Macroeconomía Internacional

Francisco Roldán
IMF

October 2021

Hoy

Código para resolver

- McCall (1970)
- El problema de la torta

McCall (1970) en Julia

Problema de búsqueda

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

Algoritmo

- Elegir una función $v^0 : \mathbb{R}_+ \to \mathbb{R}_+$
- Aplicarle a v^n la ecuación de Bellman para obtener $v^{(n+1)}$
- Comparar v^n con $v^{(n+1)}$

Problema de búsqueda

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

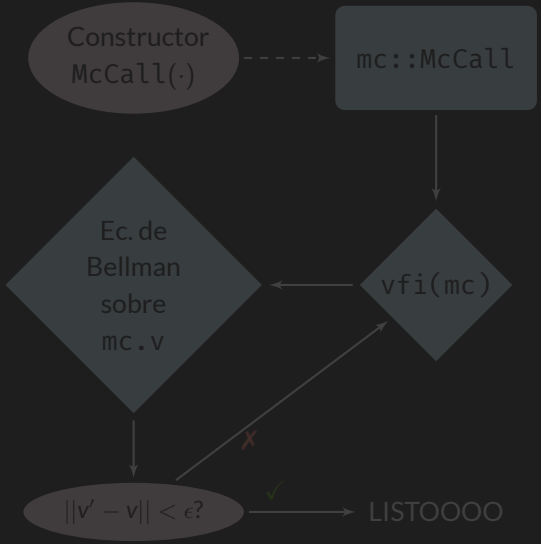
Algoritmo

- Elegir una función $v^0 : \mathbb{R}_+ \to \mathbb{R}_+$
- Aplicarle a v^n la ecuación de Bellman para obtener $v^{(n+1)}$
- Comparar v^n con $v^{(n+1)}$

Siempre escribir pseudo código

Estructura

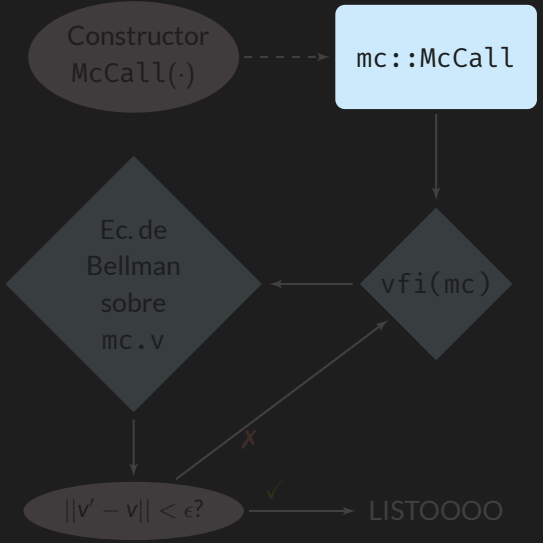
- Un objeto con el modelo
  - Parámetros
  - Grillas para las variables
  - Solución
    decisiones de los agentes, funciones de valor
- Funciones que manejen ese objeto



Siempre escribir pseudo código

Estructura

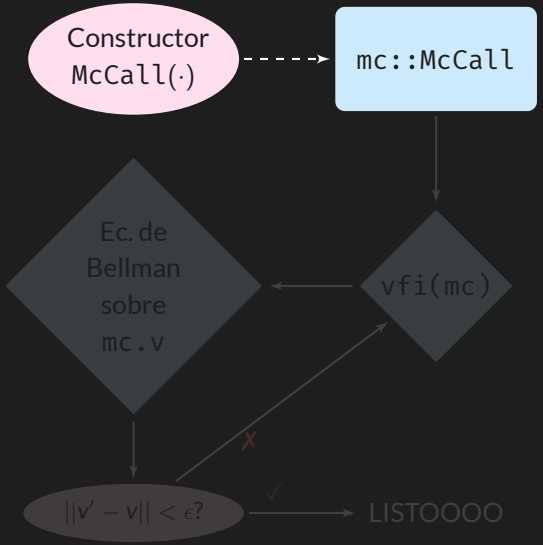
- Un objeto con el modelo
  - Parámetros
  - Grillas para las variables
  - Solución
    decisiones de los agentes, funciones de valor
- Funciones que manejen ese objeto



