# Proyecto Final: Comparación de un método de tipo Montecarlo y resolución directa de una EDP

Johnny Godoy [1] Javier Santidrián [2] Patricio Yáñez [2]

[1]Universidad de Chile-Departamento de Ingeniería Matemática

## Motivación del Problema

Este proyecto busca comparar algoritmos de tipo Montecarlo con algoritmos de diferencias finitas para la resolución de una EDP, ya sea en términos de eficacia (error de aproximación), eficiencia (tiempo de ejecución) y la complejidad de código (escalabilidad a otros parámetros y problemas). Exploramos distintos métodos y variaciones de ellos, incluso proponiendo los nuestros, estudiando las ventajas y desventajas de cada uno.

Donde se busca resolver la ecuación de Poisson con condiciones de Dirichlet:

$$\left.\begin{aligned} -\Delta u &= f \text{ sobre } \Omega \\ u &= g \text{ en } \partial\Omega \end{aligned}\right\} \tag{1}$$

El código será capaz de resolver cualquiera de estos casos, pero para la realización de experimentos consideramos la misma EDP que en el laboratorio 1:

$$\left.\begin{aligned} -\Delta u = 0 \qquad & \text{sobre } \Omega = [0,1]^2 \\ u(0,y) = u(1,y) = 0 \qquad & \text{en } 0 \le y \le 1 \\ u(x,0) = 0 \qquad & \text{en } 0 \le x \le 1 \\ u(x,1) = \sin(\pi x) \qquad & \text{en } 0 \le x \le 1 \end{aligned}\right\} \tag{1} \tag{2}$$

y como conocemos la solución analítica de este problema:

## Algoritmo de Montecarlo y visualización de un paseo aleatorio

Los algoritmos a utilizar se basan en simular un paseo aleatorio simple desde un punto $x_0 \in \text{int}(\Omega)$ hasta que llegue a un punto $\xi \in \partial\Omega$. Con esto fijando el parámetro y simulando el paseo tenemos


Figure 1. Visualización de un paseo aletorio

## Algoritmo Propuesto del Reporte de Referencia

El algoritmo propuesto recibe como entrada un $x_0 \in \text{int}(\Omega)$ y aproxima $u(x_0)$ como sigue:

0. Inicializar un diccionario que asocia los puntos de borde $\xi \in \partial\Omega$ al valor 0. 1. Repetir $W$ veces:

1.1. Correr un paseo aleatorio $w$ desde $x_0$ hasta llegar a un punto de borde $\xi$

1.2. Calcular $F(w) = \sum_{i \in w \setminus \{\xi\}} f(i)h^2$

1.3. Actualizar el diccionario $C[\xi] += 1$

2. Calcular la probabilidad de llegar a cada punto de borde como $P(\xi) = \frac{C[\xi]}{W}$ 3. Retornar $u(x_0) = \sum_{\xi \in \partial\Omega} P(\xi)g(\xi) - \sum_w F(w)$

## Estimación bayesiana

En el paso 5 calculamos la probabilidad desde un enfoque frecuentista. En particular, si el evento de caer en un punto de borde no ocurre, se le asigna probabilidad 0, lo cual puede problemático, pues el evento no necesariamente es imposible.

Esto ocurre si es que no se corren suficientes iteraciones $W$. Si queremos que se llegue a puntos del borde, necesitamos al menos $O(n)$ iteraciones para cada punto $x_0$.

Entonces, proponemos utilizar un estimador bayesiano con prior conjugado Dirichlet de parámetro $\alpha$:

$$P(\xi) = \frac{C[\xi] + \alpha_\xi}{W + \sum_{\kappa \in \partial\Omega} \alpha_\kappa}$$

En el denominador el término $\sum_{\kappa \in \partial\Omega} \alpha_\kappa$ se interpreta como nuestra **confianza en el prior**.

Nuestras dos propuestas de prior son las siguientes:

$$\alpha_\xi = \frac{1}{2\sum_{\kappa \in \partial\Omega} g(\kappa)} \forall \xi \in \partial\Omega, \qquad \alpha_\xi = \frac{1}{|i - \xi_i| + |j - \xi_j|} \forall \xi \in \partial\Omega$$

El primero es un prior constante, escalado para que valores altos de $g$ no tengan demasiada influencia. El segundo prior supone que los puntos más cercanos (en sentido $L^1$) son más probables para el paseo aleatorio - este supuesto utiliza la convexidad del dominio y falla sin tenerla.

