Online Optimization

Sebastian Gracia, Shyla Kupis, Tilly Erwin

April 2020

1 Abstract

Online convex programming is a form of convex programming in which a convex set is known ahead of time. At each iteration a point is selected from the convex set then a repeated minimization problem is solved at the point. At each step a corresponding cost function is evaluated. The cost function however is not known ahead of time. This paper explores an algorithm to minimize possible costs incurred in an online convex problem. It is called the Greedy Projection Algorithm. In this paper, we introduce the background and properties of this algorithm, then apply it by hand and numerically via MATLAB code to a set of Online optimization problems, then discuss the results of each.

2 Introduction

Online optimization is a formulation of a game played by an online player iteratively making decisions[2]. Each decision the player makes will incur a cost. The cost of each decision is not known beforehand. Instead, a point is evaluated in a convex set to observe the cost function. Thus each step will induce a cost and the objective is to minimize the total induced cost. An online optimization algorithm is a selection routine that attempts to obtain the lowest cost. This paper examines one such algorithm known as the Greedy Projection Algorithm (GPA) discussed in Zinkevich 2003 [3]. The sections of this paper will discuss the convergence, then then results and performance of the algorithm are analyzed.

There are many applications for this algorithm, for example, Bansel et al [1] applies the Greedy Projection algorithm to the problem of routing virtual circuits, where we see that the "Regret" of the problem is $O(\sqrt{T})$ where T is the number of cost functions. This low cost is what makes this algorithm so useful for the problem or routing virtual circuits. Other examples for the use of GPA might be production of a new product, the opening of a new store or restaurant, or even perhaps scheduling of classes before enrollment is fully known.

3 Problem Formulation

To simplify our calculations in the problems we worked by hand we have fixed the feasible set $\mathbf{F}$, such that $\mathbf{F} = \{\mathbf{x} \in \mathbb{R}^2 : 0 \leq x_1, x_2 \leq 1\}$. We will also fix our initial starting point $\mathbf{x^1} \in \mathbf{F}$ to be $\mathbf{x^1} = (0, 0)$. This is also done to simplify our calculations since any arbitrary $\mathbf{x} \in \mathbf{F}$ can be chosen for the initial starting point. Our numeric algorithm however is not constrained by these modifications and can handle any feasible set or starting point. For the purposes of this paper, we worked on three classes of minimization problems, linear, quadratic, and mixed. More details on the problem setup will be shown later in the example and numerical sections.

4 GPA Algorithm

Simply put, the GPA starts with a gradient descent, allowing it to move in the direction of the previous cost function [1]. It then either uses that point as its next point, or, if that point is outside the feasible region, uses the closest point to that point that is within the feasible region. It does this by projecting back into the feasible set and then using that point as the next x. A more technical explanation of the algorithm can be found in section 6.

4.1 Basic Properties

Some basic properties of the GPA are that it is an online optimization problem that works with convex sets and convex cost functions. Like any online optimization problem, only the current cost function is known. The GPA has an excellent convergence rate, as it discussed in detail in the next section. More details on the properties are given in section 6.

4.2 Performance/convergence

One way to measure the convergence of this algorithm is in terms of "Regret" or "The difference between the cost of the algorithm and the static optimal solution for cost functions g" [1]. When measured this way, the complexity can be calculated by:

$$\sqrt{T+1}(\frac{1}{2}max_{t,f\in F}||\nabla g^t(x)||^2) \tag{1}$$

Or in the notation of Zinkevich,

$$R_G(T) \leq \frac{||F||^2\sqrt{T}}{2} + (\sqrt{T} - \frac{1}{1})||\nabla c||^2 \tag{2}$$

The full proof of each of these can be found in Bansel et al and Zinkevich respectively , however the summary of the proof from Zinkevich [4] is as follows: First, we start with a set of linear cost functions $\{c_1, c_2, ..., c_n\}$. Then the

algorithm is computed using these cost functions and $x_1, ..., x_n$ is found. Next, it is shown that multiplying the linear cost functions by x would not change the behavior of the algorithm. This is done by first establishing the definition of convexity for c, that is

$$c^t(x) \geq (\nabla c^t(x^t))(x - x^t) + c^t(x^t)$$

Then, x^* is fixed as an optimal vector, and the result

$$c^t(x^t) - c^t(x^*) \leq g^t \cdot x^t - g^t \cdot x^*$$

is derived from taking $c^t(x^*)$. This establishes that the linear cost functions can be used to establish the regret. Next, $y^{t+1} = x^t - \eta_t(g^t)$ is defined, x^* is subtracted from it and then the whole thing is squared to yield

$$(y^{t+1} - x^*)^2 = (x^t - x^*)^2 - 2\eta_t(x^t - x^*) \cdot g^t + \eta_t^2 ||g^t||^2$$

