PUCE Virtual

CLASE

2

MÉTODOS NUMÉRICOS

Precisión, Exactitud y Cifras Significativas

QS WORLD UNIVERSITY RANKINGS

Educación de calidad

INTRODUCCIÓN DE LA CLASE

GRADO / POSGRADO / TECNOLOGÍAS

Estudia a tu tiempo
son interrupciones

EDUCACIÓN EN LÍNEA DE CALIDAD

En esta clase se profundiza en los conceptos de precisión y exactitud dentro del cálculo numérico, diferenciando claramente ambos términos y su impacto en la calidad de los resultados, entendiendo la exactitud como la cercanía al valor verdadero y la precisión como la consistencia de los resultados bajo las mismas condiciones, lo que permite reconocer que un resultado puede ser preciso sin ser exacto y viceversa, analizando su importancia en la interpretación de resultados y en la validación de algoritmos, junto con el uso adecuado de cifras significativas para expresar valores acordes al nivel de incertidumbre, considerando además cómo la representación numérica influye en la percepción de exactitud y cómo un mal manejo puede generar errores de interpretación, incorporando también el concepto de condicionamiento para evaluar la sensibilidad de un problema ante pequeñas variaciones en los datos y el principio de estabilidad numérica para asegurar que los errores no crezcan de forma descontrolada, con el objetivo de que el estudiante pueda evaluar la confiabilidad de los resultados y comprender la relación entre precisión, estabilidad y sensibilidad.

Además, se introduce la solución numérica de ecuaciones no lineales como un problema central en el análisis aplicado, planteado como la búsqueda de raíces de funciones de la forma f(x)=0, destacando que en muchos casos no existen soluciones analíticas y es necesario recurrir a métodos iterativos, apoyándose en ejemplos de física, ingeniería y economía para entender su importancia, comenzando con métodos cerrados como los gráficos para identificar intervalos donde existe solución, seguido del método de bisección basado en el teorema del valor intermedio que garantiza convergencia bajo ciertas condiciones, y el método de falsa posición que mejora la aproximación mediante interpolación lineal, permitiendo al estudiante formular problemas reales, aplicar estos métodos y analizar la precisión y eficiencia de las soluciones obtenidas.

Resultado o resultados de aprendizaje que será abordado con el contenido de la clase.

Analizar problemas matemáticos y computacionales utilizando métodos numéricos adecuados para su solución aproximada.

Precisión, Exactitud y Cifras Significativas

En las clases anteriores hemos visto la definición de métodos numéricos y cómo los errores son inevitables en cualquier cálculo computacional. Ahora vamos a profundizar en conceptos que determinan la calidad de esos cálculos: precisión, exactitud, cifras significativas, condicionamiento y estabilidad. Estos elementos son cruciales en ciberseguridad porque afectan directamente la fiabilidad de operaciones críticas como la generación de claves criptográficas, el análisis de canales laterales o el entrenamiento de modelos de detección de amenazas. Un mal manejo de estos conceptos puede llevar a que un algoritmo “correcto en teoría” falle en la práctica, exponiendo sistemas a ataques.

Al final de esta clase entenderás cómo evaluar la precisión de un cálculo numérico, por qué un problema puede ser inherentemente sensible (mal condicionado) y cómo elegir algoritmos estables para minimizar riesgos en implementaciones de seguridad informática.

Precisión y exactitud en el cálculo numérico

La exactitud (accuracy) mide qué tan cerca está el valor calculado del valor verdadero (o teóricamente correcto). Un resultado exacto es aquel que coincide con la realidad matemática lo más posible.

La precisión (precision) se refiere a cuántos dígitos confiables tiene el resultado, es decir, la reproducibilidad o consistencia de los cálculos repetidos. Puedes tener alta precisión (muchos dígitos iguales en repeticiones) pero baja exactitud (todos alejados del valor verdadero), o viceversa.

En ciberseguridad, la analogía del tiro al blanco es muy útil:

Alta exactitud y alta precisión: el ataque side-channel recupera la clave correcta con consistencia.

Baja exactitud pero alta precisión: un modelo de detección de malware clasifica consistentemente tráfico benigno como malicioso (falsos positivos repetidos).

Baja precisión: variabilidad alta en correlaciones de power analysis, haciendo impredecible la extracción de claves.

En computadoras, la precisión está limitada por IEEE 754 (doble precisión: ~15-16 dígitos decimales). En criptografía con números grandes, usamos precisión arbitraria para mantener exactitud en operaciones modulares.

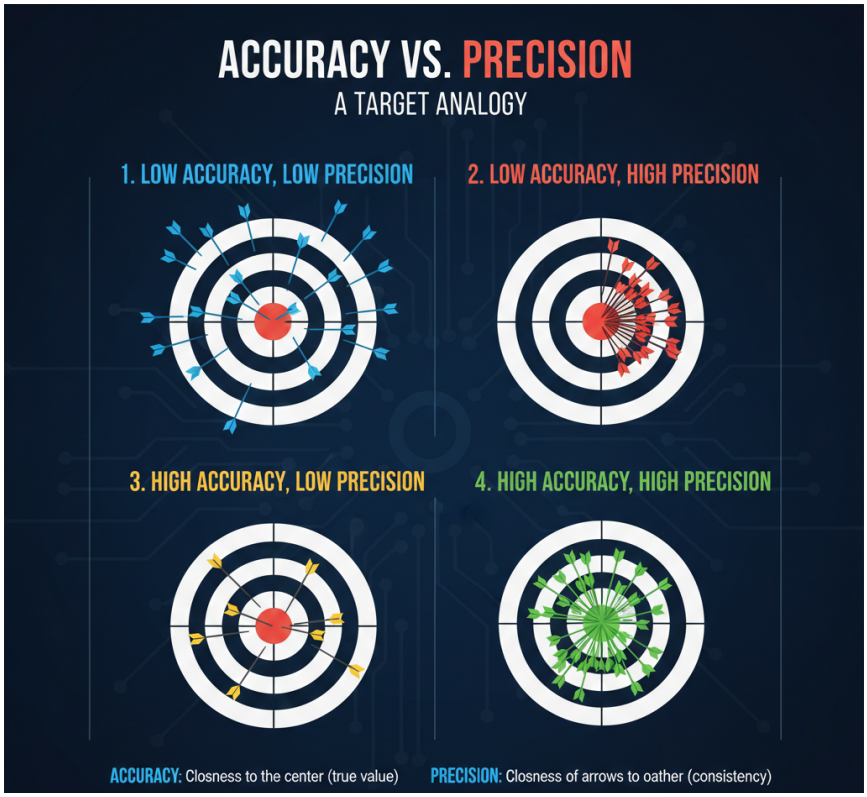


Figura 1: Analogía del tiro al blanco ilustrando precisión vs exactitud (cuatro combinaciones: baja/alta).

