BOOK

ANGELINE DEZA CUELA

DECEMBER 2025

INDICE

¿QUE ES LA PROGRAMACION NUMERICA?

La **programación numérica** es una rama de las matemáticas aplicadas y de la computación que se encarga de **resolver problemas matemáticos mediante algoritmos implementados en la computadora**, cuando no es posible obtener una solución exacta o esta resulta muy compleja.

Se basa en el uso de **métodos numéricos** para obtener **soluciones aproximadas** a problemas como:

- Resolución de ecuaciones algebraicas y no lineales
- Cálculo de derivadas e integrales
- Interpolación y aproximación de funciones
- Resolución de sistemas de ecuaciones
- Análisis de valores propios y procesos estocásticos

La programación numérica considera aspectos como el **error**, la **precisión**, la **convergencia** y la **estabilidad** de los métodos, con el fin de garantizar resultados confiables. Además, integra el conocimiento matemático con la **implementación en lenguajes de programación**, permitiendo automatizar cálculos complejos de forma eficiente.

FUNDAMENTOS DE LA PROGRAMACION NUNMERICA

Los **fundamentos de la programación numérica** son los principios básicos que permiten **resolver problemas matemáticos usando la computadora**, cuando no existe una solución exacta o esta es muy difícil de obtener.

Representación de números

Las computadoras tienen limitaciones para representar números. Usan aritmética de punto flotante con precisión finita, lo que genera errores de redondeo. Esto significa que operaciones como 0.1 + 0.2 pueden no dar exactamente 0.3 en la computadora.

ANALISIS DE ERRORES

Hay tres tipos principales de errores: errores de redondeo (por la precisión finita), errores de truncamiento (al aproximar procesos infinitos) y errores de datos (en las mediciones iniciales). Es crucial entender cómo se propagan estos errores a través de los cálculos.

ERRORES NUMERICOS

Todo método numérico implica errores. Los principales son:

- **Error absoluto**: diferencia entre el valor real y el aproximado

- **Error relativo**: error en relación con el valor real
- **Error de redondeo**: causado por la representación finita de los números
- **Error de truncamiento**: al aproximar un proceso infinito

ALGORITMOS NUMERICOS

Un algoritmo numérico es un conjunto de pasos lógicos para resolver un problema matemático. Debe ser:

- Preciso
- Finito
- Eficiente

ESTABILIDAD Y CONDICIONAMIENTO

Estos conceptos determinan si podemos confiar en el resultado:

- **Condicionamiento (del problema):** ¿Qué tan sensible es la solución a pequeños cambios en los datos de entrada? Si un cambio mínimo en la entrada altera drásticamente la salida, el problema está **mal condicionado**.
- **Estabilidad (del algoritmo):** Un algoritmo es estable si no magnifica los errores de redondeo durante el cálculo.

ALGEBRA LINEAL COMPUTACIONAL

- **Sistemas de ecuaciones:** Métodos para resolver $Ax = b$ (como la eliminación Gaussiana o la descomposición LU).
- **Vectores y Matrices ralas (Sparse):** Técnicas para manejar matrices que tienen muchos ceros, optimizando el uso de memoria.
- **Bibliotecas estándar:** Uso de herramientas de bajo nivel como **BLAS** (Basic Linear Algebra Subprograms) y **LAPACK**.

ALGORITMOS DE RESOLUCION COMUNES

- **Raíces de ecuaciones:** Métodos como Newton-Raphson o bisección.

- **Optimización:** Encontrar mínimos o máximos de una función (crucial para la Inteligencia Artificial).

- **Integración y Diferenciación numérica:** Aproximar integrales mediante la regla del trapecio o Simpson.

- **Ecuaciones Diferenciales:** Métodos como Runge-Kutta.

APROXIMACION Y DISCRETIZACION

Muchos problemas continuos se transforman en problemas discretos:

- Derivadas → diferencias finitas
- Integrales → sumas aproximadas

VARIABLES Y FUNCIONES DE LA PROGRAMACION NUMERICA

Las **variables** representan valores numéricos que pueden cambiar durante la ejecución de un algoritmo. En programación numérica se utilizan para:

- Almacenar datos de entrada
- Guardar resultados intermedios
- Controlar iteraciones y condiciones

TIPOS DE DATOS NUMERICOS

El aspecto más crítico es cuántos bits se asignan a cada número. Esto se basa en el estándar **IEEE 754**.

