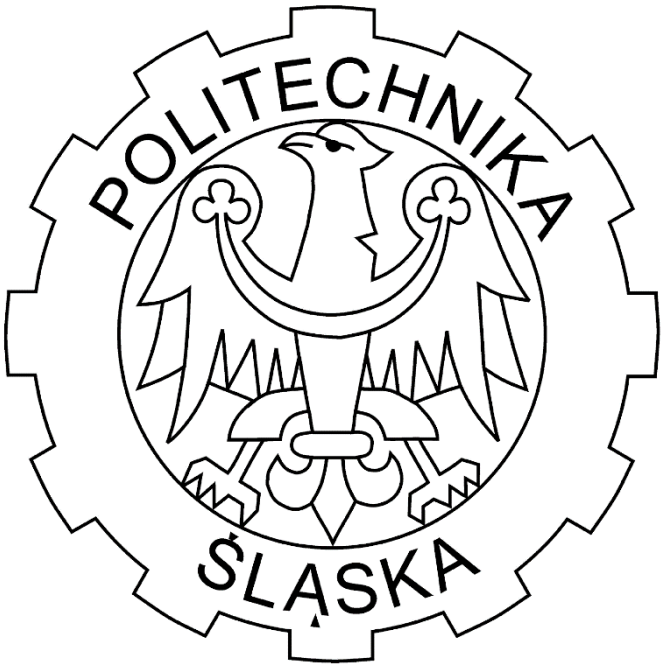
Advanced Optimization Methods

Report

Direct methods of unconstrained dynamic optimization



Authors:
Piotr Pawełko
Krzysztof Miler

Laboratory date:
26.05.2022

1. Problem statement

$$Min \leftarrow J = \sum_{n=0}^{N-1} (\mathrm{q}x^2 + \mathrm{rxu} + \mathrm{h}u^2)$$

$$x_{n+1} = ax_n + bu_n$$

Assumed parameters:

- u

Parameters to determine:

- x
- p
- b

Section 4 parameters:

- a = 1
- b = 0.5
- q = 0.5
- r = 0 or 0.5
- h = 1
- x_0 = 1
- N = 6
- K = 25

2. Essential calculation for direct and conjugated gradient methods

$$H_n = J(x,u) + p_{n+1}x_{n+1}$$

$$H_n = (\mathrm{q}x^2 + \mathrm{rxu} + \mathrm{h}u^2) + p_{n+1}(ax_n + bu_n)$$

$$p_n = \frac{\partial H_n}{\partial x_n} = 2qx + ru + 2p_{n+1}ax = 2x(q + ap_{n+1}) + ru$$

$$b_n = \frac{\partial H_n}{\partial u_n} = 2hu + rx + 2p_{n+1}bu = 2u(h + bp_{n+1}) + rx$$

Calculation of u for direct gradient:

