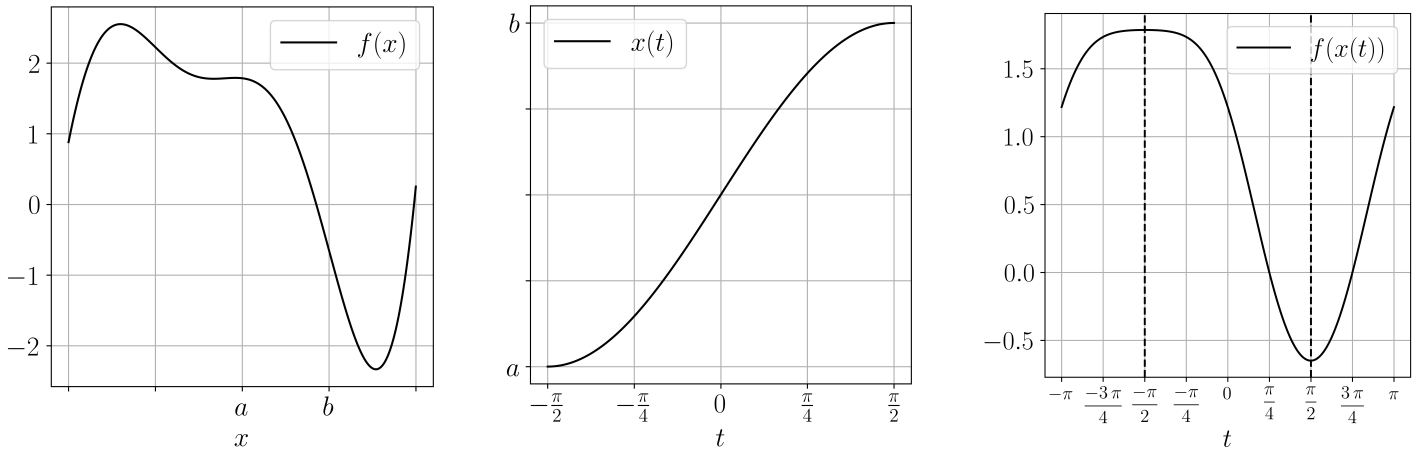
Ejercicio en clases
Scientific Computing Team - Semana - 05

1. ***Bounded Newton***: Sea $f_k : \mathbb{R} \to \mathbb{R}$ una secuencia de funciones que tienen múltiples raíces, sin embargo acá estamos interesados en encontrar una raíz particular de cada $f_k(x)$. En particular, la raíz de nuestro interés, r_k tal que $f(r_k) = 0$, está localizada en el intervalo $]a_k, b_k[$, $a_k < b_k$, y $f(a_k)\, f(b_k) < 0$, para $k \in \{1, 2, \ldots, K\}$. Una clara alternativa para encontrar las K raíces es la utilización del método de la Bisección, sin embargo exploraremos otra alternativa. En particular, adaptaremos el método de Newton para buscar raíces localizadas en un intervalo, por simplicidad lo denotaremos como $[a, b]$.

   Para la ejecución de esta tarea, considere que utilizaremos el siguiente cambio de variable:

$$x(t) = \frac{1}{2}((b - a)\sin(t) + a + b), \tag{1}$$

   ver figura 1b para referencia. Es decir, el cambio de variable lleva valores del intervalo $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ a $[a, b]$.



(a) Gráfica de la función $f(x)$, notar que se incluyó un intervalo extendido de forma referencial. Es decir, para valores de x menores a a y mayores a b.

(b) Gráfica del cambio de variable de $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ a $[a, b]$.

(c) Composición de las funciones $f(x)$ y $x(t)$.

Figura 1: Ejemplo del cambio de variable propuesto.

   Ahora, utilizando el cambio de variable, re-definiremos el problema de búsqueda de raíces en estudio reemplazando la variable independiente por el cambio de variable propuesto, es decir, realizaremos una composición de funciones y ya no buscaremos primeramente la raíz r_k, si no que buscaremos t_k tal que

$$f(x(t_k)) = 0.$$

   Lo que implica que $r_k = x(t_k)$. Es decir, podemos encontrar r_k asegurando que estará en el intervalo requerido!

   (a) Proponga un algoritmo que permite encontrar la raíz $r_k \in [a_k, b_k]$ de una función $f_k(x)$ **utilizando** el cambio de variable de la ecuación (1). Notar que la $f_k(x)$ puede tener más raíces en general pero fuera del intervalo indicado. Usted debe indicar,

   - explícitamente cómo utilizará el cambio de variable de la ecuación (1),
   - el *initial guess* a utilizar,
   - explícitamente la iteración de Newton respectiva.

