Programación probabilística y problemas inversos en ecuaciones diferenciales ordinarias

Simulación de sistemas
Escuela de Gobierno y Transformación Pública, ITESM
Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, UNAM

27 de marzo de 2022

Escuela de Gobierno y Transformación Pública
Tecnológico de Monterrey

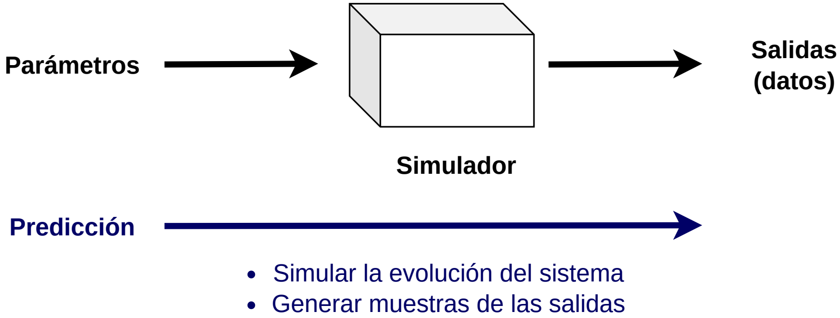


Figura: Adaptado de Atılım Güneş Baydin, *Probabilistic Programming for Inverse Problems in the Physical Sciences*. Recurso : `https://www.youtube.com/watch?v=E3_Ey0z068o`

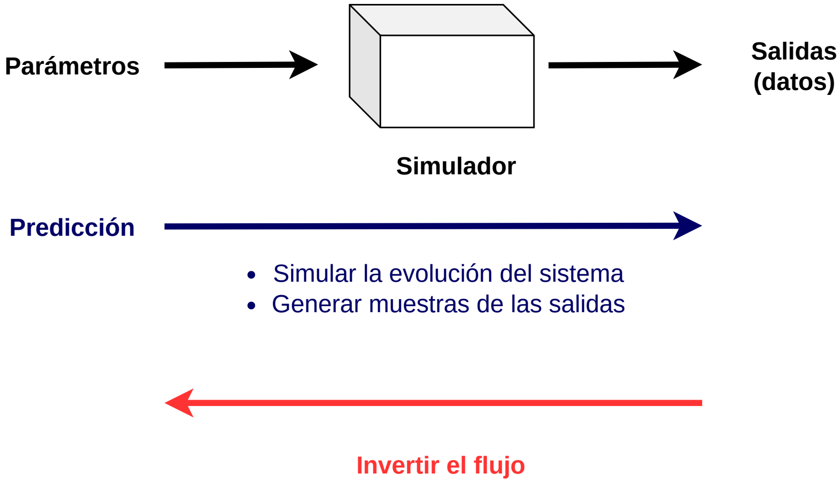
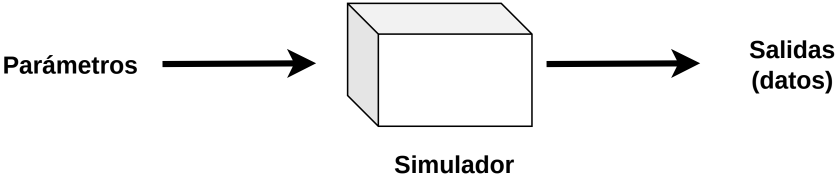


Figura: Adaptado de Atılım Güneş Baydin, *Probabilistic Programming for Inverse Problems in the Physical Sciences*. Recurso : `https://www.youtube.com/watch?v=E3_Ey0z068o`



Predicción

- Simular la evolución del sistema
- Generar muestras de las salidas

Inferencia

- **Encontrar los parámetros que producen (explican) los datos observados.**
- **Problema inverso.**

Figura: Adaptado de Atılım Güneş Baydin, *Probabilistic Programming for Inverse Problems in the Physical Sciences*. Recurso : `https://www.youtube.com/watch?v=E3_Ey0z068o`

