Introducción a Modelos basados en agentes

Florian Chávez-Juárez

Junio 2019
Curso intensivo Ibero CDMX

Este curso

Bienvenidos!

Muy bienvenidos al curso

Introducción a modelos basados en agentes

Los objetivos principales del curso son:

- Intoducir el concepto de **modelos basados en agentes**
- Introducir el paquete **Netlogo** - software especializado para modelos ABM
- **Hands-on learning**: aprender ambos conceptos de forma práctica e interactiva.

¿Quiénes somos?

Florian Chávez Juárez:

- Profesor en el Laboratorio Nacional de Políticas Públicas (LNPP)
- PhD en econometría de la Universidad de Ginebra
- Coordinador de la *Unidad de Simulación* del LNPP

Correo: `florian.chavez@cide.edu` y `florian@chavezjuarez.com`

Temas de investigación:	modelos basados en agentes, economía de la salud, economía del desarrollo, desigualdad y movilidad social, *behavioural finance*
Técnicas:	Econometría aplicada, modelos basados en agentes

Sitio del curso: `https://github.com/fwchj/CursoABMIbero`

¿Quiénes somos? (2)

Ustedes:

- ¿En qué programa están?
- ¿Cuáles son sus intereses principales?
- ¿Por qué toman este curso?
- ¿Qué se esperan del curso?
- ¿Tienen experiencia en programación? ¿Cuál?

¿Por qué modelos basados en agentes?

El enfoque tradicional de modelaje en economía tiene muchos méritos, pero también algunas limitaciones. Modelos computacionales pueden resolver algunas de esas limitaciones. Por ejemplo, modelos basados en agentes:

1. pueden ser (y normalmente son) **multi-disciplinarios** donde se pueden mezclar fenómenos de economía, sociología, biología, etc
2. pueden incluir **heterogeneidad** tanto en las características como en los comportamientos de los agentes
3. son **flexibles** y se pueden meter muchos procesos
4. pueden fácilmente incluir **interacciones** entre agentes
5. no dependen de **soluciones analíticas**
6. pueden incluir fácilmente elementos modernos como por ejemplo elementos de **economía del comportamiento**.

El menú de la semana

1. **ENTRADA: Modelos basados en agentes (1 día)**
   - Conceptos claves: agentes, reglas de comportamiento, interacción, etc
   - Ventajas y desventajas de modelos ABM
   - Validación de modelos
   - Descripción del modelo: el protocolo ODD
2. **PLATO FUERTE: Introducción a Netlogo (2 días)**
   - Lógica y definiciones en NetLogo
   - Crear agentes, relaciones y el contexto
   - Agregar acciones
   - Obtener resultados del modelo
   - Experimentos
3. **POSTRE: Proyecto práctico (2 días)**
   - Diseño del modelo
   - Implementación del modelo en Netlogo
   - Uso del modelo y análisis de resultados

Ejemplo: predator prey

Vemos un primer modelo pequeño como ejemplo

- Viven ovejas y lobos en el mismo espacio
- El espacio está dividido en pequeños cuadritos (patches)
- En cada periodo la oveja come el pasto que hay en el cuadrito en el cual se encuentra. Después toma un cierto tiempo para que el pasto se recupere.
- Ovejas ganan energía del pasto. Si tienen suficiente energía, se procrean.
- Lobos comen ovejas para ganar energía. Cuando tienen suficiente energía, se procrean.
- Lobos y ovejas sin energía se mueren.

Al final del curso van a poder hacer un modelo como este (de hecho, podrán hacer cosa más complicados).

Modelos basados en agentes

Modelos basados en agentes

Características y uso de modelos basados en agentes

Modelos basados en agentes - ¿qué son?

- Modelos basados en agentes son **modelos computacionales tipo 'bottom-up'**.
- Agentes tienen
  - **Características**: fijas (género) o variable (edad)
  - **Reglas de comportamiento**
- Agentes 'viven' en un **entorno**, **interactúan** con otros agentes y se comportan de acuerdo a las reglas de comportamiento
- Típicamente **no tienen una condición de equilibrio** ni una solución analítica
- Normalmente se implementan con algoritmos en la computadora
- Una ventaja clase es la posibilidad de incluir múltiples **heterogeneidades**:
  - caracterísicas heterogéneas en lugar de un agentes representativo
  - comportamiento heterogéneo
  - heterogeneidad en el entorno

Ejemplos de lo que se desarrolla en la USim

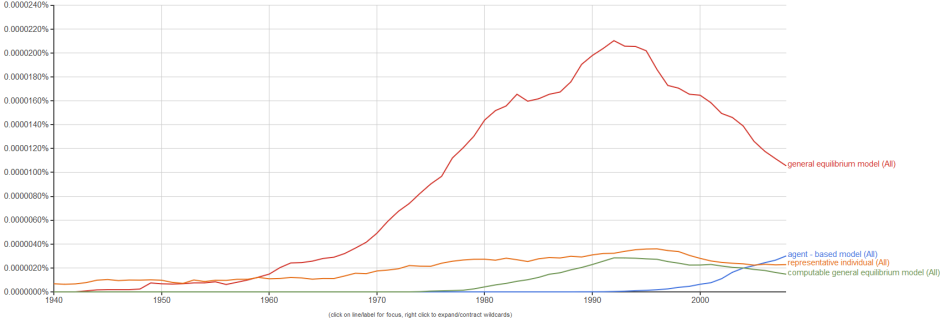
La **Unidad de Simulación** (USim) del LNPP se especializa en el desarrollo de modelos ABM para el análisis de políticas públicas. Unos ejemplo de modelos

- Modelo para estudiar los efectos de cambios as sistema de seguridad social sobre movilidad social, crecimiento y desigualdad.
- Modelo analizando la cooperación en un contexto de desigualdad
- Modelo de economía de la salud
- Modelo de tráfico de la CDMX
- Simulador de impuestos
- Modelo ABM para generar datos de una sociedad artificial para pruebas de modelos econométricos
- Modelo del comportamiento de familias en el mercado financiero

Modelos basados en agentes: una poco de historia

- ABM es un enfoque **relativamente joven**: las primeras idea surgieron en los años 1940 (Von Neumann), sin embargo los primeros modelos aparecieron en los 1980s y 1990s.
- Este periodo largo entre la idea y el primer modelo se debe por lo menos parcialmente a la falta de computadoras.
- Al principio de los 1980, Robert Axelrod organizó un **torneo del dilema del prisionero evolutivo**. Investigadores de diferentes áreas programaron propuestas de comportamiento y luego dejaron agentes con diferentes estrategias enfrentarse. Se detectó cuál estrategia es la más dominante con el tiempo.
- Con la introducción de paquetes especializados (Swarm, Netlogo, Mason, Repast, etc), el **costo para modelar con agentes bajó considerablemente**.

¿Qué dice Google Ngrams sobre la historia de los ABM?



Ver en vivo

ABM no están limitados a economía

Los modelos basados en agentes no están limitados a ciencias sociales en general o economía en particular. De hecho, muchas aplicaciones vienen de:

- biología
- dinámicas de población
- epidemiología
- aplicaciones bio-médicas
- geografía
- mercadotécnica
- Planeación urbana (por ejemplo modelos de tráfico)

Ejemplos introductorios

Vamos a ver unos ejemplos de modelos. Nos van a servir para introducir los conceptos formales con referencias a los ejemplos.

- Modelo de flujos de **tráfico** (`http://traffic-simulation.de/`)
- Tráfico en la CDMX
- Versión ABM del modelo de **hotelling**
- Schelling's **segregation** model
- Predator Prey

Modelos basados en agentes

Elementos claves: agentes, reglas de comportamiento, entorno

Agentes

Definición 1 (Agente)

El agente es la representación de una unidad en el mundo real (individuo, empresa, auto, parcela, etc) en un modelo basado en agentes. Cada agente tiene **características**, **reglas de comportamiento** y vive en un **entorno**.

Agentes forman el **elemento de base** de cada modelo basado en agentes. Todas las decisiones de modelización se hacen a nivel del agente. Los resultados a nivel macro que surgen son el fruto de los comportamientos individuales de los agentes viviendo en un entorno particular.

Agentes: características

Las características de agentes pueden ser de diferentes tipos[1] :

	Intrínsecas	**Relacionales**
Estática/constante	Género Nombre	Madre Padre
Dinámica/variable	Edad Educación Estatus de salud	Red social Hijas/hijos

De un punto de vista económico, la distinción no es muy relevante. Sin embargo, para la programación puede ser muy importante entender las diferencias:

- **Intrínsecas:** los valores están guardados en el agente (variable de instancia), mientras que valores relacionales hacen únicamente **referencia** (vínculos) entre agentes.

[1] A notar que los términos en la tabla (intrínsico, relacional, etc) no son términos bien establecidos. Podrán encontrar muchos términos distintos en la literatura.

