Numerical Algorithms (MU4IN910)

Practical 3

Assia MASTOR

February 27, 2024

Exercise 12

1- Golden search section algorithm :

```
function [xmin, fmin] = goldenSectionSearch(func, a, b)
    % Initialisation
    tol=1e-6;
    maxIter = 100;
    rho = 0.618; % Ratio de la section dorée
    iter = 0;

    % Étape 1: Initialisation des points et des valeurs de la fonction
    x1 = a + (1 - rho) * (b - a);
    x2 = a + rho * (b - a);
    f1 = func(x1);
    f2 = func(x2);

    % Boucle principale
    while abs(b - a) > tol && iter < maxIter
        if f1 < f2
            % Étape 2: Réduire l'intervalle en déplaçant le point b
            b = x2;
            x2 = x1;
            f2 = f1;
            x1 = a + (1 - rho) * (b - a);
            f1 = func(x1);
        else
            % Étape 3: Réduire l'intervalle en déplaçant le point a
            a = x1;
            x1 = x2;
            f1 = f2;
            x2 = a + rho * (b - a);
            f2 = func(x2);
        end
        iter = iter + 1;
    end

    % Sélectionner le minimum trouvé
    if f1 < f2
        xmin = x1;
        fmin = f1;
    else
        xmin = x2;
        fmin = f2;
    end
end
```

2- Newton's method:

```
function [x, iter] = newtonsMethod(func, initial_guess, tol, maxIter)
    % Initialisation
    x = initial_guess;
    iter = 0;

    % Boucle principale
    while abs(func(x)) > tol && iter < maxIter
        % Calcul de la dérivée de la fonction au point actuel
        df = (func(x + tol) - func(x)) / tol;

        % Mise à jour de x selon la formule de la méthode de Newton
        x = x - func(x) / df;

        % Incrémentation du nombre d'itérations
        iter = iter + 1;
    end
end
```

3- Tests

Initial guess = pi/2

```
Test pour f(x) = sin(x) sur [0, pi/2]:
Minimum trouvé à x = 3.2295e-07 avec f(x) = 3.2295e-07
Minimum trouvé par fminbnd: x = 6.4177e-05, f(x) = 6.4177e-05
Minimum trouvé à x avec la méthode de Newton = -0.000000 après 3 itérations.
```

Initial guess = 0

```
Test pour f(x) = (arctan(x))^2 sur [-1, 1]:
Minimum trouvé à x = -4.8658e-07 avec f(x) = 2.3676e-13
Minimum trouvé par fminbnd: x = 2.7756e-17, f(x) = 7.7037e-34
Minimum trouvé à x avec la méthode de Newton = 0.000000 après 0 itérations.
```

Initial guess = 0

```
Test pour f(x) = |x| sur [-1, 1]:
Minimum trouvé à x = -4.8658e-07 avec f(x) = 4.8658e-07
Minimum trouvé par fminbnd: x = 2.7756e-17, f(x) = 2.7756e-17
Minimum trouvé à x avec la méthode de Newton = 0.000000 après 0 itérations.
```

Initial guess =2

```
Test pour f(x) = |ln(x)| sur [1/2, 4]:
Minimum trouvé à x = 1 avec f(x) = 5.7703e-07
Minimum trouvé par fminbnd: x = 0.99999, f(x) = 9.4937e-06
Minimum trouvé à x avec la méthode de Newton = 1.000000 après 5 itérations.
```

Exercise 13

1-

```
syms x1 x2;
f = 100*(x2 - x1^2)^2 + (1 - x1)^2;

gradient_f = gradient(f, [x1, x2]);

% Hessian matrix
Hessian_f = hessian(f, [x1, x2]);


disp('Gradient of f(x):');
disp(gradient_f);

disp('Hessian matrix of f(x):');
disp(Hessian_f);
```

Résultat :

```
Gradient of f(x):
2*x1 - 400*x1*(- x1^2 + x2) - 2
            - 200*x1^2 + 200*x2

Hessian matrix of f(x):
[1200*x1^2 - 400*x2 + 2, -400*x1]
[               -400*x1,      200]
```

Le resultat donné est donc : Gradient de $f(x)$:

$$\begin{bmatrix} 2x_1 - 400x_1(-x_1^2 + x_2) - 2 \\ -200x_1^2 + 200x_2 \end{bmatrix}$$

Hessian matrix de $f(x)$:

$$\begin{bmatrix} 1200x_1^2 - 400x_2 + 2 & -400x_1 \\ -400x_1 & 200 \end{bmatrix}$$

2- Pour vérifier que $x^* = [1, 1]^T$ est un minimum local de f on vérifie que sa matrice Hessienne est définie positive.

