1-\beta} \right\}$$



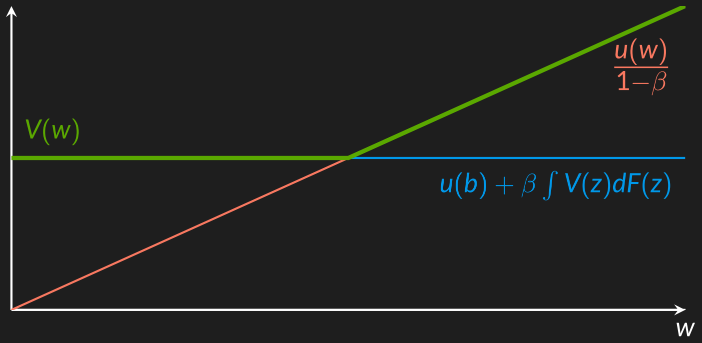
McCall (1970) escrito recursivo

$$V(w) = \max\left\{u(b) + \beta \int V(z)dF(z), \frac{u(w)}{1-\beta}\right\}$$



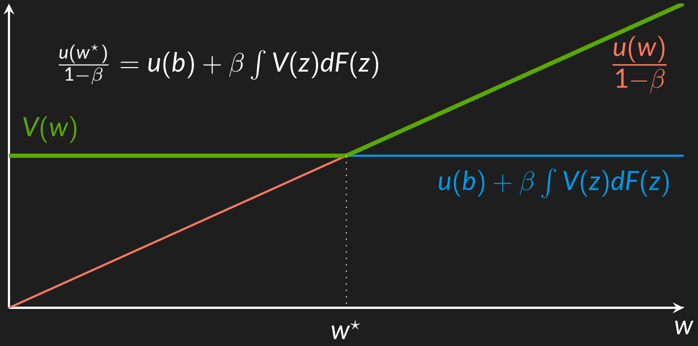
McCall (1970) escrito recursivo

$$V(w) = \max\left\{u(b) + \beta \int V(z)dF(z), \frac{u(w)}{1-\beta}\right\}$$



McCall (1970) escrito recursivo

$$V(w) = \max\left\{u(b) + \beta \int V(z)dF(z), \frac{u(w)}{1-\beta}\right\}$$

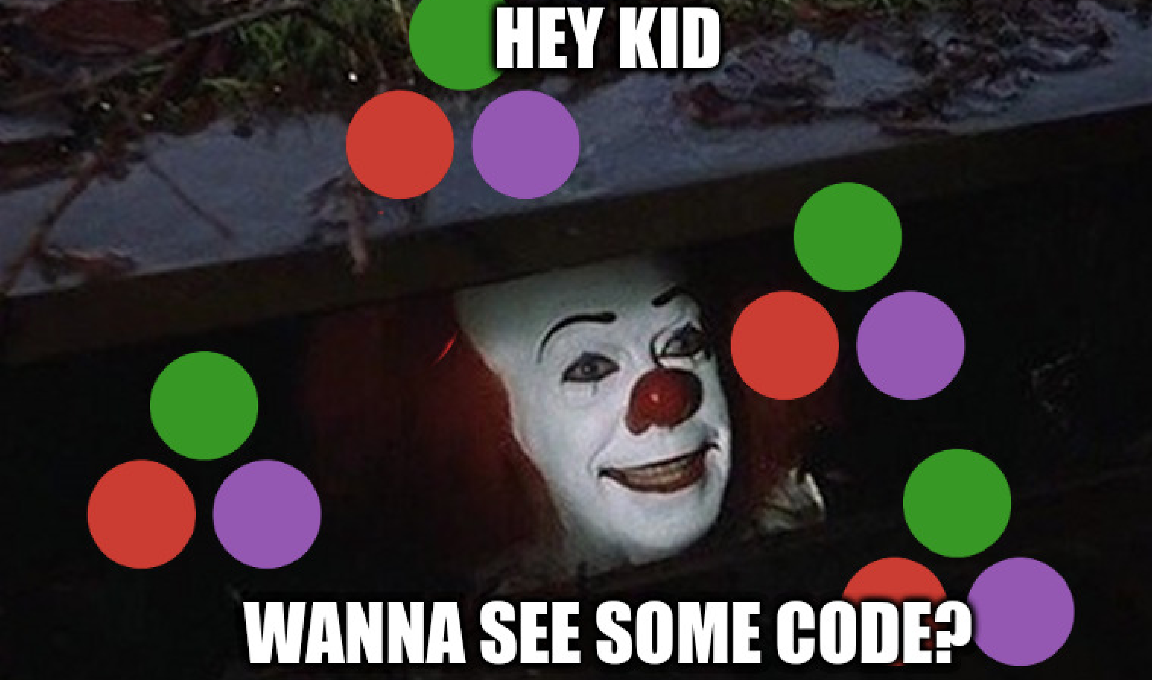



HEY KID
WANNA SEE SOME CODE?


HEY KID
WANNA SEE SOME CODE?


HEY KID
WANNA SEE SOME CODE?


HEY KID
WANNA SEE SOME CODE?


HEY KID
WANNA SEE SOME CODE?

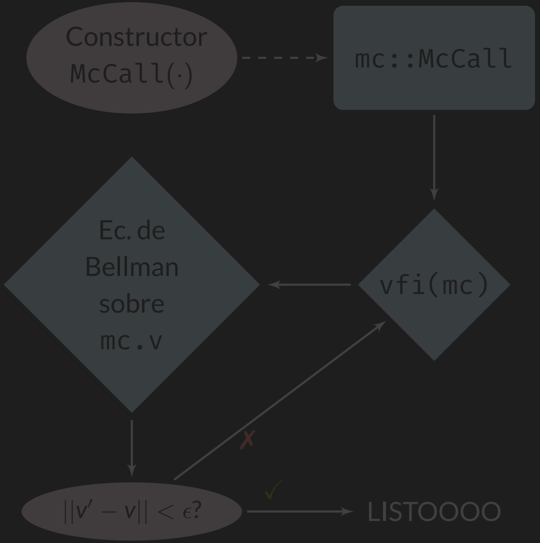

HEY KID