Siempre escribir pseudo código

Estructura

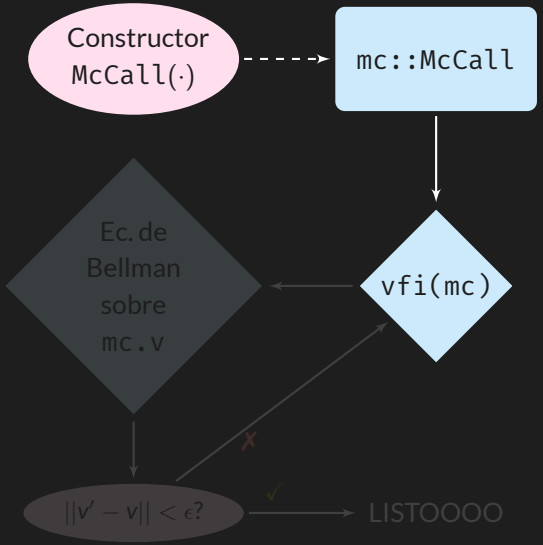
- Un objeto con el modelo
  - Parámetros
  - Grillas para las variables
  - Solución
    decisiones de los agentes, funciones de valor
- Funciones que manejen ese objeto



Siempre escribir pseudo código

Estructura

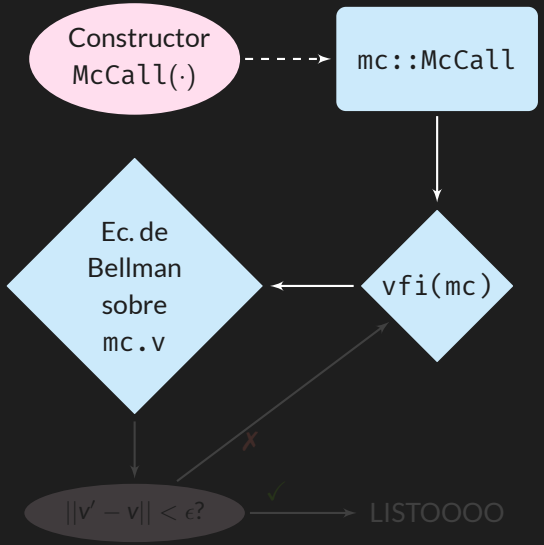
- Un objeto con el modelo
  - Parámetros
  - Grillas para las variables
  - Solución
    decisiones de los agentes, funciones de valor
- Funciones que manejen ese objeto



Siempre escribir pseudo código

Estructura

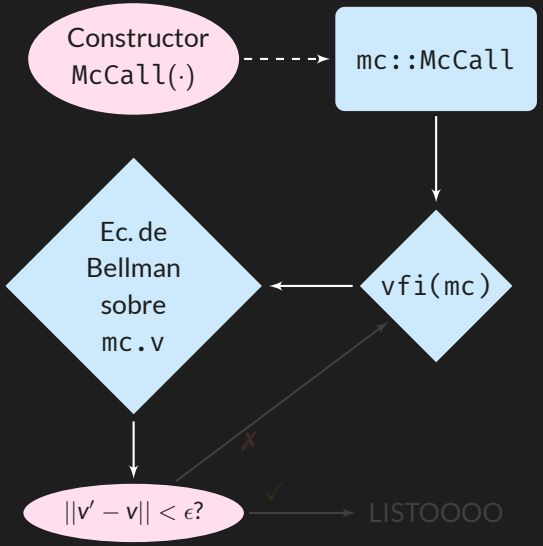
- Un objeto con el modelo
  - Parámetros
  - Grillas para las variables
  - Solución
    decisiones de los agentes, funciones de valor
- Funciones que manejen ese objeto



