**Tarea 3: Métodos Numéricos en Python**  
**Alumno:** Andrés Padrón Quintana  
**Curso:** Data Science and Machine Learning Applied to Financial Markets - Módulo III  
**Fecha:** 13 de octubre de 2025

**BLOQUE 1 — BÁSICO**

1. **Integración** de e^(−x^2) en [0, 1]  
   Se calcula I con cuadratura adaptativa (scipy.integrate.quad) y se contrasta con la regla del trapecio compuesta (numpy.trapz).

* I\_quad ≈ 0.74682413 I\_trapz ≈ 0.74682407 (m=1000)
* |I\_quad − I\_trapz| ≈ 6.13×10^−8 quad coincide con la referencia dentro del error numérico; trapecios converge al refinar mallas.

1. **Interpolación cúbica** (interp1d) y s(2.5)  
   **Datos:** {(0,1), (1,2.7), (2,5.8), (3,6.6), (4,7.5)}.  
   **Resultado:** s(2.5) ≈ 6.4328125, coherente con 5.8 < s(2.5) < 6.6.  
   La interpolación cúbica es suave y pasa por todos los puntos.
2. **Ajuste polinómico de grado 2 (usando puntos del problema 2)**  
   **Modelo:** p2(x) = a x^2 + b x + c.  
   **Coeficientes (LS):** (a, b, c) = (−0.278571, 2.804286, 0.782857).  
   **Evaluación:** p2(2.5) ≈ 6.0525.  
   El ajuste no interpola todos los puntos (minimiza error global), por eso p2(2.5) ≠ s(2.5).
3. **Interpolación polinómica completa (grado 4)**  
   Interpolador p4 de grado 4 con coeficientes aproximados (a, b, c, d, e) = (0.254167, −2.141667, 5.345833, −1.758333, 1.000000).  
   Comparación en x=2.5:

* p4(2.5) ≈ 6.4804688
* s(2.5) ≈ 6.4328125
* |p4(2.5) − s(2.5)| ≈ 4.77×10^−2  
  p4 interpola exactamente los cinco puntos, pero puede oscilar más que la cúbica (fenómeno de Runge).

1. **Raíces** de x^3 − 6x^2 + 11x − 6  
   Usando numpy.roots y Newton–Raphson (scipy.optimize.newton).  
   **Resultados:** raíces reales = {1, 2, 3}.  
   Newton converge rápido cuando la semilla cae en la cuenca adecuada

**BLOQUE 2 — INTERMEDIO**

1) **Área** de sin(x) en [0, π]  
Cálculamos la integral I = ∫\_0^π sin(x) dx.  
I\_quad ≈ 2.000000000000; valor exacto = 2; error ≈ 0.  
Coincidencia a precisión de máquina.

**2) Interpolación de cos(x) + ruido** (lineal vs. cúbica)  
y\_i = cos(x\_i) + ε\_i, con ε\_i ~ N(0, 0.1^2), n=20, x\_i en [0, 10].  
La interpolación lineal sigue más el ruido local (menos suave); la cúbica suaviza entre nodos y respeta la forma de cos(x), aunque puede ondular ligeramente.

**3) Ajuste polinómico de grado 3**  
Modelo: p3(x) = a x^3 + b x^2 + c x + d.  
Coeficientes: (a, b, c, d) = (−0.025823, 0.380188, −1.498508, 1.267758).  
Evaluación: p3(5) ≈ 0.0521.  
Comentario: el cúbico capta la tendencia global de cos(x) pese al ruido; no interpola cada punto y puede desviarse en bordes. (Fig. B2-3)

**4) Raíz de cos(x) − x** (Newton propio vs. scipy.newton)  
Ecuación: h(x) = cos(x) − x = 0.  
Resultados (x0 = 1):

* Newton propio: x\* ≈ 0.739085133215 (5 iteraciones).
* scipy.newton: x\* ≈ 0.739085133215.  
  Comentario: h′(x) = −sin(x) − 1 ≠ 0 cerca de la raíz ⇒ convergencia rápida y estable. (Fig. B2-4)

**5) Mínimo de x^4 − 3x^3 + 2**  
Óptimo (minimize\_scalar): x\* ≈ 2.25; φ(x\*) ≈ −6.542969.  
El punto marcado coincide con el valle principal en [−1, 3].

**6) Descenso de gradiente en f(x) = (x − 3)^2 + 4**  
**Gradiente:** ∇f(x) = 2(x − 3).  
x\_{k+1} = x\_k − α·2(x\_k − 3) = (1 − 2α) x\_k + 6α. Con x0 = 0 y 20 iteraciones:

* α = 0.1 → x\_20 ≈ 2.965412; f(x\_20) ≈ 4.001196 (convergencia estable y lenta).
* α = 0.5 → x\_20 = 3.000000; f(x\_20) = 4.000000 (llega en 1 paso)
* α = 0.9 → x\_20 ≈ 2.965412; f(x\_20) ≈ 4.001196 (convergencia oscilatoria)  
  Mínimo teórico: x\* = 3; f(x\*) = 4.

**Conclusiones**

1. Cuadraturas adaptativas (quad) son muy precisas; reglas compuestas convergen al refinar.
2. En interpolación, la cúbica es más suave; el polinomio global de alto grado puede oscilar.
3. Los ajustes polinómicos modelan la tendencia y no interpolan cada punto.
4. Newton–Raphson es rápido si la derivada no se anula cerca de la raíz.
5. minimize\_scalar es fiable para mínimos univariados.
6. En descenso de gradiente, la tasa α gobierna estabilidad/velocidad; para cuadráticas, α = 0.5 es ideal.