WANNA SEE SOME CODE?

McCall (1970) en Julia

Siempre escribir pseudo código

Estructura

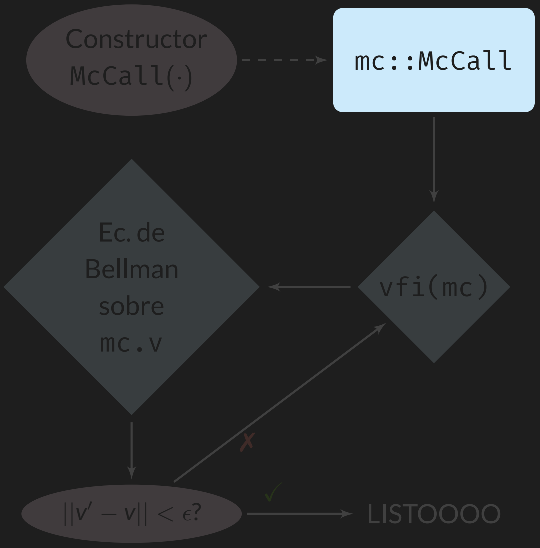
- Un objeto con el modelo
  - Parámetros
  - Grillas para las variables
  - Solución
    decisiones de los agentes, funciones de valor
- Funciones que manejen ese objeto



Siempre escribir pseudo código

Estructura

- Un objeto con el modelo
  - Parámetros
  - Grillas para las variables
  - Solución
    decisiones de los agentes, funciones de valor
- Funciones que manejen ese objeto

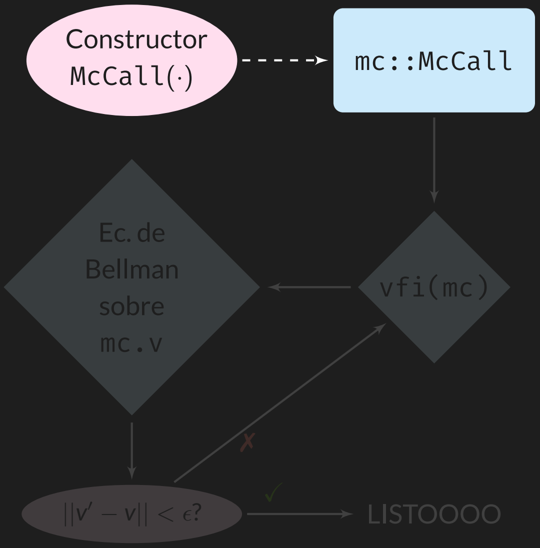


Siempre escribir pseudo código

Estructura

- Un objeto con el modelo
  - Parámetros
  - Grillas para las variables
  - Solución

