Sorbonne Université

Numerical Algorithms

Practical 3-4

Méline Trochon
Anatole Vercelloni

Teacher : Stef Graillat

Janvier 2023

Table des matières

1 Exercise 12

1. 2.

```
function x_min = golden_section(f, a, b, tol, it_max)
    to = (sqrt(5) - 1) / 2;
    x1 = a + (1 - to) * (b - a);
    x2 = a + to * (b - a);
    f1 = f(x1);
    f2 = f(x2);
    for it = 1:it_max
        if f1 > f2
            a = x1;
            x1 = x2;
            f1 = f2;
            x2 = a + to * (b - a);
            f2 = f(x2);
        else
            b = x2;
            x2 = x1;
            f2 = f1;
            x1 = a + (1 - to) * (b - a);
            f1 = f(x1);
        end
        if (b - a) < tol
            break
        end
    end
    x_min = (x1 + x2) / 2;
end
```

```
function x_min = newton(f, x0, tol, it_max)
    val = x0;
    fprim = diff(f);
    fsecond = diff(fprim);
    for it = 1:it_max
        prec = val;
        val = val - eval(fprim(val)/fsecond(val));
        if abs(prec - val) < tol
            break
        end
    end
    x_min = val;
end
```

FIGURE 1 – Implementation of the golden section and the Newton method in 1D

```
>> golden_section(f1, 0, pi/2, 1e-5, 100)

ans =

   4.6816e-06

>> golden_section(f2, -1, 1, 1e-5, 100)

ans =

  -1.4072e-06

>> golden_section(f3, 1/2, 4, 1e-5, 100)

ans =

    1.0000

>> golden_section(f4, -1, 1, 1e-5, 100)

ans =

  -1.4072e-06
```

```
>> newton(f1, 1, 1e-5, 100)

ans =

    1.5708

>> newton(f2, 0.5, 1e-5, 100)

ans =

   4.4708e+17

>> newton(f3, 2, 1e-5, 100)

ans =

   2.5353e+30

>> newton(f4, 0.5, 1e-5, 100)

ans =

   NaN
```

FIGURE 2 – Test of the algorithms on given examples

We can see that the golden section seems to give good results compare to fminbnd. But the Newton method gives different results each time we change x0, because the interval of the function is bigger than the ball in which Newton method converges.

2 Exercise 13

1. $f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$
We can use matlab to compute the gradient and the Hessien, we get :

```
>> gradient(f, x)

ans(x1, x2) =

2*x1 - 2*x1*(- 100*x1^2 + 100*x2) - 200*x1*(- x1^2 + x2) - 2
                                         - 200*x1^2 + 200*x2

>> hessian(f)

ans(x1, x2) =

[1200*x1^2 - 400*x2 + 2, -400*x1]
[                -400*x1,     200]
```

2. Again, we use matlab to check if $g(x^*) = [00]^T$ and that the eigenvalues of the hessian are non negative, which means that H is definite positive and then that $x^*$is a minimum.

```
>> g(1,1)

ans =

0
0

>> vpa(eig(H(1,1)))

ans =

0.39936076748763304521271374857l2
1001.6006392325123669547872862514
```

3 Exercise 14

1.2.

```
function x_min = gradient_method(f, alpha, x1, x2, tol, it_max)
    grad = gradient(f);
    x = [x1
        x2];
    for it = 1:it_max
        prec = x;
        x = eval(x - alpha * grad(x(1), x(2)));
        if norm(prec - x) < tol
            break
        end
    end
    x_min = x;
end
```

```
>> gradient_method(f, 1, -1, -1, 1e-5, 100)

ans =

   1.0e+47 *

   -0.0000
   -5.1538

>> gradient_method(f, 0.1, -1, -1, 1e-5, 100)

ans =

   1.0e-04 *

   -0.3484
   -0.0000

>> gradient_method(f, 0.5, -1, -1, 1e-5, 100)

ans =

     0
    -1

>> gradient_method(rose, 0.001, -1, 1.2, 1e-5, 100)

ans =

   -0.9976
    1.0033
```

Figure 3 – gradient method and example for the Rosenbrook function

We can see that if the alpha is too big, then the gradient_method doesn't converge and goes to infinity.

3.

```
function x_min = step_gradient_method(f, x1, x2, tol, it_max)
    grad = gradient(f);
    x = [x1
        x2];
    for it = 1:it_max
        prec = x;
        alpha = wolfe_method(f, -grad(x(1), x(2)), x, 0, Inf, 1, 0.1, 0.9);
        x = eval(x - alpha * grad(x(1), x(2)));
        if norm(prec - x) < tol
            break
        end
    end
    x_min = x;
end
```

```
function a_min = wolfe_method(f, d, x, tg, td, t, m1, m2)
    syms z;
    g(z) = f(x(1) + z * d(1), x(2) + z * d(2));
    dg = gradient(g);
    for it=1:100
        if (g(t) <= g(0) + m1 * t * dg(0))
            if (dg(t) >= m2 * dg(0))
                a_min = t;
                return
            end
            tg = t;
            if td == Inf
                t = 10 * tg;
            else
                t = (td + tg) / 2;
            end
            continue
        end
        if (g(t) > g(0) + m1 * t * dg(0))
            td = t;
            t = (td + tg) / 2;
        end
    end
    disp("wolfe did not converges")
end
```

Figure 4 – Implementation of the step gradient method

```
>> step_gradient_method(f, -1, 1.2, 1e-5, 100)

ans =

     0
     0

>> rose(x1, x2) = 100*(x2 - x1^2)^2 + (1 - x1)^2

rose(x1, x2) =

(x1 - 1)^2 + 100*(- x1^2 + x2)^2

>> step_gradient_method(rose, -1, -2, 1e-5, 100)

ans =

    0.7592
    0.5749
```

Figure 5 – Results of the step gradient method