- **Enteros (`int`):** Se usan principalmente para índices de matrices, contadores de bucles o dimensiones de tensores. No tienen errores de redondeo.
- **Punto Flotante (`float`):** Son las variables estrella. Representan números reales.
    - **Precisión Simple (`float32`):** Usa 32 bits. Es más rápido y consume menos memoria. Muy común en **Deep Learning** (GPUs).
    - **Precisión Doble (`float64`):** Usa 64 bits. Es el estándar en **simulaciones científicas** e ingeniería donde el error debe ser mínimo.
- **Complejos (`complex`):** Muchas librerías (como NumPy) permiten variables con una parte real y una imaginaria, esenciales para el procesamiento de señales y física cuántica.

TIPOS DE VARIABLES MAS COMUNES

- **Variables independientes**: valores que se eligen o controlan (por ejemplo, xxx)

- **Variables dependientes**: valores que dependen de otras variables (por ejemplo, f(x)f(x)f(x))

- **Constantes**: valores fijos que no cambian durante el proceso

- **Variables de control**: regulan ciclos e iteraciones en los métodos numéricos

A. Tipos por Contenedor (Estructuras)

- **Escalares:** Un único valor numérico (ej. una constante física).
- **Arreglos (Arrays/Vectores):** Colecciones de datos del mismo tipo. Es la unidad básica de cálculo.
- **Matrices y Tensores:** Variables multidimensionales. Un tensor de orden 3 podría representar la presión en un espacio 3D a lo largo del tiempo.

B. Atributos de la Variable

- **Dtype (Tipo de dato):** Determina la precisión (ej. `float64` para alta precisión científica o `int16` para ahorrar memoria en señales de audio).
- **Shape (Forma):** Las dimensiones de la variable. En programación numérica, las operaciones solo funcionan si las "formas" de las variables son compatibles (Álgebra Lineal).

FUNCIONES EN LA PROGRAMACION NUMERICA

Las funciones aquí se dividen en dos grandes grupos: las que vienen de librerías optimizadas y las que nosotros construimos para resolver modelos.

A. Funciones Vectorizadas

Es el concepto más importante. En lugar de procesar un número a la vez con un bucle `for`, las funciones numéricas operan sobre **toda la variable (vector/matriz) a la vez**.

- *Ejemplo:* Si tienes un vector V con un millón de números, la función `sin(V)` calcula el seno de todos simultáneamente aprovechando instrucciones del procesador llamadas SIMD (Single Instruction, Multiple Data).

B. Funciones de Aproximación

Son algoritmos que sustituyen operaciones matemáticas complejas:

- **Interpolación:** Funciones que "adivinan" valores intermedios entre puntos conocidos.
- **Optimización:** Funciones que buscan el valor mínimo de otra función (ej. el algoritmo de *Descenso de Gradiente*).
- **Integración Numérica:** Funciones que calculan el área bajo una curva dividiéndola en fragmentos (Regla de Simpson o Cuadratura Gaussiana).

A. Funciones Elementales (Ufuncs)

Son las funciones matemáticas básicas aplicadas a arreglos:

- **Trigonométricas:** `sin(x)`, `cos(x)`, `arctan(x)`.
- **Logarítmicas y Exponenciales:** `log(x)`, `exp(x)`.
- **Redondeo:** `floor()`, `ceil()`, `round()`.

B. Funciones de Reducción (Agregación)

Estas funciones toman una variable compleja (como una matriz) y devuelven un valor más simple (un escalar o un vector de menor dimensión):

- `sum()`: Suma de elementos.
- `mean()` / `std()`: Promedio y desviación estándar.
- `min()` / `max()`: Valores extremos.

- `norm()`: Calcula la norma de un vector (su "longitud" matemática).

C. Solvers (Resolutores)

Son funciones complejas que implementan algoritmos iterativos para resolver problemas que no tienen una solución directa:

- **Raíces:** Encuentran donde $f(x) = 0$ (ej. método de Newton).
- **Optimización:** Encuentran el mínimo de una función (crucial para entrenar Redes Neuronales).
- **Integración:** Calculan el área bajo una curva usando métodos como la Regla de Simpson.

METODO DE NEWTON RAPHSON

La expresión matemática es:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

DONDE:

- x_n es la aproximación actual
- x_{n+1} es la nueva aproximación
- $f(x)$ es la función
- $f'(x)$ es la derivada de la función

Características principales

- Es un método **rápido**, con convergencia cuadrática cuando funciona correctamente.
- Requiere que la función sea **derivable**.
- Depende fuertemente de la **aproximación inicial**.
- Puede fallar si f′(xn)=0f′(x_n)=0f′(xn)=0 o si la función no es adecuada.

