CONVEX AND CONIC OPTIMIZATION
Homework 6

NILAVA METYA
nilava.metya@rutgers.edu
nm8188@princeton.edu

May 2, 2024

Problem 1

1. Suppose you had a blackbox that given a 3SAT instance would tell you whether it is satisfiable or not. How can you make polynomially many calls to this blackbox to find a satisfying assignment to any satisfiable instance of 3SAT?
2. Suppose you had a blackbox that given a graph G and an integer k would tell you whether G has a stable set of size larger or equal to k. How can you make polynomially many calls to this blackbox to find a maximum stable set of a given graph?

Solution

1. Sum, product in boolean expressions will denote `OR`, `AND` respectively.
Let f be the blackbox. f takes in a formula and outputs 1 if satisfiable, and 0 otherwise.
Let S be a formula in CNF, with variables $x_1, \cdots, x_n$. Treat S as a polynomial in x_i's. Assume S is satisfiable, i.e., there are values $a_1, \cdots, a_n \in \{0,1\}$ such that $S(a_1, \cdots, a_n) = 1$. Often we denote $\boldsymbol{a} = (a_1, \cdots, a_n)$. We will make n calls to the blackbox. We find $\boldsymbol{a}$ using f as in Algorithm 1.

   Let's justify the correctness via the following claims. The first statement in each bullet is a claim, followed by an explanation.

   - **For each $i \in [n]$, at least one of $f(x_i \wedge S)$ or $f(\overline{x_i} \wedge S)$ is `True`.** If $a_i = 1$ then $(x_i \wedge S)(\boldsymbol{a}) = 1 \cdot S(\boldsymbol{a}) = 1$. If $a_i = 0$ then $(\overline{x_i} \wedge S)(\boldsymbol{a}) = 1 \cdot S(\boldsymbol{a}) = 1$.
   - **For any formula φ, and a variable x in it, $f(x\varphi) =$ `True` iff φ has an assignment such that $x = 1$.** Indeed if $f(x\varphi) =$ `True` then there is some $\boldsymbol{c} \in \{0,1\}^N$ (where N is the number of variables in φ) such that $\varphi(\boldsymbol{c}) = x(\boldsymbol{c}) = 1$. On the other hand if $\boldsymbol{c}$ is an assignment with $x(\boldsymbol{c}) = 1 = \varphi(\boldsymbol{c})$ then clearly $\boldsymbol{c}$ satisfies $x\varphi$. Note here that $x(\boldsymbol{c})$ is the coordinate of $\boldsymbol{c}$ that corresponds to the variable x.
   - **At the end of the k^{th} loop (for each $1 \leq k \leq n$), $y_1 \cdots y_k S$ is satisfiable with $y_1 = \cdots = y_k = 1$, where S is the input to the above procedure.** (Inductively) assuming this is true

Algorithm 1 Finding SAT assignment using decision solving blackbox

Require: Formula S (which has a satisfiable assignment)
Ensure: Boolean numbers $a_1, \cdots, a_n$ that satisfy S

```
procedure FINDSAT(S)
    declare variables y_1, ..., y_n
    for k = 1, ..., n do
        y_k ← x_k
        if f(S · ∏_{i=1}^{k} y_i) then
            a_k ← 1
        else
            y_k ← x̄_k
            a_k ← 0
        end if
    end for
    return a = (a_1, ..., a_n)
end procedure
```

till t^{th} step, we note that the **if** statement in the $(t+1)^{\text{st}}$ iteration checks $f(Sy_1 \cdots y_t \cdot x_{t+1})$. We know that $Sy_1 \cdots y_t$ is satisfiable with $y_1 = \cdots = y_t = 1$. If $f(Sy_1 \cdots y_t x_{t+1}) = 1$ then $Sy_1 \cdots y_t$ has a satisfiable assignment with $x_{t+1} = 1$ (by the previous bullet) which is assigned to a_{t+1} and y_{t+1} is taken to be x_{t+1} itself. If not, then (by first bullet) $Sy_1 \cdots y_t \overline{x_{t+1}}$ is satisfiable with $x_{t+1} = 0$ which is assigned to a_{t+1} again, but y_{t+1} is taken to be $\overline{x_{t+1}}$. In either case, $S \cdot y_1 \cdots y_{t+1}$ is satisfiable, by the end of the loop. The base case for $k = 1$ is again true due to the previous bullet point.

- $y_i = a_i \cdot x_i + \overline{a_i} \cdot \overline{x_i}$ $\forall i \in [n]$. This is clear by the construction. Note that this is exactly $\overline{a_i \texttt{ XOR } x_i}$.
- **At the end,** $y_1 \cdots y_n S$ **is satisfiable with the assignment** $x_i = a_i$ $\forall i$**.** Using the third and fourth bullet above, $y_i = 1$ exactly when $x_i = a_i$.

The above algorithm takes only $\mathcal{O}(n \cdot T(f))$ steps where $T(f)$ is the runtime of the blackbox f (or $\mathcal{O}(n^3 \cdot T(f))$ if we consider that it takes $\mathcal{O}(k)$ steps to form $y_1 \cdots y_k S$ before passing it to f). Thus it's a polynomial-time reduction.