Creación propia.

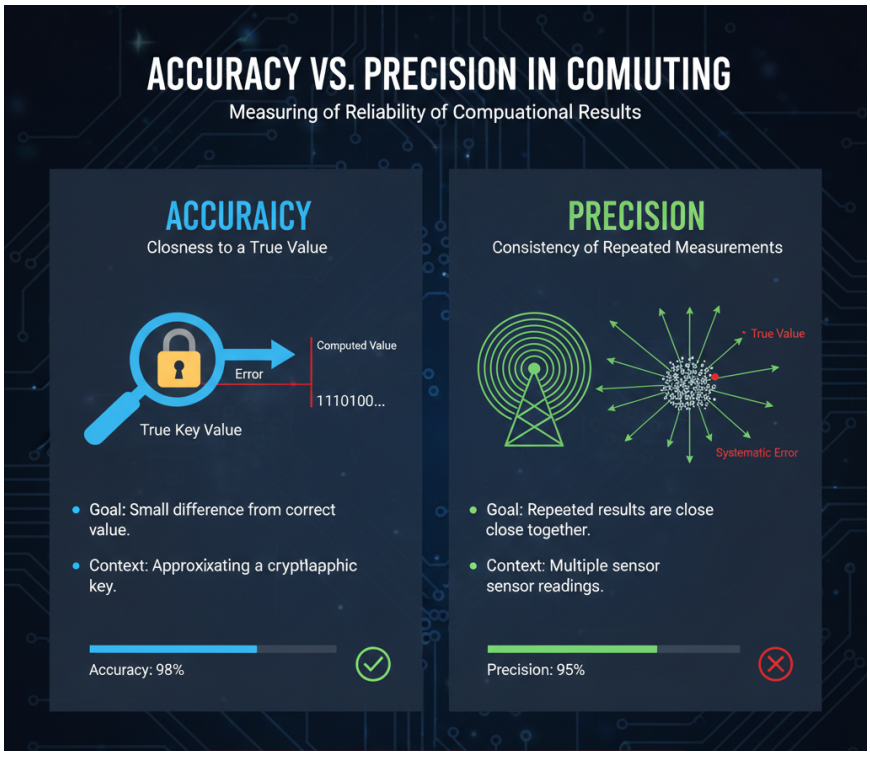


Figura 2: Representación visual de precisión y exactitud aplicada a mediciones en contextos computacionales.

Creación propia.

En los métodos numéricos utilizados en criptografía, como la búsqueda de claves en sistemas de cifrado, es posible que la precisión de los cálculos esté vinculada con la capacidad de aproximarse correctamente a una clave válida sin incurrir en errores sistemáticos derivados de representaciones numéricas limitadas. La exactitud es esencial aquí porque un pequeño desajuste en las estimaciones

de la clave puede llevar a errores de desencriptación o incluso a una vulnerabilidad explotable en el sistema criptográfico. Sin embargo, en algunos casos, un pequeño margen de imprecisión controlada podría no comprometer la seguridad, siempre y cuando el cálculo esté bien condicionado y las medidas de error sean aceptables.

Por lo tanto, la distinción entre precisión y exactitud en este contexto de mediciones computacionales no solo se refiere a la fiabilidad de los cálculos, sino también a la seguridad general del sistema, ya que una baja exactitud o una baja precisión pueden introducir errores que propagan fallos o vulnerabilidades en sistemas criptográficos, haciéndolos potencialmente más susceptibles a ataques numéricos. Así, el equilibrio entre ambos factores es esencial para mantener la seguridad y estabilidad en los sistemas de cómputo que dependen de cálculos exactos y consistentes, como es el caso de la mayoría de los protocolos criptográficos modernos.

Cifras significativas

Las cifras significativas indican cuántos dígitos del número son confiables y contribuyen a la precisión del resultado.

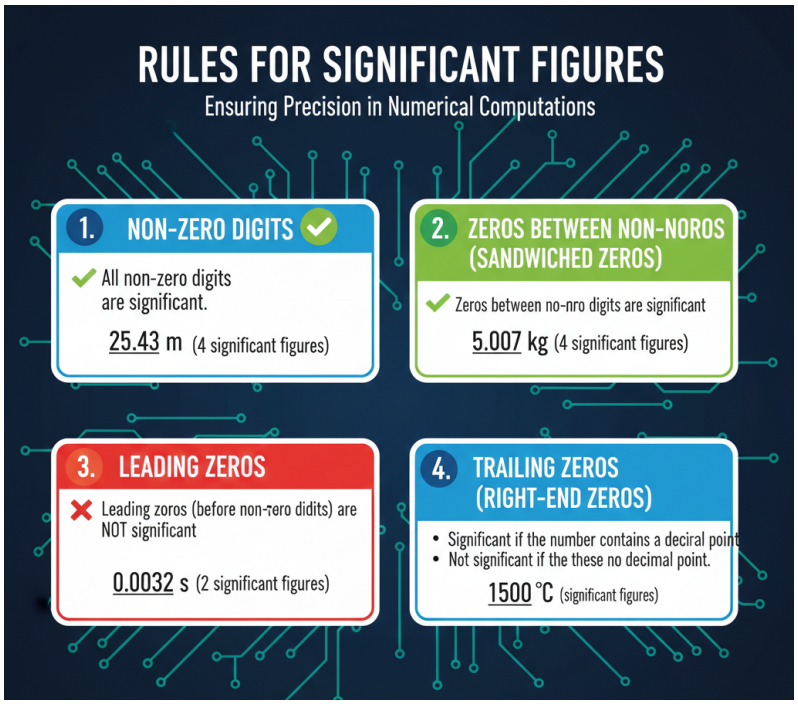
Reglas básicas:

Todos los dígitos no cero son significativos.

