Práctica 2, (Ceros de funciones)
Métodos numéricos
2n curso del grado en Matemáticas

Fecha de Reporte: **31/03/2019**
Alumno: *Graells Ricardo, Marc*
(NIU: *1388471*)

0. Resumen

En el presente documento se comentan las soluciones a los ejercicios propuestos en las sesiones de *Seminarios de Métodos Numéricos* de los días 11, 18 y 25 de marzo. Los ficheros `.c` con las implementaciones del código en lenguaje C se adjuntan tal como se detalla en documento del mismo subdirectorio `README.txt`.

1. Primer problema

1.1. Apartado a y b

Analíticamente si $f(x) = x^3 - x - 40 = 0$ y por fórmulas de Cardano α cumple la ecuación, es decir, es raíz del polinomio $P_1(x) = x^3 - x - 40$. Esperamos que $\alpha^3 - \alpha - 40$ sea cero. Implementado en C con precisión `float` y `double`. Tal como vemos en Tabla 1 obtenemos

Precisión	$\lvert\alpha^3 - \alpha - 40\rvert$
`float`	3.09×10^{-4}
`double`	3.27560×10^{-12}

Tabla 1: Resultados obtenidos $|P_1(\alpha)|$ mediante `pr1_a_fd.c`.

errores del orden de 10^{-4} y 10^{-12} cuando cabria esperar 10^{-7} y 10^{-16}, respectivamente. Por tanto comprobamos que hay error de cancelación y concluimos que *evaluar directamente* no es optimo.

Implementamos el método de Newton en lenguaje C para obtener una aproximación de α, raíz del polinomio $P_1(x) = x^3 - x - 40$.

Antes de ejecutarlo podemos estimar que, si suponemos que el método converge, la raíz del polinomio es simple y los valores de la sucesión distan de un extremo relativo próximo al eje x; esperamos que en estas condiciones los decimales correctos se dupliquen en cada iteración. Suponiendo que obtenemos 2 decimales correctos en la tercera aproximación, conjeturamos que en 7 iteraciones aproximadamente tengamos 15 decimales correctos, pongamos entorno a unas diez (iteraciones) por usar potencias, 10[1]. Compilamos y ejecutamos el programa [1]. Observamos los valores en Tabla 2. Los resultados son aproximadamente los esperados, observamos que con precisión `float` no obtenemos los 8 decimales correctos. A partir de aquí solo utilizaremos variables `double` ya que deseamos más decimales correctos; mayor precisión de la que puede ofrecer `float` .

[1] Ver `pr1_b_fd.c` .

Precisión	# Iteración suficiente necesaria	α (8 decimales correctos)	# Iteración suficiente necesaria	α (15 decimales correctos)
`float`	(5,6)	3.517393599	×	*Excede precisión de la variable*
`double`	(6,7)	3.51739351	(7,10)	3.51739351405282

Tabla 2: Resultados obtenidos para α con $x_0 = 2.0$ mediante `pr1_b_fd` .Observamos en gris número de iteraciones donde ya obtenemos los decimales correctos deseados.El segundo componente de la 2-tupla (en color negro) es el número de iteraciones que nos asegura los decimales correctos deseados aplicando que $|x_{i+1} - x_i| \approx |x_i - \alpha|$. En rojo destacamos el decimal incorrecto en `float`.

1.2. Apartado c

Obtenemos fórmula de Cardano para la ecuación $x^3 - x - 400 = 0$, al ser el discriminante positivo ($\Delta > 0$) estamos en el caso de dos raíces complejas y una real. Tenemos pues que la fórmula de Cardano para la solución es

$$\beta = \sqrt[3]{\frac{-q+\sqrt{\Delta}}{2}} + \sqrt[3]{\frac{-q-\sqrt{\Delta}}{2}} \approx 7.41330 \quad \text{donde} \begin{cases} p = -1 \\ q = -400 \\ \Delta = q^2 + \frac{4}{27}p^3 \approx 399.8 > 0 \\ 2 \leq \beta \leq 8 \end{cases}$$