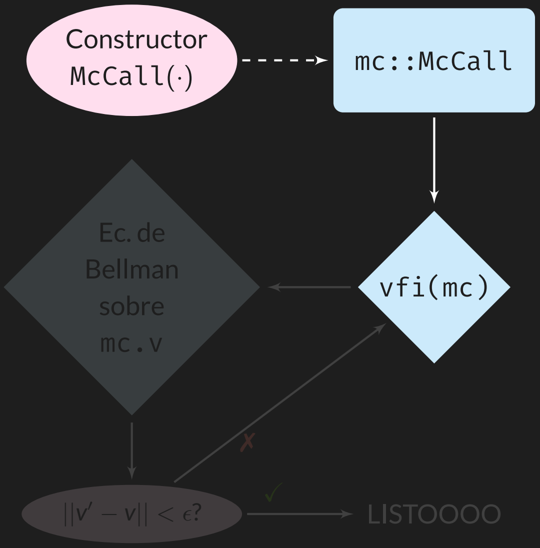
    decisiones de los agentes, funciones de valor
- Funciones que manejen ese objeto



Siempre escribir pseudo código

Estructura

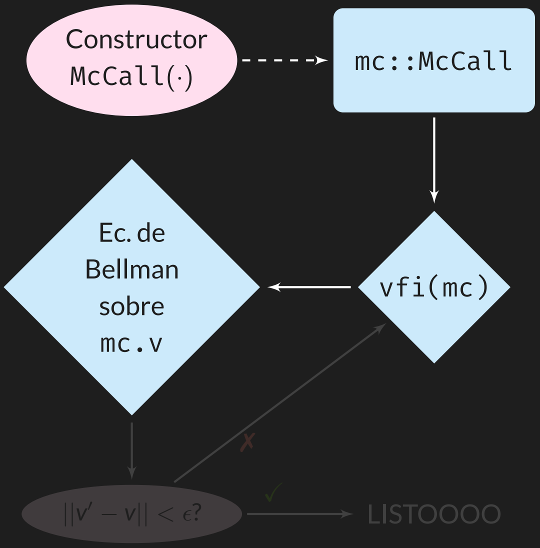
- Un objeto con el modelo
  - Parámetros
  - Grillas para las variables
  - Solución
    decisiones de los agentes, funciones de valor
- Funciones que manejen ese objeto



Siempre escribir pseudo código

Estructura

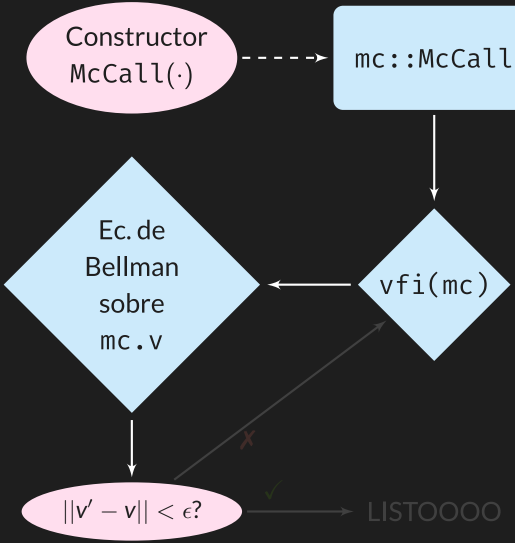
- Un objeto con el modelo
    - Parámetros
    - Grillas para las variables
    - Solución
      decisiones de los agentes, funciones de valor
- Funciones que manejen ese objeto



