Instituto Tecnológico de Costa Rica
Área de Ingeniería en Computadores
Profesor: Dr. Pablo Alvarado Moya
CE-3102 Análisis Numérico para Ingeniería — Fecha de entrega: **28 de agosto, 2018**
II Semestre 2018

Tarea 3

En esta tarea se reforzarán los conceptos vistos en clase sobre métodos cerrados y abiertos para encontrar raíces de funciones.

Para ello se implementarán los métodos vistos en clase y un método adicional expuesto en el libro *Numerical Recipes*. El lenguaje a utilizar es C++, y se hará uso particularmente de técnicas de programación genérica.

Los métodos a implementar, con sus respectivas interfaces, son los siguientes:

1. Bisección

```
template<typename T>
T rootBisection(const std::function<T(T)>& funct,
                T xl,
                T xu,
                const T eps)
```

2. Interpolación lineal

```
template<typename T>
T rootInterpolation(const std::function<T(T)>& funct,
                    T xl,
                    T xu,
                    const T eps)
```

3. Newton-Raphson

```
template<typename T>
T rootNewtonRaphson(const std::function<T(T)>& funct,
                    T xi,
                    const T eps)
```

4. Secante

```
template<typename T>
T rootSecant(const std::function<T(T)>& funct,
             T xi,
             T xii,
             const T eps)
```

5. Brent

```
template<typename T>
T rootBrent(const std::function<T(T)>& funct,
            T xl,
            T xu,
            const T eps)
```

6. Ridder

```
template<typename T>
T rootRidder(const std::function<T(T)>& funct,
             T xi,
             T xii,
             const T eps)
```

donde

`funct` función a la que se buscarán las raíces.

`xl` valor inferior del intervalo de búsqueda en métodos cerrados

`xu` valor superior del intervalo de búsqueda en métodos cerrados

`xi` valor inicial en métodos abiertos

`xii` valor anterior al inicial en métodos abiertos secuenciales

`eps` tolerancia del error

Para la implementación utilice el código colocado en Documentos/Tareas/Tarea 3/Código para Tarea 03. Lo puede descomprimir con:

```
> tar xzvf tarea03.tgz
```

En el directorio `include` encontrará los archivos `Root*.hpp` donde deberá implementar las funciones con la interfaz indicada.

Las pruebas unitarias ya están implementadas en el archivo `test/testRootFinders.cpp` y usted deberá asegurar que los métodos que usted implemente pasen todas las pruebas ya allí especificadas.

Las pruebas unitarias utilizan el marco de pruebas de Boost, así que busque información sobre dicho marco de pruebas para poder usar apropiadamente las opciones de línea de comando y poder obtener información suficiente de las pruebas.

Las pruebas unitarias ya son correctamente configuradas para compilación por el sistema CMake. Para ejecutarlas utilice la aplicación en `build/test/tester`, si es que usted utilizó `build` como carpeta de compilación.

Por ejemplo, con

```
cd build/test/tester
./tester --help
```

usted podrá ver todas las opciones diponibles para el tester. Con

```
cd build/test/
./tester --list_content
```

usted obtendrá la lista de pruebas disponibles, y con

```
./tester -t RootFinders -r detailed
```

ejecutará solo las pruebas asociadas a la búsqueda de raíces y podrá ver los detalles de todas las pruebas ejecutadas.

Analice el archivo `benchmarkRootFinders.cpp` en la carpeta de `benchmarks`. En ese archivo usted encontrará una clase que encapsula cualquier función `std::function<T(T)>`. Su tarea es interceptar llamadas a la función original e incrementar un contador, que dicha clase permita acceder para saber cuántas veces se ha llamado la función encapsulada.

Con instancias de esa clase se llama a los solucionadores para evaluar cuántas llamadas de la función son requeridas en cada uno de los métodos implementados para encontrar las respectivas soluciones.

El archivo tiene un prototipo muy básico para evaluar el número de llamadas en función de la precisión deseada para las soluciones, y se evalúa para las funciones allí definidas:

1. $|x| = e^{-x}$
2. $\exp(-x^2) = \exp(-(x-3)^2/3)$
3. $x^2 = \text{atan}(x)$
4. $(x-2)^3 + \frac{1}{100}(x-2)$

Puesto que los *benchmarks* utilizan el mismo marco de pruebas de Boost, usted puede ejecutar solo las pruebas de los buscadores de raíces con:

```
cd build/benchmarks
./benchmark -t RootFinders -r detailed
```

puede ejecutar solo las pruebas de desempeño de los buscadores de raíces.

Usted debe modificar estas pruebas de desempeño para mostrar los resultados gráficamente. Esto implica que debe exponer todos los datos generados en una matriz o vector, para poder graficarlos. Puede usar las pruebas de suma de matrices como ejemplo. Decida cómo presentar los resultados, puesto que tiene varias dimensiones que considerar, al calcularse el número de llamadas en términos de la tolerancia, la función de prueba y método de búsqueda de raíces. Las gráficas deben ser mostradas en la ejecución del código. Entregue las gráficas además en un archivo PDF.

Nota: La implementacion y uso de la clase encapsuladora requiere comprender aspectos avanzados de C++. Investigue por qué es necesiario el uso de `mutable` para el contador. Además, investigue por qué es necesario hacer esto

```
f_type c1(CallCounter<T>(t1<T>));
solver(c1,T(0),eps);
std::cout << c1.template target< CallCounter<T> >()->counter() << ";␣";
```

y por qué es necesario usar `.template` en la tercera línea.