Agente: reglas de comportamiento

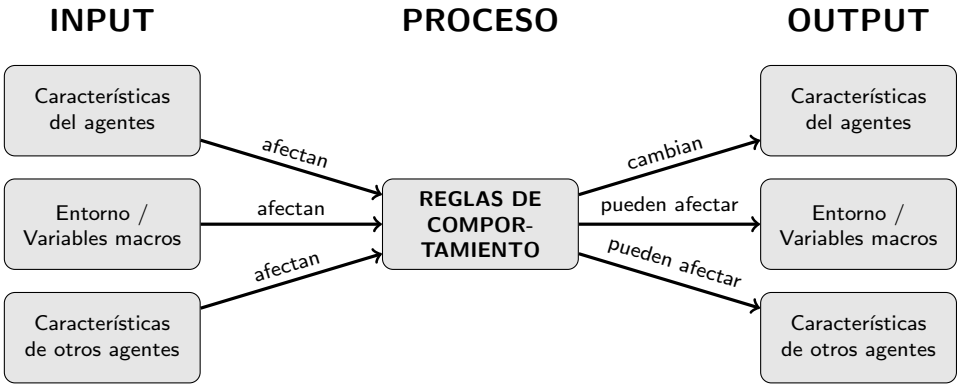
- **Regla de comportamiento** manejan toda la acción que hay en el modelo. En modelos tradicionales en economía, lo equivalente sería la decisión basada en una optimización de alguna función de utilidad.
- En muchos libros de textos sobre ABM pueden encontrar frases como: *"reglas de comportamiento son reglas sencillas que contrastan con un comportamiento muy sofisticado de maximización de utilidad..."*

$\Rightarrow$ Es cierto que en muchos modelos ABM es así, pero **ABM no se basa forzosamente en reglas sencillas.** Podemos incluir por ejemplo una maximización de utilidad sin problema.

- Vemos ABM como una técnica en lugar de un cambio de paradigma.

Reglas de comportamiento: gráficamente

Intentamos ilustrar las reglas de comportamiento gráficamente:



Reglas de comportamiento: ejemplo 1 (simple)

Modelo de segregación de Schelling

1. Analizar la vecindad y calcular un porcentaje de agentes similares
2. Si la proporción es mayor al umbral, se mueve a un lugar no ocupado. En el caso opuesto, no hace nada.

$\Rightarrow$ Moverse afecta la ubicación del agente, pero también:

- el entorno: la estructura de residencia cambia
- la situación para otros agentes: tanto en el origen como en el destino
- variables macro: puede afectar el número de agentes satisfechos

Regla de comportamiento: ejemplo 2 (algo complicado)

Chávez-Juárez et al. (2016)

1. Familia: toma el dinero disponible, la eduación actual de los niños y el estado de salud de la población como dado.
2. **Maximiza una función de utilidad**

   U(consumo,educación niños, salud de miembros de la familia, ahorros)

   a través de una optimización numérica que incluye tanto **decisiones discretas** (educación) como **continuas** (gasto en servicios de salud, consumo, ahorro)
3. La optimización implica
   - una estimación de retornos a la educación (nada vs. pública vs. privada) para cada niño.
   - una estimación de retornos a gasto en salud (función no lineal entre gasto y efecto)
   - oportunidades futuras a través del ahorro

Agente: entorno

El último elemento clave de modelos basados en agentes es el **entorno** en el cual viven los agentes:

- El entorno puede ser (no tiene que ser) una representación del entorno real: por ejemplo en el modelo de Schelling pensamos en una ciudad. En este caso tenemos un **entorno espacial**.
- En el modelo de Hotelling tenemos una proyección (grid) en el ABM que se refiere a un constructo más abstracto en la realidad (diferenciación de producto). Entonces el entorno ABM no se traduce literal a un entorno real. Podemos hablar de un **entorno espacial abstracto**.
- Podemos tener un modelo sin proyección espacial y definir en entorno de una forma aún más abstracta. Por ejemplo, podemos usar una red (social). Hablamos de un modelo con un **entorno de red**.
- Por supuesto, podemos **combinar más de un entorno** en un modelo a través de varias capas: por ejemplo gente que vive en una ciudad con una ubicación explícita que también tiene una red social.

El entorno: diferentes niveles de abstracción

La elección del entorno adecuado para un modelo depende fundamentalmente de la pregunta de investigación:

- ¿Quiero simular una situación real? (p.ej. optimizar el tráfico en la CDMX)
- ¿Mi investigación es más general y quiero implementar un entorno abstracto? (p.ej. Hotelling model)

También tenemos que decidir si queremos un entorno aleatorio o real:

- **Entorno aleatorio**: por ejemplo a través de redes sociales aleatorias, ubicación aleatoria de empesas en el espacio de productos, etc.
- **Entorno real**: redes sociales basadas en datos (p.ej. datos facebook), integración GIS de las vías en la CDMX.

Ejercicio en clase: agentes, reglas y entorno

Ejercicio 1 (Elementos en nuestros ejemplos)

Para los siguientes modelos define cuales son los **agentes**, **reglas de comportamiento** *y el* **entorno**:

- *Modelo de segregación de Schelling*
- *Flujo de tráfico*
- *Predator Prey (lobos y ovejas)*

Modelos basados en agentes

Características claves: interacción, información, tiempo

Características claves: comparación con modelos tradicionales

Para esta discusión seguimos en gran parte a Miller and Page (2007).
Presentan la siguiente tabla a la cual únicamente agregué la última columna:

Modelo tradicional	**ABM**	**Comentarios**
Preciso	Flexible	ABM normalmente sin solución analítica
Pocos procesos	Orientado en procesos	Bueno, pero también requiere mucha información de los procesos
Sin tiempo	Con tiempo	En ABM un tiempo de computadora (tick) se puede asociar a un tiempo real
Optimizando	Adaptativo	Generalmente sí, pero no siempre
Estático	Dinámico	
1,2 o ∞ agentes	1,2,..., N agentes	
En el vacío	Espacial/redes	Gracias al entorno
Homogéneo	Heterogéneo	**Crucial**: permite heterogeneidad en muchos aspectos (características, comportamiento, resultados)

Flexibilidad vs. precisión

Tomamos dos casos extremos de un modelo económico:

Descripción verbal	Modelo matemático
Muy flexible: agregar/quitar elementos es muy sencillo	**No flexible**: también cambios pequeños requieren muchas veces resolver el modelo nuevamente
Impreciso por definición	**Muy preciso**: resultados (normalmente) determinísticos y se pueden resumir con unos parámetros
No requiere simplificaciones	**Requiere simplificaciones** para obtener soluciones analíticas

ABM (y otros modelos computacionales) se encuentran **en un lugar en medio** de los dos extremos. Necesitan mucho más precisión, pero siguen flexibles por no requerir soluciones analíticas.

Tal vez podemos resumir el asunto fácilmente con la siguiente cita de Read (1914):

> *It is better to be vaguely right than exactly wrong*

Flexibilidad: ejemplos

¿Como es la flexibilidad en modelos ABM?

- Pensamos en un modelo tradicional con una función de utilidad que se optimiza en el código numéricamente. Si queremos ver que pasa con otra función, simplemente la cambiamos y corremos el modelo.
- Podemos generar ABM de forma modular, donde podemos activar/desactivar cada módulo.
- **Agregar un nuevo tipo de agente** es trivial: se programa, se agrega y listo!
- Flexibilidad también se puede referir a ser flexible para que país/región se aplica el modelo. Podemos lograrlo con diferentes elementos 'input', por ejemplo la población inicial, el entorno, etc

A notar: el programador tiene que **anticipar** este tipo de flexibilidad.

Orientado a procesos

- Las soluciones de modelos ABM normalmente no resultan de una condición de equilibrio, sino son el fruto de los procesos que se modelan.
- Modelos matemáticos basados en equilibrios muchas veces (pueden) ignorar muchos procesos. Cuando hacemos un ABM, no podemos ignorar los procesos, ya que todo depende de ellos!!
- Tomamos el modelo más sencillo en economía: encontrar el equilibrio de oferta y demanda en un mercado. Simplemente tenemos que definir las funciones de demanda y oferta y resolver el sistema. Eso no implica pensar en los procesos de producción, inversión, compra etc...
- En ABM, no podemos evitar dichos procesos:
  - ¿En qué orden deciden los agentes?
  - ¿Quién se mueve primero, el/los vendedores o los compradores?
  - ¿Qué pasa si la demanda excede la oferta o vice-versa?