Los ceros entre dígitos no cero son significativos.

Los ceros al final después del punto decimal son significativos.

Los ceros al inicio no lo son (excepto en notación científica).



Ejemplo: 0.001230 tiene 4 cifras significativas (1,2,3,0); 1230 tiene 3 o 4 dependiendo del contexto (ambiguo sin punto decimal).

Figura 3: Diagrama de reglas para determinar cifras significativas en números.

Creación propia.

Estas reglas son de gran relevancia en cálculos criptográficos y análisis de seguridad cuando los algoritmos trabajan con valores numéricos críticos, como claves o valores hash, donde la exactitud y precisión en los cálculos es esenciales para garantizar la integridad del sistema. Las cifras significativas determinan cuánta información precisa se puede extraer de los datos, lo que puede afectar directamente la robustez de un sistema criptográfico frente a posibles ataques.

Si los cálculos o mediciones dentro de un algoritmo criptográfico no se realizan con suficiente precisión (es decir, si se pierden cifras significativas debido a errores de redondeo o truncamiento), la seguridad de las claves o la fiabilidad de las funciones hash puede verse comprometida, haciendo que el sistema sea susceptible a errores numéricos o incluso a vulnerabilidades explotables.

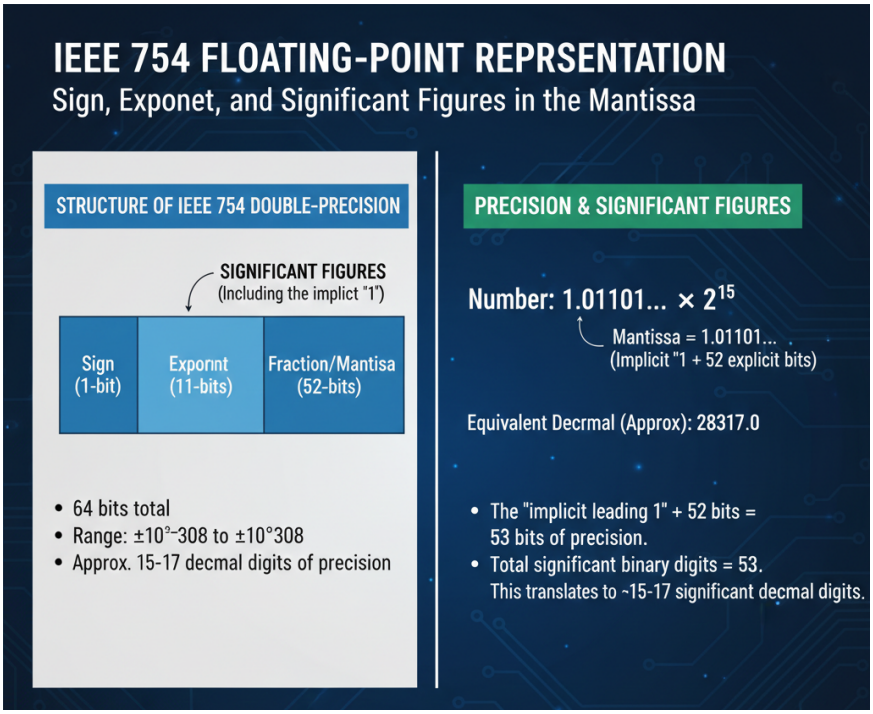


Figura 4: Representación de IEEE 754 mostrando signo, exponente y mantisa (cifras significativas en la fracción).

Creación propia.

La representación IEEE 754 en su formato de doble precisión (64 bits) es un estándar ampliamente utilizado para representar números en punto flotante en las computadoras. Este formato descompone el número en tres componentes: el signo, el exponente y la mantisa (o fracción). Cada una de estas partes tiene un papel fundamental en la precisión, el rango de valores y cómo se realiza el cálculo, especialmente cuando se trata de cálculos numéricos que involucran operaciones criptográficas y análisis en sistemas de ciberseguridad.

Enlace 1: https://www.youtube.com/watch?v=D7DN3KqNkjk Ver en YouTube: Condition Number and Numerical Stability in Matrix Computations Este video explica el número de condición y su impacto en cálculos sensibles como los usados en criptografía basada en lattices.

Condicionamiento de problemas numéricos

El condicionamiento mide la sensibilidad intrínseca de un problema matemático a pequeñas perturbaciones en los datos de entrada. Se cuantifica con el número de condición (κ):

κ ≈ 1: bien condicionado (pequeños errores de entrada generan pequeños errores de salida).

κ grande (e.g., 10^{10} o más): mal condicionado (errores pequeños se amplifican enormemente).

Para matrices (común en lattices o sistemas lineales en criptoanálisis): $\kappa(A) = ||A|| \cdot ||A^{-1}||$ (en norma inducida), o ratio de eigenvalores máximo/mínimo.