Reconstrucción de redes

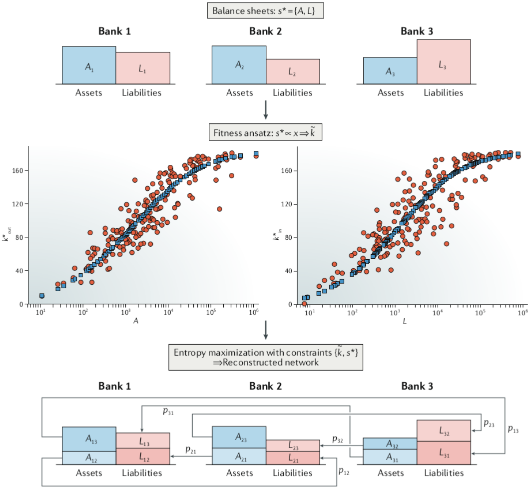


Figura: Tomado de Cimini, G., Squartini, T., Garlaschelli, D. & Gabrielli, A Systemic risk analysis on reconstructed economic and financial networks. Sci. Rep. 5, 15758 (2015).

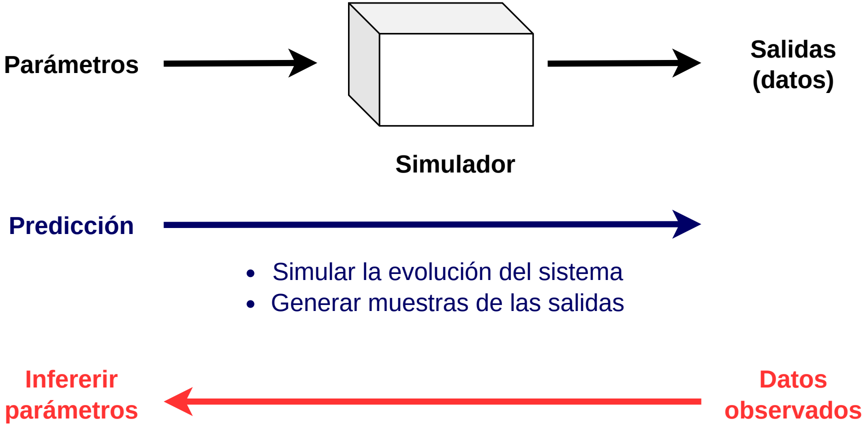


Figura: Adaptado de Atılım Güneş Baydin, *Probabilistic Programming for Inverse Problems in the Physical Sciences*. Recurso : `https://www.youtube.com/watch?v=E3_Ey0z068o`

Programación probabilística

- Es un enfoque de aprendizaje de máquinas que permite escribir programas que representan modelos probabilísticos.
- La programación probabilística soporta declaraciones probabilísticas como declaración de variables aleatorias.
- Permite la inferencia (bayesiana) de parámetros condicionados en datos observados.

```
@model gdemo(x) = begin
  s ~ InverseGamma(2,3)
  m ~ Normal(0,sqrt(s))

  for i=1:lengh(x)
    x[i] ~ Normal(m, sqrt(s))
  end

end
```

Frameworks para programación probabilística

(a) Pyro

(b) Stan

Turing.jl

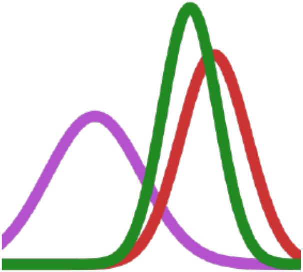


Figura: https://turing.ml/stable/

¿Por qué otro lenguaje de alto nivel?

Características típicas

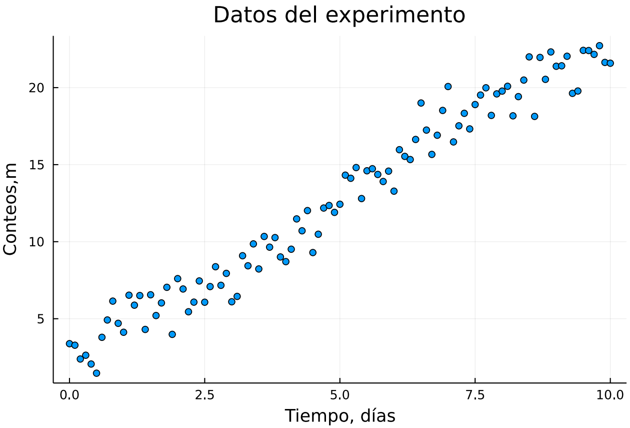
- Tipado dinámico.
- Sintaxis de alto nivel.
- Open source.
- Built-in package manager.