Next they apply the properties $(y-x)^2 \leq (P(y)-x)^2$ and $||g^t|| \leq ||\nabla c||$ to yield the result

$$(x^t - x^*) \cdot g^t \leq \frac{1}{2\eta_t}((x^t - x^*)^2 - (x^{t+1} - x^*)^2) + \frac{\eta_t}{2}||\nabla c||^2$$

Then through algebra and the above equation they get the result

$$R_G(T) = \sum_{t=1}^{T}(x^t - x^*) \cdot g^t \leq ||F||^2 \frac{1}{2\eta_T} + \frac{||\nabla c||^2}{2} \sum_{t=1}^{T} \eta_t$$

The last major step is to set $\eta_t = \frac{1}{\sqrt{t}}$, yielding

$$\sum_{t=1}^{T} \eta_t \leq 2\sqrt{T} - 1$$

Plugging this in gives the above bound for the Regret and ends the proof. This is considered to be a fairly low complexity, and as such this is considered a good algorithm for fairly large data. [1]

5 Example

In this section we will manually calculate two iterations of the GPA for three different cases of cost functions. A set of strictly linear cost functions, strictly quadratic, and finally a mixed set of cost functions.

5.1 Linear Program

Let $c^t = \{x_1 + x_2, x_1 - x_2\}, \nabla c^t = \{(1,1),(1,-1)\}$. As previously discussed $\mathbf{x^1} = (0,0)$ and $\eta_t = \{t^{-\frac{1}{2}}\}$. We can also write $c^t = (b^t)^T(x - a^t)$, where

$b^t = \{(1,1),(1,-1)\}$ and $a^t = \{(0,0),(0,0)\}$.

$$\begin{aligned}
\mathbf{x^{1+1}} &= P(\mathbf{x^1} - \eta_1 \nabla c^1(\mathbf{x^1})) \\
&= P((0,0) - 1(1,1)) \\
&= P(-1,-1) \\
&= (0,0) \\
\mathbf{x^{2+1}} &= P(\mathbf{x^2} - \eta_2 \nabla c^2(\mathbf{x^2})) \\
&= P((0,0) - \frac{1}{\sqrt{2}}(1,-1)) \\
&= P(\frac{-1}{\sqrt{2}}, \frac{1}{\sqrt{2}}) \\
&= (0, \frac{1}{\sqrt{2}})
\end{aligned}$$

5.2 Quadratic Program

Let $c^t = \{(\mathbf{x} - \mathbf{a^t})^\mathbf{T}\mathbf{Q^t}(\mathbf{x} - \mathbf{a^t}) + (\mathbf{b^t})^\mathbf{T}\mathbf{x}\}$, where $\mathbf{Q^t} = \{\begin{pmatrix} 1 & 2 \\ 2 & -1 \end{pmatrix}, \begin{pmatrix} -1 & 2 \\ 2 & 1 \end{pmatrix}\}$

$\mathbf{a^t} = \{(1,1),(-1,-1)\}$
$\mathbf{b^t} = \{(1,1),(-1,-1)\}$
$\nabla \mathbf{c^t} = \{\mathbf{2Q^t}(\mathbf{x} - \mathbf{a^t}) + \mathbf{b^t}\}$

$$\begin{aligned}
\mathbf{x^{1+1}} &= P(\mathbf{x^1} - \eta_1 \nabla c^1(\mathbf{x^1})) \\
&= P((0,0) - 1(-5,-1)) \\
&= P(5,1) \\
&= (1,1) \\
\mathbf{x^{2+1}} &= P(\mathbf{x^2} - \eta_2 \nabla c^2(\mathbf{x^2})) \\
&= P((1,1) - \frac{1}{\sqrt{2}}(7,11)) \\
&= P(1 - \frac{7}{\sqrt{2}}, 1 - \frac{11}{\sqrt{2}}) \\
&= (0,0)
\end{aligned}$$

5.3 Mixed

Let $c^t = \{-x_1 + x_2, 2x_1^2 - 2x_2^2 + 2x_1x_2\},\ \nabla c^t = \{(-1,1), (4x_1 + 2x_2, 2x_1 - 4x_2)\}$

$$\begin{aligned}
\mathbf{x^{1+1}} &= P(\mathbf{x^1} - \eta_1 \nabla c^1(\mathbf{x^1})) \\
&= P((0,0) - 1(-1,1)) \\
&= P(1,-1) \\
&= (1,0) \\
\mathbf{x^{2+1}} &= P(\mathbf{x^2} - \eta_2 \nabla c^2(\mathbf{x^2})) \\
&= P((1,0) - \frac{1}{\sqrt{2}}(4,2)) \\
&= P(1 - \frac{4}{\sqrt{2}}, 1 - \frac{2}{\sqrt{2}}) \\
&= (0,0)
\end{aligned}$$