Orientado a procesos (2)

Otro ejemplo: Pensamos en la inversión en educación de una familia. En el enfoque de Becker distribuimos los recursos de acuerdo a la habilidad cognitiva de los niños.
Pero, en ABM, necesitamos respuestas a muchas preguntas:

- ¿Qué pasa si tenemos varios niños de diferentes edades? ¿No invierto en el primer niño porque anticipo que el segundo es más inteligente?
- ¿Cómo comparo la habilidad cognitiva del primogénito con los que todavía están por llegar?
- En caso de no tener suficientes recursos y tener que sacar un niño de la escuela. El niño A es más o menos inteligente, pero a un año de graduarse, mientras que el niño B es muy inteligente, pero acaba de empezar la educación. ¿Qué niños tiene que dejar la escuela? ¿El nivel alcanzado actualmente importa para la decisión?

Orientado a procesos (3) - las dos caras de la moneda

¿La orientación a procesos es algo bueno?

Sí, porque...	**No, porque...**
Más realista: ignorar los procesos puede causar resultados poco realistas.	**Complicación no necesaria** del modelo: a nivel macro, los detalles no importan.
Es un **proceso enriquecedor**, dado que detectamos muchos procesos que normalmente no consideramos	Es **una pérdida de tiempo**, porque tenemos que entender muchos procesos
Ignorar los procesos nos hace elegir implícitamente la **opción por default**	Agregar tantos detalles genera una caja negra (**black box**).

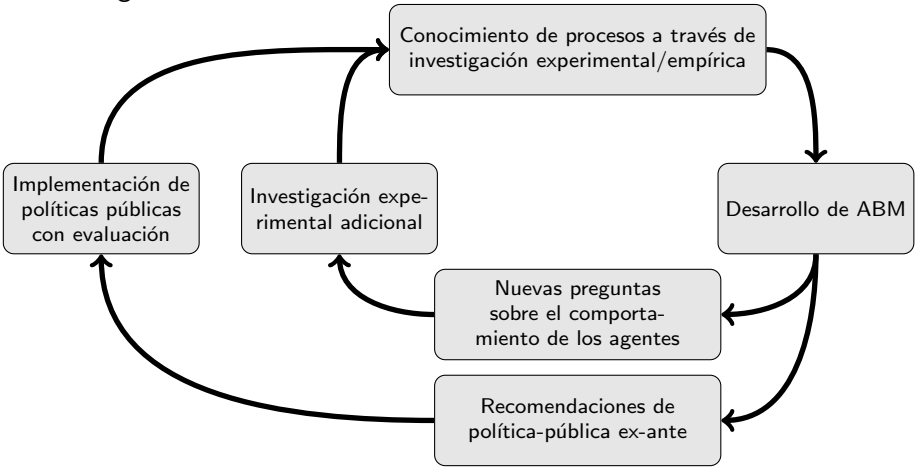
No hay uno que domina al otro - depende de lo que hacemos!

Orientado a procesos (4) - comentarios de mi experiencia

- El argumento de **un proceso enriquecedor** es muy cierto. Por ejemplo, desarrollando el modelo Chávez-Juárez et al. (2016) nos enfrentamos a muchas preguntas:
  - Mercado laboral: ¿cuál es el orden correcto? ¿Trabajadores aceptan la primera oferta o esperan?
  - ¿Cómo modelar la aprobación de un grado en la escuela?
  - ¿Empresas optimizan primero su nivel de capital o su fuerza laboral?
  - ¿Familias optimizan primero su consumo o primero su oferta laboral? ¿O optimizan condicional a conocer el estatus laboral?
- Un elemento clave en muchos modelos es el orden de decisión. Contrario a modelos matemáticos, no podemos verdaderamente **hacer decisiones simultaneas** (más sobre eso más adelante)
- La mayor disponibilidad de datos y conocimiento de otras ciencias (psicología, ciencia de la conducta, sociología, etc) pueden ayudar mucho para entender los procesos.
- ABM no sólo genera resultados, sobre todo genera nuevas preguntas!

Orientado a procesos (5) -retos y oportunidades

En Chávez-Juárez (2016) hago el argumento a favor de un enfoque iterativo de investigación:



Adaptativo

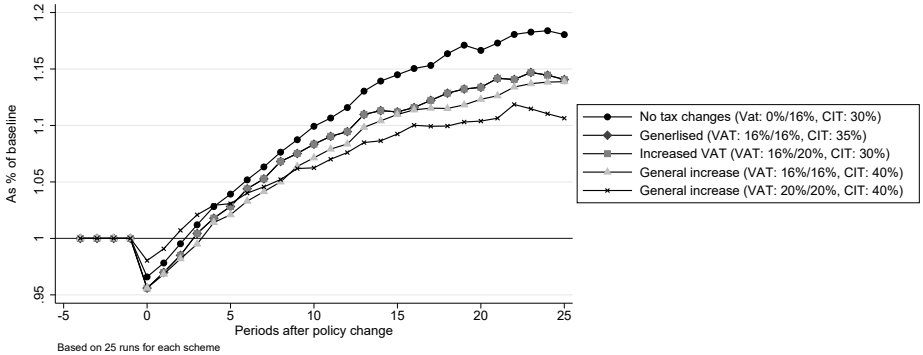
- La mayoría de los ABM no incluyen una optimización de utilidad, por lo menos no una inter-temporal.
- En lugar de la optimización, agentes toman decisiones basadas en la situación actual (entorno, otros agentes, etc) y se **adaptan** a cambios en dichos 'input'.
- Podemos incluir un concepto de **aprendizaje** donde los agentes toman el pasado para aprender cual es la mejor estrategia.
- El **aprendizaje** se puede implementar de forma muy sencilla (hacer lo que el mejor hizo antes) o usar técnicas más avanzadas como algoritmos genéticos o aprendizaje direccional.
- Sin embargo, no necesitamos aprendizaje para tener adaptación. Los agentes puedes por ejemplo adaptar su decisión de educación a cambios en los retornos a la eduación, sin que haya aprendizaje.

Dinámico

- Casi todos los **fenómenos socials son dinámicos** por naturaleza. Sin embargo, modelos de equilibrio normalmente consideran nada más un punto inicial y un punto final (equilibrio). Podemos pensar en la siguiente frase de Miller and Page (2007): *"trying to understand running water by catching it in a bucket"*
- **¿Pero qué pasa entre el punto inicial y el punto final?**
- La forma como simulamos los ABM nos permite analizar la situación en cada momento. Nada más requiere definir bien el vínculo entre tiempo de computadora y tiempo real.
- Particularmente importante cuando analizamos políticas públicas: ¿para cuando podemos esperar resultados?
- Asimismo, un sistema puede converger a diferentes estados estacionarios y no necesariamente llega siempre a lo mismo.

Dinámico: ejemplo de Chávez-Juárez et al. (2016)

Vemos una ilustración de Chávez-Juárez et al. (2016), donde analizamos los efectos de un cambio de impuestos sobre el BIP:



$\Rightarrow$ Pérdidas en el corto plazo, beneficios en el mediano y largo plazo.

Heterogeneidad

En mi opinión, una de las **ventajas principales** de ABM es poder modelar **heterogeneidad**. Pesamos en heterogeneidad en múltiples formas:

- Características (hasta cierto punto posible en modelos tradicionales)
- Comportamiento
- Información
- Circunstancias (p.ej. entorno)
- Resultados

Es muy importante en el análisis de políticas públicas: **¿Quién gana y quien pierde si implementamos una reforma?**

Riesgo: Existe el riesgo de agregar demasiado y ya no entender bien lo que pasa en el modelo. Hay que incluir lo menos posible, pero no menos de lo necesario.

Interacción

- ABM ofrece muchas posibilidades de **interacción** entre agentes. Podemos distinguir algunos tipos:
  - **Directo** (agente-a-agente) o **indirecto** (via resultados macro/entorno)
  - **Activa/estratégica** (p.ej. Hotelling) o **pasiva** (p.ej. tráfico)
- Si permitimos **interacción local** (por ejemplo espacial o en red social) podemos obtener equilibrios locales que no son idénticos para todos los agentes.
- Lo crucial de la interacción es que el **comportamiento y resultado de un agente** depende de lo que hacen o hicieron otros. Ejemplo: *con el mismo entorno (vías de circulación) viajar de Santa Fe al AICM no es lo mismo un domingo a las 4am que un viernes a las 5pm. Todo depende de la existencia/cantidad y comportamiento de otros agentes.*

Información

- El concepto de la **información** es crucial en muchos modelos económicos (p.ej. información asimétrica)
- Sin embargo, en muchos modelos el origen de la información es poco claro. Vemos unos ejemplos:
  1. En el modelo de oferta y demanda: suponemos que la empresas conocen la función de demanda. ¿De dónde?
  2. En la teoría de capital humano las familias optimizan su inversión en educación dependiendo de los ingresos futuros de los niños. ¿Por qué conocen los retornos a la educación en el futuro, si ni los economistas logran bien estimar los retornos en el presente?

$\Rightarrow$ Dicho eso, no necesariamente es malo no conocer el origen! Si los supuestos sobre la información disponible es razonable, podemos ignorar el origen de la información lo que simplifica mucho!