Características novedosas

- Buen perfomance: tiempo de ejecución rápido.
- Compilación JIT (Just in Time) a nivel de función.
- En un porcentaje alto, Julia está escrito en Julia.
- Metaprogramación (macros).

Ecuaciones diferenciales ordinarias[1]

Realizamos experimentos donde se inoculan placas de agar con bacterias en el tiempo 0.



[1] Tomado de las notas de Ben Lampert sobre inferencia bayesiana https://ben-lambert.com/bayesian-short-course/

- Queremos modelar el crecimiento de una población de bacterias.
- Se asume el siguiente modelo de crecimiento poblacional,

$$\frac{dN}{dt} = \alpha N(1 - \beta N)$$

donde $\alpha > 0$ es la tasa de crecimiento debida a la división celular y $\beta > 0$ mide la reducción en la trasa de crecimiento debido a la aglomeración.

¿Cómo inferimos los parámetros de este modelo?

Asumamos un error de medición alrededor del valor verdadero:

$$N^*(t) \sim \text{normal}(N(t), \sigma)$$

donde:

- $N^*(t)$ es el conteo de bacterias en el tiempo t.
- $N(t)$ es la solución de la ODE en el tiempo t (número verdadero de bacterias en la placa).
- $\sigma > 0$ mide la magnitud del error de medición alrededor del valor verdadero.

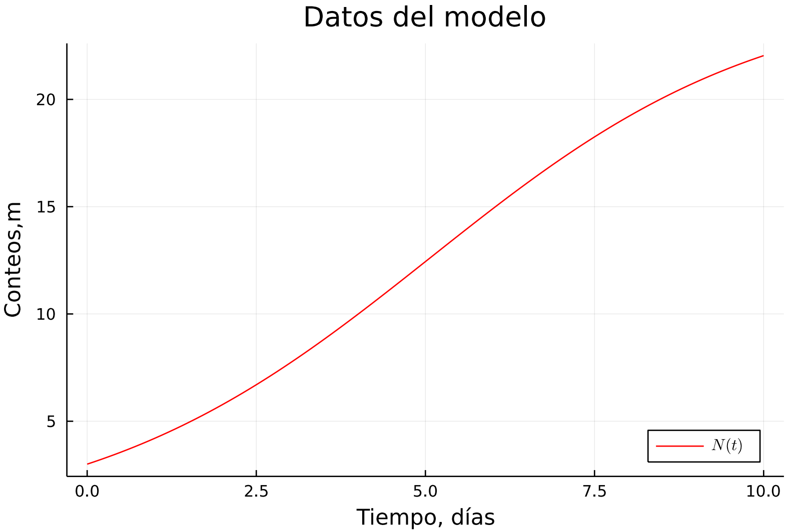
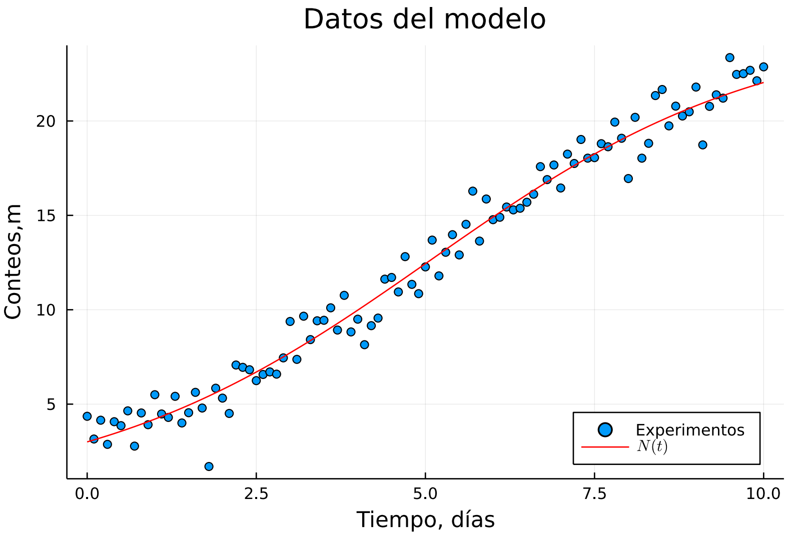


Figura: Número verdadero de bacterias, $N(t)$

Superposición de distribución de muestreo que representa el error de medición y de los datos experimentales.