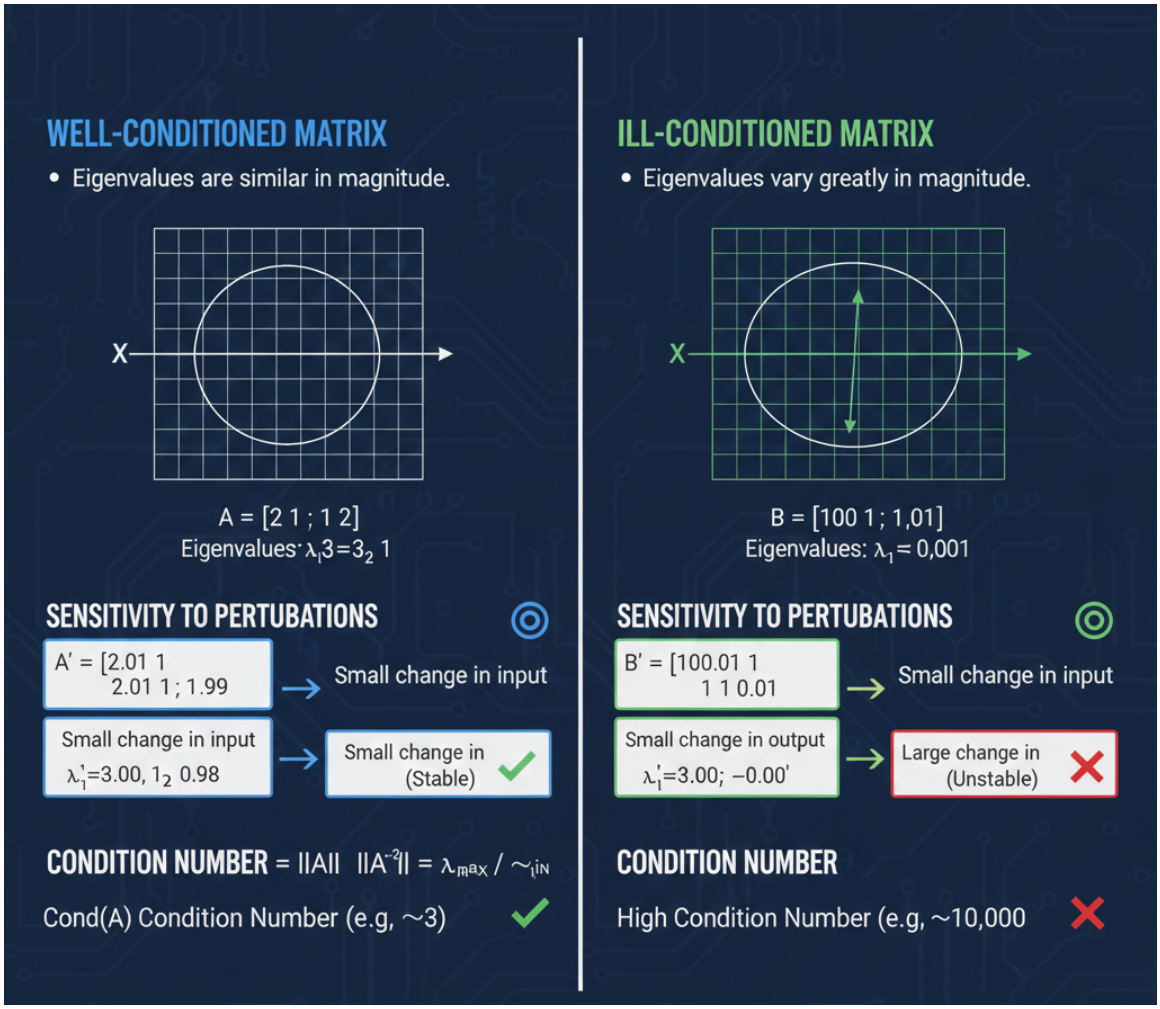


Figura 5: Ilustración de matriz bien condicionado vs mal condicionado (eigenvalores y sensibilidad).

Creación propia.

Estabilidad numérica

La estabilidad numérica se refiere a cómo un algoritmo maneja errores: un algoritmo es estable si los errores introducidos durante su ejecución no se amplifican desproporcionadamente (backward stable: el resultado es exacto para un problema ligeramente perturbado).

Algoritmo estable: forward error pequeño cuando backward error es pequeño.

Algoritmo inestable: amplifica errores incluso si el problema está bien condicionado.

En ciberseguridad:

Algoritmos estables en reducción modular evitan sesgos en timing/power.

En ML para detección: algoritmos estables (e.g., Adam optimizer bien implementado) evitan inestabilidad en gradientes.