Evaluando el valor de β obtenido por fórmulas de Cardano estimamos un error absoluto de 3.25315×10^{-10}. Implementando en lenguaje C los métodos iterativos de Bisección (intervalo $[2,8]$) **(c1)**, Secante **(c2)** (intervalo $[2,8]$) y Newton **(c3)** (pivote $x_0 = 2.0$) obtenemos 15 decimales correctos de β, vemos Tabla 3. [2] Con este ejemplo y los que siguen en el

Método	# Iteración (suficiente, necesaria)
Bisección	(54,56)
Secante	(7,8)
Newton	(10,11)

Tabla 3: Observamos en gris número de iteraciones donde ya obtenemos los decimales correctos deseados.El segundo componente de la 2-tupla (en color negro) es el número de iteraciones que nos asegura los decimales correctos mediante `pr1_c_d.c`.

documento podemos *intuir* que probablemente la mayoría de métodos de convergencia numérica que convergen de forma rápida (es decir tienen mayor **orden de convergencia**) normalmente tienen un **intervalo de convergencia** *relativamente pequeño*, es decir, si dado un pivote o intervalo numérico el método converge o no. La estrategia es sencilla, si buscamos la raíz de un polinomio probablemente será interesante comprobar su existencia de forma analítica en primer lugar, acotar la solución por método de la Bisección (orden de convergencia lineal) con unas pocas iteraciones en segundo lugar y finalmente aplicar el método de Newton (cuadrático) ; uniendo los *puntos fuertes* de todos los métodos. Opcionalmente si la derivada es tediosa se puede optar por el método de la secante (cuadrático).

[2] β con 15 decimales correctos: `7.413302725857898`.

2. Segundo problema

2.1. Apartado a

Implementamos el método propuesto en lenguaje C [3]. Probamos para tres valores concretos de x_0. Si $x_0 = 2.0$ el método no converge, de hecho esto ya lo podíamos esperar ya que se exige x_0 próximo a x^*, raíz. Si consideramos $x_0 = 7.0$ obtenemos convergencia como cabría esperar, ya que $|x_0 - x*| < 1$.

Analizamos ahora la convergencia con $x_0 = 6.0$. Observamos que el método da 15 decimales correctos en la iteración número 8. La convergencia del método la obtenemos de forma *experimental* comparando los valores [4]

$$\frac{e_k}{e_{k-1}}, \frac{e_k}{(e_{k-1})^2}, \cdots, \frac{e_k}{(e_{k-1})^4}$$

Partiendo de la definición de orden de convergencia de una succión, si $p, m \in \{1,2,3,4\}$ representa un valor de convergencia, es decir, lineal, cuadrática, cubica y cuártica respectivamente. Sea p el orden de convergencia del método, esperamos que: [5]

$$\begin{cases} \lim_{n\to\infty} \frac{e_k}{(e_{k-1})^m} = 0 & \text{si } m < p, \\ \lim_{n\to\infty} \frac{e_k}{(e_{k-1})^m} = C \in \mathbb{R} & \text{si } m = p \text{ i} \\ \lim_{n\to\infty} \frac{e_k}{(e_{k-1})^m} = \infty & \text{si } m > p \end{cases}$$

A la practica, tal como vemos en Tabla 4 con 7 filas y 4 columnas de datos, analizamos el crecimiento y decrecimiento de estos valores más que su orden de magnitud absoluto. En concreto por crecimiento comparado, es decir, si crecen o decrecen en *exceso* comparativamente. Considerando esto obtenemos que la convergencia del nuevo método para el

Iteración [#]	$\frac{e_k}{e_{k-1}}$	$\frac{e_k}{(e_{k-1})^2}$	$\frac{e_k}{(e_{k-1})^3}$	$\frac{e_k}{(e_{k-1})^4}$
k=2	0.1031033459	0.0580634632	0.0326988977	0.018414642
k=3	0.7218815351	3.942968893	21.53677984	117.63544
k=4	0.328420595	2.484974108	18.80240281	142.26722
k=5	0.08573593649	1.975258762	45.50772215	1048.4463
k=6	0.007696644473	2.068233798	555.7735007	149346.84
k=7	7.007928741E-05	2.446732305	85424.65534	2.982497E+09
k=8	4.424939312E-07	220.4521916	1.098301363E+11	5.47178E+19

Tabla 4: Resultados obtenidos con `pr2_a_d` para $x_0 = 6.0$. Estimamos convergencia como mucho cuártica. Observamos que en la iteración 7 se despejan todas las dudas respecto el orden de convergencia del método, que claramente es cuadrático.

