Desarrollo de Software Matemático

Simulador X

1. ABATRACT

Este informe presenta el desarrollo del Simulador X, una calculadora científica modular de escritorio diseñada para realizar operaciones matemáticas simbólicas y numéricas avanzadas. La aplicación incluye módulos de álgebra lineal, análisis matemático, estadística computacional, simulación y visualización gráfica. Está desarrollada en Python utilizando PyQt5, con una arquitectura escalable, una interfaz amigable y un sistema de temas personalizables. Simulador X se enfoca en ser una herramienta educativa interactiva que fomenta el aprendizaje práctico de las matemáticas aplicadas, integrando técnicas de programación estructurada, bibliotecas científicas y fundamentos matemáticos sólidos.

2. INTRODUCCION

En el contexto educativo y profesional actual, el uso de herramientas digitales para el aprendizaje y la resolución de problemas matemáticos se ha convertido en una necesidad esencial. La enseñanza de matemáticas avanzadas requiere enfoques didácticos modernos que combinen teoría y práctica, permitiendo a los estudiantes experimentar y validar sus conocimientos en tiempo real.

En este marco, se desarrolla Simulador X, una calculadora científica de escritorio que integra múltiples módulos para el tratamiento simbólico y numérico de operaciones comunes en ingeniería, física y matemáticas. A diferencia de plataformas tradicionales, Simulador X se caracteriza por su enfoque modular, su interfaz gráfica intuitiva, su portabilidad, y su capacidad de extenderse fácilmente a nuevas funcionalidades.

El sistema está construido utilizando el lenguaje Python y la biblioteca PyQt5 para la interfaz gráfica. Esto permite una integración eficiente con librerías como NumPy, SymPy, Matplotlib y SciPy, potenciando las capacidades de cálculo y visualización.

Simulador X constituye una solución innovadora y educativa que permite resolver problemas reales de forma automatizada, facilitando tanto el aprendizaje como la exploración matemática autónoma.

3. METODOLOGIA

El desarrollo de Simulador X se llevó a cabo utilizando el lenguaje de programación Python 3, debido a su versatilidad, legibilidad y amplio.

ecosistema de bibliotecas científicas. La construcción de la interfaz gráfica se realizó con PyQt5, una biblioteca robusta que permite diseñar interfaces modernas y adaptables.

4. MATRICES

A. Suma y Resta de Matrices

Sean dos matrices $A = (a_{ij})\ y\ B = (b_{ij})$ del mismo orden $m \times n$ la suma y resta se definen elemento a elemento:

$$C = A + B \rightarrow\ c_{ij} = a_{ij} + b_{ij}$$
$$D = A - B \rightarrow\ d_{ij} = a_{ij} - b_{ij}$$

B. Multiplicación de Matrices

Si A es una matriz de tamaño $m \times p$ y B es $p \times n$, su producto $E = AB$ es una matriz $m \times n$ donde:

$$e_{ik} = \sum_{j=1}^{p} a_{ij} b_{jk}$$

C. Transpuesta

La transpuesta de una matriz $A = (a_{ij})$ de tamaño $m\ x\ n$ se obtiene invirtiendo filas por columnas:

$$A^T = (a_{ji})$$

D. Determinante

Para una matriz cuadrada A, el determinante es un escalar que refleja propiedades como la invertibilidad:

- Si $\det(A) = 0$, entonces A es singular (no tiene inversa).
- Se usa en sistemas de ecuaciones (regla de Cramer), cambio de base, etc.

Para matrices 2×2:

$$\det(A) = a_{11}a_{22} - a_{12}a_{21}$$

E. Inversa de una Matriz

Una matriz cuadrada A tiene inversa A^{-1} si y solo si $\det(A) \neq 0$. Se cumple:

$$AA^{-1} = A^{-1}A = 1$$

Donde I es la matriz identidad

F. Resolución de Sistemas Lineales

Se resuelven sistemas de la forma:

$$AX = B$$

Utilizando métodos como:

- ❖ Eliminación de Gauss
- ❖ Inversa de matrices
- ❖ Función numpy.linalg.solve() para eficiencia computacional.

5. POLINOMIOS

A. REPRESENTACION DE UN POLINOMIO

Un polinomio en una variable x se representa como:

$$P(x) = \sum_{i=0}^{n} a_x x^i$$

Donde:

- a_i son los coeficientes (reales o enteros),
- n es el grado del polinomio.