2. We have a decision problem solving submodule HASSTAB(G, k) that takes input a graph G, an integer k and outputs `True` if G has a stable set of size (at least) k, and `False` otherwise. We also use a module NEIGH(G, v) which returns all neighbors of v in G.

   Let's show that this algorithm finds a maximal stable set. It is clear that the first **for** loop finds the size of maximal stable set k. From now on, when we say 'loop', we refer to the second **for** loop. In the following enumerations, the first lines contains claims in bold, followed by a proof in the rest of the paragraph.

   - S **always remains stable**. Whenever S is updated with, say, v, G' is restricted to the non-neghbors of v, ensuring that every future update will only accept vertices in S among non-neighbors of v.

Algorithm 2 Finding stable set using decision solving blackbox

Require: Graph $G = (V, E)$ with $V = [n]$
Ensure: Maximal stable set in G
  procedure MAXSTAB(G)
    $n \leftarrow$ number of vertices, $k \leftarrow 0$
    for $i = 1, \cdots, n$ **do** ▷ finding maximum stable set size
      if HASSTAB(G, i) **then**
        $k \leftarrow i$
      end if
    end for
    $S \leftarrow \varnothing, t \leftarrow k, G' \leftarrow G$ ▷ initialize a container for max stable set and make dummy variables
    for $i = 1, \cdots, n$ **do** ▷ finding maximum stable set
      $G' \leftarrow G' \setminus \{i\}$
      if (NOT HASSTAB(G', t)) **then**
        append i to S
        $t \leftarrow t - 1$
        $G' \leftarrow G' \setminus$ NEIGH(G', i)
      end if
    end for
    return S
  end procedure

- **At the start and end of every reiteration, G' has a stable set of size t, for the respective current values**. (Inductively) assume true at the start of some loop. If removing i from G reduces the stable set number (by 1 because one vertex is removed), which is exactly how t is updated. If true at the end of some loop, then automatically true for the start of the next loop.
- **At the end of every reiteration, $|S| + t = k$.** This is because t is updated to decrease by 1 exactly when S gets a new vertex.
- **After the i^{th} iteration, the only vertices remaining in G' are a subset of $[n] \setminus [i]$.** This is beacuse vertex i is removed at the t^{th} iteration.
- **At the end of the iteration corresponding to $i = n$, the value of t is 0.** At the beginning of this iteration, G' is either $\{n\}$ or $\varnothing$ (due to the previous bullet point), so that the only possible values of t are $0, 1$ (by the second bullet above). If $t = 1$ then $G' = \{n\}$ so that only the **else** brach is executed and t is updated to 0. If t was already 0, then there is no update and the iteration ends with $t = 0$. So at the end, $t = 0$ anyway.

Due to the third and fifth bullet points above, the set S we end with satisfies that $|S| = k$ whence it has the same cardinality as a maximal stable set. The first bullet point shows that it is stable. Thus the algorithm outputs a maximal stable set.

Now coming to runtime. The number of steps is $\mathcal{O}(n \cdot T(\text{HASSTAB}) + n \cdot T(\text{NEIGH}))$ where $T(\text{HASSTAB})$ is the time to run the module HASSTAB and $T(\text{NEIGH})$ is the time to find neighbors

of a vertex and remove it from the graph. The latter is clearly polynomial-time in terms of the input. Thus this is a poly-time reduction.

Problem 2

Consider a family of decision problems indexed by a positive integer k:

RANK-k-SDP

Input: Symmetric $N \times N$ matrices $A_1, \cdots, A_m$ with entries in $\mathbb{Q}$, scalars $b_1, \cdots, b_m \in \mathbb{Q}$.
Question: Is there a real symmetric matrix X that satisfies the constraints

$$\mathrm{Tr}(A_i X) = b_i, i \in [m], X \succeq 0, \mathrm{rank}(X) = k? \tag{1}$$

Solution

RANK-1-SDP:

First notice that a symmetric psd matrix $X \in S^n$ has rank 1 iff it has the form $xx^\top$ for some $x \in \mathbb{R}^n \setminus \{0\}$. So in the underlying problem, the constraints can be rewritten as $b_i = \mathrm{Tr}(A_i X) = x^\top A_i x \forall i \in [m]$ and $x \neq 0$. Thus it is clear that

$$\exists X \in S^n \text{ s.t. } X \succeq 0, \mathrm{Tr}(A_i X) = b_i \forall i \in [m], \mathrm{rank}(X) = 1 \iff \exists x \in \mathbb{R}^n \text{ s.t. } x^\top A_i x = b_i \forall i \in [m], x \neq 0. \tag{2}$$

We will show a reduction **STABLE-SET** $\longrightarrow$ **RANK-1-SDP**. Let (G, k) be an instance of **STABLE-SET** where graph G has edges E and vertices $[n]$. Recall that this is same as feasibility of some $v \in \mathbb{R}^n$ satisfying $v_i(1 - v_i) = 0 \; \forall i \in [n], v_i v_j = 0 \; \forall \{i, j\} \in E, \sum\limits_{i \in [n]} v_i \geq k$.