6 Implementation

The Greedy Projection Algorithm (GPA) was developed as a form of model prediction using the previous set of cost functions and their respective set of constraints. The theory behind the GPA involves online convex programming problems in which there is a convex feasible set F and an infinite sequence of convex cost functions $\{c^t\}_{t=1}^{\infty}$, such that $c^t : F \to \mathbb{R}$ for all $t \in \mathbb{N}$. Since we are using convex cost functions in a feasible convex set, each optimal solution minimizes the cost. We will use this fact to find local optimal solutions when our current cost function is unknown but we know our previous set of cost functions and their respective set of constraints. However, it is important to note that the GPA performs best when there is not a drastic difference between the outcome of each cost function.
Here we define the metric norm $d(x,y)$ as the 2-norm, $||x - y||_2$, which will be used in our projection function $P(y) = \min_{x \in F} d(x,y)$ for minimizing the distance between a point $y \in \mathbb{R}^n$ inside or outside of the feasible set F. There are some criteria to satisfy before implementing the GPA, which are as follows:

1. The feasible set is nonempty, bounded and closed.
2. Each $c^t(x)$ is differentiable for all $x \in F$ and $||\nabla c^t(x)|| \leq N \in \mathbb{R}$.
3. There exists $y \in \mathbb{R}^n$, such that a solution to the projection $P(y) = \min_{x \in F} d(x,y)$ exists.

Now, we are ready to present the Greedy Projection Algorithm.

1 Greedy Projection Algorithm: Choose $x^1 \in F$.
Initialize the cost of A until some stopping time T: $C_A = 0$, where
$C_A(T) = \sum_{t=1}^{T} c^t(x^t)$
Define $\eta_t = t^{-\frac{1}{2}}$ and c^t for all $t = 1, \ldots, N$.
$C_A = c^t(x^t) + C_A$.
$y = x^t - \eta^t \nabla c^t(x^t)$.
Update $x^{t+1} = \min_{x \in F} d(x, y)$, where $d(x, y) = ||x - y||_2$.
If $t > T$, STOP; otherwise $t = t + 1$.
Then, $C_x(T) = \min_{x \in F} C(T)$, where $C(T) = \sum_{t=1}^{T} c^t(x)$.
Compute the regret of the algorithm of Q until time T as
$R_A(T) = C_A(T) - C_x(T)$.
DONE!

7 Numerical Results

The numerical results presented highlight the performance of the GPA for (1) linear, (2) quadratic, and (3) mixed programs. For the first examples of (1) and (2), we ran the code using the examples demonstrated by hand above. The only constraints implemented were the lower and upper bounds of the constraints, which formed a square. The other examples were created by experimenting with parameters, such as the inequality constraints, lower and upper bounds, and coefficient matrix Q and coefficient vector q, and constants $a^t, b^t \in \mathbb{R}^{\not\leq}$.
In terms of uncertainty quantification, we examined the regret of the algorithm of matrix A for some user-specified stop time T:

$$R_A(T) = C_A(T) - C_x(T), \quad C_x(T) = \min_{x \in F} \sum_{t=1}^{T} c^t(x).$$

The regret of the algorithm can be thought of as a measure for comparing how the predicted optimal solution compares to the static feasible solution or the case when we are using the sum of all cost functions for prediction.

7.1 Linear Program

$$\text{Linear Cost Function: } \min_{x \in F} c^t = (b^t)^T (x - a^t) \text{ s.t. } A^t x \leq g^t,$$

where $F = \{x^i \in \mathbb{R}, i = 1, 2 | 0 \leq x^i \leq 1\}$.

Below are the feasible solutions at each iteration (i) for c^t for $t = 1, 2$ from the handwritten example problem and (ii) for c^t for $t = 1, \ldots, 8$ from the randomly designed example. For the second linear program, we experimented with randomly generating a^t and b^t for $t = 1, \ldots, 8$ in the GPA and creating an inequality constraint. We chose a line passing through the square as $y \leq x$, where $A^t = \begin{bmatrix} 1 & 1 \end{bmatrix}$ and $g^t = \begin{bmatrix} 0 & 0 \end{bmatrix}$. From Figure 1, you can see how the feasible

solutions $x^t, t = 1, \ldots, 8$ lie below $y = x$ as expected. Lastly, we obtained regret values of $R_A^{(i)} = -1e - 7$ and $R_A^{(ii)} = 0.2467$.