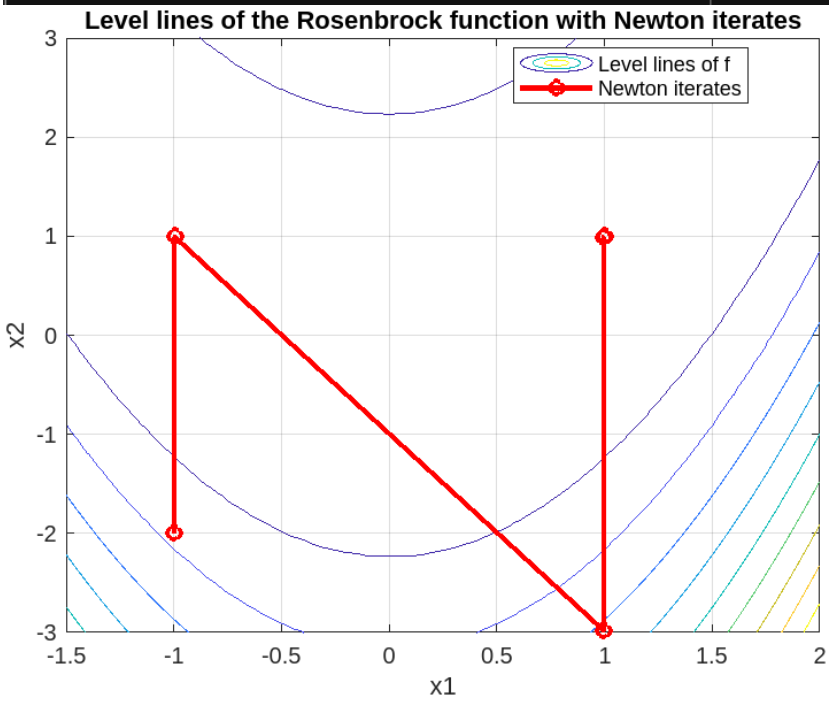
```
% x*
x_star = [1; 1];
gradient_at_x_star = double(subs(gradient_f, {x1, x2}, {x_star(1), x_star(2)}));
Hessian_at_x_star = double(subs(Hessian_f, {x1, x2}, {x_star(1), x_star(2)}));

is_positive_definite = all(eig(Hessian_at_x_star) > 0);

% Affichage du gradient à x*
disp('Gradient à x*:');
disp(gradient_at_x_star);

% Affichage de la matrice hessienne à x*
disp('Matrice hessienne à x*:');
disp(Hessian_at_x_star);


if is_positive_definite
    disp('La matrice hessienne est définie positive.');
    disp('Par conséquent, x* est un minimum local de la fonction de Rosenbrock.');
else
    disp('La matrice hessienne n''est pas définie positive.');
    disp('Par conséquent, x* pourrait ne pas être un minimum local de la fonction de Rosenbrock.
end
```

Command Window

```
Gradient à x*:
     0
     0

Matrice hessienne à x*:
   802  -400
  -400   200

La matrice hessienne est définie positive.
Par conséquent, x* est un minimum local de la fonction de Rosenbrock.
>>
```

3-

```
%fonction Rosenbrock
f = @(x1, x2) 100*(x2 - x1.^2).^2 + (1 - x1).^2;

gradient_f = @(x1, x2) [400*x1.^3 - 400*x1.*x2 + 2*x1 - 2; 200*(x2 - x1.^2)];
Hessian_f = @(x1, x2) [1200*x1.^2 - 400*x2 + 2, -400*x1; -400*x1, 200*ones(size(x1))];

%Point initial
x0 = [-1; -2];

x_iterates = zeros(2, 5);
x_iterates(:, 1) = x0;

% Méthode de Newton pour les 5 première itérations
for i = 1:4

    gradient_x = gradient_f(x_iterates(1, i), x_iterates(2, i));
    Hessian_x = Hessian_f(x_iterates(1, i), x_iterates(2, i));

   x_iterates(:, i+1) = x_iterates(:, i) - Hessian_x \ gradient_x;

end


figure;
fcontour(f, [-1.5, 2, -3, 3]);

hold on;

plot(x_iterates(1, :), x_iterates(2, :), 'ro-', 'LineWidth', 2);

xlabel('x1');
ylabel('x2');
title('Level lines of the Rosenbrock function with Newton iterates');
legend('Level lines of f', 'Newton iterates', 'Location', 'Best');
grid on;
hold off;
```