Dado que estamos usando la distribución normal,

$$N^*(t) \sim \text{normal}(N(t), \sigma)$$

la verosimilitud de las observaciones es

$$L(N(t), \sigma) = \prod_{t=t_1}^{T} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{-(N^*(t) - N(t))^2}{2\sigma^2}\right]$$

¿Cómo calculamos $N(t)$?

$$\frac{dN}{dt} = \alpha N(1 - \beta N)$$

Cualquier solución de $N(t)$ -exacta o numérica- depende de los parámetros del modelo de ODE. En este caso,

$$N(t) = f(t, \alpha, \beta)$$

¿Cómo integramos los datos observados para estimar estos parámetros?

Inferencia bayesiana

- En un contexto de inferencia bayesiana, donde se tiene un modelo muestral $Y \sim p(y|\theta)$ y una distribución inicial $p(\theta)$, la distribución final se calcula de la siguiente forma:

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{\int p(\theta')p(y|\theta')d\theta'}$$

- Aún descartando el denominador (el cual se puede interpretar como un factor de escalamiento que hace que la función integre a 1), el cálculo analítico de $p(\theta|y)$, en la mayoria de las ocasiones, es complicado.

- Hay distintas técnicas para calcular $p(\theta|y)$ (aproximación de Laplace, métodos de cuadratura, etc), pero por la capacidad de cómputo disponible las técnicas de simulación de Monte Carlo vía cadenas de Markov (MCMC) han sido mayormente utilizadas.
- La idea de MCMC es **construir una cadena de Markov que sea fácil de simular (a través de un proceso de muestreo) y cuya distribución de equilibrio corresponda a la distribución final de interés**.

- En el contexto de nuestro problema,
  - $p(\theta)$ puede como la distribución inicial de los parámetros del modelo de ODE,
  - $p(y|\theta)$ (el modelo muestral o distribución de verosimilitud) representa el modelo de ODE.
- Podemos utilizar las técnicas de MCMC para construir la distribución $p(\theta|y)$, de los parámetros del modelo de ODE.

Turing

Turing es un lenguaje de programación probabilística de propósito general para una inferencia bayesiana robusta y eficiente. Las características actuales incluyen:

- Programación probabilística de propósito general con una interfaz de modelado intuitiva;
- Muestreo de Monte Carlo hamiltoniano (HMC) robusto y eficiente para distribuciones posteriores diferenciables;
- Muestreo de partículas MCMC para distribuciones posteriores complejas que involucran variables discretas y flujo de control estocástico; y
- Inferencia composicional a través del muestreo de Gibbs que combina partículas MCMC, HMC y paseo aleatorio MH (RWMH).

Algoritmo de Metrópolis

- Considerando el caso en que se tiene un modelo muestral $Y \sim p(y|\theta)$ y una distribución inicial $p(\theta)$.
- La distribución final es difil de calcular por la integral en el denominador:
$$p(\theta|y) = \frac{p(\theta)p(y|\theta)}{\int p(\theta')p(y|\theta')d\theta'} \tag{1}$$
- Si se pudiera obtener muestras de $p(\theta|y)$, se podría generar $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(S)} \sim$ i.i.d.$p(\theta|y)$ y obtener aproximaciones con el método de Monte Carlo.
- El problema es cuando no se pueden obtener muestras directamente de $p(\theta|y)$.