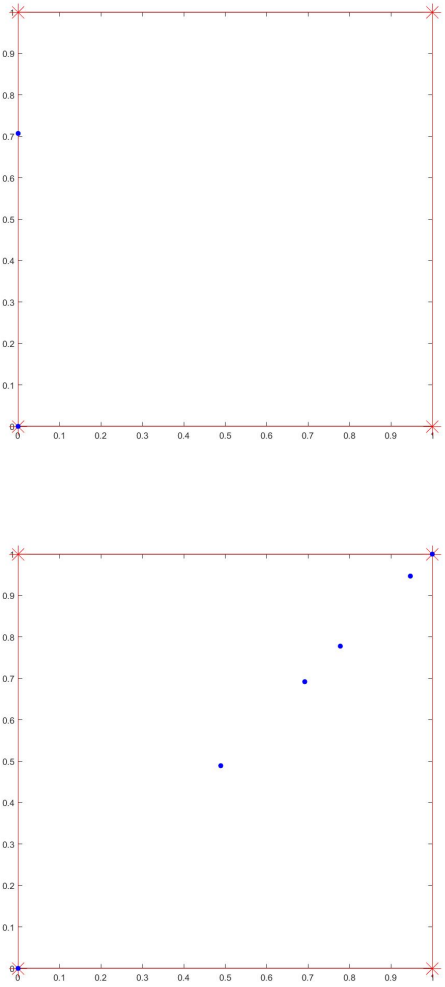

Figure 1: Linear Examples of (top) Handwritten and (bottom) Randomly Designed Problems

7.2 Quadratic Program

Quadratic Cost Function: $\min_{x \in F} c^t(x) = (x - a_1^t)^T Q^t (x - a_1^t) + (b^t)^T (x - a_2^t)$ s.t. $A^t x \leq g^t$,

where $F = \{x^i \in \mathbb{R}, i = 1, 2 | 0 \leq x^i \leq 1\}$.

Below are the feasible solutions at each iteration (i) for c^t for $t = 1, 2$ from the handwritten example problem and (ii) for c^t for $t = 1, \ldots, 8$ from the randomly designed example. For the second quadratic program, we experimented with randomly generating Q^t, a_1^t, a_2^t and b^t for $t = 1, \ldots, 8$ in the GPA and creating an inequality constraint. We chose a line passing through the square as $y \leq x$, where $A^t = \begin{bmatrix} 1 & 1 \end{bmatrix}$ and $g^t = \begin{bmatrix} 0 & 0 \end{bmatrix}$. From Figure 1, you can see how the feasible solutions $x^t, t = 1, \ldots, 8$ lie below $y = x$ as expected. Lastly, we obtained regret values of $R_A^{(i)} = 2.0020$ and $R_A^{(ii)} = 0.0179$.

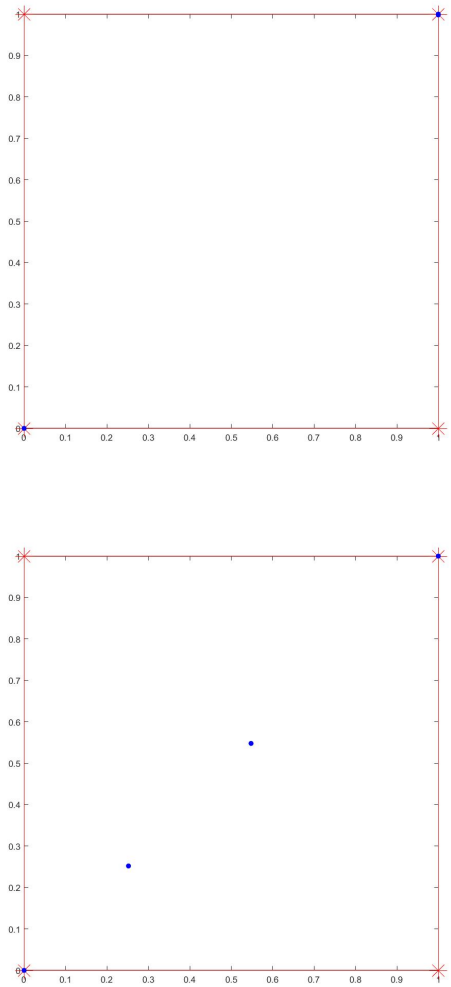

Figure 2: Quadratic Examples of (top) Handwritten and (bottom) Randomly Designed Problems

7.3 Mixed Quadratic and Linear Program

$$\min_{x \in F} c^t(x) = (x - a_1^t)^T Q^t (x - a_1^t) + (b^t)^T (x - a_2^t) + a_3^t \text{ s.t. } A^t x \leq g^t,$$

where $F = \{x^i \in \mathbb{R}, i = 1, 2 | 0 \leq x^i \leq 1\}$.