Siempre escribir pseudo código

Estructura

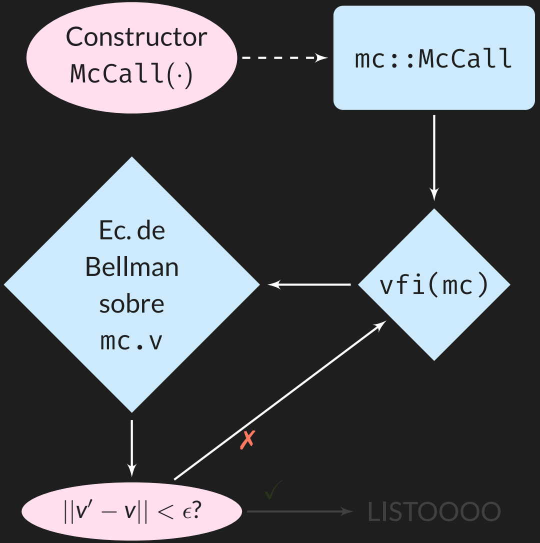
- Un objeto con el modelo
  - Parámetros
  - Grillas para las variables
  - Solución
    decisiones de los agentes, funciones de valor
- Funciones que manejen ese objeto



Siempre escribir pseudo código

Estructura

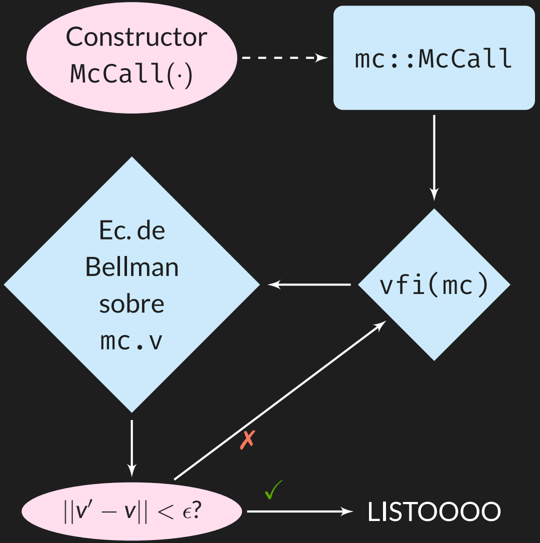
- Un objeto con el modelo
  - Parámetros
  - Grillas para las variables
  - Solución
    decisiones de los agentes,
    funciones de valor
- Funciones que manejen ese objeto



Siempre escribir pseudo código

Estructura

- Un objeto con el modelo
  - Parámetros
  - Grillas para las variables
  - Solución
    decisiones de los agentes, funciones de valor
- Funciones que manejen ese objeto



Tipos abstractos y concretos

Estructuras de datos

- Simples
  - `Float64`, `Int64`, `String`, `Function`, ...
- Compuestos
  - `Vector{Int64}`, `Matrix{String}`, `DataFrame`, ...
- Abstractos
  - `Any`, `Number` (`Float64 <: Number <: Any`)
  - `AbstractArray`
- Creados por uno
  - Vamos a introducir un tipo `McCall` para guardar el planteo y la solución del modelo

- Multiple dispatch
  - La misma función puede tener efectos diferentes según el tipo de datos de sus argumentos
  - En ese caso *f* tiene distintos métodos

Tipos abstractos y concretos

Estructuras de datos

- Simples
  - `Float64`, `Int64`, `String`, `Function`, ...
- Compuestos
  - `Vector{Int64}`, `Matrix{String}`, `DataFrame`, ...
- Abstractos
  - `Any`, `Number` (`Float64 <: Number <: Any`)
  - `AbstractArray`
- Creados por uno
  - Vamos a introducir un tipo `McCall` para guardar el planteo y la solución del modelo

- Multiple dispatch
  - La misma función puede tener efectos diferentes según el tipo de datos de sus argumentos
  - En ese caso f tiene distintos métodos

Tipos concretos

```
mutable struct McCall
    β::Float64
    γ::Float64

    b::Float64

    wgrid::Vector{Float64}
    pw::Vector{Float64}

    w_star::Float64
    v::Vector{Float64}
end
```

- Declara un nuevo tipo `McCall`
- Una vez declarado, permite ‘`x::McCall`’
- Objetos de tipo `McCall` pueden ser modificados (`mutable`)
- Si `x::McCall`, entonces `x.β` debe ser `Float64`
- Queremos guardar
  1. Parámetros de la función de utilidad
  2. Grilla de puntos para w
  3. Probabilidad de cada w
  4. Estimación de $w^\star$, $v(w)$