- En términos de aproximar la distribución posterior, lo crítico no es tener muestras i.i.d. de $p(\theta|y)$ sino que poder construir una gran colección de valores de θ, cuya distribución empírica aproxime $p(\theta|y)$
- Suponiendo que se cuenta con una colección $\{\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(s)}\}$ a la cual queremos agregar un valor nuevo θ^{s+1}.
- Se considera agregar un valor θ^{*} que es cercano a θ^{s}.
- **¿Qué regla de decisión utilizamos?**

- Si $p(\theta^*|y) > p(\theta^{(s)}|y)$, se prefieren más valores de θ^* en el conjunto que los valores de $\theta^{(s)}$.
- Dado que $\theta^{(s)}$ ya está en el conjunto, entonces parece que deberíamos incluir también a θ^*.
- En el caso contrario, si $p(\theta^*|y) < p(\theta^{(s)}|y)$ no es necesario incluir θ^*.
- Así, parece que la decisión de incluir o no a θ^* depende de la comparación entre $p(\theta^*|y)$ y $p(\theta^{(s)}|y)$.
- Afortunadamente, esta comparación se puede hacer incluso si no podemos calcular $p(\theta|y)$

$$r = \frac{p(\theta^*|y)}{p(\theta^{(s)}|y)} = \frac{p(y|\theta^*)p(\theta^*)}{p(y)} \frac{p(y)}{p(y|\theta^{(s)})p(\theta^{(s)})} = \frac{p(y|\theta^*)p(\theta^*)}{p(y|\theta^{(s)})p(\theta^{(s)})} \tag{2}$$

Habiendo calculado r, ¿qué tenemos que hacer?

- Si $r > 1$:
  - **Intuición**: Dado que $\theta^{(s)}$ ya está en el conjunto, debemos incluir a θ^* ya que tiene una probabilidad mayor que $\theta^{(s)}$.
  - **Decisión**: Aceptar θ^* en nuestro conjunto, *i.e.* definir $\theta^{s+1} = \theta^*$.
- Si $r < 1$:
  - **Intuición**: La frecuencia relativa de valores θ en el conjunto que es igual a θ^*, comparado con aquellos valores iguales a $\theta^{(s)}$ está dada por la expresión $\dfrac{p(\theta^*|y)}{p(\theta^{(s)}|y)} = r$. Esto significa que por cada instancia de $\theta^{(s)}$, deberíamos tener sólo una fracción de valores θ^*.
  - **Decisión**: Definir θ^{s+1} igual a θ^* o $\theta^{(s)}$ con probabilidad r y $1 - r$ respectivamente.

Algoritmo Metropolis

- Lo anterior es la intuición básica del algoritmo de **Metropolis**.
- El algoritmo de Metropolis procede al muestrear un valor θ^* cercano al valor actual $\theta^{(s)}$ usando una distribución simétrica $J(\theta^*|\theta^{(s)})$.
- En este contexto, simetría significa que $J(\theta_b|\theta_a) = J(\theta_a|\theta_b)$, *i.e.* la probabilidad de proponer $\theta^* = \theta_b$ dado que $\theta^{(s)} = \theta_a$ es igual a la probabilidad de proponer $\theta^* = \theta_a$ dado que $\theta^{(s)} = \theta_b$.
- Algunos ejemplos de $J(\theta^*|\theta^{(s)})$ son :
  - $J(\theta^*|\theta^{(s)}) = \text{uniform}(\theta^{(s)} - \delta, \theta^{(s)} + \delta)$
  - $J(\theta^*|\theta^{(s)}) = \text{normal}(\theta^{(s)}, \delta^2)$
- El valor del parámetro δ generalmente se elige para hacer que el algoritmo se ejecute de manera eficiente.