Below are the feasible solutions at each iteration (i) for c^t for $t = 1, 2$ from the handwritten example problem and (ii) for c^t for $t = 1, \ldots, 8$ from the randomly designed example. For the second mixed program, we experimented with

randomly generating Q^t, a_1^t, a_2^t, a_3^t and b^t for $t = 1, \ldots, 8$ in the GPA and creating an inequality constraint. We chose a line passing through the square as $y \leq x$, where $A^t = \begin{bmatrix} 1 & 1 \end{bmatrix}$ and $g^t = \begin{bmatrix} 0 & 0 \end{bmatrix}$. From Figure 1, you can see how the feasible solutions $x^t, t = 1, \ldots, 8$ lie below $y = x$ as expected. Lastly, we obtained regret values of $R_A^{(i)} = 2.9964$ and $R_A^{(ii)} = 0.6480$.

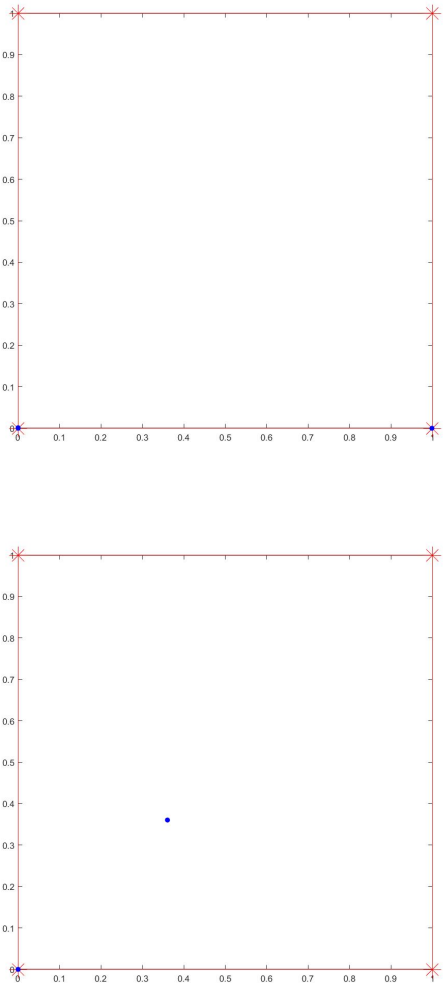

Figure 3: Mixed Examples of (top) Handwritten and (bottom) Randomly Designed Problems

8 Conclusion

In conclusion we have seen in this paper that the Greedy Projection Algorithm is a simple but powerful tool in online optimization. Its low convergence rate and ability to handle an unknown future make it an excellent choice for many real word applications. Our numerical results show that it is accurate and fast, with results for a small number of iterations running in fractions of a second. The Greedy Projection Algorithm is overall an excellent algorithm to study and consider for use.

8.1 Future Work

Perhaps future works include more robust testing of the GPA with more complicated feasible sets. Our work on this project was also limited to problems of two variables. Expanding our MATLAB code to handle more than two variables would be the next step. Observing run times and functionality of the algorithm under more than two dimensions would be a reasonable next step.