Ventajas

- Alta velocidad de convergencia
- Pocas iteraciones para obtener buena precisión
- Muy usado en problemas científicos e ingenieriles

Desventajas

- Necesita calcular la derivada
- Puede no converger si el punto inicial no es apropiado
- No siempre garantiza solución

El método Newton-Raphson es ampliamente utilizado en:

- Ingeniería y ciencias aplicadas
- Optimización numérica
- Resolución de ecuaciones no lineales
- Modelos físicos, biológicos y económicos

EJERCICIO METODO RAPHSON:

1. Determinar la raíz positiva de la función $f(x) = x^2 - 2$ mediante el algoritmo de Newton-Raphson, partiendo de una estimación inicial $x_0 = 2$.

1. Definición Matemática

El método se basa en la siguiente relación de recurrencia:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Dada la función:

$$f(x) = x^2 - 2$$

Su derivada con respecto a x es:

$$f'(x) = 2x$$

Sustituyendo en la fórmula general, obtenemos la función de iteración específica para este problema:

$$x_{n+1} = x_n - \frac{x_n^2 - 2}{2x_n}$$

2. Desarrollo de las Iteraciones

Partimos del valor inicial $x_0 = 2$.

Primera Iteración ($n = 0$):

$$x_1 = 2 - \frac{2^2 - 2}{2(2)} = 2 - \frac{2}{4} = 1.5$$

Segunda Iteración ($n = 1$):

$$x_2 = 1.5 - \frac{1.5^2 - 2}{2(1.5)} = 1.5 - \frac{2.25 - 2}{3} = 1.5 - \frac{0.25}{3}$$

$$x_2 \approx 1.41666667$$

Tercera Iteración ($n = 2$):

$$x_3 = 1.41666667 - \frac{(1.41666667)^2 - 2}{2(1.41666667)}$$

ANALIZAMOS EL ERROR:

Podemos observar que el valor real es: $\sqrt{2} \approx 1.41421356.$ El error absoluto en la tercera interacion es:

$$E_{abs} = |1.41421356 - 1.41421569| = 0.00000213$$

Esto demuestra la **convergencia cuadrática** del método, donde el número de decimales exactos aumenta rápidamente en cada paso.

IMPLEMENTAMOS EN PSEUDOCODIGO:

Algoritmo : Newton-Raphson
Entradas : $f, f', x_0, \text{tol}, \text{max\_iter}$
Para $i = 1$ hasta max_iter :
  1. $\Delta x = \frac{f(x_{i-1})}{f'(x_{i-1})}$
  2. $x_i = x_{i-1} - \Delta x$
  3. **Si** $|x_i - x_{i-1}| < \text{tol}$:
    Retornar x_i
FinPara

EJERCICIO EN CODIGO:

```
import numpy as np

def newton_raphson(f, df, x0, tol=1e-7, max_iter=100):
    """
    Implementación del método de Newton-Raphson.

    Parámetros:
    f       : Función a la que se le busca la raíz.
    df      : Derivada de la función f.
    x0      : Estimación inicial (float).
    tol     : Tolerancia para la convergencia (error aceptable).
    max_iter : Número máximo de iteraciones.
    """
    print(f"{'Iteración':<12} | {'x_n':<15} | {'f(x_n)':<15}")
    print("-" * 45)

    xn = x0
    for n in range(max_iter):
        fxn = f(xn)
        dfxn = df(xn)

        # Imprimir el estado actual de la iteración
        print(f"{n:<12} | {xn:<15.8f} | {fxn:<15.8e}")

        # 1. Verificar si la derivada es demasiado pequeña (evitar división por cero)
        if abs(dfxn) < 1e-12:
            print("Error: Derivada cercana a cero. El método no puede continuar.")
            return None

        # 2. Aplicar la fórmula: x_{n+1} = x_n - f(x_n) / f'(x_n)
```

```
        x_next = xn - fxn / dfxn

        # 3. Criterio de parada: Si la diferencia es menor a la tolerancia
        if abs(x_next - xn) < tol:
            print("-" * 45)
            print(f"Convergencia alcanzada en la iteración {n}.")
            return x_next

        xn = x_next

    print("Se alcanzó el máximo de iteraciones sin converger.")
    return xn

# --- Ejemplo de uso ---

# Definimos la función f(x) = x^2 - 2 y su derivada f'(x) = 2x
f_ejemplo = lambda x: x**2 - 2
df_ejemplo = lambda x: 2*x

# Valor inicial aproximado
valor_inicial = 2.0

# Llamada a la función
raiz = newton_raphson(f_ejemplo, df_ejemplo, valor_inicial)

if raiz:
    print(f"\nLa raíz aproximada es: {raiz:.10f}")
    print(f"Valor real de sqrt(2): {np.sqrt(2):.10f}")
```

TIPOS DE INDICE EN PROGRAMACION NUMERICA

1. INDICE H

¿Qué es el índice H?

