Quiz 1
Algoritmos y Complejidad

«Búsqueda de raíces»

Algorithm Knaves

2020-09-21

Sea $f(x) = 54x^6 + 45x^5 - 102x^4 - 69x^3 + 35x^2 + 16x - 4$ con 5 ceros en el intervalo $[-2,2]$.

1. **(30 pts.)** Utilice el **método de Newton** para encontrar los 5 ceros de $f(x)$. Indique el punto inicial (initial guess) con el que halló cada cero. Informe en cada iteración del método de Newton sobre la aproximación actual y el respectivo error.

   Respuesta.

   No podemos saber de antemano si para un initial guess x newton converge a un cero r. Solo nos queda intentar con algunos valores.

   Luego de algunas prueba y error, se consigue el siguente arreglo de initial guess $[-2, -1, 0, 0{,}8, 1]$ con el cual se converge a los 5 ceros de $f(x)$. Claramente esta arreglo no es único.

   El siguiente código encuentra los ceros pedidos. La función `newton_backward` informa en cada iteración la solución aproximada y su error asociado, esta función retorna una tupla de la forma (aproximación, backward error asociado). Para el output utilizamos la función `zip` para asignar a cada initial_guess el cero al cual converge newton.

```
epsilon_backward = 0.5 * (10 ** -13)
initial_guesses = [-1, 0.8, -2, 0]
approximate_roots = []

for initial_guess in initial_guesses:
    approximate_roots.append( newton_backward(f, f_prime, initial_guess,
                                epsilon_backward)[0] )

print("initial guesses and approximate roots: ")
print(list(zip(initial_guesses, approximate_roots)))
```

2. **(30 pts.)** Determine las raíces para las que Newton converge linealmente, y para las cuales la convergencia es cuadrática.

   Respuesta. Sabemos que Newton es cuadrático para raices simples. El siguiente código analiza la existencia de raices de multiplicidad doble:

```
roots = (-2/3,-1.38129848204399, 0.205182924689048, 0.5, 1.17611555735495)
multiple_roots = []
for root in roots:
    if ( abs(f_prime(root) - 0) < epsilon_backward):
        multiple_roots.append(root)

print("roots for which newton is linear")
print(multiple_roots)
```

   Este programa solo nos entrega $r = -2/3$ **como candidato** a tener multiplicidad doble. Como $f''(-2/3) \neq 0$ concluimos multiplicidad doble y por lo tanto convergencia lineal con razón $1/2$ para el cero $r = -2/3$. Los demás ceros son simples (si tuvieran multiplicidad > 2 el programita señalado debería informalos), por lo tanto newton converge cuadráticamente a ellos. El valor epsilon_backward utilizado fue $0{,}5 \cdot 10^{-13}$.

   No era necesario programar para esta pregunta, sin embargo el código recién expuesto explica el proceso mediante el cuál podíamos deducir los ceros con multiplicidad mayor a 1. En caso de basarse en un script para responder, hay que tener en cuenta que no contamos con todas las raíces exactas. Además, fraciones como $2/3$ solo las podemos computar con un límite de precisión. Por lo tanto no tiene sentido preguntar $f''(x) = 0$ a la hora de programar, es necesaria una holgura.

3. **(40 pts.)** Elija *adecuadamente* un intervalo para cada cero obtenido en 1. Luego compare el desempeño del **método de la Bisección** con el **método Regula Falsi** para los 5 intervalos recién elegidos, indique sus conclusiones. Para medir el error puede usar backward o forward error. Indique cuál medida utilizó y argumente el porqué.

   Respuesta.

   Debemos tomar intervalos $[a,b]$ que horquillen a cada cero y solo ese cero. Notar que esto es imposible para el cero de multiplicidad doble encontrado en 2.

   Para comparar estos dos métodos utilizaremos el mismo conjunto $[a,b]$ inicial, el mismo criterio de error y el mismo mínimo error epsilon (forward) como tolerancia.

   El siguiente código hace el trabajo, donde se ha programado el método de la bisección y el método regula falsi utilizando el forward error:

```
epsilon_forward = 0.5 * (10 ** -10) # soluciones correctas en 10 cifras

roots = (-1.38129848204399, -2/3, 0.205182924689048, 0.5, 1.17611555735495)
intervals = [(-2, -1, roots[0]), (0, 0.4, roots[2]), (0.3, 0.6, roots[3]),
             (0.6, 1.5, roots[4])]

solutions_bisection = []
solutions_regulafalsi = []
for a,b,root in intervals:
    solutions_bisection.append(bisection_forward(f, a, b, epsilon_forward, root))
    solutions_regulafalsi.append(regulafalsi_forward(f, a, b, epsilon_forward, root))

print("comparison between bisection and regulafalsi (solution, associated error,
                                                      it_numbers): ")
for j in range(len(intervals)):
    print(f"Bisección:  {solutions_bisection[j]}" )
    print(f"Regula falsi: {solutions_regulafalsi[j]}")
    print("**********")
```

   Como se utiliza la misma cota para el forward error no se obtienen diferencias en el orden del error por lo que parece necesario estudiar cuánto trabajo le tomó al método (en iteraciones) llegar a la solución que cumple la cota del error. Se concluye que el método de la bisección es más "estable" para los intervalos elegidos pues en promedio bisección toma 32 iteraciones, regula falsi 73.

   Estas conclusiones son abiertas pues depende del intervalo elegido y el criterio de detención. Es interesante también obtener el valor S para regula falsi en cada intervalo, aunque en el fondo esto se traduce en más o menos iteraciones.

1. Condiciones de entrega

- El quiz se realizará *individualmente* (esto es grupos de una persona), sin excepciones.
- La entrega debe realizarse vía Moodle en un *tarball* en el área designada al efecto, bajo el formato `quiz-1-`*`rol`*`.tar.gz` (`rol` con dígito verificador y sin guión).

  Dicho *tarball* debe contener las fuentes en LaTeX 2_ε (al menos `quiz-1.tex`) de la parte escrita de su entrega, además de un archivo `quiz-1.pdf`, correspondiente a la compilación de esas fuentes.
- En la portada de su texto deberá incluir una tabla como la siguiente:

Concepto	**Tiempo [min]**
Investigación	
Desarrollo	
Informe	

  Acá *investigación* incluye revisión de apuntes, búsquedas, lectura de otras referencias; *desarrollo* es el tiempo invertido en hallar la solución pedida; *informe* se refiere al tiempo requerido para confeccionar el entregable.
- Si usa material adicional al discutido en clases, detállelo. Agregue información suficiente para ubicar ese material (en caso de no tratarse de discusiones con compañeros de curso u otras personas).
- En caso de haber programas, su ejecutable *debe* llamarse `quiz1`, de haber varias preguntas solicitando programas, estos deben llamarse `quiz1-1`, `quiz1-2`, etc. Si hay programas compilados, incluya una `Makefile` que efectúe las compilaciones correspondientes.

  Los programas se evalúan según que tan claros (bien escritos) son, si se compilan y ejecutan sin errores o advertencias según corresponda. Parte del puntaje es por ejecución correcta con casos de prueba. Si el programa no se ciñe a los requerimientos de entrada y salida, la nota respectiva es cero.
- La entrega debe realizarse el día indicado en Moodle. No se aceptarán entregas atrasadas.