$$u_n^{(i+1)} = u_n^{(i)} - tb_n^{(i)}$$

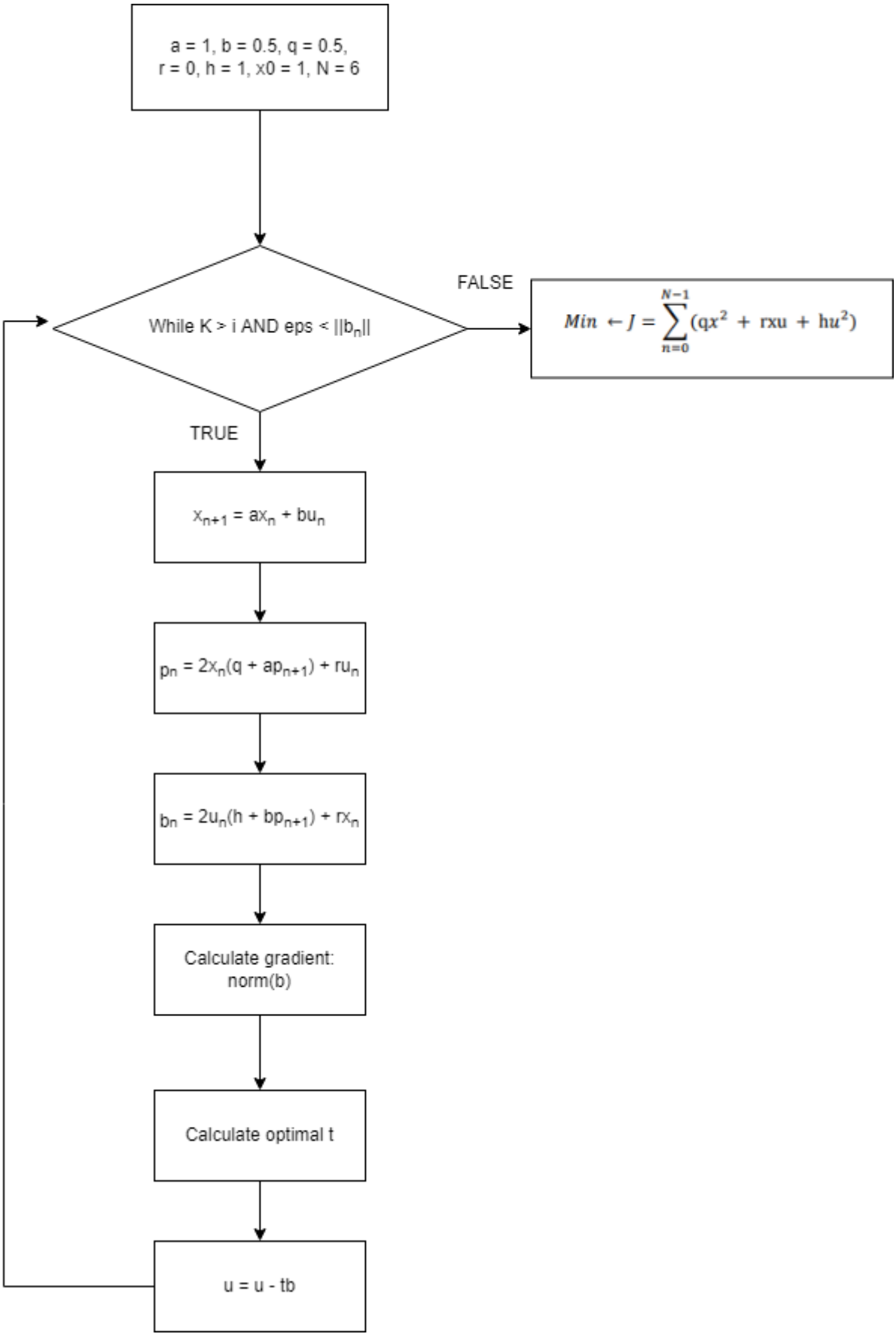
Calculation of u for conjugated gradient:

$$d_n^{(i)} = -b_n^{(i)} + c^{(i)}d_n^{(i-1)}$$

$$c^{(i)} = \frac{\left\|b^{(i)}\right\|^2}{\left\|b^{(i-1)}\right\|^2}$$

$$u_n^{(i+1)} = u_n^{(i)} - td_n^{(i)}$$

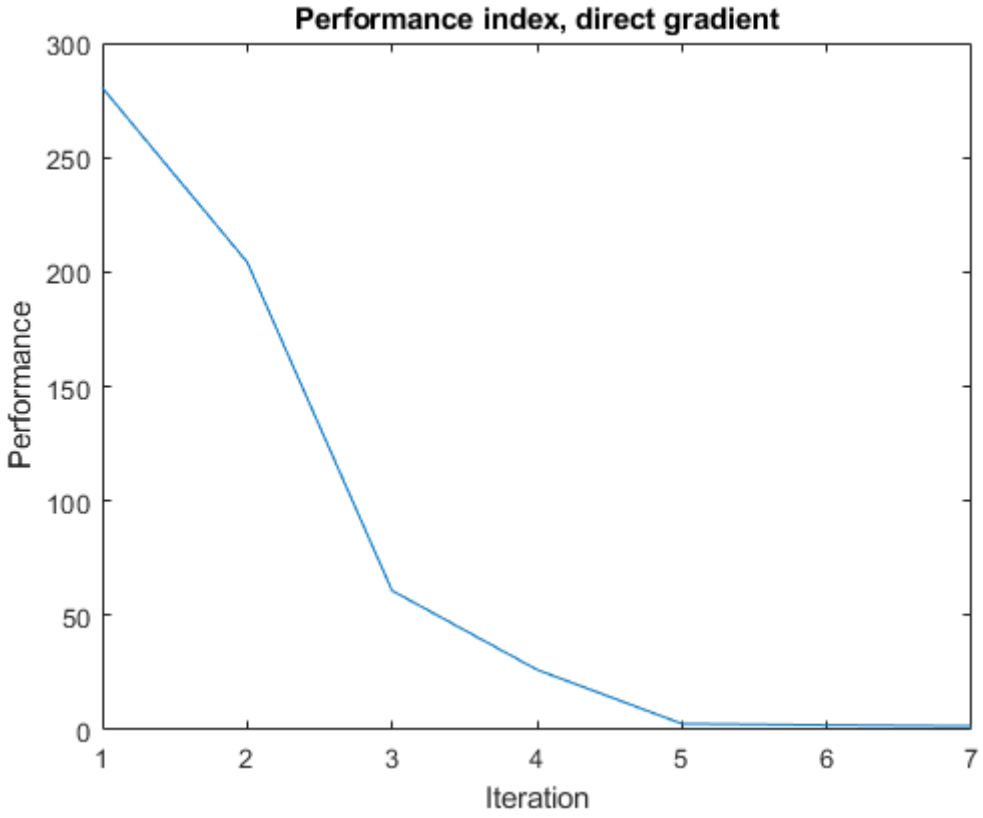
3. Conjugated gradient method



Tests:

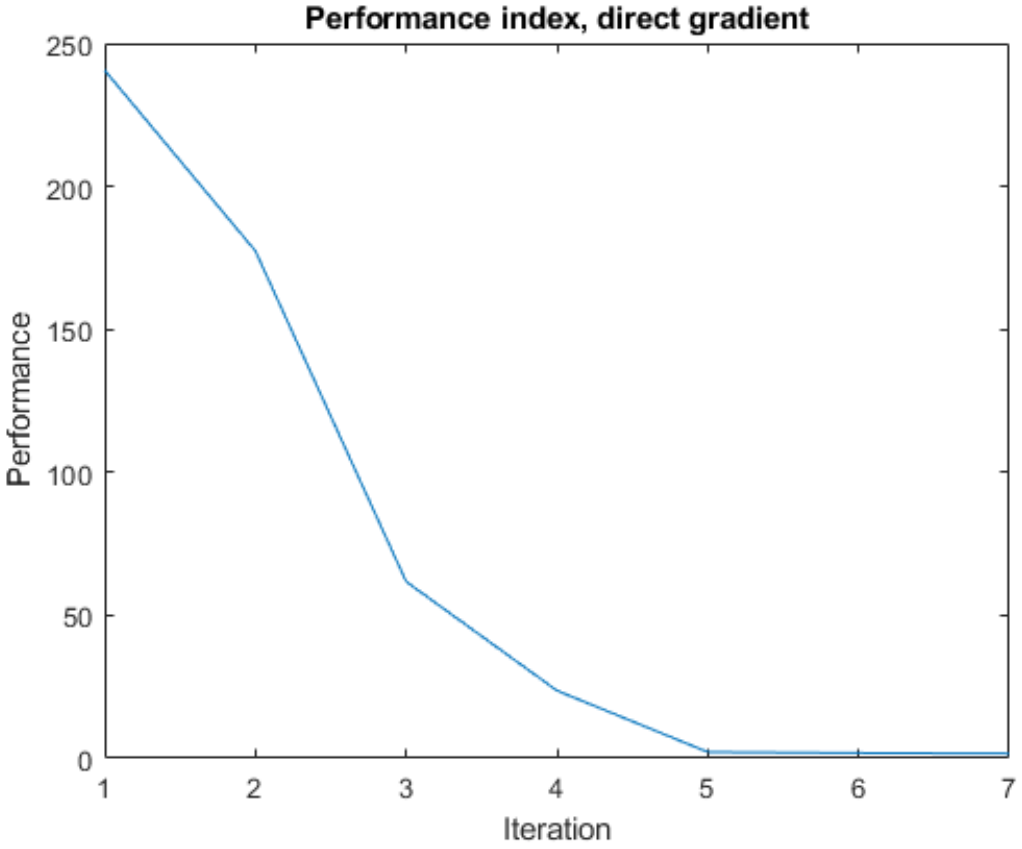
N = 10

Iteration	u	x	J
1	1, 2, 3, 1, 2, 4, 5, 2, 0	1, 1.5, 2.5, 4, 4.5, 5.5, 7.5, 10, 11, 11	280.5
7	-0.5828, -0.4141, -0.3073, -0.2115, -0.1501, -0.097, -0.0579, -0.0153, 0	1, 0.7136, 0.5110, 0.3603, 0.2438, 0.1587, 0.1044, 0.0728, 0.0573, 0.0529, 0.042	1.6872