Funciones básicas

```
function u(mc::McCall, c)
    γ = mc.γ
    if γ == 1
        return log(c)
    else
        return c^(1-γ) / (1-γ)
    end
end
function R(mc::McCall, w)
    β = mc.β
    return u(mc, w) / (1-β)
end
```

- `u` usa el modelo y un número `c` y calcula la función de utilidad en ese número
- `R` usa el modelo y un número `w` y calcula el valor de aceptar una oferta `w`
- `R` usa la definición de `u`

Constructores

- La declaración del tipo `McCall` que hicimos también crea un constructor
- El constructor permite

```
x = McCall(β, γ, b, wgrid, w_star, pw, v)
```

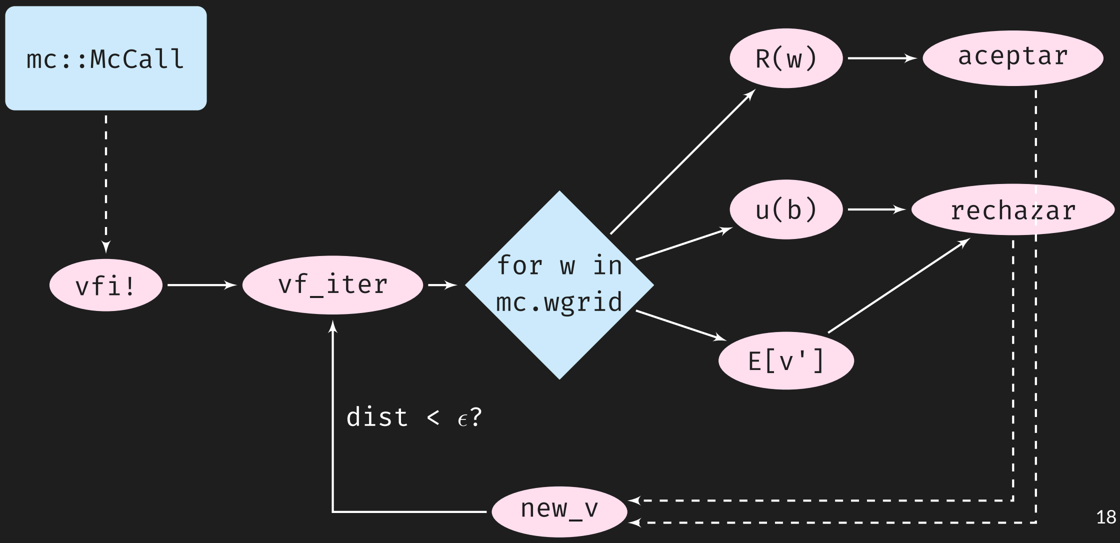
siempre que los argumentos sean de los tipos correctos (el constructor tiene un único método)

Constructor

```julia
function McCall(; β = 0.96, γ = 0, b = 1, μw = 1, σw = 0.05,
    wmin = 0.1, wmax = 4, Nw = 50)

    wgrid = range(wmin, wmax, length=Nw)
    w_star = first(wgrid)
    d = Normal(μw, σw)
    pw = [pdf(d, wv) for wv in wgrid]
    pw = pw / sum(pw)
    v = zeros(Nw)
    return McCall(β, γ, b, wgrid, w_star, pw, v)
end
```

Pseudo-código



Valor de continuación

```
function E_v(mc::McCall)
    Ev = 0.0
    for (jwp, wpv) in enumerate(mc.wgrid)
        Ev += mc.pw[jwp] * mc.v[jwp]
    end
    return Ev
end

E_v2(mc::McCall) = sum( [ mc.pw[jwp]*mc.v[jwp] for (jwp,wpv)
    in enumerate(mc.wgrid) ] )

E_v3(mc::McCall) = mc.pw'*mc.v
```

Iterador


```julia
function vf_iter(mc::McCall, flag = 0)
    new_v = similar(mc.v)
    for (jw, wv) in enumerate(mc.wgrid)
        aceptar = R(mc, wv)
        rechazar = u(mc, mc.b) + mc.β * E_v(mc)
        new_v[jw] = max(aceptar, rechazar)
        if flag == 0 && aceptar >= rechazar
            mc.w_star = wv
            flag = 1
        end
    end
    return new_v
end
```

```julia
function vfi!(mc::McCall; maxiter = 200, tol = 1e-6)
    dist = 1+tol
    iter = 0
    while dist > tol && iter < maxiter
        iter += 1
        new_v = vf_iter(mc)
        dist = norm(mc.v - new_v)
        mc.v = new_v
    end
    print("Finished in $(iter) iterations. Dist = $(dist)")
end
```