Siempre escribir pseudo código

Estructura

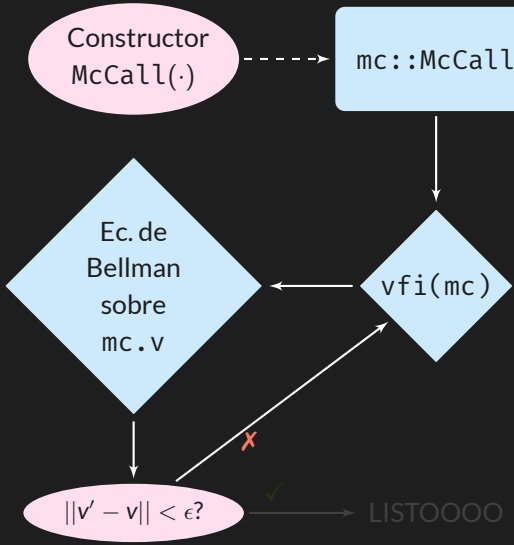
- Un objeto con el modelo
  - Parámetros
  - Grillas para las variables
  - Solución
    decisiones de los agentes, funciones de valor
- Funciones que manejen ese objeto



Siempre escribir pseudo código

Estructura

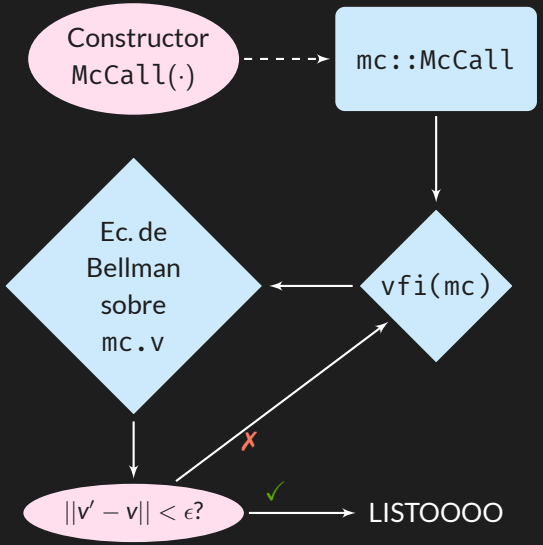
- Un objeto con el modelo
  - Parámetros
  - Grillas para las variables
  - Solución
    decisiones de los agentes, funciones de valor
- Funciones que manejen ese objeto



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 y probabildad para w
  3. Estimación de $w^\star$, $v(w)$

Constructores

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

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

siempre que los argumentos sean de los tipos correctos

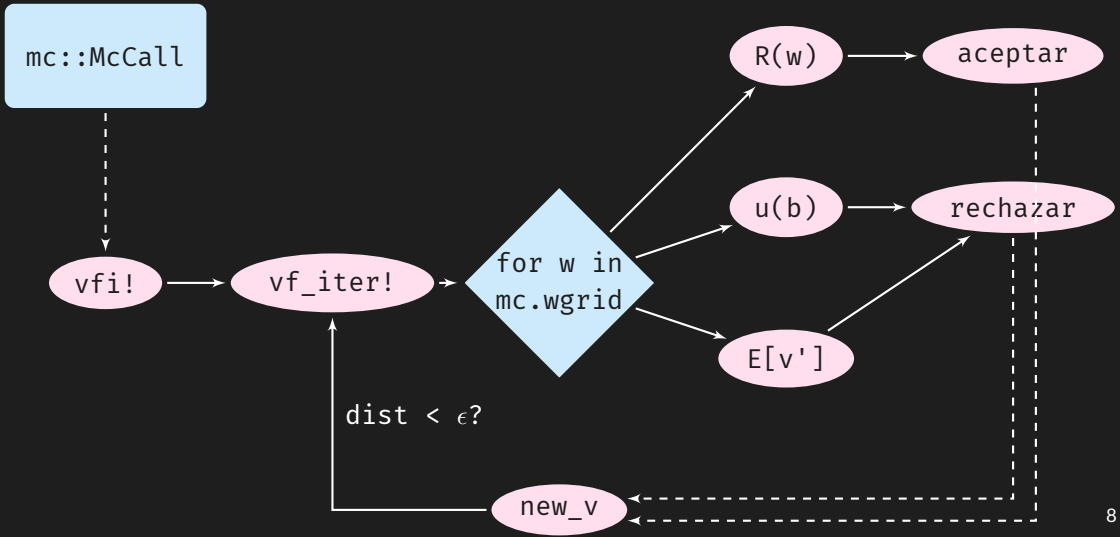
(el constructor tiene un único método)