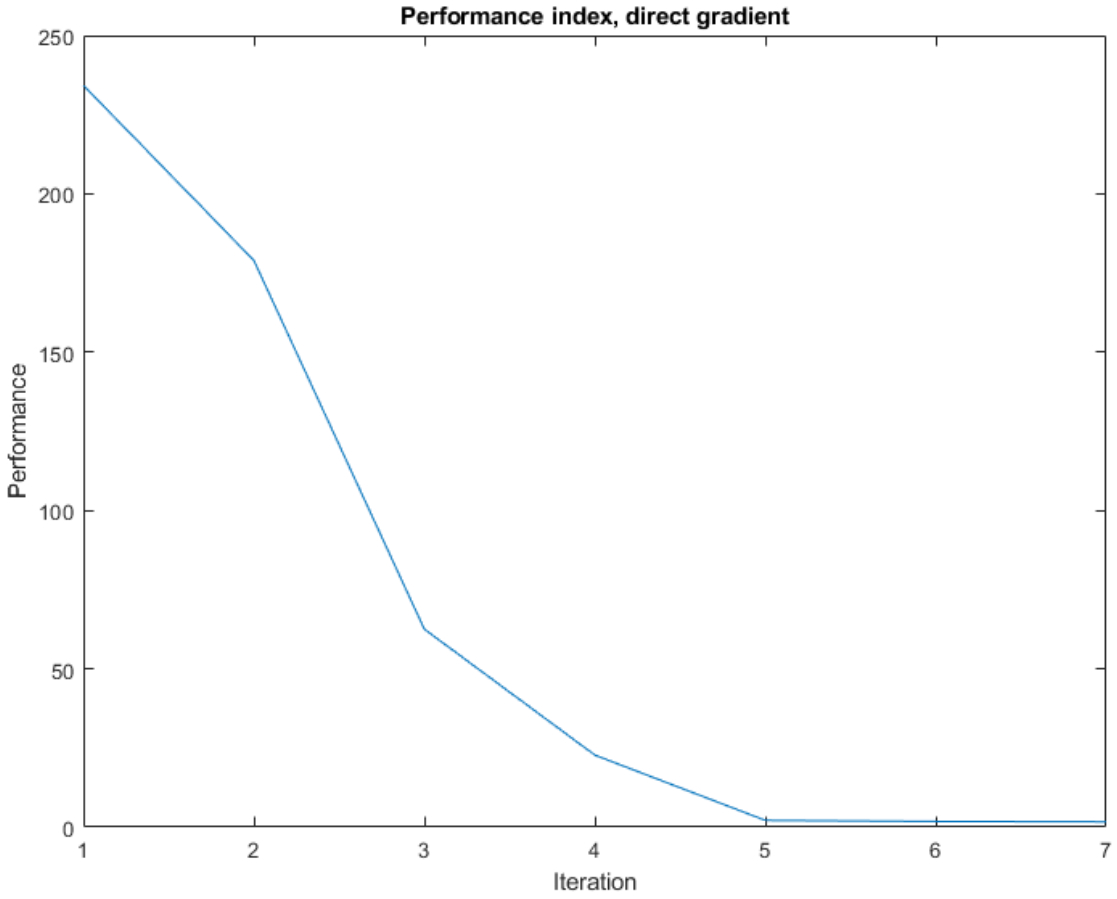
N = 9

Iteration	u	x	J
1	1, 2, 2, 1, 2, 4, 4, 2, 3, 3	1, 1.5, 2.5, 3.5, 4, 5, 7, 9, 10, 11.5	280.5
7	-0.5727, -0.4053, -0.3014, -0.2330, -0.1701, -0.1088, -0.0632, -0.0309, -0.0089, -0.0218	1, 0.7085, 0.5015, 0.3478, 0.2420, 0.1670, 0.1185, 0.0895, 0.0818, 0.0818	1.6809