9 Appendix: MATLAB Codes

```
%%% Greedy Projection Algorithm Script %%%

%% Case 1a - Linear Program using Handwritten Problem
clc; clear all;

% Define coefficient matrix for quadratic and/or coefficient vector for
% linear example
q = [1 1; 1 -1];

% Define inequality constraints
A = []; % coefficient matrix
b = []; % right-hand side

% Define equality constraints
Aeq = []; % coefficient matrix
beq = []; % right-hand side

% Define bounds
lb = [0; 0]; % lower bounds
ub = [1; 1]; % upper bounds

cfun = @(x,a,b,q) q*(x-a); % cost functions
cgradfun = @(x,a,b,q) q'; % gradient of cost functions

n = 2; % dimension of space in Rn, i.e., number of variables
```

```
N = 2; % number of cases or cost functions
% initial guess (must be in feasible set)
x(:,1) = zeros(2,1);
coeff = [0 0; 0 0];

[xt,RA] = GPAfunclinear(A,b,Aeq,beq,lb,ub,cfun,cgradfun,x,coeff,q);

if (n==2) % plot in 2D only!
    figure;
    plot([lb(1) ub(1)],[lb(2) lb(2)],'-*r','MarkerSize',20);
    hold on;
    plot([lb(1) ub(1)],[ub(2) ub(2)],'-*r','MarkerSize',20);
    hold on;
    plot([lb(1) lb(1)],[lb(2) ub(2)],'-*r','MarkerSize',20);
    hold on;
    plot([ub(1) ub(1)],[lb(2) ub(2)],'-*r','MarkerSize',20);
    hold on;
    % Set of feasible points computed without knowing the new cost function
    plot(xt(1,:),xt(2,:),'.b','MarkerSize',20);
    hold off;
    axis square;
end

%% Case 1b - Linear Program
clc; clear all;

n = 2; % dimension of space in Rn, i.e., number of variables
N = 8; % number of cases or cost functions

% Define coefficient matrix for quadratic and/or coefficient vector for
% linear example
q = randn(N,n);

% Define inequality constraints
A = []; % coefficient matrix
b = []; % right-hand side

% Define equality constraints
Aeq = [-1 1]; % coefficient matrix
beq = [0]; % right-hand side

% Define bounds
lb = [0; 0]; % lower bounds
ub = [1; 1]; % upper bounds

cfun = @(x,a,b,q) q*(x-a); % cost functions
```

```
cgradfun = @(x,a,b,q) q'; % gradient of cost functions

% initial guess (must be in feasible set)
x(:,1) = zeros(2,1);
coeff = randn(N,n);

[xt,RA] = GPAfunclinear(A,b,Aeq,beq,lb,ub,cfun,cgradfun,x,coeff,q);

if (n==2) % plot in 2D only!
    figure;
    plot([lb(1) ub(1)],[lb(2) lb(2)],'-*r','MarkerSize',20);
    hold on;
    plot([lb(1) ub(1)],[ub(2) ub(2)],'-*r','MarkerSize',20);
    hold on;
    plot([lb(1) lb(1)],[lb(2) ub(2)],'-*r','MarkerSize',20);
    hold on;
    plot([ub(1) ub(1)],[lb(2) ub(2)],'-*r','MarkerSize',20);
    hold on;
    % Set of feasible points computed without knowing the new cost function
    plot(xt(1,:),xt(2,:),'.b','MarkerSize',20);
    hold off;
    axis square;
end

%% Case 2a - Quadratic Program using Handwritten Problem
clc; clear all;

% Define coefficient matrix
Q = [1 2; 2 -1; -1 2; 2 1];
q = [1 -1;1 -1];

% Define inequality constraints
A = []; % coefficient matrix
b = []; % right-hand side

% Define equality constraints
Aeq = []; % coefficient matrix
beq = []; % right-hand side

% Define bounds
lb = [0; 0]; % lower bounds
ub = [1; 1]; % upper bounds

cfun = @(x,a,b,Q,q) (x-a)'*Q*(x-a) + q'*x; % cost functions
cgradfun = @(x,a,b,Q,q) 2*Q*(x-a) + q; % gradient of cost functions
```

```
n = 2; % dimension of space in Rn, i.e., number of variables
N = 2; % number of cases or cost functions
% initial guess (must be in feasible set)
x(:,1) = zeros(2,1);
coeff = [1 -1; 1 -1];

[xt,RA] = GPAfuncquad(A,b,Aeq,beq,lb,ub,cfun,cgradfun,x,coeff,Q,q);

if (n==2) % plot in 2D only!
    figure;
    plot([lb(1) ub(1)],[lb(2) lb(2)],'-*r','MarkerSize',20);
    hold on;
    plot([lb(1) ub(1)],[ub(2) ub(2)],'-*r','MarkerSize',20);
    hold on;
    plot([lb(1) lb(1)],[lb(2) ub(2)],'-*r','MarkerSize',20);
    hold on;
    plot([ub(1) ub(1)],[lb(2) ub(2)],'-*r','MarkerSize',20);
    hold on;
    % Set of feasible points computed without knowing the new cost function