Problema 1 es **cuadrática**.

[3] Ver `pr2_a_d.c` .

[4] Cogemos tan solo cuatro valores, ya que no esperamos convergencia mayor que cubica.

[5] Si p=1 exigiremos además que $C < 1$.

3. Tercer problema

3.1. Apartado a y b

Implementamos el método de Halley en lenguaje C [6]. Probamos para solucionar ecuación del apartado c) del Problema 1 con tres valores concretos de x_0. Procedemos como en Problema 2. Para $x_0 = 2.0$ obtenemos convergencia con 15 decimales correctos en la quinta iteración. Para $x_0 = 6.0, 7.0$ obtenemos 15 decimales correctos en 3 iteraciones. Utilizamos criterio del problema anterior, resumido en Tabla 5. Sutilmente vemos que en este caso el

Iteración [#]	$\frac{e_k}{e_{k-1}}$	$\frac{e_k}{(e_{k-1})^2}$	$\frac{e_k}{(e_{k-1})^3}$	$\frac{e_k}{(e_{k-1})^4}$
k=2	0.03366356137	0.02462092936	0.01800730933	0.0131702254
k=3	2.634236837E-05	0.000572320289	0.01243436082	0.2701517524

Tabla 5: Resultados obtenidos con `pr3_ab_d` para $x_0 = 6.0$. Estimamos convergencia como mucho cuártica. Concluimos orden de convergencia cubico.

orden de convergencia es cubico, esto lo podemos observar ya que los dos primeros valores disminuyen 4 y 2 ordenes de magnitud, el tercero mantiene el orden de magnitud y el cuarto aumenta un orden de magnitud. Por tanto, descartando, obtenemos numéricamente que el orden de convergencia del método es de orden 3 o **cubica**.

4. Cuarto problema

Con ayuda de `pr4_a_d.c` Observamos que con tan solo `k=5` iteraciones ja obtenemos 14 decimales[7] correctos de π. Efectuando el estudio de convergencia, como en Problema 2 y 3, obtenemos orden de convergencia cuadrático. A partir de aquí el valor resultante se va alejando de π, es decir, $|\pi - p_k|$ crece. En las iteración `k=45`, `k=46`, y `k=47` obtenemos valores de aproximadamente 3.2528, 3.3722, 3.6393. En las iteraciones `k=48` y `k=49`; 4.3462 y 6.9367, respectivamente. El la iteración `k=50` obtenemos un valor de -33.34322 aproximadamente. A partir de aquí cada vez obtenemos valores negativos muy pequeños, en la iteración `k=100` obtenemos un valor de orden de 10^{-15}; concluyendo que el método ha degenerado totalmente, probablemente por error de cancelación.

El método comienza a dar problemas bastante cerca de la precisión máxima para `double` incentivando pensar que el error es causado por la precisión finita. Por D.A.G[8] tenemos que $b_k \leq a_k$, de hecho, el límite de estos existe y coincide $\lim_{k\to\infty} a_k = \lim_{k\to\infty} b_k$, es decir, esperamos valores cada vez más próximos y como la convergencia es rápida en la quinta iteración estos valores ,que cada vez son más próximos, no mejoran el resultado que queda degenerada al augmentar el número de operaciones. Algunas curiosidad sobre el método ,ver .[9]

[6] Ver `pr3_ab_d.c` .

[7] 3.141592653589871 donde el decimoquinto decimal correcto debería ser 9.

[8] Desigualdad aritmético-geométrica.

[9] A.Gasull, *Fes matemàtiques!* , pág: 26-27 ,(2000).