B. Suma de Polinomios

Dados dos polinomios:

$$P(x) = \sum_{i=0}^{n} a_i x^i, \quad Q(x) = \sum_{i=0}^{m} b_i x^i$$

La suma se define como:

$$R(x) = P(x) + Q(x) = \sum_{i=0}^{\max(n,m)} (a_i + b_i) x^i$$

C. Multiplicación de Polinomios

El producto de P(x) y Q(x) se obtiene como:

$$S(x) = P(x) * Q(x) = \sum_{k=o}^{n+m} c_k x^k,$$

$$donde\ c_k = \sum_{i+j=k} a_i b_j$$

D. Derivación de Polinomios

La derivada simbólica de un polinomio es:

$$\frac{d}{dx} P(x) = \sum_{i=1}^{n} i a_i x^{i-1}$$

Este cálculo es automático usando sympy.diff().

E. Integración Indefinida

La integral indefinida de un polinomio es:

$$\int P(x)dx = \sum_{i=0}^{n} \frac{a_i}{i+1} x^{i+1} + c$$

Donde C es la constante de integración.

F. Evaluación Numérica

Usando la regla de Horner o directamente evaluando:

$$P(x_o) = a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_n x_0^n$$

Esto se implementa en Python con métodos directos (evalf) o numpy.polyval.

6. VECTORES

A. Suma y Resta de Vectores

Sean dos vectores $\vec{a} = (a_x, a_y, a_z) y\ \vec{b} = (b_x, b_y, b_z)$:

Suma:

$$\vec{a} + \vec{b} = (a_x + b_x, a_y + b_y, a_z + b_z)$$

Resta:

$$\vec{a} - \vec{b} = (a_x - b_x, a_y - b_y, a_z - b_z)$$

B. Producto Punto

El producto escalar (o punto) se define como:

$$\vec{a} * \vec{b} = (a_x b_x + a_y b_y + a_z b_z)$$

Este valor es un escalar que mide la proyección de un vector sobre otro.

C. Producto Cruzado

Para vectores en R^3el producto vectorial es:

$$\vec{a}\ x\ \vec{b} = (a_y b_z - a_z b_{y,} a_z b_x - a_x b_z, a_x b_y - a_z b_x)$$

Este resultado es un vector perpendicular a $\vec{a}\ y\ \vec{b}$

D. Magnitud de un Vector

La magnitud (o norma) de un vector es:

$$||\vec{a}| = \sqrt{a_x^2 + a_y^2 + a_z^2})$$

Esta operación es fundamental en normalización, distancias, y física.

7. GRÁFICAS 2D Y 3D

A. Gráfica 2D

Las funciones de una sola variable f(x) se representan en el plano cartesiano como:

$$y = f(x)$$

Estas gráficas permiten analizar comportamiento, continuidad, raíces, máximos/mínimos y puntos de inflexión.

- Se usa matplotlib.pyplot.plot() para generar curvas suaves.
- El rango de x es configurable desde la interfaz.

B. Gráfica 3D

Las funciones de dos variables $f(x, y)$ se representan en el espacio tridimensional como:

$$z = f(x, y)$$

Estas gráficas permiten observar superficies, cuencas, crestas y comportamientos multivariables.

- Se emplea matplotlib.pyplot.figure() con Axes3D.
- La malla se genera con numpy.meshgrid.

8. Derivación e integración simbólica

Dada una función $f(x)$ su derivada se define como:

$$\frac{d}{dx} f(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Para funciones polinómicas o expresiones algebraicas, se aplica la regla de potencias:

$$\frac{d}{dx}(ax^n) = anx^{n-1}$$

A. *Integración Indefinida*

La integral indefinida de una función es el proceso inverso de la derivación:

$$\int f(x)dx = F(x) + c$$

Para un polinomio:

$$P(x) = \sum_{i=0}^{n} a_i x^i \rightarrow \int P(x)dx = \sum_{i=0}^{n} \frac{a_i}{i+1} x^{i+1} + c$$

B. *Evaluación de Funciones*

Una función simbólica también puede evaluarse en un valor específico:

$$f(x_0) = reemplazar\ x\ por\ x_0$$

9. *ecuaciones diferenciales ordinarias (edo)*

A. *Planteamiento General*

Una ecuación diferencial de primer orden se expresa como:

$$\frac{dy}{dx} = f(x,y), y(x_0) = y_0$$

El objetivo es aproximar el valor de y para distintos valores de x, partiendo de una condición inicial.

B. *Método de Euler*

Es el método más sencillo, basado en la fórmula de avance:

$$y_i + 1 = y_i + hf(x_i y_i)$$

Donde h es el tamaño de paso.