    plot(xt(1,:),xt(2,:),'.b','MarkerSize',20);
    hold off;
    axis square;
end

%% Case 2b - Quadratic Program
clc; clear all;

n = 2; % dimension of space in Rn, i.e., number of variables
N = 8; % number of cases or cost functions

% Define coefficient matrix
Q = repmat(eye(n,n),N,1);
q = randn(n,N);
% q = repmat([1 -1; 1 -1],1,N);

% Define inequality constraints
A = []; % coefficient matrix
b = []; % right-hand side

% Define equality constraints
Aeq = [-1 1]; % coefficient matrix
beq = [0]; % right-hand side

% Define bounds
lb = [0; 0]; % lower bounds
ub = [1; 1]; % upper bounds
```

```
cfun = @(x,a,b,Q,q) (x-a)'*Q*(x-a) + q'*(x-a); % cost functions
cgradfun = @(x,a,b,Q,q) 2*Q*(x-a) + q; % gradient of cost functions

% initial guess (must be in feasible set)
x(:,1) = zeros(2,1);
coeff = randn(N,n);

[xt,RA] = GPAfuncquad(A,b,Aeq,beq,lb,ub,cfun,cgradfun,x,coeff,Q,q);

if (n==2) % plot in 2D only!
    figure;
    plot([lb(1) ub(1)],[lb(2) lb(2)],'-*r','MarkerSize',20);
    hold on;
    plot([lb(1) ub(1)],[ub(2) ub(2)],'-*r','MarkerSize',20);
    hold on;
    plot([lb(1) lb(1)],[lb(2) ub(2)],'-*r','MarkerSize',20);
    hold on;
    plot([ub(1) ub(1)],[lb(2) ub(2)],'-*r','MarkerSize',20);
    hold on;
    % Set of feasible points computed without knowing the new cost function
    plot(xt(1,:),xt(2,:),'.b','MarkerSize',20);
    hold off;
    axis square;
end

%% Case 3a - Mixed Program using Handwritten Problem
clc; clear all;

% Define coefficient matrix
q1 = [-1 0; 1 0];
q2 = [0 2];
q3 = [0 2; 0 -2];

% Define inequality constraints
A = []; % coefficient matrix
b = []; % right-hand side

% Define equality constraints
Aeq = []; % coefficient matrix
beq = []; % right-hand side

% Define bounds
lb = [0; 0]; % lower bounds
ub = [1; 1]; % upper bounds
```

```
cfun = @(x,q1,q2,q3) q1'*x + q2*x(1)*x(2) + q3'*x.^2; % cost functions
% Gradient of cost functions
cgradfun = @(x,q1,q2,q3) q1 + [q2*x(1); q2*x(2)] + 2*q3.*x;

n = 2; % dimension of space in Rn, i.e., number of variables
% initial guess (must be in feasible set)
x(:,1) = zeros(2,1);

[xt,RA] = GPAfuncmixed(A,b,Aeq,beq,lb,ub,cfun,cgradfun,x,q1,q2,q3);

if (n==2) % plot in 2D only!
    figure;
    plot([lb(1) ub(1)],[lb(2) lb(2)],'-*r','MarkerSize',20);
    hold on;
    plot([lb(1) ub(1)],[ub(2) ub(2)],'-*r','MarkerSize',20);
    hold on;
    plot([lb(1) lb(1)],[lb(2) ub(2)],'-*r','MarkerSize',20);
    hold on;
    plot([ub(1) ub(1)],[lb(2) ub(2)],'-*r','MarkerSize',20);
    hold on;
    % Set of feasible points computed without knowing the new cost function
    plot(xt(1,:),xt(2,:),'.b','MarkerSize',20);
    hold off;
    axis square;
end

%% Case 3b - Mixed Program
clc; clear all;

n = 2; % dimension of space in Rn, i.e., number of variables
N = 8; % number of cases or cost functions

% Define coefficient matrix
q1 = randn(n,N);
q2 = randn(1,N)
q3 = randn(n,N);

% Define inequality constraints
A = []; % coefficient matrix
b = []; % right-hand side

% Define equality constraints
Aeq = [-1 1]; % coefficient matrix
beq = [0]; % right-hand side

% Define bounds
```

```
lb = [0; 0]; % lower bounds
ub = [1; 1]; % upper bounds

cfun = @(x,q1,q2,q3) q1'*x + q2*x(1)*x(2) + q3'*x.^2; % cost functions
% Gradient of cost functions
cgradfun = @(x,q1,q2,q3) q1 + [q2*x(1); q2*x(2)] + 2*q3.*x;

% initial guess (must be in feasible set)
x(:,1) = zeros(2,1);

[xt,RA] = GPAfuncmixed(A,b,Aeq,beq,lb,ub,cfun,cgradfun,x,q1,q2,q3);

if (n==2) % plot in 2D only!
    figure;
    plot([lb(1) ub(1)],[lb(2) lb(2)],'-*r','MarkerSize',20);
    hold on;
    plot([lb(1) ub(1)],[ub(2) ub(2)],'-*r','MarkerSize',20);
    hold on;
    plot([lb(1) lb(1)],[lb(2) ub(2)],'-*r','MarkerSize',20);
    hold on;
    plot([ub(1) ub(1)],[lb(2) ub(2)],'-*r','MarkerSize',20);
    hold on;
    % Set of feasible points computed without knowing the new cost function