- Sin embargo, muchas veces en ABM no tiene sentido hacer supuestos de este tipo:
  - Si modelamos los procesos en detalle, no tiene sentido no considerar el origen de la información.

Información (2)

En general, en ABM el investigador tiene que poner más atención al origen de la información, lo cual aumenta el costo de modelización, pero también viene con nuevas oportunidades

- ¿Queremos la misma información para todos o dependiendo de algunos factores como la red social?
- ¿En qué momento y a qué tipo de información tienen acceso lo agentes? Por ejemplo, empresas conocen las preferencias de empresas o las tienen que estimar con *trial-and-error*?

Tiempo: tiempo en la simulación vs. tiempo real

ABM son *tick-based*, es decir que un 'tick' es una unidad de tiempo durante la cual toda la secuencia de procesos se ejecuta.

- Por ejemplo: en cada 'tick' del modelo de Schelling, cada agentes analiza la situación y se mueve si le conviene.
- Una vez que el 'tick' termina, inicia el siguiente.
- Puede ser importante asociar un 'tick' a una duración en el tiempo real, pero no siempre es fácil. Tomamos el ejemplo del efecto sobre el GDP en Chávez-Juárez et al. (2016) donde tenemos 4 periodos de efectos negativos. Es importante saber si un periodo es una semana o 10 años!!!
- A priori, un 'tick' puede corresponder a cualquier unidad de tiempo:
  - Modelos de tráfico: 1 tick es probablemente 1 segundo
  - Chávez-Juárez et al. (2016): 1 tick = 1 año
  - Schelling: no queda claro, pero podría ser bastante largo, dado que la gente no se mueve cada mes.
  - Wendelspiess Chávez Juárez (2014): 1 tick = 1 generación

Tiempo: Sincronicidad

Una dificultad en relación al tiempo es que en ABM los algoritmos se **ejecutan de forma secuencial** (caso normal: un agente tras otro), mientras que en términos conceptuales podemos querer **decisiones simultáneas**. Ejemplo: modelo de Hotelling

Hay tres formas de 'resolver' este problema:

- **Ejecución asincrónica secuencial**: Ejecutar secuencialmente las reglas de comportamiento de los agentes: A1, A2, A3,... A1, A2, A3,..., etc. Pocas veces es una buena solución, porque pocas veces queremos dar prioridad a un agente sobre los demás.
- **Ejecución asincrónica aleatoria**: Seguimos ejecutando las acciones secuencialmente, pero en cada ocasión cambiamos el orden aleatoriamente. Ventaja: con múltiples simulaciones podemos tratar de entender la importancia del orden.
- **Ejecución sincrónica simulada:** Primero dejamos que los agentes deciden pero todavía no ejecutamos la acción. Una vez que cada uno decidió, todos ejecutan su acción. Alternativa: todos deciden en función de la información que tenían antes de que el primero decidió. **Es la mejor solución** pero por supuesto necesita algoritmos más avanzados. Además, a veces es simplemente imposible: matching market.

Aleatoriedad

Usar elementos aleatorios puede ser útil por varias razones:

- Simular elementos para los cuales **no contamos con suficiente información**: p. ej. iniciar una red social si no la conocemos: podemos hacer los vínculos aleatorios.
- Simular elementos que son **verdaderamente aleatorios**: de acuerdo a la literatura, el IQ de las personas tiene un elemento aleatorio/estocástico
- **Simular errores** en el comportamiento (p. ej. malentendidos en comunicación)

Al incluir elementos aleatorios, es importante simular el modelo con diferentes **random seeds** (semillas aleatorias) para analizar la importancia de los elementos aleatorios. *Random seeds* son números que inician el generador de elementos aleatorios y así permiten reproducir resultados con elementos aleatorios.

También puede ser una buena idea comparar simulaciones con y sin elementos aleatorios.

Modelos basados en agentes

Ventajas y desventajas de ABM

Ventajas y desventajas de ABM: resumen

Basado en las láminas anteriores, podemos tratar de hacer una lista de ventajas y desventajas de ABM:

Ventajas

- Más **posibilidades de modelización** dado que no tenemos límites que se deben a soluciones analíticas
- **Heterogeneidad** múltiple en input, acción y output
- **Análisis dinámico** (el camino al equilibrio)
- Muy **flexible** y con bajo costo (cambiando elementos)

Desventajas [y soluciones]

- **No genera resultados exactos** (analíticos) [no hay solución directa, pero tal vez no necesario]
- Requiere **mucha información sobre procesos**[Ver el reto como una oportunidad: permite incluir conocimiento de otros áreas]
- Riesgo de crear una **black box** donde ya no aprendemos nada del modelo. [Documentar y analizar bien el modelo usando enfoques estandarizados]

Modelos basados en agentes

Verificación de un modelo ABM

Verificación: quitar los 'bugs' (errores)!

La **verificación** se refiere a la eliminación de *bugs* [2] (el código no hace lo que se esperaba - o dicho de otra manera: un error del programador).

- Como en cualquier actividad humana, el código final puede contener errores. Eso resulta en resultados erróneos de la simulación y finalmente en ciencia errónea!
- Por lo tanto es importante identificar errores y corregirlos.
- En lo que sigue veremos técnicas que permiten evitar y corregir errores en un programa. La elección del método óptimo depende del proyecto, pero muchas veces usar más de una técnica ayuda.
- Pero antes: **Cuidado con el 'debugging bias'**. El *debugging bias* se refiere al hecho de buscar errores cuando los resultados no son lo esperado y no buscar errores (no verificar el código) si los resultados corresponden a lo esperado ($\neq$ no hay errores!)

[2] El término *bug* se debe a una polilla que bloqueó un relé en una de las primeras computadoras en Harvard

Verificación: técnicas y consejos (1)

En las siguientes láminas presentamos técnicas que reducen el riesgo de tener 'bugs' en el modelo. Dichas recomendaciones se basan en Gilbert and Troitzsch (2005) y en mi propia experiencia.

- **Escribir buen código**: la mejor manera de deshacerse de 'bugs' es nunca introducirlos al código 😉. Tómate el tiempo para planear bien tu código, usa 'best practices', nombres de variables y métodos que tienen sentido y no te apures en tener un modelo que corre. Cada minuto que 'ahorres' programando genera muchos minutos de búsqueda de errores!
- **Agrega comentarios y JavaDoc**: para cualquier programación, es crucial bien documentar el código. Es particularmente importante para proyectos grandes que se llevan a cabo durante un periodo largo o con interrupciones largas. Comentarios ayudan a evitar 'bugs' y también a identificarlos más fácilmente. Al cambiar el código, no hay que olvidar adaptar los comentarios y particularmente el JavaDoc.

Verificación: técnicas y consejos (2)

- **Incluir 'output' y diagnosis**: exporta (consola, base de datos) resultados/información y analiza los datos cuidadosamente durante la fase de programación (no te esperes al final!). Eso nos permite identificar los errores poco después de haberlos introducido. Mi regla de oro: **programar un elemento, verificarlo, ir al siguiente elemento**.
- **Correr el modelo paso por paso**: Intenta correo el modelo tick-por-tick y/o activando/desactivando elementos y analiza si el modelo se comporta de la forma esperada.
- **Agrega 'assertions' (red flags)**: muchas veces conocemos los valores (o la lógica) de algunos elementos. Podemos agregar un código de diagnostico que busca valores que no pueden ser. Al encontrar un valor de ese tipo, manda una alerta. Por ejemplo: verificar que los padres son más grandes (en edad) que los niños.

Verificación: técnicas y consejos (2)

- **Usa 'unit testing'**: es un método popular en ciencia de la computación y también se debería aplicar en ABM. La idea es verificar el código con unos ejemplos numéricos.
  Ejemplo: programamos la optimización de una función de utilidad en un archivo auxiliar y luego comparamos los valores a lo que calculamos en otro lugar (p.ej. en Excel). La ventaja es que así no mezclamos dicha función con lo demás del modelo!
- **Comparar con resultados conocidos**: si tienes información sobre lo que el modelo (o una versión sencilla del modelo) debería producir, puedes comparar los resultados del modelo con dicha información. Por ejemplo, en el modelo de Hotelling conocemos la solución analítica para dos jugadores y entonces podemos primero ver si el modelo reproduce eso antes de ir a casos más complejos (p.ej. 8 jugadores).

Por supuesto, la idea es combinar todas las técnicas que se pueden aplicar!

Modelos basados en agentes

Validación de ABM

Validación: ¿qué tan válido es el modelo para describir un fenómeno social?