## Comparación de Priors

Luego de implementar los diferentes estimadores, logramos visualizar cada uno y comparar los priors, con resultados como Error cuadrático Frequentist Solver $6.302253279629374$, Laplace Smoothing Solver, $6.180990015724915$ y Manhattan Smoothing Solver $5.381444368557077$

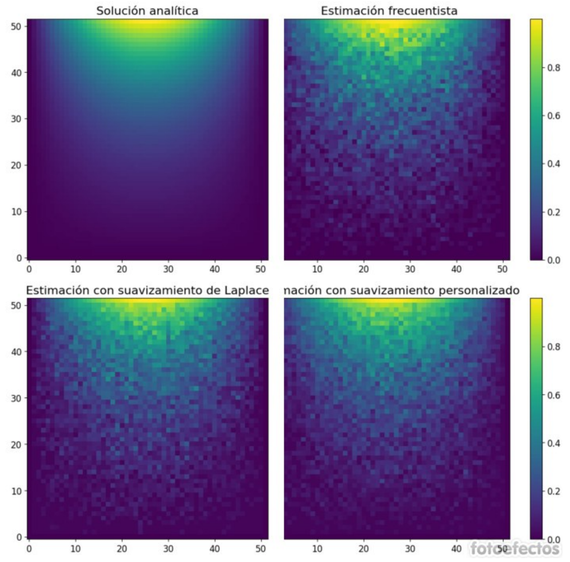


Figure 2. Solución analítica y estimadores[Frecuentista, con suavizamiento de Laplace y suavizamiento personalizado]

## Algoritmo Alternativo

Se propone un algoritmo distinto que aprovecha la propiedad de Markov: Cada uno de los caminos tiene subcaminos que pueden usarse para mejorar la estimación.

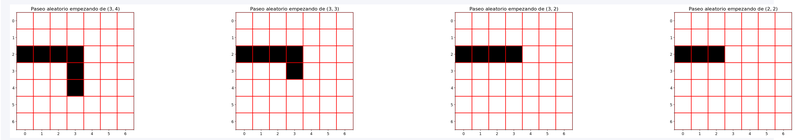


Figure 3. Visualización de paseos aleatorios

Nuestro método mejorado busca generar de inmediato $u$ en $\Omega$ realizando también Montecarlo en el punto inicial.

0. Inicializar $U, P, R$ todos arreglos nulos en todas sus componentes 1. Inicializar arreglo $F$ con $F[x,y] = -f(x,y)h^2$ 2. Repetir $W$ veces:

2.1. Elegir aleatoriamente un punto interior $x_0$

2.2. Correr un paseo aleatorio $w$ desde $x_0$ hasta $\xi$

2.3. Para cada subcamino $w_i$ desde $x_i$ hasta $\xi$:

2.3.1. Actualizar $P[x_i] += 1$

2.3.2. Actualizar $R[x_i] += g(\xi)$

2.3.2. Actualizar $U[x_i] += \sum_{i \in w \setminus \{\xi\}} F[x_i]$

3. Actualizar $U += C/P$

Acá $R$ se interpreta como los valores $g(\xi)C[\xi]$ del numerador de la expresión del algoritmo anterior. $P$ se interpreta por otro lado como los caminos totales que pasaron por el punto, y corresponde al denominador de la expresión del algoritmo anterior. $U$ almacena la solución final, durante las iteraciones se actualiza el término de $F(w)$ y al terminar todas estas se actualiza la otra sumatoria.

El último paso está bien definido si es que todos los puntos interiores han sido visitados (no se actualizan los puntos de borde pues los tenemos fijos por la condición de Dirichlet), por lo tanto debería elegirse $W$ suficientemente grande para garantizar esto.

## Conclusiones

- Métodos clásicos como diferencias finitas logran una aproximación más veloz y más exacta (menor error) a la solución de la EDP. Una de las razones de la mejora de velocidad es que los métodos clásicos de manejo de matrices están implementadas muy eficientemente con el paquete SciPy, desempeño que no se puede lograr con Python puro.
- Métodos de Montecarlo son más fáciles de modificar: Cambiar a un dominio perforado solamente requiere cambiar la detección de bordes en la simulación del paseo, mientras que en el caso de diferencias finitas hay que asegurarse de ajustar las matrices de tal forma que los puntos de la grilla no se salgan del dominio.
- El código para realizar simulaciones de Montecarlo no es complejo, pero nuestras modificaciones que mejoran velocidad y desempeño aumentan la complejidad del código.
- Los métodos de diferencias finitas sufren de la llamada maldición de la dimensionalidad, pues la matriz tiene tamaño $O(n^{2k})$ con $k$ la dimensión del espacio. Los métodos de Montecarlo no sufren esta debilidad de manera tan marcada, por lo cual podrían verse aventajados cuando $k > 2$. Esto no se probó por limitaciones de tiempo.

## References