- **Habiendo obtenido un valor propuesto θ^* se añade al conjunto este valor o a una copia de $\theta^{(s)}$, dependiendo de la proporción** $r = \dfrac{p(\theta^*|y)}{p(\theta^{(s)}|y)}$

De forma específica, dado $\theta^{(s)}$, el algoritmo Metropolis genera un valor $\theta^{(s+1)}$ de la siguiente forma:

1. Tomar una muestra $\theta^* \sim J(\theta|\theta^{(s)})$
2. Calcular la proporción de aceptación

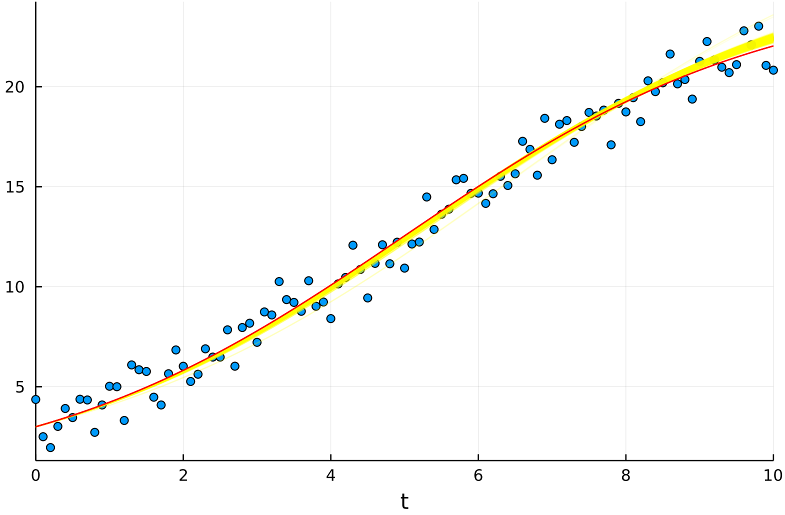
$$r = \frac{p(\theta^*|y)}{p(\theta^{(s)}|y)} = \frac{p(y|\theta^*)p(\theta^*)}{p(y|\theta^{(s)})p(\theta^{(s)})}$$

3. Sea

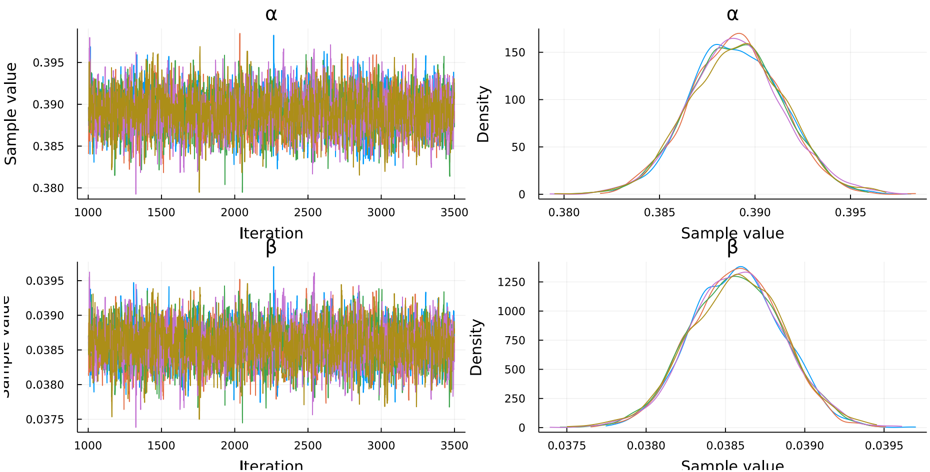
$$\theta^{(s+1)} = \begin{cases} \theta^* & \text{con probabilidad } \min(r, 1) \\ \theta^{(s)} & \text{con probabilidad } 1 - \min(r, 1) \end{cases} \tag{3}$$

El paso 3 se puede obtener al tomar una muestra $u \sim \text{uniform}(0, 1)$ y definir $\theta^{s+1} = \theta^*$ si $u < r$ y $\theta^{s+1} = \theta^{\theta^{(s)}}$ en otro caso.

¡Vamos a programar!



La distribución posterior (línea amarilla) reproduce con bastante precisión la solución *verdadera* del ODE.


α
Sample value
0.395
0.390
0.385
0.380
1000
1500
2000
2500
3000
3500
Iteration
α
Density
150
100
50
0
0.380
0.385
0.390
0.395
Sample value
β
Sample value
0.0395
0.0390
0.0385
0.0380
0.0375
1000
1500
2000
2500
3000
3500
Iteration
β
Density
1250
1000
750
500
250
0
0.0375
0.0380
0.0385
0.0390
0.0395
Sample value