Una vez que hemos verificado el modelo, es importante analizar la validez del modelo. La **validez se refiere a qué tan correcto es el modelo de un punto de vista de ciencia social:**

¿el modelo captura bien el fenómeno social?

validación no es un concepto que aplica únicamente para los ABM, sino básicamente a cualquier método de modelización. Por lo tanto, aquí nos enfocamos en lo que puede ser particular para los ABM:

- **Parámetros y calibración:** lo que metemos al modelo es razonable?
- **Comparación con datos reales**: el modelo reproduce satisfactoriamente la evidencia empírica?

Validación: encontrar los parámetros correctos (calibrar)

Cada ABM depende en gran medida de los parámetros que usamos. Es importante usar los valores 'correctos'

- Si es posible, tratamos de usar parámetros que se estiman con datos o bien que vienen del contexto que estamos analizando (p.ej. la función real de imposición).
- El uso de datos es la mejor solución, ya que acerca el modelo más a la realidad del país/de la región que estamos analizando.
- Sin embargo, muchas veces no es posible por falta de datos. En este caso se recomiendo usar **valores plausibles** y correr análisis de sensibilidad. Correr el modelo varias veces (diferentes random seeds) para cada combinación de parámetros y ver como afecta los resultados.
- Este ejercicio se vuelve rápidamente tedioso, sobre todo si consideramos interacciones entres parámetros. Tomamos un ejemplo de dos parámetros con valores entre 0.0 y 0.5 y queremos ver pasos de 0.1. Para cada uno de los dos parámetros tenemos 6 valores, lo que genera $6 \times 6 = 36$ simulaciones para cada *random seed*. Si queremos 20 simulaciones por *random seed*, ya tenemos que simular el modelo 720 veces!

Validación: comparar los resultados con el mundo real

Tal vez lo más complicado es cuando intentamos contrastar el modelo con el mundo real.

- La factibilidad depende mucho de que tan abstracto es el modelo. Para modelos muy abstractos (p.ej. Schelling) es muy complicado sino imposible. Para modelos que buscan reproducir un fenómeno muy particular en un país, sí podemos comparar los datos generados por el modelo con datos reales.
- La idea es comparar los resultados del modelo de base (baseline) con los datos y hechos esterilizados del mundo real. Lo podemos hacer con **variable macro** pero también **comparando distribuciones** y **relaciones** entre variables.
- Podemos usar técnicas estándares de **estadística y econometría** para hacer dichas comparaciones.
- No hay que limitarse a la comparación de las variables más importantes sino también ver si **variables auxiliares** se acerquen a la realidad.

Modelos basados en agentes

Descripción de modelos: el protocolo ODD

ODD: motivación e idea general

Como último tema vemos como podemos presentar modelos basados en agentes en un trabajo académico. Introducimos el protocolo **Overview, Design concepts and Details** (ODD-protocol). El ODD es una propuesta de Grimm et al. (2006) y Grimm et al. (2010) y fue analizado por Polhill (2013).

Punto de partida	Mayor uso de ABM en muchos áreas, pero no existe una forma estandarizada de presentar los modelos. Eso reduce la **comparabilidad** y en particular la **reproducibilidad** de los modelos, lo que pone en riesgo las ventajas del enfoque
Objetivos	Proponer una forma armonizada de presentar modelos ABM en cualquier área de uso.
Propuesta	Un protocolo dividido en tres bloques: Overview, Conceptos de diseño y Detalles. La idea es reportar siempre todo, independientemente de su relevancia en un proyecto en particular.

ODD: una vista general

Podemos resumir el ODD con la siguiente tabla:

Overview	Purpose	Breve descripción del modelo (similar a un abstract) sin entrar a los detalles.
	State variables and scales	Presenta el tipo de agentes junto con sus variables de estado y sus escalas
	Process overview and scheduling	Vista general de todas las reglas de comportamiento e indica el orden en el cual se ejecutan.
Design concepts	Design concepts	Información sobre temas muy específicos de los ABM (p.ej. interacción)
Details	Initialisation	¿Cómo inicia el modelo?
	Input	¿Qué otros elementos exógenos se usan como input durante la simulación?
	Submodels	Descripción de los diferentes elementos del modelo.

ODD: Overview: Purpose

El primer elemento del ODD es una pequeña descripción (1-2 párrafos) sobre los objetivos principales del modelo. Eso debería permitir al lector entender lo que hace el modelo y porque algunos elementos se hacen con mayor o menor detalle.

Un ejemplo de Grimm et al. (2006):

> *The purpose of the model is to understand how the social behaviour of the marmots - in particular territoriality, reproductive suppression, and hibernation as a group - affects population dynamics and in particular extinction risk if populations are small.*

ODD: Overview: State variables and scales

- Aquí introducimos los **diferentes tipos de agentes** que tenemos en el modelo.
- También indicamos si existe algún tipo de jerarquía entre los agentes.
- Describimos las **variables low-level** (variables de instancia), no estadísticas macro.
- En caso de modelos grandes, hacer una tabla podría ser muy buena idea.
- **Unified Modelling Language** (UML) también se puede usar aquí.
- También es importante mencionar la **escala** de las variables que estamos usando.

ODD: Overview: Process overview and scheduling

En el último elemento de 'overview' presentamos los procesos que se llevan a cabo en cada periodo.

- Indicar **que procesos** (comportamiento) se realiza por **quién** y en **qué orden**. Por ejemplo en el modelo de los lobos y ovejas podría ser:
  1. Lobo busca la oveja más cercana.
  2. Lobo corre una unidad de distancia hacia la oveja
  3. En caso de estar suficientemente cerca, la come
  4. Ovejas que sobreviven, corren una unidad de distancia en la dirección opuesta al lobo más cercano.
- Mi manera preferida para hacer eso es una **tabla** con el **orden**, el **agente**, la **descripción** del método y tal vez un **vínculo** a una descripción más detallada en las columnas.
- **Flow charts** (diagrama de flujo) pueden ser una buena alternativa (para modelos no muy complejos)

ODD: Design concepts (conceptos de diseño)

Para modelos ABM, tenemos **11 conceptos de diseño** que se refieren a una característica particular de cada modelo. En esta sección, el autor debe indicar como cada uno de dichos conceptos está implementado (o no) en el modelo. A continuación se presenta una lista de los elementos. Para los detalles, favor de consultar las referencias:

1. **Basic principles**: Cuales son los conceptos generales, las teorías, las hipótesis o enfoques de modelización que se usan en el modelo?
2. **Emergence**: ¿Qué resultados claves o otros 'outputs' del modelo emergen del comportamiento individual de los agentes?
3. **Adaptation**: ¿Cómo se adaptan agentes en el modelo? ¿Dichos cambios se basan en cambios del mismo individuo y/o del entorno?
4. **Objectives**: ¿Cuál es el objetivo de cada agente (p.ej. optimizar una función de utilidad)
5. **Learning**: ¿Agentes aprenden en el modelo? ¿Están cambiando su comportamiento en función de sus experiencias?

ODD: Design concepts (continuación)

6. **Prediction**: ¿Agentes basan sus decisiones en predicciones de valores futuros (como consecuencia de sus acciones)? Si sí, ¿cómo lo hacen?
7. **Sensing**: ¿Qué variables internas o del entorno los agentes conocen y pueden consierar para su decisión? Ejemplo: conocen lo agentes su IQ?
8. **Interaction**: ¿Qué tipo de interacción existe en el modelo? ¿Se trata de una interacción directa o indirecta?
9. **Stochasticity**: ¿En dónde el modelo tiene elementos estocásticos? ¿Por qué razón se usa un elemento estocástico?
10. **Collectives**:¿Agentes pertenecen a agregados de agentes que afectan su comportamiento? (por ejemplo individuos pueden formar parte de una familia que toma decisiones para todos los miembros)?
11. **Observation**: ¿Qué datos se obtienen/extraen del modelo para hacer pruebas y análisis? ¿Cómo se exportan los datos?

Autores pueden agregar más *design concepts* si creen que hace falta algo.

NO se debe omitir nada de la lista anterior.

ODD: Details: Initialisation

Esta sección contesta la siguiente pregunta:

¿Cuál es el estado inicial de la simulación?

- ¿Cuántos agentes hay?
- ¿Cómo se definen sus valores (basado en datos o aleatorio?)
- ¿El inicio siempre es el mismo?
- Si se usan datos, hay que poner las referencias y una explicación aquí.

ODD: Details: Input data

¿El modelo usa 'input' de fuentes externas como por ejemplo bases de datos u otros modelos para representar procesos que van cambiando a lo largo de la simulación?

- Aquí **no nos referimos** a cambios exógenos de alguna política.
- Hacemos más referencia a cambios por ejemplo en el clima, en el número de coches que entran a una zona, la hora, la luz solar,etc...
- Cambios en parámetros **no se incluyen**!

