Certamen - 3 - INF-285
SCT - Sa.06.07.24 - V.1.00

Instrucciones: *Usted tiene 120 minutos para responder el Certamen con letra clara y de forma ordenada.* ***Usted tiene que mostrar todo su trabajo para obtener todos los puntos****. Puntos parciales serán entregados a preguntas incompletas. Respuestas finales sin desarrollo o* ***sin nombre*** *reciben 0 puntos. Buena letra, claridad, completitud,* ***ser conciso*** *y orden en* ***todo el certamen*** *recibe 10 puntos adicionales, excepcionalmente se considerarán puntos parciales. Copy-and-Paste de algoritmos reciben 0 puntos. ¡Éxito!*

- **Contexto Pregunta 1**: En el contexto de la interpolación polinomial de funciones trigonométricas se requiere tener acceso a la evaluación de estas en ciertos puntos, por ejemplo en los puntos de Chebyshev respectivos. En este caso, por simplicidad solo consideraremos las funciones $\sin(t)$ y $\cos(t)$ definidas en $t \in [0, 2\,\pi]$.

  Hasta ahora, hemos dependido del acceso a alguna función de alto nivel que nos entregue la evaluación de funciones trigonométricas, por ejemplo `np.cos(t)` o `np.sin(t)`, sin embargo, **nosotros** podemos construir tales evaluaciones sin depender de esas funciones de alto nivel.

  Considere la siguiente ecuación diferencial,

$$\ddot{y}(t) + y(t) = 0, \quad t \in \,]0, 2\,\pi]\,, \tag{1a}$$
$$y(0) = \alpha, \tag{1b}$$
$$\dot{y}(0) = \beta, \tag{1c}$$

  que tiene como solución general la siguiente expresión

$$y(t) = \alpha\, \cos(t) + \beta\, \sin(t).$$

  Es decir, la solución de la ecuación diferencial nos entrega la evaluación de la función $\cos(t)$, $\sin(t)$ o una combinación lineal de estas a partir de la elección **conveniente** de las condiciones iniciales respectivas.

  ¿Para que nos sirve esto en interpolación polinomial? Nos sirve para encontrar la evaluación de $\cos(t)$ o $\sin(t)$ para cualquier $t \in [0, 2\,\pi]$, que es justamente lo que se trabajará en la pregunta.

  ¿Qué se debe hacer? Se debe resolver numéricamente la ecuación diferencial hasta un tiempo $t = \tau$ para obtener la evaluación de $\cos(\tau)$ o $\sin(\tau)$ o según se solicite.

  ¿Existe alguna restricción? Sí, se sabe que el Método de Euler no se puede utilizar en este caso por razones de estabilidad. Por lo cual se debe utilizar alguno de los otros métodos disponibles para obtener una aproximación numérica de la ecuación diferencial.

- **Contexto Pregunta 2**: En un mundo post-apocalíptico se ha perdido información valiosa, pero a la vez se ha salvado otra. Este mundo post-apocalíptico ha sido producto de la colisión de un meteoro contra la Tierra. Pero eso no es todo, viene otro y pronto!

  Lo que se sabe es que la trayectoria $y(x)$ del próximo meteoro depende de la radiación x que emite el mismo y que viene modelada por:

$$y''(x) = 2\, \exp(-2\, y(x))\, (1 - x^2), \quad x \in]0, 1[, \tag{2a}$$
$$y(0) = 0, \tag{2b}$$
$$y(1) = 0. \tag{2c}$$

  Un grupo de científicos ha recurrido al ingenioso curso de Computación Científica para estimar la trayectoria $y(x)$ del meteoro y ver si impactará con nuestro planeta. El problema es que en la información que se ha perdido, también se perdieron algunos métodos computacionales para resolver este tipo de problema, por lo tanto, solamente se cuenta con un reducido número de métodos para utilizar. El desafío es ayudar a este grupo de científicos a estimar $y(x)$.

  ¿Qué se debe hacer? Se debe resolver numéricamente la ecuación diferencial para $y(x)$ con $x \in [0, 1]$.

  ¿Existe alguna restricción? Sí, se sabe que dentro de los métodos que se salvaron para resolver algún BVP son **diferencias finitas** y **GMRes**.