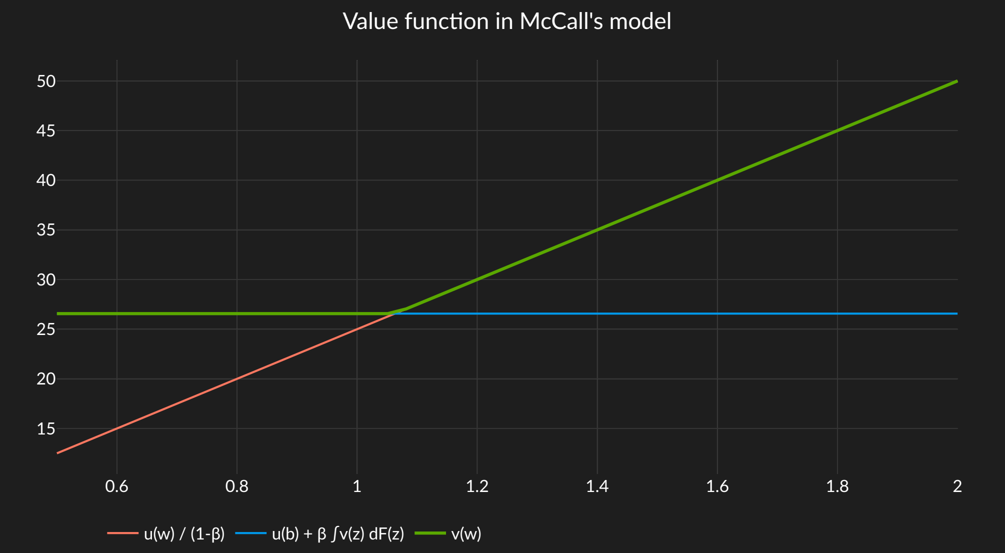
Ensamblar todo

```
include("mccall.jl")
mc = McCall()
vfi!(mc)
make_plots(mc)
```

1. Cargar/actualizar códigos
2. Crear instancia de `McCall`

   ... usando el método default
3. Usar la función `vfi!` modificando a `mc`
4. (No vimos cómo graficar) PlotlyJS

7 diferencias

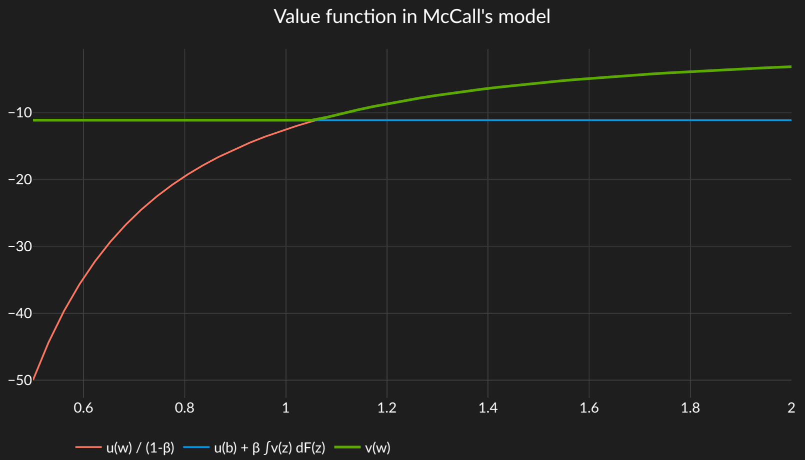


Ahora con aversión al riesgo

```
mc = McCall(γ = 3)
```

Ahora con aversión al riesgo

```
mc = McCall(γ = 3)
```



Eficiencia

Iterador

```julia
function vf_iter!(new_v, mc::McCall, flag = 0)
    rechazar = u(mc, mc.b) + mc.β * E_v(mc)
    for (jw, wv) in enumerate(mc.wgrid)
        aceptar = R(mc, wv)
        new_v[jw] = max(aceptar, rechazar)
        if flag == 0 && aceptar >= rechazar
            mc.w_star = wv
            flag = 1
        end
    end
end
```

