20211046
Hyeonjang An

Homework 1

EC6301
Numerical Opimization
September 15, 2021

Problem

Discuss thier comparative performance for at least four different problems you generate.

1. Target functions and derivative
   Function1 is general quaratic function, which has derivative as cubic. function2 is log and function3 is trigonometric function. function4 is the minus signed version of gaussian function, which is usally used as kernel. It's σ value is set as 1.4

function	original function	derivation of function	interval
function1	$f(x) = x^4 + 2x^3 - 3x^2 - 10x + 7$	$f'(x) = 4x^3 + 6x^2 - 6x - 10$	[-5, 5]
function2	$f(x) = x\ln(x)$	$f'(x) = \ln(x) + 1$	[0.1, 5]
function3	$f(x) = \sin(x) + x^2 - 10$	$f'(x) = \cos(x) + 2x$	[-5, 5]
function4	$f(x) = -\exp(-\frac{x^2}{\sigma^2})$	$f'(x) = \frac{x}{\sigma^2}\exp(-\frac{x^2}{2\sigma^2})$	[-1, 1]

2. Conditions
   - Within the interval, all functions are continuous and the first order derivative of those are also continuous.
   - In the case of bracketing method (bisection & regular falsi)
     Within the interval the function has the value zero. This can be calculated analytically.
   - In the case of straight line method (Newton's & secant)
     For the comparision pairness, the initial points of each methods are same as the maximum point of interval, which is used in bracketing method.

3. Peformance comparision

	bisection	Newton's	secant	regular falsi
function1	2072ns	546ns	789ns	11241ns
function2	2639ns	189ns	892ns	5778ns
function3	2428ns	376ns	400ns	1310ns
function4	105ns	213ns	36.9ns	238ns

4. Analysis
   Apparently, the convergence of Newton's method is the fastest. Moreover, the overhead of regular falsi method is bigger than I thought. In the case of function1, regular falsi method has the slowest convergence rate.
   The speical thing is function 4. In the case of gaussian function, Newton's method has the slowest. I think that it is because of the calculation overhead from derivation.

Implementation

Implement the method of bisection , Newtons's, secant, regular falsi.

1. Optimizing Method Class

```
class Method {
public:
  Method(std::function<float(const float&)> f):function(f){};

  // optimization methods
  float bisection(float start, float end);
  float newtons(float x);
  float secant(float x1, float x0);
  float regular_falsi(float start, float end);

protected:
  // target function as member
  std::function<float(const float&)> function;
};
```

2. Bisection method

method 1: bisection

```
float Method::bisection(float start, float end) {
  assert( function(start)*function(end)<0 );

  auto midpoint = (start + end)/2.f;

  if(function(midpoint)==0 || end-start<MIN)
    return midpoint;

  if(function(midpoint)*function(start)<0)
    midpoint = bisection(start, midpoint);
  else
    midpoint = bisection(midpoint, end);

  return midpoint;
}
```

3. Newtons's method

method 2: Newton's

```
float Method::newtons(float x0) {
  // approximattion of derivative lambda function
  auto d =
  [](std::function<float(const float&)> func, float x, float eps=1e-6)
  {
      return (func(x+eps) - func(x))/eps;
  };

  float x1 = x0;
  while(function(x1)>0.f) {
      float t = x1;
      x1 = t - function(t)/d(function, t);
  }
  return x1;
}
```

4. Secant method

method 3: secant

```
// Two point approximation method
float Method::secant(float x1, float x0){
    // no matter which one is bigger
    float t1 = std::min(x1, x0);
    float t0 = std::max(x1, x0);

    // initial two points
    float x2 = MAX;
    while(function(x2)>0.f)
    {
        x2 = t1 - ((t1-t0)/(function(t1)-function(t0))) * function(t1);

        t0 = t1;
        t1 = x2;
    }

    return x2;
}
```

5. Regular-falsi method

method 4: regular false

```
// recursive version
float Method::regular_falsi(float start, float end){
    // secant method lambda
    auto sec = [](std::function<float(const float&)> func, float x1,
        float x0)
    {
        return x1 - ((x1-x0)/(func(x1)-func(x0))) * func(x1);
    };

    assert( function(start)*function(end)<0 );

    // new x-axis intersection point
    float x = sec(function, start, end);

    if ( end-start<MIN )
        return x;

    // almost zero
    if( function(x)==0 || -MIN<function(x) && function(x)<MIN )
        return x;

    // do recursivly until the end
    if( function(start) * function(x) < 0)
        x = regular_falsi(start, x);
    else if ( function(end) * function(x) < 0)
        x = regular_falsi(x, end);

    return x;
}
```