Métodos Computacionales

Tarea 2 — 2018–20

Los archivos:
`ApellidoNombre_Hermite.py`,
`ApellidoNombre_Interpola.py`
`ApellidoNombre_NR.py`,
`ApellidoNombre_SistEcua.py` y
`ApellidoNombre_MinCua.py`
deben estar comprimidos en `ApellidoNombre_hw2.zip` y deben descomprimirse en un directorio `ApellidoNombre_hw2`. Este directorio comprimido debe contener únicamente los archivos mencionados anteriormente y debe subirlo a SICUA antes de las 8:00 pm del lunes 24 de septiembre de 2018 (10 puntos). Recuerde que es un trabajo completamente individual (no debe usar códigos que usted no haya escrito en su totalidad ni debe recibir ayuda de nadie cuando esté escribiendo su código). Recuerde también que los códigos deben correr sin botar errores en los computadores de compufísica (Y110B). Si su código bota errores que usted no sabe cómo corregir, comente la sección de código problematica y haga un print con un mensaje que indique que hay código que usted considera que merece ser revisado en detalle durante la correción. Recuerde que es buena práctica comentar el código y elegir buenos nombres para sus variables. Eso será tenido en cuenta durante el proceso de corrección de esta tarea. No se aceptarán trabajos entregados tarde ni por otro medio distinto a SICUA.

1. (10 points) **Recurrencia y derivación**
   Este ejercicio es una generalización del que hicieron en el laboratorio de métodos. Recuerden que es un trabajo individual. Los polinomios de Hermite se pueden definir de la siguiente manera

$$H_n(x) = (-1^n)e^{x^2}\frac{d^n}{dx^n}e^{-x^2} \tag{1}$$

   Su script, llamado `ApellidoNombre_Hermite.py` debe, usando derivadas numéricas y recurrencia, calcular los polinomios de Hermite hasta grado $n = N$ (use N=7, pero tenga en cuenta que su código debe funcionar para un N cualquiera). Haga una gráfica de los N=7 primeros polinomios de Hermite. Guarde esa gráfica sin mostrarla en `ApellidoNombre_Hermite.pdf`. Repita el procedimiento anterior para 10 veces más puntos de su arreglo de x. Guarde su nueva gráfica sin mostrarla en `ApellidoNombre_Hermite_10.pdf`.

2. (20 points) **Derivación e interpolación**
   La interpolación permite completar los puntos de una función en ordenadas en las que no hay datos. Un ejemplo de esto se da en el diseño de carreteras. Se puede realizar interpolación cuadrática que asegura que la primera derivada es continua, pero el problema es que al manejar eso no garantiza que la segunda derivada sea continua, por lo que es necesario mover el timón del carro de manera brusca para manejar por estos tipos de carretera.

   a. Aprenda los conceptos básicos de interpolación (puede ser el libro de Landau o alguna otra fuente) y en particular entienda cómo funcionan los "cubic splines". En base a eso escriba un script llamado `"ApellidoNombre_Interpola.py"`donde implemente una función que, usando la función interp1d de scipy, calcule las splines lineal, cuadrática y cúbica (para esta última

tanto la primera como la segunda derivada son continuas en una dimensión de tal manera que los conductores en las carreteras tengan una mejor experiencia en la carretera). Esta función debe recibir los datos de la curva original (que se encuentran en el archivo `chicamocha.txt`) y un array de los puntos para los cuales se quieren obtener los valores interpolados de nuestra curva. Su código debe graficar las tres interpolaciones de los datos y los datos originales. Guarde dicha gráfica sin mostrarla en `ApellidoNombre_Interpola.pdf`.

b. Usando arrays y slicing, escriba una función que calcule la derivada central y/o Forward/Backward para los arrays del punto anterior (datos originales y datos interpolados con splines lineal, cuadrática y cúbica). asegurándose que los arreglos de derivadas tengan igual número de datos que los arreglos que está derivando. Grafique dichas derivadas y guarde dicha gráfica sin mostrarla en `ApellidoNombre_Deriv.pdf`.