4-

```
ratio = zeros(4, 1);
prev_error_norm = norm(x - [1; 1]); % Erreur initiale avant la première itération

for iter = 1:5

    g_current = double(subs(g, [x1, x2], x'));
    H_current = double(subs(H, [x1, x2], x'));
    x = x - H_current\g_current; % mise à jour de x
    error_norm = norm(x - [1; 1]);
    disp(error_norm)

    if iter > 1
        ratio(iter-1) = error_norm / prev_error_norm^2;
    end
    prev_error_norm = error_norm;

end
disp('les ratios :')
disp(ratio)

if all(abs(ratio) > 0) % Vérifie si tous les éléments du ratio sont non nuls
    disp('Le taux de convergence est quadratique.');
else
    disp('Le taux de convergence n''est pas quadratique.');
end

hold off
```

```
     1.9967

     3.9779

     0.0098

    1.9421e-05

    2.3500e-13

les ratios :
     0.9978
     0.0006
     0.2007
     0.0006

Le taux de convergence est quadratique.
```

Exercise 14

1-

```octave
f = @(x) x(1)^2 + 2*x(2)^2;

% Gradient de la fonction
gradient_f = @(x) [2*x(1); 4*x(2)];

% Point initial
x0 = [-1; -1];

num_steps = 10;
alphas = [0.1, 0.3, 0.5, 1];

% Effectuer la descente de gradient pour chaque alpha
for alpha = alphas
    x = x0;
    fprintf('For alpha = %.1f:\n', alpha);
    for i = 1:num_steps
        gradient = gradient_f(x);
        x = x - alpha * gradient;
        fprintf('Step %d: x = (%.4f, %.4f), f(x) = %.4f\n', i, x(1), x(2), f(x));
    end
    fprintf('\n');
end
```

```
For alpha = 0.1:
Step 1: x = (-0.8000, -0.6000), f(x) = 1.3600
Step 2: x = (-0.6400, -0.3600), f(x) = 0.6688
Step 3: x = (-0.5120, -0.2160), f(x) = 0.3555
Step 4: x = (-0.4096, -0.1296), f(x) = 0.2014
Step 5: x = (-0.3277, -0.0778), f(x) = 0.1195
Step 6: x = (-0.2621, -0.0467), f(x) = 0.0731
Step 7: x = (-0.2097, -0.0280), f(x) = 0.0455
Step 8: x = (-0.1678, -0.0168), f(x) = 0.0287
Step 9: x = (-0.1342, -0.0101), f(x) = 0.0182
Step 10: x = (-0.1074, -0.0060), f(x) = 0.0116

For alpha = 0.3:
Step 1: x = (-0.4000, 0.2000), f(x) = 0.2400
Step 2: x = (-0.1600, -0.0400), f(x) = 0.0288
Step 3: x = (-0.0640, 0.0080), f(x) = 0.0042
Step 4: x = (-0.0256, -0.0016), f(x) = 0.0007
Step 5: x = (-0.0102, 0.0003), f(x) = 0.0001
Step 6: x = (-0.0041, -0.0001), f(x) = 0.0000
Step 7: x = (-0.0016, 0.0000), f(x) = 0.0000
Step 8: x = (-0.0007, -0.0000), f(x) = 0.0000
Step 9: x = (-0.0003, 0.0000), f(x) = 0.0000
Step 10: x = (-0.0001, -0.0000), f(x) = 0.0000

For alpha = 0.5:
Step 1: x = (0.0000, 1.0000), f(x) = 2.0000
Step 2: x = (0.0000, -1.0000), f(x) = 2.0000
Step 3: x = (0.0000, 1.0000), f(x) = 2.0000
Step 4: x = (0.0000, -1.0000), f(x) = 2.0000
Step 5: x = (0.0000, 1.0000), f(x) = 2.0000
Step 6: x = (0.0000, -1.0000), f(x) = 2.0000
Step 7: x = (0.0000, 1.0000), f(x) = 2.0000
Step 8: x = (0.0000, -1.0000), f(x) = 2.0000
Step 9: x = (0.0000, 1.0000), f(x) = 2.0000
Step 10: x = (0.0000, -1.0000), f(x) = 2.0000

For alpha = 1.0:
Step 1: x = (1.0000, 3.0000), f(x) = 19.0000
Step 2: x = (-1.0000, -9.0000), f(x) = 163.0000
Step 3: x = (1.0000, 27.0000), f(x) = 1459.0000
Step 4: x = (-1.0000, -81.0000), f(x) = 13123.0000
Step 5: x = (1.0000, 243.0000), f(x) = 118099.0000
Step 6: x = (-1.0000, -729.0000), f(x) = 1062883.0000
Step 7: x = (1.0000, 2187.0000), f(x) = 9565939.0000
Step 8: x = (-1.0000, -6561.0000), f(x) = 86093443.0000
Step 9: x = (1.0000, 19683.0000), f(x) = 774840979.0000
Step 10: x = (-1.0000, -59049.0000), f(x) = 6973568803.0000
```