Since each $v_i \in \{0, 1\}$, $\sum_{i \in [n]} v_i$ whence the last constraint is equivalent to the existence of some $s \in \mathbb{R}$ such that $\left(\sum\limits_{i \in [n]} v_i \right)^2 - k^2 = s^2$. So **STABLE-SET** is the feasibility of some v, s subject to

$$\begin{aligned} v \in \mathbb{R}^n, s \in \mathbb{R} \\ v_i(1 - v_i) = 0 \; \forall i \in [n] \\ v_i v_j = 0 \; \forall \{i, j\} \in E \\ \left(\sum_{i \in [n]} v_i \right)^2 = k^2 + s^2. \end{aligned} \tag{(Q)}$$

Claim 1
Feasibility of (Q) is equivalent to the feasibility of v, c, s subject to

$$\begin{aligned} v \in \mathbb{R}^n, c \in \mathbb{R}, s \in \mathbb{R} \\ c^2 = 1 \\ v_i(c - v_i) = 0 \; \forall i \in [n] \\ v_i v_j = 0 \; \forall \{i, j\} \in E \\ \left(\sum_{i \in [n]} v_i \right)^2 - s^2 = k^2. \end{aligned} \tag{(QQ)}$$

Proof. If (v, s). is feasible to (Q), then $(v, 1, s)$ is clearly feasible to (QQ).
Now suppose (v, c, s) is feasible to (QQ). So each $v_i \in \{0, c\}\, \forall i \in [n]$. This means $k^2 + s^2 = (\sum iv_i)^2 = \left(\sum_{\substack{i\in[n]\\ v_i=c}} v_i\right)^2 = c^2 \left(\sum_{\substack{i\in[n]\\ v_i\neq 0}} 1\right)^2 = \left(\sum_{\substack{i\in[n]\\ v_i\neq 0}} cv_i\right)^2 = (\sum_i cv_i)^2$. This shows that (cv, s) is feasible to (QQ). ■

Notice how all constraints in (QQ) are polynomial expressions with only constant terms and homogeneous quadratic terms. This perfectly matches with what we want in the original problem using eq. (2). To get an instance of **RANK-**1**-SDP**, take:

- The size of matrices $N = n + 2$,
- $Q = e_{n+1}e_{n+1}^\top$ and $q = 1$,
- $B_i = E_{i,n+1} - 2e_ie_i^\top$ and $r_i = 0$ for each $i \in [n]$
- $A_{ij} = E_{ij} = e_ie_j^\top + e_je_i^\top$ (which is the $N \times N$ matrix with 1 in positions $(i, j), (j, i)$ and 0 elsewhere), $b_{ij} = b_{ji} = 0$ for each edge $\{i, j\} \in E$,
- $T = \sum_{1\leq i,j\leq n} e_ie_j^\top - e_Ne_N^\top, b = k^2$.

The above are clearly all rational.

Now consider the constraints on x

$$\begin{aligned} x &\in \mathbb{R}^N \\ x^\top Qx &= q \\ x^\top B_i x &= r_i \forall i \in [n] \\ x^\top A_{ij} x &= b_{ij} \forall \{i, j\} \in E \\ x^\top Tx &= b \end{aligned} \tag{(HQ)}$$

We show that this answers the same question as **STABLE-SET** gives on (G, k).

Claim 2
(QQ) is feasible iff (HQ) is feasible.

Proof. A solution $(v, c, s) \in \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}$ of (QQ) corresponds to a solution $x = (v, c, s) \in \mathbb{R}^N$ of (HQ).

$$\begin{aligned} x^\top Qx &= c^2 & 1 &= q \\ x^\top B_i x &= 2v_ic - 2v_i^2 & 0 &= r_i \\ x^\top A_{ij} x &= 2v_iv_j & 0 &= b_{ij} \\ x^\top Tx &= (v_1 + \cdots + v_n)^2 - s^2 & k^2 &= b. \end{aligned}$$

Left column matches the LHS and right column matches the RHS of each constraint in (QQ) and (HQ). ■

Notice that every solution x of (HQ) is automatically nonzero because of the first constraint: $x_{n+1}^2 = 1$.

The above shows that G has a stable set of size (at least) k iff (HQ) is feasible, which is a problem of **RANK-**1**-SDP**. This completes the proof that **RANK-**1**-SDP** is NP hard because we showed a reduction from an NP hard problem.

RANK-k-SDP:

We will show a reduction **RANK-**1**-SDP** $\longrightarrow$ **RANK-**k**-SDP** for $k \geq 2$. Indeed, consider an instance of **RANK-**1**-SDP** with inputs which are $N \times N$ matrices $A_1, \cdots, A_m \in \mathbb{Q}^{N \times N}$ and scalars $b_1, \cdots, b_m \in \mathbb{Q}$. Consider the following constraints:

$$\begin{aligned} X \in & S^{N+k-1} \\ \operatorname{rank}(X) &= k \\ X &\succeq 0 \\ \operatorname{Tr}(E_{ij}X) &= 0 \ \forall N < i < N+k, 1 \leq j \leq N \\ \operatorname{Tr}(E_{ij}X) &= 0 \ \forall N < i < j < N+k \\ \operatorname{Tr}(E_{ii}X) &= 0 \ \forall N < i < N+k \\ \operatorname{Tr}(\tilde{A}_i X) &= b_i \ \forall i \in [m]. \end{aligned} \tag{3}$$

Here $\tilde{A}_i = \begin{bmatrix} A_i & 0 \\ 0 & 0 \end{bmatrix} \in S^{N+k-1}$ obtained by putting appropriate number of 0's.