ODD: Details: Submodels

Esta sección introduce todos los detalles de los submodelos que se mencionan en la sección *Process overview and scheduling*. Debería contestar las siguientes preguntas:

- **¿Qué - en detalle - son los submodelos** que repersentan procesos en "Process overview and scheduling"?
- ¿Cuáles son los **parámetros** del modelo, su **dimensión** y los **valores de referencia**?
- **¿Cómo se eligió el diseño de los submodelos?** ¿Qué parámetros tienen, como se hicieron las pruebas?
- Si los parámetros no se discutieron suficiente en otro lugar, se puede hacer aquí
- Básicamente aquí hay que poner todo lo que no cabe en otra sección.

ODD: unos comentarios

- Poner la descripción del modelo en un nivel 'económico' y poner los detalles técnicos en el ODD que típicamente se encuentra en un apéndice.
- Aunque a veces parece ser ridículo, **siempre hay que incluir todos los elementos**
- La información en el ODD puede ser **redundante** con lo que se pone en el paper principal. No hay problema.
- Escribir el ODD **no siempre es sencillo** y a veces nos cuesta trabajo entender lo que hay que poner. También es una cuestión de experiencia (cada vez que lo veo, me doy cuenta que mi ODD anterior tenía un error)
- Escribir el ODD **durante (o incluso antes)** de escribir el código! Puede ayudar a hacer la 'buenas preguntas' para la implementación!

Modelos basados en agentes

Comentarios finales

ABM: comentarios finales y consejos

Para terminar con la parte conceptual, aquí unos consejos[3]:

- **Enfocarse en la ciencia, no la computadora**: los modelos ABM existen para resolver problemas científicos. El enfoque debería estar en el modelo y no en su implementación técnica. El objetivo no es mostrar que tan buen programador eres ;-)
- **Enfocarse en programación eficiente**: rápidamente los modelos ABM se vuelven muy pesados en términos de computación. Por eso es importante programar de forma eficiente! Lo mejor es programar el modelo en una computadora vieja, ya que te obliga a hacer las cosas bien!
- **Evitar cajas negras** aceptando algunas simplificaciones que no cambian todo el modelo y describiendo muy bien el modelo (p.ej. ODD)

[3] Se basa en Gilbert and Troitzsch (2005), Miller and Page (2007) y mi propia experiencia

ABM: comentarios finales y consejos (continuado)

- **Incluir parámetros** para fenómenos claves lo que permite activar/desactivar algunos procesos. En general, hay que modelar lo más flexible posible para permitir cambios. Usar una lógica de modelos 'nested' podría ser muy bueno.
- **Usar código existente y compartir tu código**: no hay que inventar la rueda nuevamente! Usa código disponible y distribuye tu código en la red.
- **Hacer una lista de cambios y próximos pasos**. Es bueno tener una lista de los cambios que se hicieron y lo que viene. Puede ser también usando sistemas como GitHub y Java Tasks
- **Have fun!**

Introducción a la programación en NetLogo

¿Qué es la programación?

Programación es un término muy general para describir el proceso de desarrollar un algoritmo ejecutable en la computadora. Incluye una variedad de actividades:

- Análisis del problema
- Diseño/desarrollo de algoritmos para resolver el problema
- *Coding*[4] (Implementación del código en el lenguaje de programación)
- Verificación y *debugging*

Enfoque en este curso

1. Primero nos enfocamos en aprender *coding* en Netlogo (día 2 y 3)
2. A través de un proyecto práctico, hacemos las 4 etapas de la programación (días 4 y 5)

[4] Muchas veces el *coding* es lo que la gente tiene en mente cuando se trata de programación. Es únicamente una parte de lo que es la programación!

Introducción a la programación en NetLogo

Qué es NetLogo y qué alternativas hay?

Qué es NetLogo?

- NetLogo es un **software especializado** a modelos ABM con un **costo de entrada muy bajo** ($\Rightarrow$ nos podemos concentrar en ABM)
- Se desarrolla en Northwestern University y es de **código abierto**
- Usa un lenguaje de programación particular (un tipo de Logo) y atrás lo convierte y lo corre en **Java Virtual Machine**
- Incluye un catálogo muy amplio de ejemplos/modelos (**Model library**)
- Todo el modelo se guarda en **un sólo archivo** (vs cientos de archivos en RePast).
- Cuenta con una serie de **extensiones** que se pueden activar muy fácilmente

$\Rightarrow$ Vamos a ver una introducción, pero es imposible ver todos los comandos!

Alternativas

- **NetLogo** es el líder para cursos introductorios a modelos basados en agentes.
- Sin embargo, existen alternativas que se basan en lenguajes de programación más generales.
  - **Java:** RePast, JasMine, Mason, Swarm
  - **Python:** Mesa, (RepastPy)

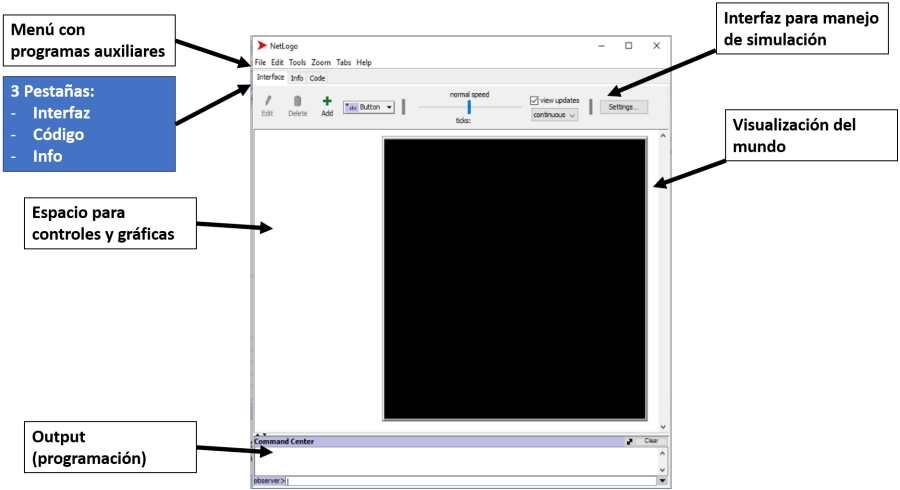
⇒ **Por qué NetLogo en este curso?**

- Nos queremos enfocar en ABM, no en programación
- No hay tiempo para aprender a programar en 5 días!

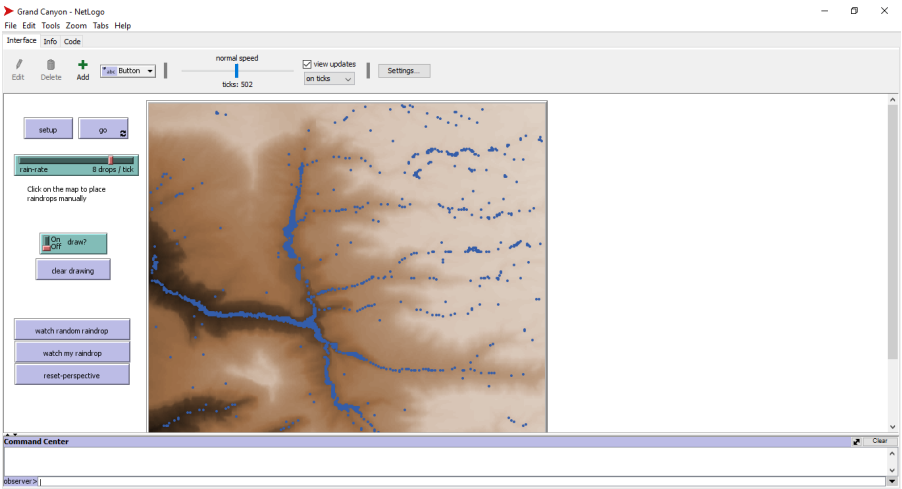
Introducción a la programación en NetLogo

Los elementos claves de Netlogo

La interfaz: proyecto nuevo



La interfaz: proyecto activo



Introducción a la programación en NetLogo

Elementos y sus nombres en Netlogo

Agentes, Entorno, Relaciones, etc

Cada software tiene sus propios nombres para los conceptos que hemos visto en la parte de ABM. NetLogo usa los siguientes:

Agente(s)	**Breed** (default: **turtle**/turtles). Se pueden definir agentes de diferentes nombres. Siempre hay que indicar el singular y el plural: `breed[alumnas alumna]`
Entorno:	Netlogo usa **patches** que se comportan básicamente como rectángulos estáticos en el mundo

Vínculos/relaciones:	NetLogo se llaman **link** y distingue de relaciones dirigidas (con origen y destino) de un lado y relaciones bidireccionales de otro.
	`undirected-link-breed [friendships friendship]`
	`directed-link-breed[debts debt]`

«Mundo»	El **observer** se refiere a algo ajeno al modelo que maneja todo el modelo (el simulador)