Eficiencia

Para usar `vf_iter!(·)` hay que modificar el loop principal

```
function vfi2!(mc::McCall; maxiter = 200, tol = 1e-8)
    dist, iter = 1+tol, 0
    new_v = similar(mc.v)
    while dist > tol && iter < maxiter
        iter += 1
        vf_iter!(new_v, mc)
        dist = norm(mc.v - new_v)
        mc.v = copy(new_v)
    end
    print("Finished in $iter iterations. Dist = $dist")
end
```

Simulación

▸ Truco

```julia
function simul(mc::McCall, flag = 0; maxiter = 2000)
    t = 0
    while flag == 0 && t < maxiter
        t += 1
        jw = findfirst(cumsum(mc.pw) .>= rand())
        wt = mc.wgrid[jw]
        print("Salario en el período $t: $wt. ")
        wt >= mc.w_star ? flag = 1 : println("Sigo buscando")
    end
    flag == 1 && println("Oferta aceptada en $t períodos")
end
```

Cierre

Cierre

Vimos

- Programación dinámica
  - Finding the state is an art
  - Iterar sobre la ecuación de Bellman es 90% de un algoritmo
  - Por qué funciona?
- El modelo de búsqueda más sencillo posible
- Código en Julia para resolverlo
  - Cómo cambia la solución en función de los parámetros?
  - Simulación.
    - Ejercicio: cuántos períodos espero tener que buscar? cuánto es $\mathbb{E}[T]$?

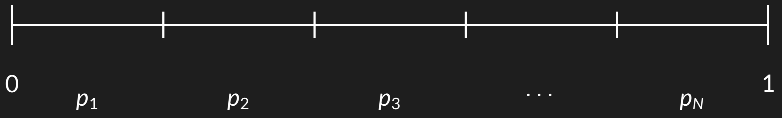
Cierre

Vimos

- Programación dinámica
  - Finding the state is an art
  - Iterar sobre la ecuación de Bellman es 90% de un algoritmo
  - Por qué funciona?
- El modelo de búsqueda más sencillo posible
- Código en Julia para resolverlo
  - Cómo cambia la solución en función de los parámetros?
  - Simulación.
    - Ejercicio: cuántos períodos espero tener que buscar? cuánto es $\mathbb{E}[T]$?

Por qué funciona `cumsum`?

- **Problema:** quiero sacar un número aleatorio con
  - valores en un vector $x = (x_1, \ldots, x_N)$
  - probabilidades $p = (p_1, \ldots, p_N)$



  - Los nodos son $\{0, p_1, p_1 + p_2, \ldots, \sum_i p_i = 1\}$
  - Si saco $z \sim \mathcal{U}(0, 1)$ y defino y como el intervalo en el que cayó z, y tiene la distribución que quiero
  - $\mathbb{P}(y = x_j) = \sum_{i=1}^{j} p_i - \sum_{i=1}^{j-1} p_i = p_j$.

PlotlyJS


```
function make_plots(mc::McCall)
  Rw = [R(mc, wv) for wv in mc.wgrid]
  at = scatter(x=mc.wgrid, y=Rw, line_color="#f97760", name = "u(w) / (1-β)")
  Ub = [u(mc, mc.b) + mc.β * E_v(mc) for wv in mc.wgrid]
  rt = scatter(x=mc.wgrid, y=Ub, line_color="#0098e9", name = "u(b) + β∫
v(z) dF(z)")
  opt = scatter(x=mc.wgrid, y=mc.v, line_color="#5aa800", line_width = 3, name = "v(w)")
  traces = [at, rt, opt]

  layout = Layout(title = "Value function in McCall's model",
    width = 1920*0.5, height = 1080*0.5,
    legend = attr(orientation = "h", x = 0.05),
    xaxis = attr(zeroline = false, gridcolor="#353535"),
    yaxis = attr(zeroline = false, gridcolor="#353535"),
    paper_bgcolor="#1e1e1e", plot_bgcolor="#1e1e1e",
    font_color="white", font_size = 16, font_family = "Lato",
    )
  plot(traces)
end
```