We discuss the above linear constraints one by one. On slight inspection, the above linear constraints deal with four different blocks of $X = \begin{bmatrix} A & B \\ B^\top & C \end{bmatrix}$ where $A \in S^N, B \in \mathbb{R}^{N \times (k-1)}, C \in S^{k-1}$.

- $\operatorname{Tr}(E_{ij}X) = 0 \ \forall N < i < N+k, 1 \leq j \leq N$: here $E_{ij} = e_i e_j^\top + e_j e_i^\top$. This is just saying that $X_{ij} = X_{ji} = 0$ whenever $i > N$ and $j \in [N]$. In other words $B = 0$.
- $\operatorname{Tr}(E_{ij}X) = 0 \ \forall N < i < j < N+k$: This looks at the block C because $j > i > N$. The constraint says that all off-diagonal entries of C are 0, that is, C is a diagonal matrix.
- $\operatorname{Tr}(E_{ii}X) = 0 \ \forall N < i < N+k$: This says that the diagonal entries of C are all 1. Using the previous point, C is the identity matrix of size $(k-1) \times (k-1)$.

So this makes X look like $\begin{bmatrix} A & 0 \\ 0 & I_{k-1} \end{bmatrix}$. Note that $\operatorname{rank}(X) = \operatorname{rank}(A) + \operatorname{rank}(I_{k-1}) = \operatorname{rank}(A) + k - 1$, whence $\operatorname{rank}(X) = k \iff \operatorname{rank}(A) = 1$. Further $\begin{bmatrix} A & 0 \\ 0 & I_{k-1} \end{bmatrix} \succeq 0 \iff A \succeq 0$ because eigenvalues of a block matrix formed of two blocks stacked diagonally is the union of the eigenvalues of the two individual blocks.

Claim 3
1 is feasible iff 3 is feasible.

Proof. From the above bullet points and the rank discussion, it is clear that a rank 1 solution $A \succeq 0$ of 1 corresponds to a rank k solution $\begin{bmatrix} A & 0 \\ 0 & I_{k-1} \end{bmatrix} \succeq 0$. ■

This completes the proof that **RANK-k-SDP** is NP hard because we showed a reduction from an NP hard problem.

Problem 3

A polynomial $p(x) \coloneqq p(x_1, \cdots, x_n)$ is nondecreasing with respect to a variable x_i if $\frac{\partial p}{\partial x_i}(x) \geq 0 \forall x \in \mathbb{R}^n$. Show that the problem of deciding whether a degree-d polynomial with rational coefficients is nondecreasing with respect to a particular variable (e.g., x_1) is

(i) in P if $d < 5$.

(ii) NP-hard if $d \geq 5$.

Solution

Set $d = \deg(p)$. Here $p \in \mathbb{Q}[x_1, \cdots, x_n]$. For short, denote $\partial_i p \coloneqq \frac{\partial p}{\partial x_i}$.

First note that $\deg(\partial_i(p)(x)) < d$ for each $i \in [n]$. This is because (fixing some $i \in [n]$) if $p(x) = x_i^k q_k(x_{-i}) + \cdots + x_i q_1(x_{-i}) + q_0(x_{-i})$ with all $q_j \in \mathbb{Q}[x_{-i}]$ (x_{-i} denotes the vector with all variables in x without x_i), then $d = \max\limits_{0 \leq j \leq k} \{j + \deg(q_j)\}$ whence $\deg(\partial_i p) = \max\limits_{1 \leq j \leq k} \{j - 1 + \deg(q_j)\} \leq \max\limits_{0 \leq j \leq k} \{j - 1 + \deg(q_j)\} = d - 1$ as a polynomial in $\mathbb{Q}[x]$.

The problem as mentioned, for each degree d, takes input n, the number of variables, the rational coefficients that make a polynomial, and an index i; and answers the question if the polynomial is non-decreasing in variable x_i. Denote the input size by N - that is the number of coefficients till degree d. The degree d of this polynomial can in fact be found in polynomial time in N. So there is a problem for each degree. Call it **MONO-d**.

(i) We will show this by cases on degree of $d' = \deg(\partial_i p(x))$. Note that $d' \leq 3$ if $d \leq 4$.

$d = 0$: Then $\partial_i p = 0 \geq 0$. So non-increasing. Thus we answered in constant time.

$d' = 0$: Then $\exists b \in \mathbb{Q}$ such that $\partial_i p(x) = b$. Non-negativity of the rational constant b can be checked in constant time.

$d' = 1$: The required derivative looks like $\partial_i p(x) = b^\top x + c$ for some $b \in \mathbb{Q}^n \setminus \{0\}, c \in \mathbb{Q}$. This is not everywhere non-negative because degree 1.