Turtles

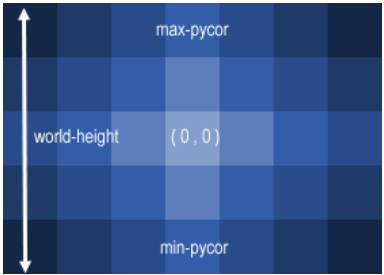
Turtles son los agentes en NetLogo. Además de las variables que indicamos, siempre tienen las siguientes variables:

breed	Tipo de turtle (podemos tener multiples tipos de agentes)
color	Color del agente (visualización)
heading	Dirección en la que mira
hidden?	Dummy para agentes ocultos
label	Etiqueta (para visualización)
label-color	Color de la etiqueta
pen-mode	Para dejar la ruta que caminó el agente
pen-size	Tamaño de la línea de la ruta
shape	Forma del agente (hay muchas formas ya instaladas)
size	Tamaño del agente (para la visualización)
who	Es un ID de cada agente
xcor	Posición en el espacio (coordenada X)
ycor	Posición en el espacio (coordenada Y)

Patches

Patches son cuadros estáticos en el mundo (en NetLogo también se consideran como agentes). Tienen los siguientes valores

pcolor	Color
plabel	Eqitueta
plabel-color	Color de la etiqueta
pxcor	Coordenada X
pycor	Coordenada Y



Links

Links son «agentes» que conectan otros agentes. Se pueden usar en **redes sociales** pero también en cosas más complejas como un contrato entre dos agentes. Tienen los siguientes valores

breed	Tipo de vínculo
color	Color de la visualización
end1	Agente del primer lado (first endpoint)
end2	Agent del segundo lado (second endpoint)
hidden?	Dummy para vínculos ocultos
label	Etiqueta del vínculo
label-color	Color de la etiqueta
shape	Forma del vinculo (default: línea o flecha)
thickness	Ancho de la línea
tie-mode	Variable binaria para indicar si los 2 agentes están fijados (se mueven juntos)

Convenciones y particularidades de Netlogo

El lenguaje de programación en Netlogo tiene algunas cosas un poco particulares:

- El símbolo para iniciar un comentario es ;
- Operadores matemáticos requieren un espacio:
  - `a-b` se interpreta como una variable con el nombre 'a-b'
  - `a - b` se interpreta como la operación matemática de 'a' menos 'b'
- No es necesario indicar el fin de una linea
- El tipo de variable no se tiene que definir (*implicit type definition*)
- El editor de código no indica en vivo si hay un error, pero sí existe un botón [Check] para verificar el código

Definir una variable y asignar un valor

Para definir una variable (que todavía no existe) set usa `let`:

```
let x 5
```

Si ya existe la variable, poner `let` resulta en un error. Para asignar un nuevo valor, hay que usar `set`:

```
set x 4
```

Un ejemplo completo (`00_DefinirVariablesImprimirConsola.nlogo`):

```
let x 4       ; definimos la variable x y asignamos el valor 4
let y 8       ; definimos la variable y y asignamos el valor 8
show y + x    ; imprimimos la suma en la consola (show tambien imprime quien lo
    calcula)
set y 3       ; asignamos el valor de 3 a la variable existente y
print y * x   ; imprimimos el resultado de y * x en la consola
print (word "El resultado de y * x es: " (y * x ))
```

Definir funciones en NetLogo

Un elemento clave que necesitaremos mucho son funciones. En NetLogo se definen con las palabras claves `to`, `to-report` y `end`

- Con argumentos, sin regresar un valor:

```
to suma2 [a b]
  print(a + b)
end
```

- Sin argumentos, regresando un valor:

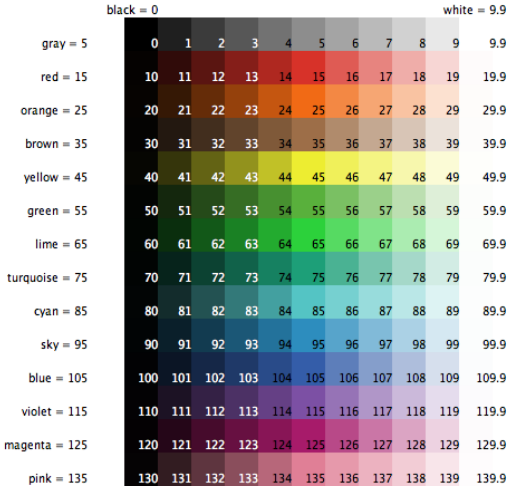
```
to-report getRandomValue
 report random 500
end
```

- Con argumentos, regresando un valor:

```
to-report suma [a b]
  let c a + b
  report c
end
```

Las funciones se activas por agentes o bien desde un botón de la interfaz.

Colores en Netlogo



Números aleatorios

Netlogo viene con un generador de números aleatorios:

`random 3`	Uniforme, valores enteros: 0, 1, 2
`random-float 5`	Uniforme en $[0, 3)$
`random-exponential 2`	Distribución exponencial con $\epsilon = 2$ (promedio $=$ 2)
`random-normal 10.1 5.2`	$N(10{,}1, 5{,}2)$
`random-gamma 5.2 0.9`	Distribución Gammma ($\Gamma(\alpha, \lambda)$)
`random-poisson 3.4`	Distribución de Poisson con promedio 3.4

Introducción a la programación en NetLogo

Crear agentes

Crear agentes

1. Primero se **define un agente** (y sus variables) de forma general (e.g. coches, casas, empresas). Aquí hacemos el *blueprint*, no cada agente de forma individual
2. Iniciamos **instancias** de dicho tipo de agentes (e.g. 100 agentes del tipo coche)

- Las acciones - contrario a lenguajes de programación orientados a objetos - no se programan en el agente, sino desde el **observer**. Veremos más adelante como se hace.

Definir un agente

La forma más sencilla de definir un agente (aquí ejemplo de un coche) es:

```
breed[coches coche]
```

Para definir que variables tiene un agente, usamos `turtles-own` con el nombre del agente:

```
coches-own[
  marca
  precio
  numero-de-puertas
  ]
```

Crear agentes: un ejemplo

```
1  breed[coches coche] ; here we define the types of agents
2  coches-own[ precio numero-de-puertas ]
3
4  to inicio
5    clear-all ; clear all existings elements
6    reset-ticks ; reset the ticks
7    create-coches 50 [ ; create 50 cars
8      setxy random-xcor random-ycor          ; ponerlos en un lugar aleatorio en el espacio
9      set precio (100000 + random 200000)    ; poner el precio aleatoriamente entre 100K y
           300K
10     set numero-de-puertas (2 + random 3) ; poner el numero de puertas aleatorio entre 2 y
           4
11     set label numero-de-puertas            ; poner una etiqueta al coche ( visualizacion )
12     set label-color blue                   ; poner la etiqueta en azul
13     set shape "car"                        ; poner el agente como 'coche' en el espacio
           ( visualizacion )
14     set color red                          ; poner el coche en rojo
15     set size 2                             ; aumentar el tamano del icono
16   ]
17 end ; end de inicio
```

Ejercicio

Ejercicio 2

Crear un mundo con 50 aviones (puede ser otra cosa si prefieren) de color azul con una etiqueta que indica la coordenada X.

Introducción a la programación en NetLogo

Reglas de comportamiento

Reglas de comportamiento

- En la parte conceptual vimos que las **reglas de comportamiento** son el fundamento de todo movimiento en un ABM.
- En NetLogo usamos **procedures** para implementar las reglas de comportamiento. El comando `ask` (y sus versiones) nos permiten pedir a los agentes (turtles, patches, links) de ejecutar alguna acción.
- Un ejemplo muy básico que se puede activar con un botón (go):

```
1        ask coches[
2           right -5 + random 20 ; cambiar 'heading' en un rango -5,15
3           pen-down             ; trazar la ruta
4           forward 1                ; moverse una unidad en el 'heading'
5           ]
```

Default procedures

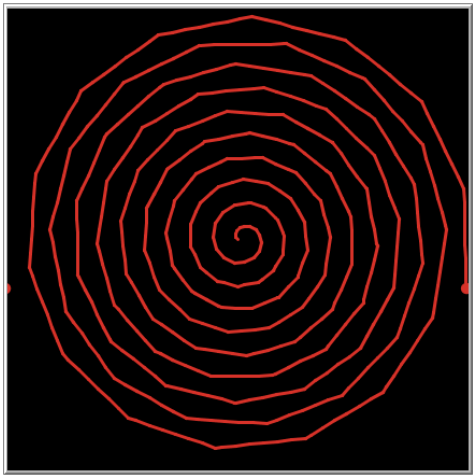
Como en el caso de variables, existen muchas *procedures* ya implementadas. Una lista no-exhaustiva:

back (bk)	Moverse hacia atrás
forward (fd)	Moverse hacia adelante
right (rt)	Turn right
left (lt)	Turn left
is-<breed>?	Preguntar si es de un tipo?
self	Indica quien es el turtle (yo)
myself	Indica quien solicitó la acción

Lista completa en:
`https://ccl.northwestern.edu/netlogo/docs/dictionary.html#turtlegroup`

Ejercicio

Sin mucha explicación, replicar algo similar a esto:



Introducción a la programación en NetLogo

Crear un entorno

Generar un entorno

Otro elemento importante en los modelos ABM es el entorno. Ya vimos que NetLogo trabaja con **patches**. Ahora vemos como podemos trabajar con ellos. Empezamos a ver simplemente como podemos poner colores diferentes por patch:

1. 'Manual':

```
ask patches[
  let suma ( pxcor + pycor)
  ifelse (remainder suma 2 = 0)
       [set pcolor white]
       [set pcolor blue]
 ]
```

2. Importar una imágen que se traduce en 'entorno'
3. Importar datos de una base de datos para generar el entorno

Vemos eso con ejemplos: `06_InitialiseEnvironment`

Generar un entorno (2)

Netlogo considera *patches* como agentes, así que también podemos agregar valores y poner por ejemplo una etiqueta.