C. *Método de Heun (Euler Mejorado)*

Corrige el método de Euler calculando un promedio:

$$y_i + 1 = y_i + \frac{h}{2}[f(x_i, y_i) + f(x_{i+1}, y_i + hf(x_i, y_i))]$$

D. Método de Runge-Kutta de 4to Orden (RK4)

Uno de los métodos más precisos, utiliza 4 evaluaciones por paso:

$$k_1 = hf(x_i, y_i)$$

$$k_1 = hf(x_i + \frac{h}{2}, y_i + \frac{k_1}{2})$$

$$k_1 = hf(x_i + \frac{h}{2}, y_i + \frac{k_2}{2})$$

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

E. *Método de Taylor de Segundo Orden*

Aproxima usando derivadas parciales:

$$y_{i+1} = y_i + hf(x_i, y_i) + \frac{h^2}{2}(\frac{\vartheta f}{\vartheta x} + \frac{\vartheta f}{\vartheta y} f)$$

F. Implementación

Simulador X permite:

- Ingresar $f(x,y)$, condición inicial (x_0, y_0), paso h y número de iteraciones.
- Generar una tabla paso a paso.
- Dibujar la curva aproximada $y(x)$ con matplotlib.

10. *Valores y vectores propios*

Cálculo de Valores Propios

Para encontrar los valores propios se resuelve:

$$\det(A - \lambda I) = 0$$

Esta es la ecuación característica, cuyo grado depende del tamaño de la matriz A.

11. *Generación de números aleatorios y distribución.*

A. *Congruencial lineal mixto:*

$$x_{n+1} = (aX_n + c)\ mod\ m$$

B. *Congruencial multiplicativo:*

$$x_{n+1} = (aX_n) mod\ m$$

C. *Producto medio:*

1. Tomar dos números X_n, Y_nde igual longitud
2. Multiplicar: $Z = X_n * Y_n$
3. Extraer los dígitos del medio como X_{n+1}

No tiene fórmula cerrada, pero es:

$$X_{n+1} = digitos\ centrales(X_n * Y_n)$$

D. *Cuadrado medio*

$$\mathrm{X_{x+1}} = \text{digitos centrales}(\mathrm{X_n^2})$$

E. *Xorshift*

$$\mathrm{X_n} = \mathrm{X_{n-1}} \oplus (\mathrm{X_{n-1}} \ggg \mathrm{a})$$
$$\mathrm{X_n} = \mathrm{X_n} \oplus (\mathrm{X_n} \lll \mathrm{b})$$
$$\mathrm{X_n} = \mathrm{X_n} \oplus (\mathrm{X_n} \ggg \mathrm{c})$$

F. *PCG (Permuted Congruential Generator)*

$$\mathrm{X_{n+1}} = \mathrm{aX_n} + \mathrm{c} \bmod 2^{\mathrm{m}}$$

Salida: $\text{permute}(\mathrm{X_n})$

Añade una permutación no lineal a un generador congruencial.

G. *WELL (Well Equidistributed Long-period Linear)*

$\mathrm{X_n}$ = combinacion lineal de estados anteriores usando XOR y shift

H. Tausworthe (LFSR binario)

$$X_n = (X_{n-r} + X_{n-s}) \bmod 2$$

I. Mersenne Twister

- Algoritmo complejo que genera bloques de 32 bits a partir de una matriz de estado de gran longitud (624).
- Fórmulas internas con multiplicaciones, bitmasks y desplazamientos.

12. Montecarlo

A. Principio del método

Para una función continua f(x) definida en un intervalo [a, b], se puede aproximar la integral:

$$\int_a^b f(x)dx \approx (b-a) * \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

Donde:

- x_i *son números aleatorios uniformemente distribuidos en* $[a, b]$
- *N es el número de muestras.*

13. Modulo de preddicion Torricelli

- **Justificación**

La predicción del comportamiento de sistemas físicos es una necesidad constante en el ámbito de la ingeniería, especialmente en procesos relacionados con la hidrodinámica, el diseño de tanques, la gestión de recursos hídricos, y la automatización de sistemas de vaciado.

A pesar de que existen herramientas industriales avanzadas para modelar estos fenómenos, muchas de ellas son complejas, cerradas o inaccesibles para fines educativos. En este contexto, se planteó la necesidad de implementar un módulo de simulación simple, visual y confiable que permita estimar el vaciado de un tanque en función de parámetros físicos reales, utilizando un modelo matemático conocido, validado y científicamente actual.

La elección del modelo de Torricelli revisado responde a la oportunidad de integrar teoría clásica con ajustes modernos, permitiendo al usuario observar en tiempo real cómo pequeños factores como la fricción o el perfil del flujo afectan el resultado final. Esto no solo aporta valor académico, sino que también entrena la intuición física y el criterio ingenieril del usuario.