On peut voir pour $\alpha = 0.1$ que les itérations descendent lentement vers le minimum, il faut beaucoup d'itérations avant d'atteindre le minimum souhaité. On

en déduit alors que le pas est trop petit malgré la convergence. Pour $\alpha = 0.3$, les itérations convergent rapidement vers le minimum. On a un pas modéré qui donne alors une convergence rapide et efficace. Lorsque l'on a $\alpha = 0.5$, les itérations oscillent entre 2 points sans convergence vers le minimum, il y a divergence car le pas est trop grand. Finalement, pour $\alpha = 1.0$, les itérations divergent rapidement vers l'infini, le pas est beaucoup trop grand, il n'y a donc pas de convergence.

On peut donc conclure qu'un α trop petit entraîne une convergence lente, tandis qu'un α trop grand peut entraîner une divergence de l'algorithme.

2- Test avec la fonction de Rosenbrock et α= 0.001

```
f = @(x) x(1)^2 + 2*(x(2)-x(1)^2)^2;

% Gradient de la fonction
gradient_f = @(x) [2*x(1) - 4*x(1)*(x(2)-x(1)^2); 2*(x(2)-x(1)^2)];

% Point initial
x0 = [-1; 1.2];


num_steps = 10;
alpha = 0.001;

% Effectuer la descente de gradient
x = x0;
fprintf('Starting point: x = (%.4f, %.4f), f(x) = %.4f\n', x(1), x(2), f(x));
for i = 1:num_steps
    gradient = gradient_f(x);
    x = x - alpha * gradient;
    fprintf('Step %d: x = (%.4f, %.4f), f(x) = %.4f\n', i, x(1), x(2), f(x));
end
```

```
Starting point: x = (-1.0000, 1.2000), f(x) = 1.0800
Step 1: x = (-0.9988, 1.1996), f(x) = 1.0792
Step 2: x = (-0.9976, 1.1992), f(x) = 1.0784
Step 3: x = (-0.9964, 1.1988), f(x) = 1.0777
Step 4: x = (-0.9953, 1.1984), f(x) = 1.0769
Step 5: x = (-0.9941, 1.1980), f(x) = 1.0762
Step 6: x = (-0.9929, 1.1975), f(x) = 1.0755
Step 7: x = (-0.9918, 1.1971), f(x) = 1.0748
Step 8: x = (-0.9907, 1.1967), f(x) = 1.0741
Step 9: x = (-0.9895, 1.1963), f(x) = 1.0734
Step 10: x = (-0.9884, 1.1958), f(x) = 1.0728
>>
```