$d' = 2$: The required derivative looks like $\partial_i p(x) = \frac{1}{2} x^\top A x - b^\top x + c$ for some $A \in \mathbb{Q}^{n \times n}, b \in \mathbb{Q}^n, c \in \mathbb{Q}$ and A symmetric. Then:

- Say $A \not\succeq 0$. This is checked in $\mathcal{O}(n^3)$ time. Then the function $\partial_i p$ is unbounded below: for example, along the direction of some eigenvector with negative eigenvalue.
- So $A \succeq 0$ now. We have reduced to the problem of minimizing the convex function $\partial_i p(x)$ where $x \in \mathbb{R}^n$. Recall that if $x \in \mathbb{R}^n$ is a minima of this function then it's a critical point. Conversely if $x \in \mathbb{R}^n$ is a critical point, then $\nabla(\partial_i p)(x) = 0 \implies \nabla(\partial_i p(x))^\top (y - x) \geq 0 \forall y \in \mathbb{R}^n$ whence it is a minima (convexity was needed here). If there is no $x \in \mathbb{R}^n$ such that $(Ax - b =) \partial_i p(x) = 0$, then there is no minima and the problem is unbounded below. Otherwise assume there is a critical point $v \in \mathbb{R}^n$, that is, $Av = b$. In other words, b is in the column span (= row transpose span because A is symmetric) of A. Then we claim every critical point takes the same objective. Indeed if u is another critical point, then $Av = Au = b$ whence

$A^\top(u-v) = A(u-v) = 0$. Since b is in the column span of A, $b^\top(u-v) = 0$. But $\partial_i p(v) = -\frac{1}{2}b^\top v + c = \frac{1}{2}b^\top u + c = p(u)$. It follows that any critical point v gives a global minima with value $-\frac{1}{2}b^\top v + c$.

Thus our polynomial time algorithm is:

(a) Differentiate $p(x)$ to get A, b, c such that $\partial_i p(x) = \frac{1}{2}x^\top Ax - b^\top x + c$.

(b) Check if $A \succeq 0$. If not, conclude that p is not non-decreasing wrt x_i and `EXIT` (because $\partial_i p$ is then unbounded below). Otherwise go to the next step.

(c) Check if b is in the column span of A. If not, conclude that p is not non-decreasing wrt x_i and `EXIT` (because $\partial_i p$ is then unbounded below). Otherwise go to the next step.

(d) Otherwise $\{x \in \mathbb{R}^n \mid Ax = b\} \neq \varnothing$. Find a solution $\overline{x}$ to $Ax = b$. The objective $\partial_i p(\overline{x})$ becomes $-\frac{1}{2}b^\top \overline{x} + c$. If this value is ≥ 0, conclude that p is non-decreasing wrt x_i and `EXIT`. Otherwise conclude that p is not non-decreasing wrt x_i and `EXIT`.

This is correct because of the above discussion. This runs in polynomial time in size of input because step (a) takes same number of steps as number of coefficients; step (b) takes $\mathcal{O}(n^3) \leq \mathcal{O}(N^3)$ steps; step (c) can be done in polynomial time (in n) again (say Gaussian elimination); and step (d) can again be done in polynomial steps in n. So overall, the number of steps this algorithm takes is polynomial in the input size N.

$d' = 3$: So $\partial_i p(x)$ restricted to $\{x \in \mathbb{R}^n \mid x_1 = x_2 = \cdots = x_n\}$ gives a $3-$degree polynomial in one variable which, we know from elementary theory, is unbounded below.

(ii) We'll now show a reduction **COPOS** $\longrightarrow$ **MONO-**d for $d \geq 5$, where **COPOS** takes input a matrix M and decides whether it is copositive. Given an input $M \in \mathbb{Q}^{n\times n}$ to **COPOS**, we'll give the input $p(x_1, \cdots, x_{n+1}) \coloneqq x_{n+1}\left(\sum_{1\leq i\leq j\leq n} M_{ij}x_i^2x_j^2\right) + x_1^d$ and variable index $n+1$ to **MONO-**d. This makes $\deg$ of the first term to be 5 which is why we want $d \geq 5$. Then the following is immediate.

Claim 4
M is copositive iff p is non-decreasing wrt x_{n+1}.

Proof. Note that $\partial_{n+1}p(x) = \sum_{1\leq i\leq j\leq n} M_{ij}x_i^2x_j^2 = \begin{bmatrix} x_1^2 \\ \vdots \\ x_n^2 \end{bmatrix}^\top M \begin{bmatrix} x_1^2 \\ \vdots \\ x_n^2 \end{bmatrix} \geq 0$ for all $x \in \mathbb{R}^{n+1}$, iff $v^\top Mv \geq 0 \forall v \in \mathbb{R}^n$ satisfying $v \geq 0$, iff M is copositive. ■

Since **COPOS** is known to be NP-hard, the above reduction shows that **MONO-**d is NP-hard.

Problem 4

1. In the file `regression_data.mat`, you are given 20 points (x_i, f_i) in $\mathbb{R}^2$ where $(x_i)_{i=1,\cdots,20}$ are the entries of the vector `xvec` and $(f_i)_{i=1,\cdots,20}$ are the entries of the vector `fvec`. The goal is to fit a polynomial of degree 7

$$p(x) = c_0 + c_1 x + \cdots + c_7 x^7 \tag{4}$$

to the data to minimize least square error:

$$\min_{c_0,\cdots,c_7} \sum_{i=1}^{20} (f_i - p(x_i))^2. \tag{5}$$

The data comes from noisy measurements of an unknown function that is a priori known to be nondecreasing (e.g., the number of calories you intake as a function of the number of Big Macs you eat).