El módulo de predicción incluido en **Simulador X** permite resolver un problema clásico de la hidráulica mediante un enfoque teórico-práctico validado por literatura científica reciente. El modelo base fue extraído del artículo "Torricelli's Law Revisited" (D'Alessio, 2021), el cual introduce una mejora sobre la ley original para aproximar mejor los resultados experimentales, sin necesidad de simulaciones complejas.

Además:

- El simulador permite ingresar variables físicas directamente, haciendo el modelo personalizable.
- Se presentan ambas versiones del modelo: el ideal y el ajustado, fomentando el análisis comparativo.
- El diseño visual incluye gráfica dinámica, tabla de datos, interpretación numérica y acceso al artículo original, lo cual convierte el módulo en una herramienta educativa y de predicción aplicada.

Por lo tanto, este módulo no solo fortalece el proyecto desde el punto de vista funcional, sino que también constituye un ejemplo claro de cómo integrar ciencia, simulación y experiencia de usuario en un entorno académico digital.

- ***Contexto***

Una planta de tratamiento de agua necesita diseñar un sistema de emergencia para vaciar un tanque cilíndrico de almacenamiento de 16 litros en el menor tiempo posible. Para ello, se debe estimar cuánto tardará en vaciarse dicho tanque mediante un orificio en la base, considerando tanto condiciones ideales como factores reales de fricción y turbulencia.

- ***Problema***

Determinar el tiempo estimado de vaciado de un tanque vertical de área $A = 201\ cm^2$ y altura inicial $H = 16cm$, mediante un orificio circular de área $a = 0.1963cm^2$. Comparar los resultados entre el modelo ideal y el modelo corregido (con coeficiente CD ≈ 0.8
), y evaluar el error relativo entre ambos.

- ***Objetivo práctico***

Con base en los resultados obtenidos, decidir qué modelo es más adecuado para el diseño del sistema y justificar la necesidad de usar modelos mejorados en ingeniería hidráulica.

- ***Simulación desarrollada***

El presente análisis se realizó mediante el módulo de predicción del Simulador X, el cual permite ingresar los parámetros físicos del sistema, simular el vaciado, visualizar el comportamiento de la altura del fluido con respecto al tiempo y comparar ambos modelos de predicción.

- ***Parametros usados en la simulación***
  - Altura inicial del fluido: $H = 16cm$
  - Área del orificio: $a = 0.1963cm^2$
  - Área del tanque: $A = 201\ cm^2$
  - Gravedad: $g = 981cm/s^2$
  - Coeficiente de descarga corregido: $CD = \frac{8}{10} = 0.8$

❖ ***Formulas aplicadas***

- ***Modelo ideal:***

$$T_{ideal} = \frac{A}{a}\sqrt{\frac{2H}{g}}$$

- ***Modelo corregido:***

$$T_{corregido} = \frac{A}{a} * \frac{1}{CD} * \sqrt{\frac{2H}{g}}$$

- ***Cálculos paso a paso:***

$$T_{ideal} = \frac{201}{0.1963} * \sqrt{\frac{2 * 16}{981}}$$
$$= 1024.1 * \sqrt{0.0326}$$
$$= 1024.1 * 0.1806$$
$$= 185.0\ segundos$$

- ***Modelo corregido (con CD=0.8):***

$$T_{corregido} = \frac{T_{ideal}}{0.8} = \frac{185.0}{0.8}$$
$$= 231.3\ segundos$$

❖ ***Análisis de error***

- ***Error absoluto:***

$$E_{abs} = T_{corregido} - T_{ideal} = 231.3 - 185.0$$
$$= 56.3\ segundos$$

- ***Error relativo***

$$E_{rel} = \frac{E_{abs}}{T_{ideal}} * 100\% \approx \frac{46.3}{185.0} \approx 25.0\%$$

❖ ***Interpretación***

La diferencia entre el modelo ideal y el corregido representa un error relativo del 25%, lo cual demuestra que ignorar la fricción y el perfil no uniforme del flujo lleva a una subestimación considerable del tiempo real de vaciado. Esto valida la necesidad de emplear modelos físicos más completos en aplicaciones reales.

14. conclusiones

El desarrollo del Simulador X ha permitido integrar múltiples áreas de las matemáticas aplicadas en una plataforma digital de fácil uso, visualmente atractiva y académicamente sólida. Cada módulo ha sido diseñado con enfoque modular, permitiendo una estructura escalable, mantenible y adaptada a necesidades reales de aprendizaje.