3- Méthode optimal step gradient avec la méthode de Wolfe et test sur la fonction de Rosenbrock

```
% Méthode de gradient avec pas optimal selon la méthode de Wolfe
function alpha = optimal_step_gradient_wolfe(x, f, gradient_f, m1, m2)
    alpha = 1;
    t0 = 0;
    t_max = inf;
    g0 = f(x);
    g0_prime = gradient_f(x)' * [-1; 0];

    while true
        g = f(x + alpha * (-gradient_f(x)));
        g_prime = gradient_f(x + alpha * (-gradient_f(x)))' * (-gradient_f(x));

        if g <= g0 + m1 * alpha * g0_prime && g_prime >= m2 * g0_prime
            break;  % Conditions de Wolfe satisfaites
        elseif g > g0 + m1 * alpha * g0_prime
            t_max = alpha;
            alpha = (t0 + t_max) / 2;
        else
            t0 = alpha;
            if t_max == inf
                alpha = 10 * t0;
            else
                alpha = (t0 + t_max) / 2;
            end
        end
    end
end
```

```
% Fonction Rosenbrock
f = @(x) x(1)^2 + 2*(x(2)-x(1)^2)^2;

% Gradient de Rosenbrock
gradient_f = @(x) [2*x(1) - 4*x(1)*(x(2)-x(1)^2); 2*(x(2)-x(1)^2)];

% paramètres de Wolfe
m1 = 0.1;
m2 = 0.9;

x0 = [-1; 1.2];
num_steps = 10;

% Effectuer la descente de gradient avec une taille de pas optimale
x = x0;
fprintf('Starting point: x = (%.4f, %.4f), f(x) = %.4f\n', x(1), x(2), f(x));
for i = 1:num_steps
    alpha = optimal_step_gradient_wolfe(x, f, gradient_f, m1, m2);
    x = x - alpha * gradient_f(x);
    fprintf('Step %d: x = (%.4f, %.4f), f(x) = %.4f, alpha = %.4f\n', i, x(1), x(2), f(x), alpha);
end
```

```
Starting point: x = (-1.0000, 1.2000), f(x) = 1.0800
Step 1: x = (0.8750, 0.5750), f(x) = 0.8383, alpha = 1.5625
Step 2: x = (0.5729, 0.6227), f(x) = 0.5016, alpha = 0.1250
Step 3: x = (0.1020, 0.0337), f(x) = 0.0115, alpha = 1.0000
Step 4: x = (0.0047, 0.0104), f(x) = 0.0002, alpha = 0.5000
Step 5: x = (0.0046, 0.0101), f(x) = 0.0002, alpha = 0.0156
Step 6: x = (0.0046, 0.0101), f(x) = 0.0002, alpha = 0.0000
Step 7: x = (0.0046, 0.0101), f(x) = 0.0002, alpha = 0.0000
Step 8: x = (0.0046, 0.0101), f(x) = 0.0002, alpha = 0.0000
Step 9: x = (0.0046, 0.0101), f(x) = 0.0002, alpha = 0.0000
Step 10: x = (0.0046, 0.0101), f(x) = 0.0002, alpha = 0.0000
>>
```

Exercise 15 1) Algorithme de Nelder-Meade

```
function [xmin,fmin] = nelder_mead(f, x_start, max_iter, tol)
    N = size(x_start, 2);
    alpha = 1; gamma = 2; rho = -0.5; sigma = 0.5;
    x = [x_start; x_start + diag(0.1*ones(1,N))]; % Initialisation du simplex initial

    for i = 1:max_iter
        % Évaluation de la fonction objectif aux points du simplex
        f_val = zeros(size(x, 1), 1);
        for i = 1:size(x, 1)
            f_val(i) = f(x(i, :));
        end
        [f_sort, idx] = sort(f_val);
        x = x(idx,:);

        % Calcul du centre de gravité du simplex
        x0 = mean(x(1:end-1,:), 1);

        % Calcul du point de réflexion
        xr = x0 + alpha*(x0 - x(end,:));

        % Mettre à jour le simplex
        if f(xr) < f_sort(end-1)
            xe = x0 + gamma*(x0 - x(end,:));
            if f(xe) < f_sort(1)
                x(end,:) = xe;
            else
                x(end,:) = xr;
            end
        else
            if f(xr) < f_sort(end)
                x(end,:) = xr;
            else
                xc = x0 + rho*(x0 - x(end,:));
                if f(xc) <= f_sort(end)
                    x(end,:) = xc;
                else
                    x(2:end,:) = x(1,:) + sigma*(x(2:end,:) - x(1,:));
                end
            end
        end
        if norm(x(1,:) - x0, Inf) < tol, break, end
    end
    xmin = x(1,:);
    fmin = f(xmin);
end
```

2- Test sur la fonction de Rosenbrock

```
% Test on the Rosenbrock function
f = @(x) (1 - x(1))^2 + 100*(x(2) - x(1)^2)^2;
x_start = [-1.2, 1];
max_iter = 200;
tol = 1e-6;
[xmin, fmin] = nelder_mead(f, x_start, max_iter, tol);
disp(['xmin = ', num2str(xmin), ', fmin = ', num2str(fmin)]);
```

```
xmin = 1           1, fmin = 1.8131e-13
```

3- Résultats avec la commande fminsearch :

```
Optimization terminated:
 the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-04
 and F(X) satisfies the convergence criteria using OPTIONS.TolFun of 1.000000e-04

xmin_fminsearch = 1          1, fmin_fminsearch = 8.1777e-10
```

On constate alors que les résultats sont quasiment équivalents avec xmin= 1 1 et un fmin quasiment égal à 0.