   (a) If the underlying function is truly monotone and the noise is not too large, one may hope that least squares would automatically respect the monotonicity constraint. Solve (5) to see if this is the case. Plot the optimal polynomial you get and report the optimal value.

   (b) Resolve (5) subject to the constraint that the polynomial (4) is nondecreasing. Plot the optimal polynomial you get and report the optimal value.

2. In the file `regression_data.mat`, you are also given 30 points (x_i^1, x_i^2, g_i) in $\mathbb{R}^3$ where $(x_i^1)_{i=1,\cdots,30}$ are the entries of the vector `x1vec`, $(x_i^2)_{i=1,\cdots,30}$ are the entries of `x2vec` and $(g_i)_{i=1,\cdots,30}$ are the entries of the vector `gvec`. The goal in this case is to fit a polynomial of degree 4:

$$p \coloneqq p(x_1, x_2) = c_0 + c_1 x_1 + c_2 x_2 + c_3 x_1^2 + c_4 x_1 x_2 + c_5 x_2^2 + \cdots + c_{15} x_2^4$$

to the data to minimize least square error:

$$\min_{c_0,\cdots,c_{15}} \sum_{i=1}^{30} (g_i - p(x_i))^2. \tag{6}$$

   (a) Solve 6 and plot the resulting polynomial together with the data points. Report the optimal value of the problem (denoted by η^*). Is the optimal polynomial convex?

   (b) Find a convex polynomial p of degree no more than 4 such that its least squares error

$$\eta \coloneqq \sum_{i=1}^{30} (g_i - p(x_i))^2$$

satisfies $\eta < 1.75\eta^*$.

Solution

1. (a) For this, we simply create an unconstrained minimization problem for 5 with parameters being c_i. The optimal value comes out to be 125.032580376 and the optimal polynomial comes out to be $-54.6891282979 - 115.453894584x + 433.37004128 * x^2 - 371.217387861 * x^3 + 143.250393347 * x^4 - 28.3156335072 * x^5 + 2.79906045685 * x^6 - .0.109762103312 * x^7$. Here's the plotted figure:

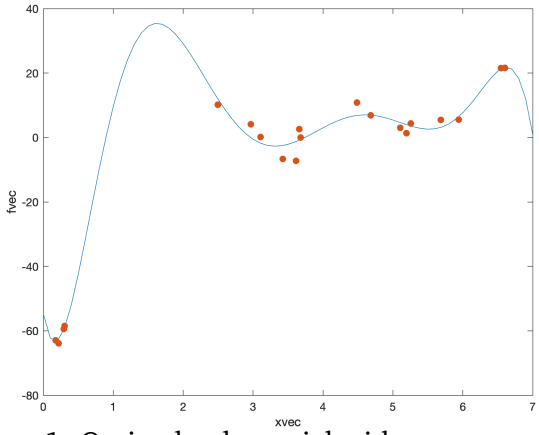

Figure 1: Optimal polynomial without constraints

```
clc;
clear all;

load("regression_data.mat");    %loading data
sdpvar x;   %declaring variable
d = 7;  %degree of our polynomial
[p, cp, mp] = polynomial(x,d);  %getting the polynomial p, coefficients ...
    cp, monomials mp

obj = 0;    %initializing objective
for i = 1:20    %for loop to add to consecutive sum of squares error
    obj = obj + (fvec(i) - replace(p,x,xvec(i)) )^2;
end

%setting up the problem and solving the optimization problem
options = sdpsettings('verbose',0,'solver','sdpt3');
optimize([], obj, options);

%displaying optimal value and optimal polynomial
sdisplay(double(obj));
sdisplay(double(cp)' * mp);

%plotting the polynomial along with data
plot(0:0.1:7, polyval(fliplr(double(cp)'), 0:0.1:7));
hold on;
scatter(xvec,fvec,'filled');
xlabel("xvec");
ylabel("fvec");
xlim([0,7]);
ylim([-80,40]);
f = gcf;
exportgraphics(f,'leastsquare.png','Resolution',300);
hold off;
```

(b) For this, we simply create a constrained minimization problem for 5 with parameters being c_i with the constraint that the derivative is non-negative everywhere. Due to Hilbert, non-negativity everywhere is equivalent to the polynomial being a SOS because it's univariate in this case. The optimal value comes out to be 329.348105777 and the optimal polynomial comes out to be $-77.1285656362+67.5949992492*x-10.6081243942*x^2-5.07857316587*x^3+1.41408502117*x^4+0.116990340254*x^5-0.0625905841674*x^6+0.00476800398802*x^7$.