En particular, el módulo de predicción basado en el modelo de Torricelli representa una aplicación clara de cómo un principio físico clásico puede ser refinado, visualizado e implementado computacionalmente. La comparación entre el modelo ideal y el modelo corregido ha demostrado la importancia de incorporar factores reales, como la fricción y el perfil de velocidad, para obtener resultados más cercanos a la realidad.

A lo largo del proyecto, se han utilizado algoritmos numéricos, métodos simbólicos y técnicas de visualización que consolidan las habilidades requeridas para el desarrollo de software científico. Además, se ha fomentado el análisis crítico al incluir validaciones, análisis de errores y comparación entre modelos.

Como trabajo futuro, se plantea la posibilidad de:

- Incluir mediciones experimentales reales para ajustar el parámetro k de forma dinámica.
- Ampliar el modelo a sistemas multietapa o con entrada de flujo.
- Conectar con sensores o bases de datos externas para hacer predicciones automatizadas.

Este proyecto no solo ha reforzado los conocimientos en matemática y programación, sino que también ha demostrado el valor de unir teoría, simulación y diseño de software en un solo entorno interactivo.

References

1. *Strang, G. (2016). Introduction to Linear Algebra (5th ed.). Wellesley-Cambridge Press.*
2. *Lay, D. C. (2011). Linear Algebra and Its Applications (4th ed.). Pearson.*
3. *Horn, R. A., & Johnson, C. R. (2012). Matrix Analysis (2nd ed.). Cambridge University Press.*
4. *Anton, H., & Rorres, C. (2014). Elementary Linear Algebra (11th ed.). Wiley.*
5. *Noble, B., & Daniel, J. W. (1988). Applied Linear Algebra (3rd ed.). Prentice Hall.*
6. *Burden, R. L., & Faires, J. D. (2011). Numerical Analysis (9th ed.). Brooks/Cole.*
7. *Stoer, J., & Bulirsch, R. (2002). Introduction to Numerical Analysis (3rd ed.). Springer.*
8. *Gear, C. W. (1971). Numerical Initial Value Problems in Ordinary Differential Equations. Prentice-Hall.*
9. *Butcher, J. C. (2016). Numerical Methods for Ordinary Differential Equations (3rd ed.). Wiley.*
10. *Kreyszig, E. (2011). Advanced Engineering Mathematics (10th ed.). Wiley.*
11. *McKinney, W. (2017). Python for Data Analysis (2nd ed.). O'Reilly Media.*
12. *VanderPlas, J. (2016). Python Data Science Handbook. O'Reilly Media.*
13. *Oliphant, T. E. (2007). Guide to NumPy. Trelgol Publishing.*
14. *SymPy Development Team. (2023). SymPy Documentation. https://docs.sympy.org*
15. *NumPy Developers. (2023). NumPy Documentation. https://numpy.org/doc/*
16. *Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering, 9(3), 90–95.*

17. *Matplotlib Team. (2023). Matplotlib Documentation. https://matplotlib.org/*
18. *Reitz, D. (2019). Effective Python: 90 Specific Ways to Write Better Python (2nd ed.). Addison-Wesley.*
19. *D'Alessio, S. (2021). Torricelli's Law Revisited. European Journal of Physics, 42(6), 065808.*
20. *Kundu, P., Cohen, I., & Dowling, D. (2015). Fluid Mechanics (6th ed.). Academic Press.*
21. *Halliday, D., Resnick, R., & Walker, J. (2013). Fundamentals of Physics (10th ed.). Wiley.*
22. *Young, H. D., & Freedman, R. A. (2016). University Physics with Modern Physics (14th ed.). Pearson.*
23. *Knuth, D. E. (1997). The Art of Computer Programming, Vol. 2: Seminumerical Algorithms (3rd ed.). Addison-Wesley.*
24. *Von Neumann, J. (1951). Various techniques used in connection with random digits. Applied Math Series, 12.*
25. *Marsaglia, G. (2003). Xorshift RNGs. Journal of Statistical Software, 8(14).*
26. *O'Neill, M. (2014). PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation.*
27. *Matsumoto, M., & Nishimura, T. (1998). Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudo-Random Number Generator. ACM TOMACS.*
28. *Panneton, F., L'Ecuyer, P., & Matsumoto, M. (2006). Improved long-period generators based on linear recurrences modulo 2. ACM Trans. Math. Softw.*
29. *Ross, S. M. (2014). Simulation (5th ed.). Academic Press.*
30. *Papoulis, A., & Pillai, S. U. (2002). Probability, Random Variables and Stochastic Processes (4th ed.). McGraw-Hill.*