Como para agentes, podemos facilmente definir variables que tiene cada patch:

```
patches-own[capital]
```

Y a continuación podemos usar esta variable como en el caso de agentes:

```
ask patches[
    set capital random 0 10
    set label capital
]
```

Introducción a la programación en NetLogo

Crear vínculos y redes

Generar vinculos entre agentes

En Netlogo los vínculos se llaman `link` y se distinguen dos tipos:

- `directed-link-breed [streets street]`
- `undirected-link-breed [friendships friendship]`

Igual como agentes o patches, podemos agregar variables a los `link`

Para crear los `link` tenemos que conocer tu tipo y su inicio y final:

```
ask turtle 1 [create-friendship-with turtle 3
    ask friendship-with turtle 3 [set label 6]
]
```

En el caso de links dirigidos es un poco diferente:

```
ask turtle 1 [create-debt-to turtle 3
    ask out-debt-to turtle 3 [set amount 5000]
]
```

Extensión para redes (NW)

No es necesario programar todo desde cero, Netlogo tiene una **extensión** que permite generar redes muy fácilmente:[5]

```
extensions [ nw ]
to setup
  clear-all
  nw:generate-watts-strogatz turtles links 25 2 0.1 [ set color red
  setxy random-xcor random-ycor]
end
```

Existen diferentes tipos de redes y funciones para calcular métricas: :

- **Redes**: preferential-attachement, random, watts-strogatz, small-world,lattice-2d, etc
- **Métricas**: centrality measures, closeness-centrality, clustering mesaures, etc
- **Agentsets**: vecions a una distancia en reg (e.g. amigos de amigos)

[5] Para detalles `https://ccl.northwestern.edu/netlogo/docs/nw.html`

Introducción a la programación en NetLogo

`ask` y agent-sets

ask con Agentsets

Tal vez lo más importante en NetLogo es la combinación del comando `ask` con lo que Netlogo llama **Agentsets**. Un **Agentset** es un conjunto de agentes (también puden ser `link` o `patches`).

Vemos el ejemplo más sencillo:

```
1 to step
2    ask coches[
3    forward 1
4    ]
5 end
```

En este ejemplo al **Agentset** se define con `coches`, que en este caso es el conjunto de todos los 'coches'

Existen muchas maneras de definir Agentsets y así definir todas las interacciones. Veamos unos ejemplos.

Diferentes Agentsets

Es imposible hacer una lista exhaustiva de las posibilidades de como hacer Agentsets, pero aquí una lista de los más interesantes:

`other cars`	Selecciona todos los agentes del tipo 'car' diferentes al solicitante
`humans with [color = red]`	Todos los 'humanos' con color azul
`cars in-radius 3`	Autos a menos de 3 patches de distancia
`persons-on neighbors4`	Personas en la vecindad inmediata (4 patches)
`[my-links] of turtle 0`	Todos los links (cualquier tipo) del agente 0
`out-street-neighbors`	Todos los vecinos que estás conectados a través de un link tipo 'street' hacia el otro
`friendship-with turtle self`	Encuentra el 'link' de tipo 'friendship' entre 'turtle' y el solicitante

Diferentes Agentsets - otros elementos

- También podemos seleccionar un agente en función de sus valores: `ask max-one-of cars [sum speed] [ die ]` elimina el coche con mayor velocidad.
- También podemos usar el comando `of` en combinación de Agentsets: `let totalWealth [sum wealth] of persons` nos da la suma de todas las riquezas.
- El comando `any?` permite ver si el AgentSet está vació: `ifelse (any? humans with [speed > 0 ]) [][stop]` termina la simulación cuando ya nadie se mueve.
- `count` permite obtener el número de elementos en un Agentset: `if count turtles = 0 [ stop ]`

Ejercicio

Ejercicio 3 (Red de calles)

Inicio

- *Crear un espacio con un número pequeño de* **casas**
- *Crear calles (directed-link) a un numero pequeño (variable) de otras casas y así crear una pequeña red de calles.*
- *Crear* **personas en los patches de las casas**

Acción

- *Cuando se encuentran en una casa, las personas eligen aleatoriamente una de las calles y ajustan su 'heading' de acuerdo a la calle. Si no hay calle que sale de la casa, se quedan (speed=0)*
- *Caminan una pequeña unidad en su 'heading'.*

Introducción a la programación en NetLogo

Output del modelo

Integrar la interfaz (y el usuario)

Hasta aquí casi no usamos la interfaz. Ahora vemos las opciones que tenemos para hacer el modelo mucho más interactivo con el usuario:

- INPUT (del usuario) - siempre se vinculan a una variable global:
  - **Button**: botones para activar alguna secuencia de comandos (setup,step, etc).
  - **Slider**: Seleccionar valores numéricos en un rango determinado
  - **Switch**: Input para variables binarias (como una checkbox)
  - **Chooser**: Permitir seleccionar de un número limitado de opciones
  - **Input**: Campo de texto para poner información
- Output
  - **Monitor**: mostrar un número en particular
  - **Plot**: gráficas!
  - **Output/Note**: un texto más amplio

Introducción a la programación en NetLogo

Experimentos

Experimentos

Una vez que el modelo funciona, se pueden hacer los experimentos (científicos). Netlogo tiene una herramienta integrada que ayuda mucho:

- Ir a Tools $\Rightarrow$ BehaviorSpace (CTRL+Shift+B)
- Crear un nuevo experimento
- Elegir los parámetros y el número de repeticiones
- Elegir lo que se debe exportar
- Guardar
- Correr el experimento y elegir como guardar la información.

Referencias

Chávez-Juárez, Florian, "On the Role of Agent-based Modeling in the Theory of Development Economics," Review of Development Economics, Online first, Available at `http://dx.doi.org/10.1111/rode.12264` 2016.

__ , **Raquel Yunoen Badillo Salas, and Victoria Hernández Sistos**, "Social Mobility, Economic Growth and Socioeconomic Inequality in an Economy without Informality and with Social," Laboratorio Nacional de Políticas Públicas. Mimeo 2016.

Gilbert, Nigel and Klaus G. Troitzsch, *Simulation for the Social Scientist*, 2nd ed., Open University Press, 2005.

Grimm, Volker, Uta Berger, Donald L. DeAngelis, J. Gary Polhill, Jarl Giske, and Steven F. Railsback, "The ODD protocol: A review and first update," *Ecological Modelling*, 2010, *221* (23), pp.2760–2768.

__ , __ , **Finn Bastiansen, Sigrunn Eliassen, Vincent Ginot, Jarl Giske, John Goss-Custard, Tamara Grand, Simone K. Heinz, Geir Huse, Andreas Huth, Jane U. Jepsen, Christian Jørgensen, Wolf M. Mooij, Birgit Müller, Guy Peer, Cyril Piou, Steven F. Railsback, Andrew M. Robbins, Martha M. Robbins, Eva Rossmanith, Nadja Rüger, Espen Strand, Sami Souissi, Richard A. Stillman, Rune Vabø, Ute Visser, and Donald L. DeAngelis**, "A standard protocol for describing individual-based and agent-based models," *Ecological Modelling*, 2006, *198* (1), pp. 115–126.

Miller, John H. and Scott E. Page, *Complex Adaptive Systems - An introduction to computational model of social life*, Princeton University Press, 2007.

Polhill, J. Gary, "ODD Updated," *Journal of Artificial Societies and Social Simulation*, 2013, *4* (9), Aritcle 9, http://jasss.soc.surrey.ac.uk/13/4/9.html.

Read, Carveth, *Logic: Deductive and Inductive*, Simpkin, Marshall, Hamilton, Kent & Co. Ltd., London, 1914.

Wendelspiess Chávez Juárez, Florian, "Three essays on inequality of opportunity and social mobility," PhD Thesis, University of Geneva. Available at `http://archive-ouverte.unige.ch/unige:40023` 2014.