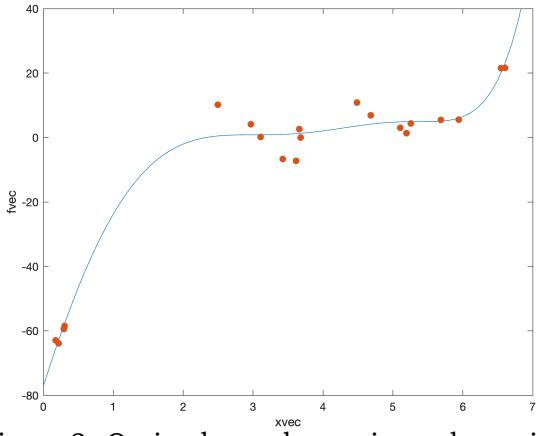

Figure 2: Optimal non-decreasing polynomial

```
clc;
clear all;

load("regression_data.mat");    %loading data
sdpvar x;   %declaring variable
d = 7;  %degree of our polynomial
[p, cp, mp] = polynomial(x,d);  %getting the polynomial p, coefficients ...
    cp, monomials mp

obj = 0;    %initializing the objective
for i = 1:20
    obj = obj + (fvec(i) - replace(p,x,xvec(i)))^2;
end

%setting up the problem and solving the optimization problem
options = sdpsettings('verbose',0,'solver','sdpt3');
constr = [sos(jacobian(p, x))];  %sos = non-negative because univariate
optimize(constr, obj, options);

%displaying optimal value and optimal polynomial
sdisplay(double(obj));
sdisplay(double(cp)' * mp);

%plotting the polynomial along with data
plot(0:0.1:7, polyval(fliplr(double(cp)'), 0:0.1:7));
hold on;
scatter(xvec,fvec,'filled');
xlabel("xvec");
ylabel("fvec");
xlim([0,7]);
ylim([-80,40]);
f = gcf;
exportgraphics(f,'leastsquareinc.png','Resolution',300);
hold off;
```

2. (a) We solve an unconstrained optimization as in 6 with parameters being $c_0, \cdots, c_{15}$. The

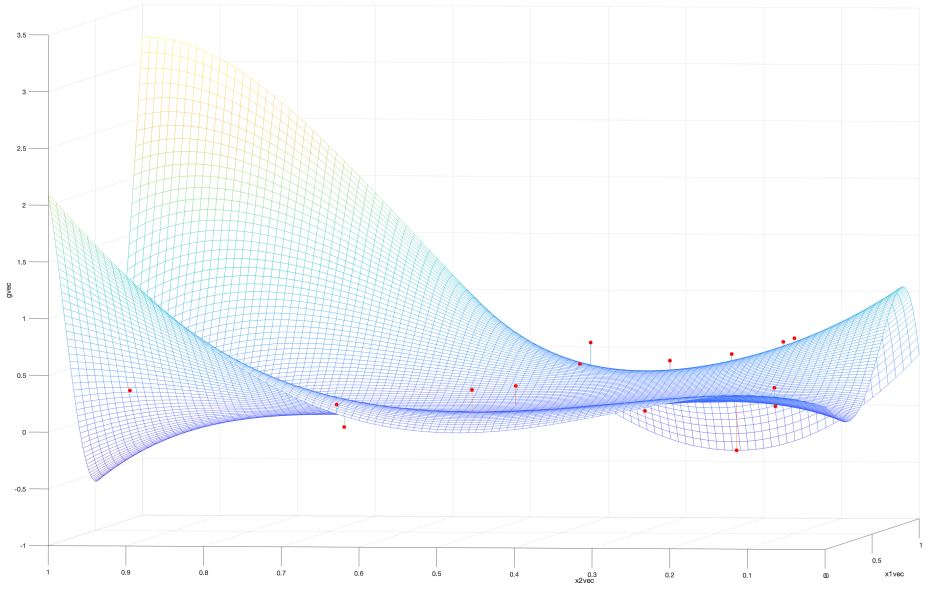

Figure 3: Optimal polynomial without constraints

optimal error came out to be $\eta^* = 0.607567008796$ and the optimal polynomial came out to be $p^* = 0.255317133343 - 0.969928883268 * x_1 + 2.14288177956 * x_2 - 2.67668839065 * x_1^2 + 8.62632262807 * x_1 * x_2 - 13.9522662329 * x_2^2 + 17.0926456078 * x_1^3 - 34.1236189852 * x_1^2 * x_2 + 6.32033706645 * x_1 * x_2^2 + 21.1619340693 * x_2^3 - 13.2527468384 * x_1^4 + 16.0599795058 * x_1^3 * x_2 + 23.1485809245 * x_1^2 * x_2^2 - 19.1076202561 * x_1 * x_2^3 - 7.511276463 * x_2^4$. It's clear from the picture that it's non-convex.

```
clc;
clear all;

load("HW6/regression_data.mat");    %loading data
sdpvar x_1 x_2 s t;   %declaring variable
inp = [s, t];
x = [x_1, x_2];
d = 4;  %degree of our polynomial
[p, cp, mp] = polynomial(x,d);  %getting the polynomial p, coefficients ...
    cp, monomials mp

obj = 0;    %initializing the objective
for i = 1:30
    obj = obj + (gvec(i) - replace(p, x, [x1vec(i), x2vec(i)]))^2;
end

%setting up the problem and solving the optimization problem
options = sdpsettings('verbose',0,'solver','sdpt3');
optimize([], obj, options);

%modifying to display polynomial in 3D
f = sdisplay(replace(p,cp,value(cp)));
f = f{1};
f = replace(f,'^','.^');
f = replace(f,'*','.*');
f = str2func(['@(x_1,x_2)' f]);
[X,Y] = meshgrid(0:0.01:1,0:0.01:1);
F = f(X,Y);
mesh(X,Y,F);
hold on
scatter3(x1vec,x2vec,gvec, 'filled', 'red');
xlabel("x1vec");
ylabel("x2vec");
zlabel("gvec");
for i = 1:30
    plot3([x1vec(i) x1vec(i)], [x2vec(i) x2vec(i)], [gvec(i) ...
        replace(double(cp)'*mp, x, [x1vec(i), x2vec(i)])]);
end
hold off

%printing optimal values
sdisplay(value(obj));
sdisplay(double(cp)' * mp);
```