(b) Implemente en Python utilizando adecuadamente la librería NumPy (en especial su capacidad de vectorización) el algoritmo propuesto en la Pregunta 1 ítem (a), es decir, en la pregunta anterior. 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.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`.
- `np.cos(x)`: Entrega la evaluación de la función coseno de los valores en radianes en el vector o escalar `x`.
- `np.sin(x)`: Entrega la evaluación de la función seno de los valores en radianes en el vector o escalar `x`.
- `np.log(x)`: Entrega la evaluación de la función logaritmo natural de los valores en el vector o escalar `x`.
- `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.

Notar que 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:
fk  : (callable) Input function f_k(x).
ak  : (float) Left limit for interval.
bk  : (float) Right limit for interval.
m   : (int) Number of iterations to be used in Newton's method.

output:
rk  : (float) The approximation of the root of f_k(x) such that f(rk)=0.0
'''
def compute_bounded_root(fk,ak,bk,m):
    # Your own code.
    return rk
```

2. ***Anomalía Excéntrica***: Dos interesantes propiedades para observar de la órbita de un cuerpo celeste alrededor del sol son el período τ y la excentricidad elíptica ϵ (Un círculo tiene $\epsilon = 0$). A partir de esto es posible encontrar, para cualquier tiempo t, el ángulo θ que se forma entre la posición del cuerpo y el eje mayor de la elipse (órbita). Esto se mediante la siguiente ecuación:

$$\tan\left(\frac{\theta}{2}\right) = \sqrt{\frac{1+\epsilon}{1-\epsilon}}\tan\left(\frac{\psi}{2}\right), \tag{2}$$

donde la *anomalía excéntrica* ψ satisface la ecuación de Kepler:

$$\psi - \epsilon\,\sin(\psi) - \frac{2\,\pi\,t}{\tau} = 0. \tag{3}$$

Notar que tanto la función θ como la función ψ dependen del tiempo, sin embargo se omite usar la notación $\theta(t)$ como $\psi(t)$ por simplicidad, pero en rigor ambas dependen del tiempo en las ecuaciones (2) y (3).

Por otro lado, podemos observar que para encontrar el ángulo θ para algún tiempo t, periodo τ y excentricidad ϵ, se debe resolver la ecuación (3) para obtener ψ y luego obtener θ a partir de la ecuación (2).

a) Proponga un algoritmo que permita obtener el valor de $\theta(t)$ para algún tiempo $0 < t \leq \tau$ mediante el método de Newton. Considere que el algoritmo recibe además como input la excentricidad ϵ, el período τ y un *initial guess* de θ llamado θ_0.

b) Implemente en Python utilizando adecuadamente la librería NumPy (en especial su capacidad de vectorización) el algoritmo propuesto en el primer ítem. 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.arange(n)`: Para `n` un número entero positivo entrega un vector de largo `n` con números enteros desde `0` a `n-1`.
- `np.sqrt(x)`: Entrega la evaluación de la raíz cuadrada no negativa de un vector o escalar `x`.
- `np.cos(x)`: Entrega la evaluación de la función coseno de los valores en radianes en el vector o escalar `x`.
- `np.sin(x)`: Entrega la evaluación de la función seno de los valores en radianes en el vector o escalar `x`.
- `np.tan(x)`: Entrega la evaluación de la función tangente de los valores en radianes en el vector o escalar `x`.
- `np.arctan(x)`: Entrega la evaluación de la función arcotangente de los valores en radianes en el vector o escalar `x`.
- `np.log(x)`: Entrega la evaluación de la función logaritmo natural de los valores en el vector o escalar `x`.
- `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.
- `Newton1D(f,fp,x0)`: Implementa el método de Newton en 1D que entrega la aproximación de la raíz de `f`. Esta función recibe como parámetros la función `f`, la derivada `fp` de `f` y el *initial guess* `x0`. Por simplicidad se omite criterio de detención.

Notar que 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:
t          : (float) Input value time.
eps        : (float) Elliptical eccentricity.
tau        : (float) Period of the celestial body.
theta0     : (float) Initial guess for psi.

output:
theta      : (float) The value of the angle theta.
'''
def find_theta(t,eps,tap,theta0):
    # Your own code.
    return theta
```