NOMBRE:_____ ROL:_____ PARALELO:_____

- **Desarrollo Pregunta 1**:

  (a) **[10 puntos]** Explique cómo debe elegir las condiciones iniciales de la ecuación (1) para obtener como solución la función trigonométrica $\cos(t)$ o $\sin(t)$, respectivamente.

  (b) **[15 puntos]** Considere que usted debe construir la aproximación numérica de la ecuación (1) con un método numérico, es decir conoce $y_k \approx y(t_k)$, para $t_k = h\,k$ y $k \in \{0, 1, 2, \ldots, K\}$, donde $t_K \leq \tau$ (notar que τ se definió en el contexto de la pregunta), y que $\tau < t_{K+1}$. Esto significa que τ **no** está alineado exactamente con la *grilla temporal* definida por t_k.

  Explique, concisamente, cómo puede obtener las aproximaciones de $y(t_k)$ y de $y(\tau)$, tal que asegure que se mantenga el orden del método numérico utilizado para obtener todos los y_k previos. Es decir, si usted decide utilizar un método $\mathcal{O}(h)$, $\mathcal{O}(h^2)$, o $\mathcal{O}(h^4)$ obtener los y_k, entonces la aproximación de $y(\tau)$ debe ser $\mathcal{O}(h)$, $\mathcal{O}(h^2)$, o $\mathcal{O}(h^4)$, respectivamente.

  En resumen, usted debe:

  - Considerar que la grilla temporal considera un equiespaciado h, es decir $t_j - t_{j-1} = h$ para $j \in \{1, 2, \ldots, K\}$.
  - Explicitar cómo matemática y computacionalmente construirá los y_k utilizando la ecuación (1).
  - Explicitar cómo matemática y computacionalmente obtendrá una aproximación de $y(\tau)$ manteniendo el orden del error en las aproximaciones y_k.

(c) **[25 puntos]** Implemente su propuesta de algoritmo para estimar $\cos(\tau)$ o $\sin(\tau)$ según lo indicado en la firma de la función solicitada más abajo. Para su implementación, considere que **solo** tiene a su disposición las siguientes funciones de la librería NumPy, además de las operaciones elementales, ciclos y condicionales propios de Python:

- `np.zeros(n)`: Genera un vector nulo de dimensión `n`.
- `np.arange(n)`: Para `n` un número entero positivo entrega un vector de largo `n` con números enteros desde `0` a `n-1`.
- `np.abs(x)`: Entrega el valor absoluto de `x` de forma *element-wise*.
- `np.floor(a)`: Entrega la parte entera inferior de cada elemento en el vector `a` o, respectivamente, la parte entera inferior del escalar `a`.
- `np.ceil(a)`: Entrega la parte entera superior de cada elemento en el vector `a` o, respectivamente, la parte entera superior del escalar `a`.
- `np.linspace(a, b, n)`: Entrega `n` puntos equiespaciados en el dominio $[a, b]$.
- `myChebyshev(a, b, n)`: Entrega los `n` puntos de Chebyshev en el intervalo definido $[a, b]$. Si se utiliza `myChebyshev(-1, 1, n)`, se retorna los puntos de Chebyshev "originales".
- `{eulerMethod_one_step,backwardEuler_one_step,RK2_one_step,RK4_one_step}(xi,yi)`: Implementa **un paso** del {método de Euler, *backward* Euler, RK2, RK4}, respectivamente, para *IVP* (y sistemas dinámicos) donde `yi` $\in \mathbb{R}^m$ corresponde al vector de estado en el tiempo `ti`, `f`: $\mathbb{R}^m \to \mathbb{R}^m$ es la función del lado derecho del IVP (o sistema dinámico) y `h` es el paso en el tiempo. Esta función retorna la aproximación numérica de la solución $\mathbf{y}(t_i + h)$. Notar que si la dimensión m del problema es 1 entonces es un IVP pero si $m > 1$ entonces es un sistema dinámico, considerando m un número entero. La implementación es trasparente a la dimensión del problema, por lo cual se puede usar para ambos casos.

Al momento de implementar usted **debe decidir** qué componentes se deben vectorizar y qué componentes no, considerando las funciones de NumPy antes mencionadas.