    plot(xt(1,:),xt(2,:),'.b','MarkerSize',20);
    hold off;
    axis square;
end

%%% Greedy Projection Algorithm %%%
function [x,RA] = GPAfunclinear(A,b,Aeq,beq,lb,ub,cfun,cgradfun,x,coeff,q)
% Function for generating learning weights
etafun = @(t) 1/sqrt(t);
% Stopping time
T = size(coeff,1);

for t = 1:T
    eta(t) = etafun(t); % learning weight at time t
    % Cost function at time t
    cA(t) = cfun(x(:,t),coeff(t,1),coeff(t,2),q(t,:));
    % Projection point at time t
    y = x(:,t)-eta(t)*cgradfun(x(:,t),coeff(t,1),coeff(t,2),q(t,:));
    projfun = @(x) norm(x-y,2); % l2-norm or euclidean distance
    % Feasible point computed without knowing the new cost function
    x(:,t+1) = fmincon(projfun,x(:,t),A,b,Aeq,beq,lb,ub);
end
```

```
% Computing regret of algorithm unitl time T
% Static feasible solution
ctfun = @(x) costfunlinear(x,cfun,coeff(:,1),coeff(:,2),q);
xs = fmincon(ctfun,x(:,1),A,b,Aeq,beq,lb,ub);
RA = sum(cA)-ctfun(xs);

end

function [c] = costfunlinear(x,cfun,a,b,q)
% Function for summing all of cost functions
c = 0;
T = size(a,1);
for t = 1:T
    c = c + cfun(x,a(t),b(t),q(t,:));
end

end

%%% Greedy Projection Algorithm %%%
function [x,RA] = GPAfuncquad(A,b,Aeq,beq,lb,ub,cfun,cgradfun,x,coeff,Q,q)
% Function for generating learning weights
etafun = @(t) 1/sqrt(t);
% Stopping time
T = size(coeff,1);
n = size(x,1);

for t = 1:T
    r1 = 1+(t-1)*n;
    r2 = t*n;
    eta(t) = etafun(t); % learning weight at time t
    % Cost function at time t
    cA(t) = cfun(x(:,t),coeff(t,1),coeff(t,2),Q(r1:r2,1:n),q(:,t));
    % Projection point at time t
    y = x(:,t)-eta(t)*cgradfun(x(:,t),coeff(t,1),coeff(t,2),...
        Q(r1:r2,1:n),q(:,t));
    projfun = @(x) norm(x-y,2); % l2-norm or euclidean distance
    % Feasible point computed without knowing the new cost function
    x(:,t+1) = fmincon(projfun,x(:,t),A,b,Aeq,beq,lb,ub);
end

% Computing regret of algorithm unitl time T
% Static feasible solution
ctfun = @(x) costfunquad(x,cfun,coeff(:,1),coeff(:,2),Q,q);
xs = fmincon(ctfun,x(:,1),A,b,Aeq,beq,lb,ub);
RA = sum(cA)-ctfun(xs);
```

```
end

function [c] = costfunquad(x,cfun,a,b,Q,q)
% Function for summing all of cost functions
c = 0;
T = size(a,1);
n = size(x,1);
for t = 1:T
    r1 = 1+(t-1)*n;
    r2 = t*n;
    c = c + cfun(x,a(t),b(t),Q(r1:r2,1:n),q(:,t));
end

end

%%% Greedy Projection Algorithm %%%
function [x,RA] = GPAfuncmixed(A,b,Aeq,beq,lb,ub,cfun,cgradfun,x,q1,q2,q3)
% Function for generating learning weights
etafun = @(t) 1/sqrt(t);
% Stopping time
T = size(q1,1);
n = size(x,1);

for t = 1:T
    eta(t) = etafun(t); % learning weight at time t
    % Cost function at time t
    cA(t) = cfun(x(:,t),q1(:,t),q2(:,t),q3(:,t));
    % Projection point at time t
    y = x(:,t)-eta(t)*cgradfun(x(:,t),q1(:,t),q2(:,t),q3(:,t));
    projfun = @(x) norm(x-y,2); % l2-norm or euclidean distance
    % Feasible point computed without knowing the new cost function
    x(:,t+1) = fmincon(projfun,x(:,t),A,b,Aeq,beq,lb,ub);
end

% Computing regret of algorithm unitl time T
% Static feasible solution
ctfun = @(x) costfunmixed(x,cfun,q1,q2,q3);
xs = fmincon(ctfun,x(:,1),A,b,Aeq,beq,lb,ub);
RA = sum(cA)-ctfun(xs);

end


function [c] = costfunmixed(x,cfun,q1,q2,q3)
% Function for summing all of cost functions
c = 0;
```

```
T = size(q1,1);
for t = 1:T
    c = c + cfun(x,q1(:,t),q2(:,t),q3(:,t));
end

end
```

References

[1] Blum A. Chawla S. Meyerson A. Bansal, B. Online oblivious routing. 2003.

[2] Elad Hazan. *Introduction to Online Convex Optimization*. Princeton University Press, 2016.

[3] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. *Proceedings of the 20th International Conference on Machine Learning*, pages 928–936, 2003.

[4] M. Zinkevich. Online convex programmingand generalized infinitesimal gradient ascent. 2003.