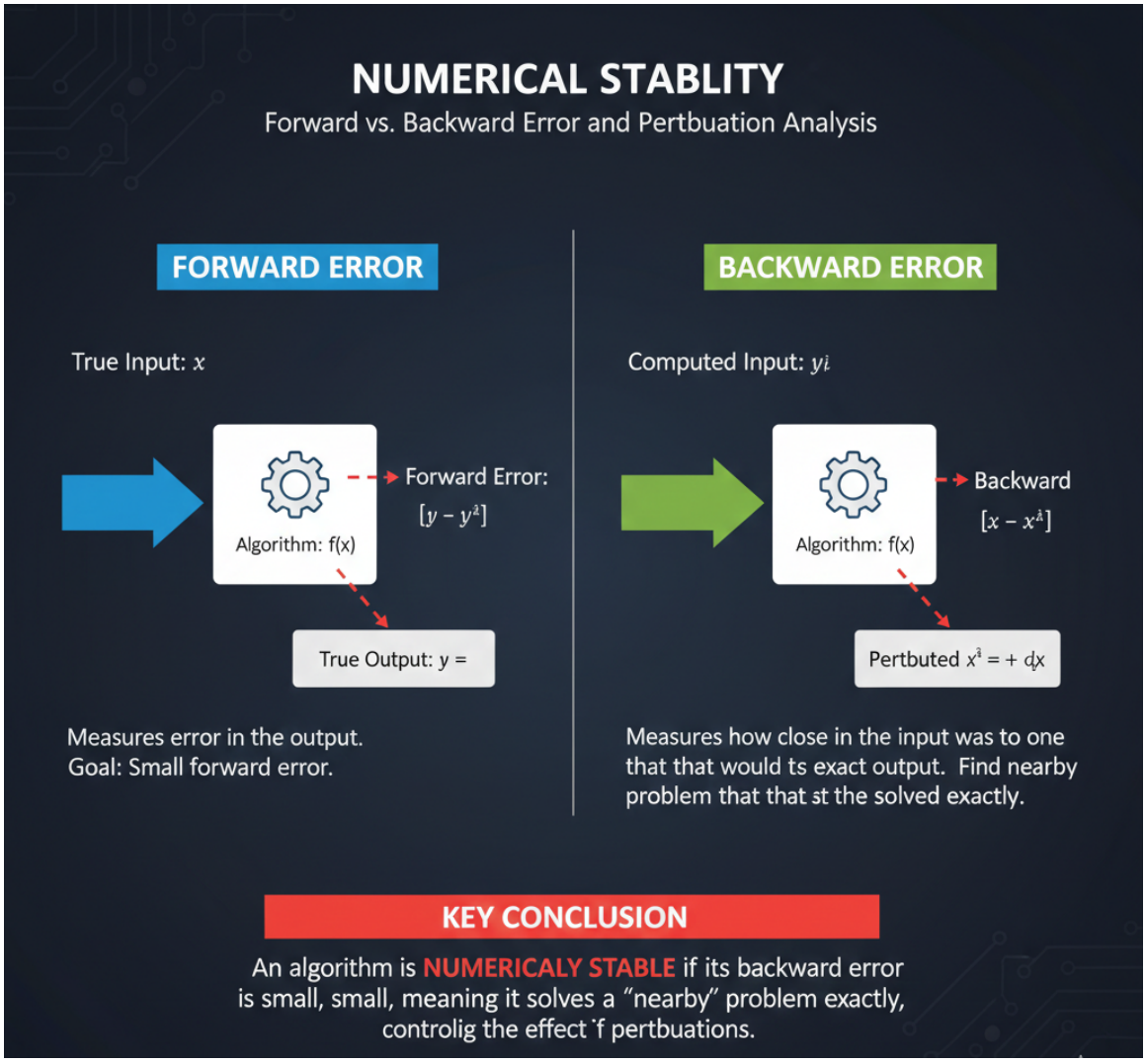


Figura 6: Diagrama de estabilidad numérica (forward vs backward error y perturbaciones).

Creación propia.

Enlace 2: https://nhigham.com/2020/08/04/what-is-numerical-stability/ Ver en Blog: What is Numerical Stability? (Nick Higham) Este artículo detalla estabilidad forward/backward con ejemplos relevantes para implementaciones criptográficas.

La estabilidad numérica de un algoritmo se refleja en cómo los errores hacia adelante y errores hacia atrás se comportan cuando se aplican pequeñas perturbaciones a los datos. Si un algoritmo es numéricamente estable, los errores hacia adelante deben crecer lentamente a medida que las perturbaciones aumentan, lo que indica que el algoritmo no amplifica excesivamente los errores. Sin embargo, si el algoritmo es inestable, una pequeña perturbación en la entrada puede provocar un gran aumento en el error hacia adelante, lo que sugiere que el algoritmo no es robusto y es susceptible a fallos en escenarios donde se realizan aproximaciones numéricas o se manejan datos imprecisos.

Solución de Ecuaciones No Lineales I

Hasta ahora hemos cubierto los fundamentos de los métodos numéricos, los errores y la precisión. Ahora entramos en uno de los pilares prácticos: la solución de ecuaciones no lineales de la forma $f(x) = 0$. Estas ecuaciones aparecen constantemente en ciberseguridad: desde encontrar raíces de polinomios en ataques a RSA (e.g., Coppersmith para claves con bits bajos conocidos), hasta resolver ecuaciones en criptoanálisis diferencial o recuperar parámetros en side-channel attacks mediante aproximaciones.

En esta primera parte nos enfocamos en métodos de acotamiento (bracketing): gráficos para visualización inicial, bisección (garantizado pero lento) y falsa posición (más rápido en muchos casos). Al final de la clase podrás formular una ecuación no lineal típica en ciberseguridad, usar métodos gráficos para localizar raíces aproximadas y aplicar bisección o falsa posición para refinarlas con control de error.

Formulación de ecuaciones no lineales

Una ecuación no lineal se expresa como $f(x) = 0$, donde $f(x)$ es una función no lineal (polinomios de grado >1, exponenciales, trigonométricas, modulares, etc.). En ciberseguridad, estas ecuaciones surgen en:

Criptoanálisis polinómico: encontrar raíces pequeñas de polinomios modulares en ataques Coppersmith (e.g., RSA con mensajes pequeños o claves con patrón).

Ataques a curvas elípticas o lattices: resolver ecuaciones para aproximar puntos cortos o logs discretos parciales.

Side-channel: ajustar parámetros en modelos de leakage (e.g., ecuaciones para correlación máxima en CPA).

ML para seguridad: encontrar umbrales óptimos en funciones de pérdida no lineales.

La formulación implica: definir $f(x)$ claramente, identificar intervalos donde f cambia de signo (por teorema del valor intermedio), y considerar múltiples raíces o singularidades. En práctica, usamos software como SageMath o Python (SciPy) para prototipar, pero entender los métodos manuales ayuda a depurar vulnerabilidades numéricas.