3. (20 points) **Newton-Raphson**
La idea de este ejercicio es que exploren la convergencia del método de Newton-Raphson. Para esto debe escribir un script de python llamado `ApellidoNombre_NR.py` que, usando el método de Newton-Raphson, encuentre los ceros del siguiente polinomio:
$f(x) = x^5$–$2x^4$–$10x^3 + 20x^2 + 9x$–18.
Para esto haga un script que:

   - Haga una gráfica del polinomio en el intervalo [-4:4]. Guarde dicha gráfica sin mostrarla en `ApellidoNombre_NRpoli.pdf`.
   - Imprima el valor de una raiz $x0_r$ del polinomio encontrada si usa como x_{guess} inicial el valor -3. Imprima el valor de $x0_r$ encontrado y de $f(x0_r)$.
   - Repita lo anterior para un $x_{guess} = -1$. Imprima el valor de la nueva raiz $x1_r$ encontrada y de $f(x1_r)$.
   - Repita lo anterior para 1000 valores de x_{guess} generados aleatoriamente en el intervalo [-4:4]. Cuente cuantas iteraciones necesita su código para encontrar una raíz x_r del polinomio (tal que $f(x_r)$ sea menor a 10^{-10}) para cada x_{guess}. Haga una gráfica (use un scatter) del número de iteraciones en función del x_{guess} inicial. Guarde dicha gráfica sin mostrarla en `ApellidoNombre_NR_itera.pdf`.
   - Haga una gráfica (use un scatter) de los valores de x_r econtrados para los distintos x_{guess} en función de x_{guess}. Guarde dicha gráfica sin mostrarla en `ApellidoNombre_NRxguess.pdf`.
   - Imprima un mensaje en donde explique por qué cree que para ciertos valores de x_{guess} el número de iteraciones necesarios para encontrar la raiz es mayor. Además haga un análisis y describa qué pasa con los valores de x_r encontrados en esos puntos "problema" comprados con los encontrados para otros valores de x_{guess}.

4. (20 points) **Sistemas de ecuaciones lineales**
Use su propia implementación de eliminación gaussiana para solucionar un sistema de ecuaciones general $\mathbf{A}\vec{x} = \vec{b}$ (creado para fines de este ejercicio usando números aleatorios). Preste particular atención al hecho de que puede haber ceros en las diagonales. Usted debe modificar el script `CAMBIARNombre_SistEcua.py`. Su código debe:

   - Imprimir la matriz de coeficientes original $\mathbf{A}$ y el vector $\vec{b}$

- Obtener la matriz con unos en la diagonal y ceros en la parte inferior izquierda e imprimirla. Imprimir también el vector $\vec{b}'$ resultante.
- Imprimir cuál es el vector solución del sistema y **comparar** el resultado obtenido usando su propia implementación con el resultado obtenido con las funciones de linalg de numpy.

5. (20 points) **Mínimos cuadrados**
   Antes que nada lo más fácil es recordar de qué se trata la idea de los mínimos cuadrados. Si no lo recuerdan bien de algebra lineal, pueden buscar sus notas, o algún video/notas en internet. Por ejemplo esta serie de videos explica la idea en bastante detalle:

   `https://www.khanacademy.org/math/linear-algebra/alternate-bases/orthogonal-projections/v/linear-algebra-least-squares-approximation`

   El resumen es esencialmente que, para resolver $\mathbf{A}\vec{x} = \vec{b}$ cuando no hay solución (el sistema está sobre-determinado), se multiplica por la transpuesta en ambos lados $\mathbf{A}^T\mathbf{A}\vec{\mathbf{x}} = \mathbf{A}^\mathbf{T}\vec{\mathbf{b}}$.

   a. Realice una regresión lineal (m*x + b) sobre los datos `data_mean_sq.txt` para encontrar la pendiente m y el punto de corte b que use este método de mínimos cuadrados, haciendo las manipulaciones necesarias de matrices con las funciones de numpy (pero no con la función de mínimos cuadrados de numpy).

   b. Ahora vamos a hacer un ajuste a una exponencial. Normalmente esto requeriría usar un método de mínimos cuadrados más sofisticado, pero podemos "linealizar" nuestros datos a través del logaritmo y ajustar los datos linealizados. Usted debe generar sus datos para ajustar tal que: se generen usando una función que reciba un arreglo `x` y retorne la Y para cada punto de dicho arreglo con las siguientes características:

   $$y = Ae^{(-x/\tau)} + Ruido \tag{2}$$

   donde $A = 10$, $\tau = 0.5$ y el ruido es Gaussiano de amplitud 0.16 (asegúrese de que es distinto para cada punto del arreglo).

   Queremos establecer qué tan confiable es este método. Para esto vamos a ver qué tanto varían los parámetros que ajustamos para 10 muestreos. Escriba un programa que: Genere estas 10 muestras de datos (que son distintos debido al ruido adicionado en la función Y), realice los ajustes necesarios en el rango x=[0:6) con 30 puntos. Calcule e imprima un mensaje con la varianza de τ y A.