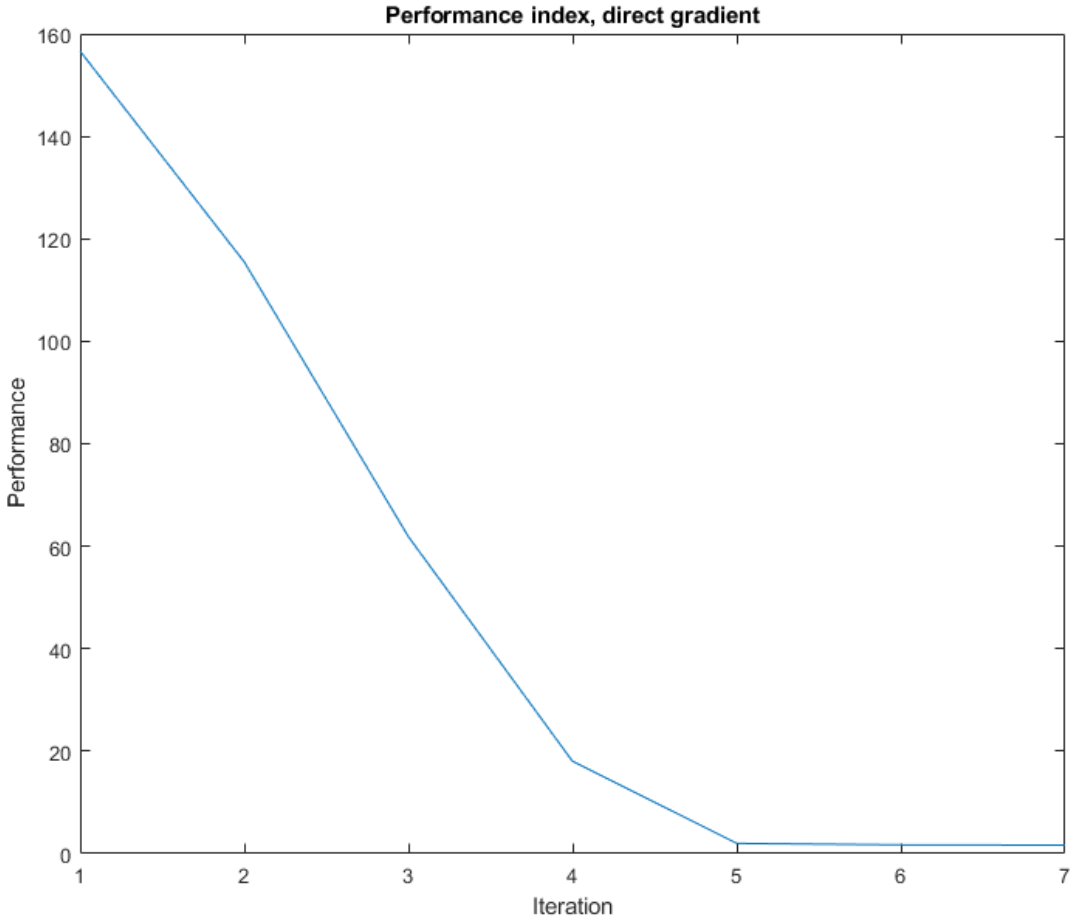
N = 8

Iteration	u	x	J
1	1, 2, 6, 2, 2, 4, 1, 2	1, 1.5, 2.5, 5.5, 6.5, 7.5, 9.5, 10, 11	234.25
7	-0.5985, -0.4106, -0.2699, -0.2047, -0.1433, -0.0778, -0.0399, -0.039	1, 0.7007, 0.4955, 0.3605, 0.2582, 0.1865, 0.1476, 0.1277, 0.1257	1.6827