Constructor

```
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, pw, w_star, v)
end
```

Pseudo-código



Funciones básicas

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

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

```
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
```

Loop

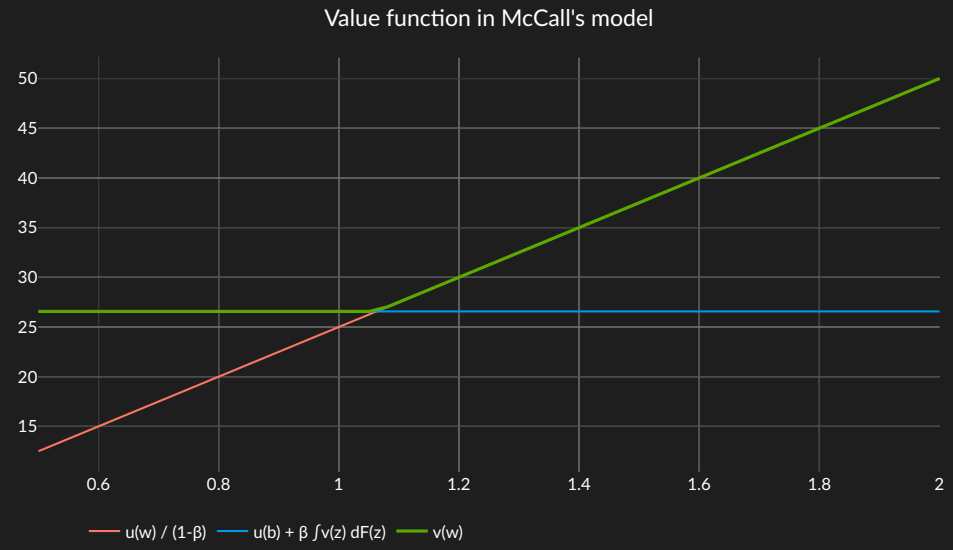
```
function vfi!(mc::McCall; maxiter = 200, tol = 1e-8)
   dist, iter = 1+tol, 0
   new_v = similar(mc.v)
   while dist > tol && iter < maxiter
      iter += 1
      vf_iter!(new_v, mc)
      dist = norm(mc.v - new_v)
      mc.v .= new_v
   end
   print("Finished in $iter iterations. Dist = $dist")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

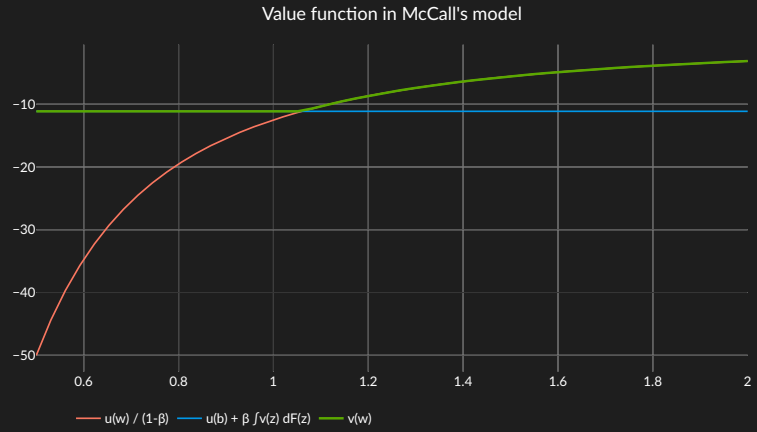


Ahora con aversión al riesgo

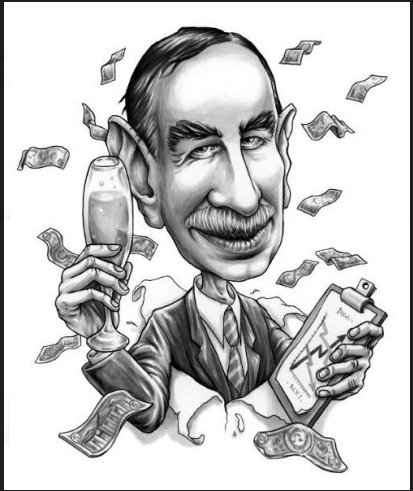
```
mc = McCall(γ = 3)
```

Ahora con aversión al riesgo