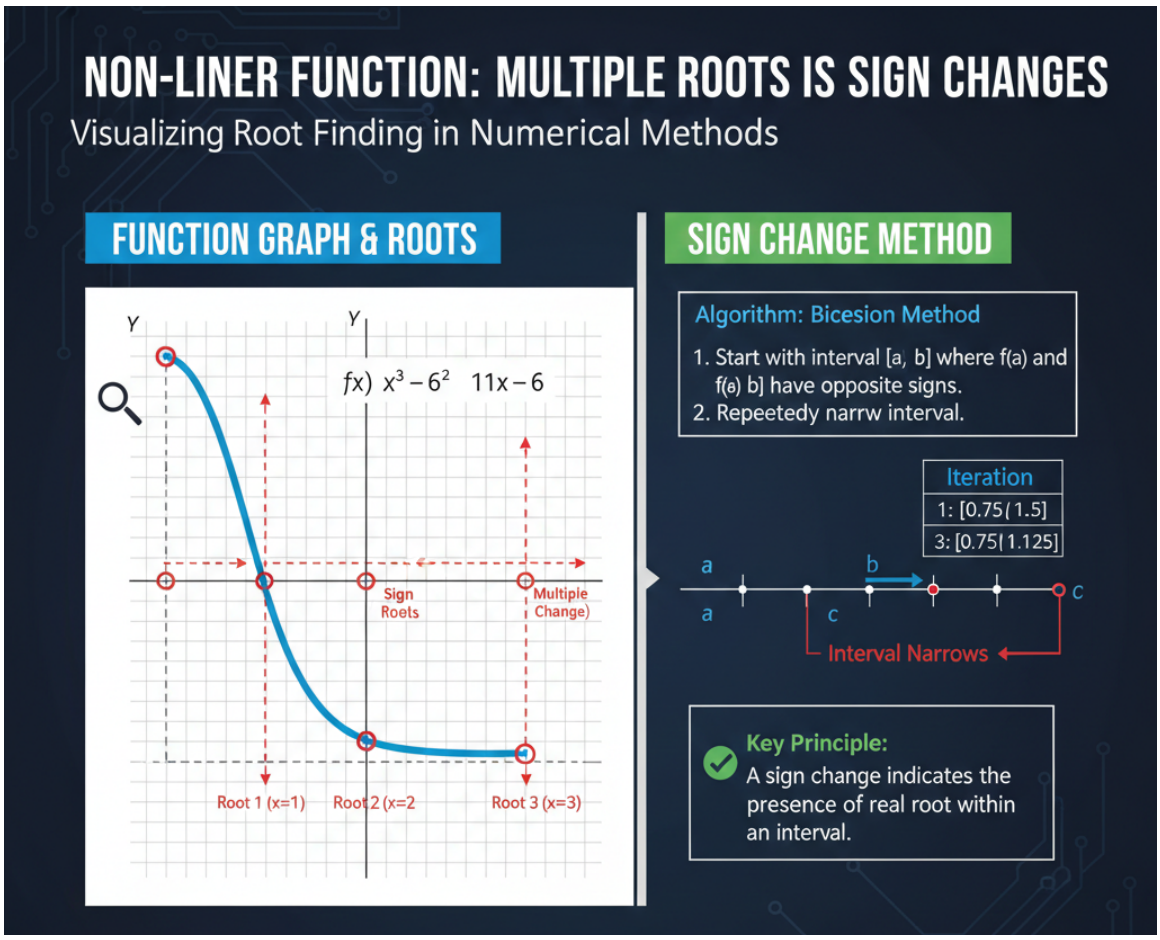


Figura 7: Representación gráfica de una función no lineal típica con raíces múltiples y cambio de signo.

Creación propia.

En cálculos numéricos o métodos iterativos para encontrar raíces de ecuaciones no lineales, como el método de bisección, las raíces múltiples pueden complicar la convergencia, ya que estos métodos suelen basarse en detectar cambios de signo entre dos valores. En casos de raíces de multiplicidad mayor, el cambio de signo puede no ser tan evidente, lo que podría hacer que el método de bisección, por ejemplo, no converja adecuadamente a una solución precisa. En aplicaciones como la generación de claves criptográficas o la validación de valores de autenticación en ciberseguridad, un comportamiento de raíz múltiple y cambios de signo podría dar lugar a errores de cálculo o problemas de convergencia en algoritmos numéricos.

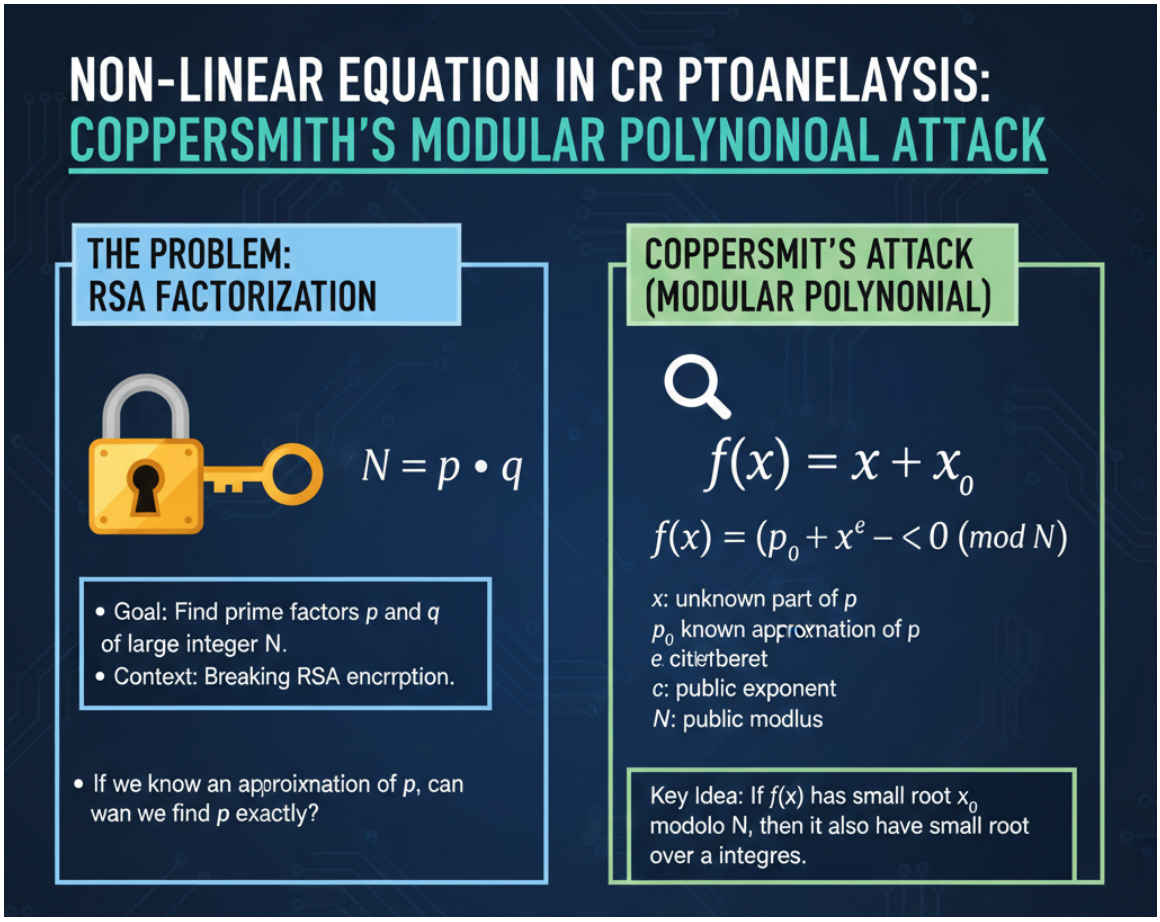


Figura 8: Ejemplo de formulación de ecuación no lineal en criptoanálisis: polinomio modular para ataque Coppersmith.