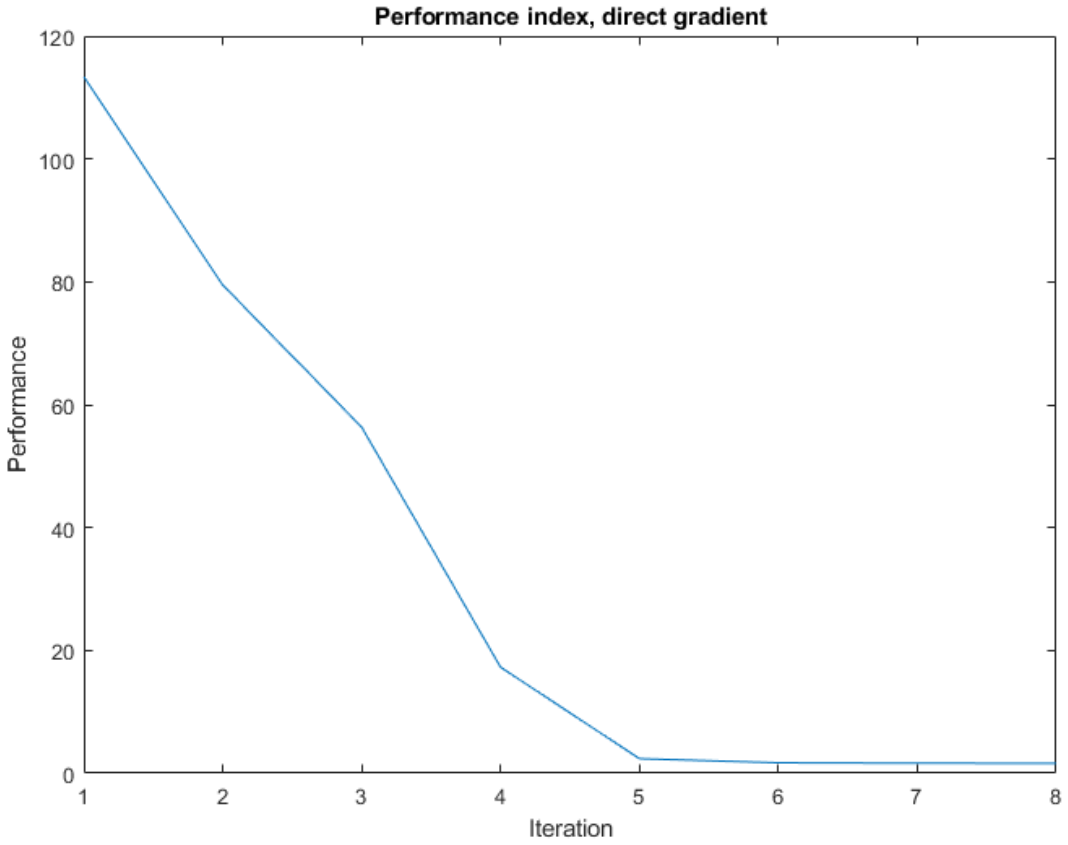
N = 7

Iteration	u	x	J
1	1, 2, 2, 3, 5, 4, 1	1, 1.5, 2.5, 3.5, 5, 7.5, 9.5, 10	156.6250
7	-0.5916, -0.4168, -0.2888, -0.1907, -0.1137, -0.0513, -0.0068	1, 0.7042, 0.4958, 0.3514, 0.2561, 0.1992, 0.1736, 0.1702,	1.6594



N = 6

Iteration	u	x	J
1	1, 3, 2, 3, 2, 6	1, 1.5, 2.5, 3.5, 5, 7.5, 9.5, 10	113.3750
8	-0.5528, -0.3819, -0.2634, -0.1687, -0.0842, -0.0139,	1, 0.7236, 0.5326, 0.4009, 0.3166, 0.2745, 0.2815	1.6284



4. Conclusion

From the considerations on the direct gradient method, it shows that the method of creating the search direction, which is always minus gradient, causes a number of unfavorable effects, e.g. reduction of the base dimensionality, difficulties in finding the correct gradient direction. Moreover, this method is characterized by the fact that each subsequent iteration starts with zero information about the examined surface. In order to counteract this effect, the conjugate gradient method was created, in which new directions are created conjugate to all previous directions used. That makes the search more efficient.

5. Code

```
A = 1;
B = 0.5;
Q = 0.5;
R = 0; % 0 or 0.5
H = 1;
x0 = 1;
N = 6;
K = 25;
eps = 0.25;

u = zeros(K, N);
u(1, :) = [1, 3, 2, 3, 2, 6];
x = zeros(K, N + 1);
x(:, 1) = x0;
p = zeros(1, N + 1);
b = zeros(1, N);
J = zeros(K, 1);
t = 0.01;

for iter = 1 : K
    for i = 2 : size(x, 2)
        x(iter, i) = A * x(iter, i - 1) + B * u(iter, i - 1);
    end
    for i = N : -1 : 1
        p(i) = 2 * x(iter,i) * Q + A * p(i + 1) + R * u(iter,i);
    end
    b_old = b;
    for i = 1 : length(b)
        b(i) = 2 * u(iter, i) * H  + B * p(i + 1) + R * x(iter, i);
    end
    b_norm(i) = sqrt(b*b');
    b_norm_old(i) = sqrt(b_old*b_old');
    if iter == 1
        d = -b;
    else
        d = -b + b_norm(i) ^ 2 / b_norm_old(i) ^ 2 * d;
    end
    J = sum(Q * x(iter, 1 : end - 1).^2 + R*x(iter, end-1)*u(iter, :) + H *
u(iter, :).^2);
    J_i(iter) = J;
    if norm(d) < eps
        break;
    else
        syms t;
        x_t = calx(A, B, x0, u(iter, :), -d, t);
        J_t = simplify(calJ(Q,H,x_t,u(iter, :),R, -d,t));
        deriv = diff(J_t);
        T = solve(deriv == 0, t);
        T = double(T);
        u = u + T * d;
    end
end

plot(J_i(1 : iter));
xlabel('Iteration')
ylabel('Performance')
```

```
title('Performance index, direct gradient');

function x = calx(A,B,x0,u,b,T)
    N = size(u,2);
    x = sym(zeros(1,N));
    x(1) = sym(x0);
    for i=1:N-1
       x(i+1)=A*x(i)+B*(u(i)-T*b(i));
    end
end
function J=calJ(Q, H, x,u,r, b,T)
    N = length(x);
    J = sym(zeros(1,N));
    for i=1:N
       J(i) = (Q*x(i)^2 + r*x(i)*(u(i)-T*b(i)) + H*(u(i)-T*b(i))^2);
    end
    J = sum(J);
end
```