(b) We solve a constrained optimization as in 6 with parameters being $c_0, \cdots, c_{15}$ with the constraints that the Hessian is PSD and the error is at most $1.75\eta^*$. The first constraint is enforced by asking that $\begin{bmatrix} s & t \end{bmatrix} H(x_1, x_2) \begin{bmatrix} s \\ t \end{bmatrix}$ being non-negative for every $x_1, x_2, s, t \in \mathbb{R}$ (equivalently SOS because it's a binary quartic in (x_1, x_2, s, t)). The optimal error came out to be 1.0473433212 and the optimal polynomial came out to be $p^* = 0.573755005423 - 1.99144979766 * \mathrm{x}_1 - 2.21859111356 * \mathrm{x}_2 + 5.09138334665 * \mathrm{x}_1^2 + 7.0384501806 * \mathrm{x}_1 * \mathrm{x}_2 + 3.92848978664 * \mathrm{x}_2^2 - 5.03233518376 * \mathrm{x}_1^3 - 9.12724553595 * \mathrm{x}_1^2 * \mathrm{x}_2 - 11.6513144747 * \mathrm{x}_1 *$

$x_2^2 - 2.34405914206 * x_2^3 + 1.91289158409 * x_1^4 + 3.73216359112 * x_1^3 * x_2 + 8.86040781842 * x_1^2 * x_2^2 + 2.87325038847 * x_1 * x_2^3 + 1.14003374755 * x_2^4$.

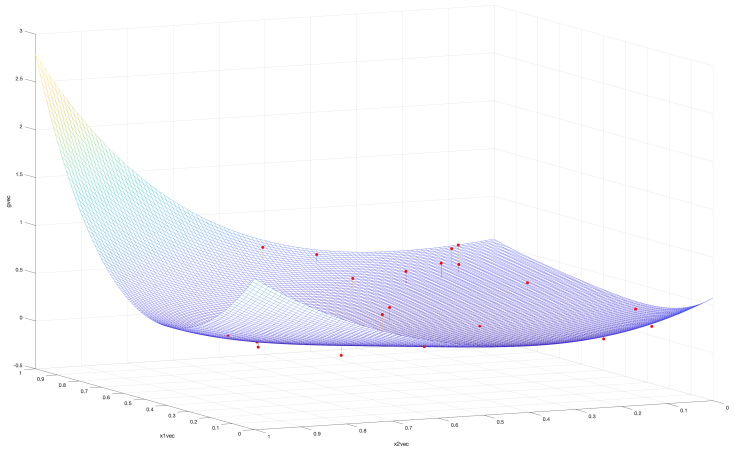

Figure 4: Optimal convex polynomial

```
clc;
clear all;

load("regression_data.mat");    %loading data
sdpvar x_1 x_2 s t;   %declaring variable
inp = [s, t];
x = [x_1, x_2];
d = 4;  %degree of our polynomial
[p, cp, mp] = polynomial(x,d);  %getting the polynomial p, coefficients ...
    cp, monomials mp
eta = 0.607567008796;   %optimal value from previous problem

obj = 0;    %initializing the objective
for i = 1:30
    obj = obj + (gvec(i) - replace(p, x, [x1vec(i), x2vec(i)]))^2;
end

%setting up the problem and solving the optimization problem
options = sdpsettings('verbose',0,'solver','sdpt3');
H = hessian(p,x);
constr = [sos(inp * H * inp')]; %sos = non-negative because bivariate ...
    quartic
constr = constr + [obj ≤ eta * 1.75];
optimize(constr, obj, options);

%modifying to display polynomial in 3D
f = sdisplay(replace(p,cp,value(cp)));
f = f{1};
f = replace(f,'^','.^');
f = replace(f,'*','.*');
f = str2func(['@(x_1,x_2)' f]);
[X,Y] = meshgrid(0:0.01:1,0:0.01:1);
F = f(X,Y);
mesh(X,Y,F);
hold on
scatter3(x1vec,x2vec,gvec, 'filled', 'red');
xlabel("x1vec");
ylabel("x2vec");
zlabel("gvec");
for i = 1:30
    plot3([x1vec(i) x1vec(i)], [x2vec(i) x2vec(i)], [gvec(i) ...
        replace(double(cp)'*mp, x, [x1vec(i), x2vec(i)])]);
end
hold off

sdisplay(value(obj));
sdisplay(double(cp)' * mp);
```