Considere la siguiente firma:

```
'''
input:
    tau     : (float) Value for Tau
    h       : (float) Temporal discretization h
    outputSelector   : (int)
                0 -> returns cos(tau),
                1 -> returns sin(tau),
                2 -> returns alpha*cos(tau)+beta*sin(tau)
    alpha   : (float) Value for alpha if used.
    beta    : (float) Value for beta if used.

    output:
    out_value  : (double) If:
               outputSelector=0, it returns cos(tau)
               outputSelector=1, it returns sin(tau)
               outputSelector=2, it returns alpha*cos(tau)+beta*sin(tau)
'''
def find_cos_sin_tau(tau, h=1e-5, outputSelector=0, alpha=1.0, beta=1.0):
        # Your own code.
    return out_value
```

En la siguiente página se repite la firma de la función

Considere la siguiente firma:

```
    '''
    input:
        tau     : (float) Value for Tau
        h       : (float) Temporal discretization h
        outputSelector   : (int)
                    0 -> returns cos(tau),
                    1 -> returns sin(tau),
                    2 -> returns alpha*cos(tau)+beta*sin(tau)
        alpha   : (float) Value for alpha if used.
        beta    : (float) Value for beta if used.

        output:
        out_value  : (double) If:
                   outputSelector=0, it returns cos(tau)
                   outputSelector=1, it returns sin(tau)
                   outputSelector=2, it returns alpha*cos(tau)+beta*sin(tau)
    '''
    def find_cos_sin_tau(tau, h=1e-5, outputSelector=0, alpha=1.0, beta=1.0):
            # Your own code.









        return out_value
```

NOMBRE:_____ ROL:_____ PARALELO:_____

- **Desarrollo Pregunta 2**:

(a) **[25 puntos]** Proponga un algoritmo que ayude al grupo de científicos para resolver numéricamente la ecuación diferencial (2), utilizando solamente los métodos indicados, es decir, **diferencias finitas** y **GMRes**. En caso de que decida utilizar Newton en $\mathbb{R}^n$, se recuerda que una aproximación de primer orden del producto **matriz-vector** entre la matriz Jacobiana y un vector arbitrario $\mathbf{v}$ es:

$$J_{\mathbf{F}}(\mathbf{y})\,\mathbf{v} \approx \frac{\mathbf{F}(\mathbf{y}+\varepsilon\,\mathbf{v}) - \mathbf{F}(\mathbf{y})}{\varepsilon}, \qquad \varepsilon = 10^{-8}. \tag{3}$$

En resumen lo que se solicita es:

- Explicitar cómo utilizará diferencias finitas para la ecuación diferencial (2).
- Explicitar el problema de búsqueda de raíz para $\mathbf{F}(\mathbf{y}) = \mathbf{0}$.
- Explicitar cómo obtendrá la aproximación numérica $y(x)$, específicamente cuál método utilizará para esta tarea. En caso de que decida utilizar Newton en $\mathbb{R}^n$, debe explicitar cómo obtener la aproximación de la solución en cada iteración y cómo se actualiza la solución.

BONUS (10 pts.) Implementación del cálculo del producto **matriz-vector** entre la matriz Jacobiana y un vector arbitrario **v** de forma **exacta**, es decir, construir una función `Jv_exact(v,y)` donde retorne el cálculo "exacto" (hasta precisión de máquina) de $J_{\mathbf{F}}(\mathbf{y})\,\mathbf{v}$ para ser utilizada por GMRes, por ejemplo en su implementación en la siguiente pregunta.

(b) **[25 puntos]** Implemente su propuesta de algoritmo para resolver numéricamente la ecuación diferencial (2) según lo indicado en la firma de la función solicitada más abajo. Para su implementación, considere que **solo** tiene a su disposición las siguientes funciones de la librería NumPy, además de las operaciones elementales, ciclos y condicionales propios de Python:

- `np.zeros(n)`: Genera un vector nulo de dimensión `n`.
- `np.arange(n)`: Para `n` un número entero positivo entrega un vector de largo `n` con números enteros desde `0` a `n-1`.
- `np.linspace(a, b, n)`: Entrega `n` puntos equiespaciados en el dominio $[a, b]$.
- `np.dot(a,b)`: Obtiene el producto interno entre el vector `a` y `b`. En caso de que `a` sea una matriz, entrega el producto matriz-vector respectivo. Para esto último también es posible utilizar el operador `@`.
- `np.ones((n_rows, n_cols))`: Entrega un `ndarray` de dimensión `n_rows`×`n_cols` donde cada coeficiente es igual a `1`. En caso de que solo se entrega un número entero como *input*, es decir, `np.ones(n)`, entonces retorna un vector de largo n con 1s en cada coeficiente.
- `np.power(x,y)`: Evalúa la expresión x^y si `x` e `y` son escalares. En caso de que `x` e `y` sean vectores, deben tener la misma dimensión y entrega la evaluación elemento a elemento. Si solo uno de los términos es un vector, entrega el vector donde el término constante se consideró para cada término de vector.
- `np.linalg.norm(x)`: Entrega la norma Euclidiana del vector `x`.
- `GMRes_explicit_matrix(A,b,x0,threshold)`: Esta función implementa el algoritmo GMRes para resolver un sistema de ecuaciones lineales de la forma $A\,\mathbf{x} = \mathbf{b}$ considerando que se tiene acceso a la matriz A almacenada explícitamente en la memoria RAM, al vector $\mathbf{b}$, `x0` como *initial guess* y que retorna la aproximación numérica $\widetilde{\mathbf{x}}$ tal que se asegura que $\|\mathbf{b} - A\,\widetilde{\mathbf{x}}\| \leq$`threshold`, es decir menor o igual que el valor `threshold`.
- `GMRes_matrix_free(afun,b,x0,threshold)`: Esta función implementa el algoritmo GMRes para resolver un sistema de ecuaciones lineales de la forma $A\,\mathbf{x} = \mathbf{b}$ considerando que solo se tiene acceso a la función `afun`, que recibe como parámetro un vector `v` y retorna el vector `w` que corresponde al producto entre la matriz A y el vector `v`, al vector $\mathbf{b}$, `x0` como *initial guess* y que retorna la aproximación numérica $\widetilde{\mathbf{x}}$ tal que se asegura que $\|\mathbf{b} - A\,\widetilde{\mathbf{x}}\| \leq$`threshold`, es decir menor o igual que el valor `threshold`.

Al momento de implementar usted **debe decidir** qué componentes se deben vectorizar y qué componentes no, considerando las funciones de NumPy antes mencionadas.

Considere la siguiente firma:

```
'''
input:
    n        : (int)     number of interior points to obtain the numerical
                         aproximations, notice that y(x_0) and y(x_{n+1}) are the
                         boundary conditions.
    gmres_th : (float)   value for threshold used in GMRes method.
    rel_tol  : (float)   value for relative tolerance where norm(F(y))/norm(F(0)) <= rel_tol.
                         If it is achieved, the code needs to return the approximation.
    n_newton : (int)     max number of iterations used in Newton method.
    output:
    y        : (ndarray) numerical approximation for y(x).
'''
def bvp_meteor(n,gmres_th=1e-6,rel_tol=1e-7,n_newton=100):
        # Initialization of output variable.
        # Notice that y[0] and y[-1] must be kept equal to 0 since they are
        # the boundary conditions.
        y = np.zeros(n+2)
        # Your own code.
    return y
```

En la siguiente página se repite la firma de la función

Considere la siguiente firma:

```
    '''
    input:
        n        : (int)     number of interior points to obtain the numerical
                             aproximations, notice that y(x_0) and y(x_{n+1}) are the
                             boundary conditions.
        gmres_th : (float)   value for threshold used in GMRes method.
        rel_tol  : (float)   value for relative tolerance where norm(F(y))/norm(F(0)) <= rel_tol.
                             If it is achieved, the code needs to return the approximation.
        n_newton : (int)     max number of iterations used in Newton method.
        output:
        y        : (ndarray) numerical approximation for y(x).
    '''
    def bvp_meteor(n,gmres_th=1e-6,rel_tol=1e-7,n_newton=100):
            # Initialization of output variable.
            # Notice that y[0] and y[-1] must be kept equal to 0 since they are
            # the boundary conditions.
            y = np.zeros(n+2)
            # Your own code.
        return y
```

```
        return y
```