Creación propia.

El ataque de Coppersmith es un ataque en criptoanálisis que se utiliza para encontrar soluciones pequeñas a ecuaciones no lineales, específicamente en el contexto de problemas como la factorización de números grandes o la resolución de ecuaciones modulares en sistemas criptográficos. Uno de los enfoques clave del ataque es el uso de polinomios modulares para encontrar soluciones a ecuaciones no lineales que tienen raíces pequeñas en un contexto modular. Este enfoque es muy útil, por ejemplo, en sistemas criptográficos basados en la RSA o en la reducción de ecuaciones en espacios reducidos.

El ataque se puede formular utilizando una ecuación no lineal modular del tipo:

$$\mathbf{f(x) \equiv 0 (mod N)}$$

Donde $f(x)$es un polinomio de x, Nes un módulo grande (como un número compuesto que se descompone en sus factores primos), y el objetivo es encontrar valores pequeños de x(posibles

soluciones que corresponden a la clave privada en RSA o a otras variables críticas en sistemas criptográficos) que satisfacen esta ecuación.

Un ejemplo típico de un polinomio modular utilizado en el ataque de Coppersmith es el siguiente:

$$\mathbf{f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0}$$

Donde $x$es la variable desconocida, y $a_0, a_1, \ldots, a_n$ son los coeficientes del polinomio. El objetivo es resolver la ecuación:

$$\mathbf{f(x) \equiv 0 (mod N)}$$

Aquí, el módulo N es típicamente un número compuesto, como el producto de dos grandes números primos en el contexto de RSA. En un ataque de Coppersmith, la idea es encontrar valores de x que sean pequeños (en comparación con N) y que aún así satisfagan la ecuación.

El método de Coppersmith se basa en el Teorema de Coppersmith, que permite encontrar raíces de ecuaciones polinomiales en un cuerpo modular cuando estas raíces son relativamente pequeñas en comparación con el módulo N. La idea es utilizar una forma de reducción modular para aproximar la raíz de la ecuación polinómica, utilizando el hecho de que si $x_0$es una raíz, entonces:

$$\mathbf{f(x_0) \equiv 0 (mod N)}$$

El teorema de Coppersmith establece que para un polinomio $f(x)$, se puede encontrar una raíz $x_0$pequeña si se conocen las factores del módulo N, o si se puede construir un polinomio de reducción que lleve a una solución pequeña a la ecuación.

Consideremos el caso clásico de RSA, donde el módulo Nes el producto de dos números primos grandes py q, y la clave pública es el par (e, N). Supongamos que queremos encontrar la clave privada d, que satisface la ecuación modular:

$$\mathbf{e \cdot d \equiv 1 (mod \varphi(N))}$$

Donde $\varphi(N) = (p-1)(q-1)$es la función totiente de Euler. Para resolver esta ecuación en el ataque de Coppersmith, primero formulamos un polinomio $f(x)$de la forma:

$$\mathbf{f(x) = e \cdot x - 1}$$

El objetivo es encontrar un valor pequeño de x(en este caso, d) que haga que:

$$\mathbf{f(d) = e \cdot d - 1 \equiv 0 (mod \varphi(N))}$$

Este tipo de ecuación modular puede ser resuelto utilizando el algoritmo de Coppersmith si se conoce una aproximación adecuada para la raíz de la ecuación, que en el contexto de RSA es la clave privada d.

Métodos gráficos

Los métodos gráficos consisten en graficar $f(x)$ y observar dónde cruza el eje x (raíces). Son útiles para:

Localizar intervalos iniciales [a, b] donde $f(a)$ y $f(b)$ tienen signos opuestos (garantía de raíz por Bolzano).

Identificar múltiples raíces o comportamientos (oscilaciones, asíntotas).

Visualizar convergencia o problemas (e.g., raíces dobles donde el método puede estancarse).