```
mc = McCall(γ = 3)
```



Perdón si te reusé el mismo código



Simulación

Truco

```
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
```

Problema de la torta

Problema de la torta recursivo

$$v(k) = \max_{c,k'} u(c) + \beta v(k')$$

$$\text{sujeto a } c + k' = k(1 + r)$$

$$k' \geq 0$$

- La función v es desconocida
- Podemos
  1. meter una f cualquiera del lado derecho (reemplazar v por f)
  2. usar la ec. de Bellman para encontrar una nueva f
  3. comparar la f que entró con la f que salió
  4. usar la f que salió del lado derecho

Problema de la torta recursivo

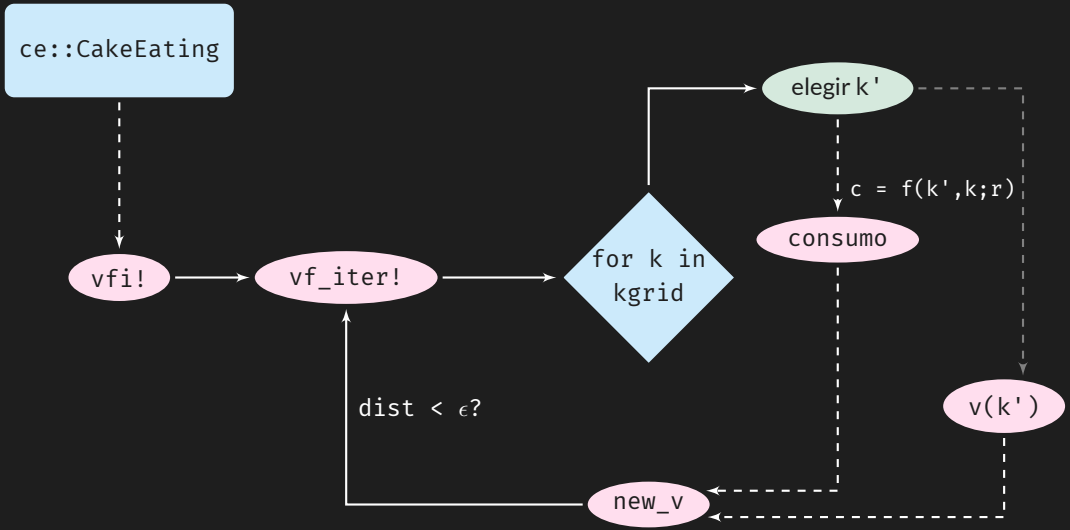
$$v(k) = \max_{c,k'} u(c) + \beta v(k')$$

$$\text{sujeto a } c + k' = k(1 + r)$$

$$k' \geq 0$$

- La función v es desconocida
- Podemos
  1. meter una f cualquiera del lado derecho (reemplazar v por f)
  2. usar la ec. de Bellman para encontrar una nueva f
  3. comparar la f que entró con la f que salió
  4. usar la f que salió del lado derecho

Pseudo-código



Cómo elegir k'?

Dos formas

- Forma lenta
  - Recorrer `kgrid` elemento por elemento y probarlos a todos
  - Super robusto, fácil de implementar
  - Iterar la ecuación de Bellman requiere K^2 combinaciones de (k, k')
- Forma un poco mejor
  - Elegir $k' \in [\underline{k}, \bar{k}]$ con algún algoritmo continuo
    - Precisión en la elección de k' *no depende* de la cantidad de puntos de `kgrid`
  - Requiere evaluar $v(k')$ para valores de $k \notin$ `kgrid`
    - `Interpolations.jl`

Cómo elegir k'?