- Es el t**amaño del paso de discretización** que se utiliza cuando se transforma un problema continuo en uno **discreto** para poder resolverlo numéricamente.
- Es una medida que evalúa la productividad e impacto científico de un investigador. Un científico tiene índice H = n si ha publicado n artículos que han recibido al menos n citas cada uno.

Si un intervalo continuo $[a, b]$ se divide en n subintervalos iguales, el índice h se define como:

$$h = \frac{b - a}{n}$$

Cada punto de la malla se expresa como:

$$x_i = a + i\,h \quad \text{con } i = 0, 1, 2, \ldots, n$$

- El índice **h** es el **paso de discretización**
- Controla **precisión, error y estabilidad**
- Aparece en casi todos los métodos numéricos
- Elegir un buen hhh es una decisión clave en programación numérica

2. INDICE I

¿Qué es el índice i?

- Es una **variable de conteo** que se utiliza para **identificar la posición de un elemento** dentro de un conjunto discreto de datos o para **recorrer secuencias de valores** generadas durante un método numérico. No representa una magnitud física como hhh, sino un **contador lógico**.

El índice i se usa para enumerar elementos de una secuencia:

$$x_i, \quad i = 0, 1, 2, \ldots, n$$

Cada valor de i indica la **posición** del punto o dato dentro de la malla o arreglo.

DERIVACION NUMERICA DEL INDICE I:

En diferencias finitas:

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_i)}{h}$$

El índice i indica el punto donde se calcula la derivada.

3. INDICE N

¿Qué es el índice N?

- Representa la **cantidad total de elementos**, **divisiones**, **puntos** o **iteraciones** que se utilizan en un método numérico. A diferencia de *h* (que es el tamaño del paso), *n* indica **cuántos pasos se realizan**.

Cuando se discretiza un intervalo continuo [a,b]:

n = numero de subintervalos

y se cumple que:

$$h = \frac{b-a}{n}$$

Por tanto:

- n controla la resolucion del método.
- h depende directamente de n.

4. INDICE K

¿Qué es el índice K?

- Se utiliza principalmente para **identificar el número de iteración** en los **métodos iterativos**. Sirve para distinguir cada aproximación sucesiva que se va obteniendo cuando la solución exacta no se calcula en un solo paso, sino mediante repeticiones.

En un método iterativo, la solución aproximada se expresa como:

$$x^{(k)}, \quad k = 0, 1, 2, \ldots$$

DONDE:

- X(k) es la aproximación en la iteración K.
- K indica cuentas veces se ha repetido el proceso

5. INDICE J

¿Qué es el índice J?

- Se utiliza como un **segundo índice de conteo**, especialmente cuando se trabaja con **estructuras bidimensionales** o con **dos recorridos simultáneos**. Es muy común en **matrices, mallas 2D y problemas con dos variables independientes**.

El índice J sirve para identificar una segunda posición dentro de un arreglo, tabla o malla:

$$a_{ij}$$

DONDE:

- i = fila
- j = columna

GRADIENTES EN LA PROGRAMACION NUMERICA:

¿Qué es la gradiente?

- Es un **operador vectorial** que se aplica a una función escalar de varias variables y mide **cómo cambia la función en cada dirección** del espacio.

Para una función de dos variables:

$$f(x, y)$$

su gradiente es:

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$$

Para tres variables:

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right)$$

- El **gradiente** de una función escalar de varias variables es un **vector** formado por sus **derivadas parciales**. Indica la **dirección de mayor crecimiento** de la función y la **magnitud del cambio** en ese punto.

Si la función es $f(x_1, x_2, \ldots, x_n)$, su gradiente se define como:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right)$$

¿Cuál es la interpretación del gradiente?

- **Dirección**: hacia donde la función aumenta más rápido
- **Sentido contrario**: indica la dirección de mayor descenso
- **Magnitud**: qué tan fuerte es el cambio

GRADIENTE Y DISCRETIZACION:

Como en muchos casos no se conoce la derivada exacta, el gradiente se **aproxima numéricamente** usando diferencias finitas y el índice h.

Aproximación numérica del gradiente

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x_i + h) - f(x_i)}{h}$$

o de forma centrada:

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x_i + h) - f(x_i - h)}{2h}$$

Aquí, h es el **paso de discretización**.

GRADIENTE NUMERICO:

En la programación numérica, casi nunca se tienen derivadas exactas, por eso el gradiente se aproxima.

Diferencias finitas hacia adelante

$$\left.\frac{\partial f}{\partial x}\right|_{i,j} \approx \frac{f_{i+1,j} - f_{i,j}}{h}$$

Diferencias finitas centradas (más precisas)

$$\left.\frac{\partial f}{\partial x}\right|_{i,j} \approx \frac{f_{i+1,j} - f_{i-1,j}}{2h}$$

Aquí:

- i, j: posición en la malla
- h: tamaño del paso