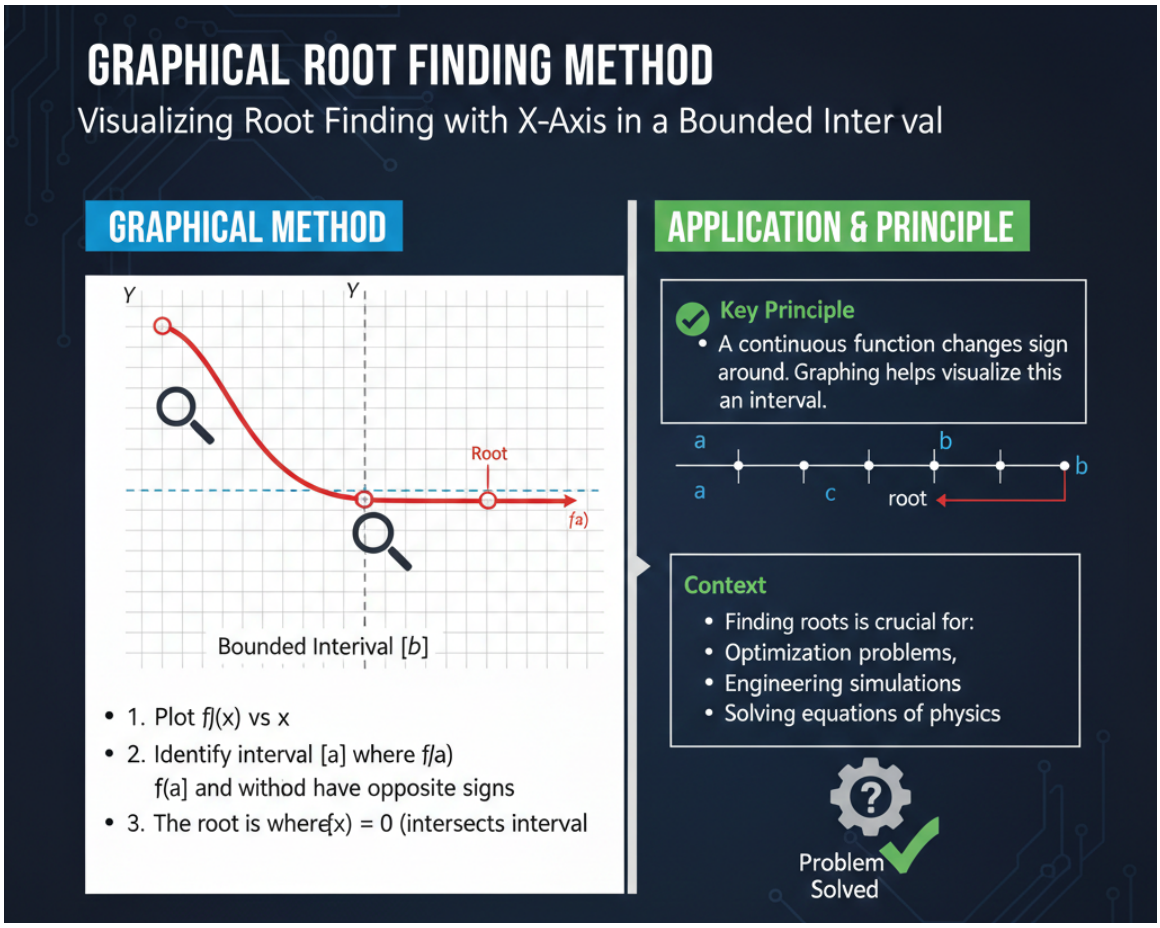


Figura 8: Método gráfico para encontrar raíces: intersección de la función con el eje x en un intervalo acotado.

Creación propia.

Método de bisección

El método de bisección (o dicotomía) es un bracketing garantizado: requiere f continua y $f(a)f(b) < 0$ en $[a, b]$.

Algoritmo:

Calcular punto medio $c = (a + b)/2$.

Evaluar $f(c)$.

Si $f(c) = 0$ → raíz encontrada.

Si $f(a)f(c) < 0$ → raíz en $[a, c]$ → $b = c$.

Si no → raíz en $[c, b]$ → $a = c$.

Repetir hasta $|b - a|$ < tolerancia ε o número máximo de iteraciones.

Convergencia lineal: error se reduce a la mitad cada iteración. Número de iteraciones aproximado: $n \geq \log^2((b - a)/\varepsilon)$.

Ventajas: siempre converge si hay raíz y intervalo correcto; robusto. Desventajas: lento (lineal), no aprovecha pendiente.

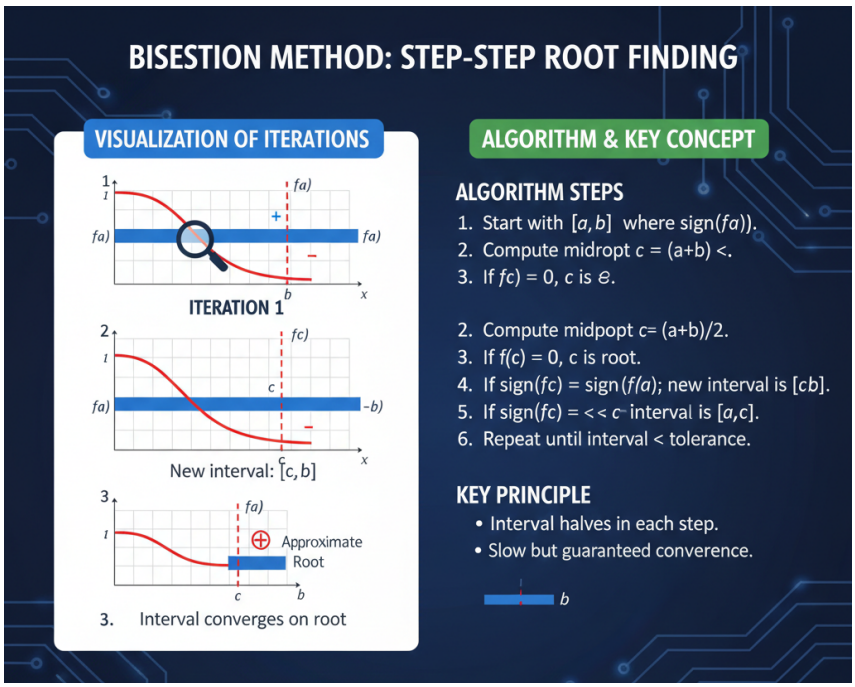


Figura 9: Ilustración paso a paso del método de bisección: halving del intervalo y convergencia al root.

Creación propia.

Método de la falsa posición (Regula Falsi)

El método de falsa posición combina bracketing con interpolación lineal (secante). Asume que la función es aproximadamente lineal entre a y b.

Algoritmo:

Calcular punto $c = b - f(b)(b-a)/\big(f(b) - f(a)\big)$.

Evaluar $f(c)$.

Si $f(c) = 0$ →raíz.

Si $f(a)f(c) < 0 \rightarrow b = c$.

Si no → $a = c$.

Repetir hasta convergencia.

Ventajas: más rápido que bisección cuando la función es casi lineal (convergencia superlineal en un lado).

Desventajas: puede estancarse si una raíz es muy cercana a un extremo (intervalo se reduce muy lento en un lado).

REFERENCIAS

• **Higham, N. J. (2002).** Accuracy and stability of numerical algorithms (2ª ed.). SIAM.
• **Overton, M. L. (2001).** Numerical computing with IEEE floating point arithmetic. SIAM.
• **Paar, C., & Pelzl, J. (2010).** Understanding cryptography: A textbook for students and practitioners. Springer.
• **Goodfellow, I., Bengio, Y., & Courville, A. (2016).** Deep learning. MIT Press. (Capítulo sobre aspectos numéricos).
• **Boneh, D., & Shoup, V. (2020).** A graduate course in applied cryptography. https://toc.cryptobook.us

PUCE Virtual

síguenos en:

www.pucevirtual.puce.edu.ec