Dos formas

- Forma lenta
  - Recorrer `kgrid` elemento por elemento y probarlos a todos
  - Super robusto, fácil de implementar
  - Iterar la ecuación de Bellman requiere K^2 combinaciones de (k, k')
- Forma un poco mejor
  - Elegir $k' \in [\underline{k}, \bar{k}]$ con algún algoritmo continuo
    - Precisión en la elección de k' *no depende* de la cantidad de puntos de `kgrid`
  - Requiere evaluar $v(k')$ para valores de $k \notin$ `kgrid`
    - `Interpolations.jl`

Forma lenta

cakeeating.jl

Forma más rápida

`itpcake.jl`

Ojo con multiple dispatch!!

Forma más rápida

`itpcake.jl`

Ojo con multiple dispatch!!

Elaine y las esponjas

Elaine y las esponjas

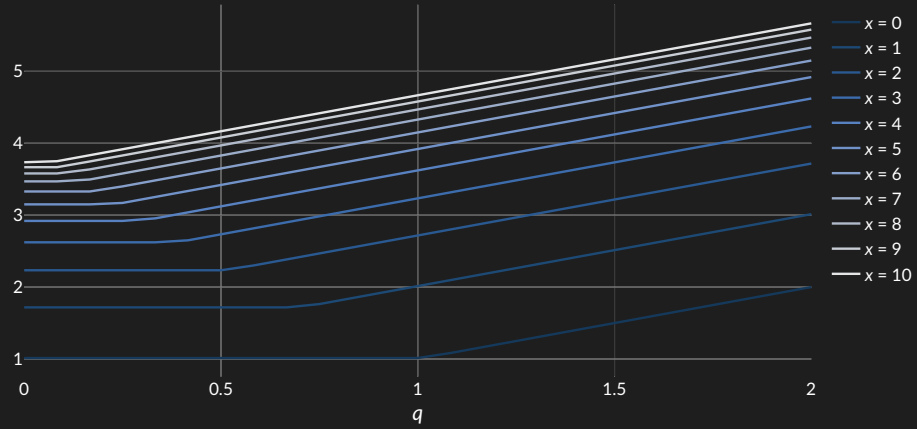
$$v(x,q) = \max\left\{u(q) + \delta\mathbb{E}\left[v(x-1,q')\right], \delta\mathbb{E}\left[v(x,q')\right]\right\}$$

Elaine y las esponjas

$$v(x, q) = \max\left\{u(q) + \delta\mathbb{E}\left[v(x-1, q')\right], \delta\mathbb{E}\left[v(x, q')\right]\right\}$$

Elaine y las esponjas

$$v(x,q) = \max\left\{u(q) + \delta\mathbb{E}\left[v(x-1,q')\right], \delta\mathbb{E}\left[v(x,q')\right]\right\}$$



Cierre

Cierre

Vimos

- Estructuras de datos
- Siempre escribir pseudo-código
- Un poquito de multiple dispatch

- Dos modelos
  - McCall
  - Problema de la torta
    - Eligiendo k' en la grilla original
    - Eligiendo k' de forma continua con interpolaciones

- Problem sets
  - Estática comparada para McCall, usando el simulador
  - Reglas de decisión en el problema de la torta