EECS 127: Optimization Models in Engineering

Tyler Zhu

January 28, 2021

"A good stock of examples, as large as possible, is indispensable for a thorough understanding of any concept, and when I want to learn something new, I make it my first job to build one."

– Paul Halmos.

These are course notes for the Spring 2021 rendition of EECS 127, Optimization Models in Engineering, taught by Professor Laurent El Ghaoui.

Contents

1 Tuesday, January 19th

1.1 Introduction

In this course, we'll be talking primarily about *optimization*. A standard form of optimization is the following:

$$p^* = \min_{\boldsymbol{x}} f_0(\boldsymbol{x}) \quad \text{subject to: } f_i(\boldsymbol{x}) \le 0,\ i = 1, \dots, m,$$

where

- vector $\boldsymbol{x} \in \mathbb{R}^n$ is the *decision variable*;
- $f_0 : \mathbb{R}^n \to \mathbb{R}$ is the *objective* function, or *cost*;
- $f_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \dots, m$, represent the *constraints*;
- p^* is the *optimal value.*

Realistically, $\boldsymbol{x} = [x_1 \ \dots \ x_n]$ represents different decisions, i.e. x_2 would be our decision at time $t = 2$. Also note that we can easily solve instead to maximize some $r(x)$ by setting $f_0(x) = -r(x)$. This above setup is known as the *standard form.*

Oftentimes we will have multiple optimal solutions to the constraint, in which case any $x^* \in \arg\min f_0(x)$ for which $f_i(x^*) \le 0, i = 1, \dots, m$ is satisfied acts as an optimizer.

In this class, we're not as concerned with algorithms for optimization, but more so translating problems from the real world into this language.

Example 1.1 (Least-squares regression)**.** A classic example in machine learning is when we have a given vector y and we're trying to express it as a linear function of an input vector z, i.e. data points. The goal is the solve the objective

$$\min_x \sum_{i=1}^m (y_i - x^\top z^{(i)})^2$$

where

- $z^{(i)} \in \mathbb{R}^n$, $i = 1, \dots, n$ are data points;
- $y \in \mathbb{R}^m$ is a "response" vector;
- $x^\top z$ is the scalar product $z_1 x_1 + \cdots + z_n x_n$ b/w the two vectors $x, z \in \mathbb{R}^n$.

One example of constraints we could be working with are $x \ge 0$ and $x^\top \mathbb{1} = 1$, which corresponds to modeling a discrete distribution.

Example 1.2 (Support Vector Machines (SVMs)). In SVMs, we instead are trying to optimize a "hinge" loss, i.e.

$$\min_{x,b} \sum_{i=1}^{m} \max(0, 1 - y_i(x^\top z^{(i)} + b))$$

where

- $z^{(i)} \in \mathbb{R}^n$, $i = 1, \ldots, n$ are data points;
- $y \in \{-1, 1\}^m$ is a *binary* response vector;
- $x^\top z + b = 0$ defines a "separating hyperplane" in data space.

We could imagine that our points are colored green and red. Then at a conceptual level, we're trying to create a hyperplane that separates our data points into two different classes as clearly as possible. Once we find the best x, b, we can predict the binary output $\hat{y}$ corresponding to a new point's predicted class.

While we just gave a few machine learning examples which were problems without constraints, we can often use optimization to act on certain situations. One example is energy production. There's a **lot** of other ones.

1.2 Optimization Problems

There is more nomenclature we need to know:

- *Feasible set*, i.e. the set of possible values satisfying the constraints.
- *Unconstrained minimizer*:x_0, i.e. minimizing the cost function without constraints.
- *Optimal Point*: x^*.
- *Level sets* of objective functions, i.e. sets $\{x|R(x) = c\}$ for some c.
- *Sub-level sets*, i.e. sets $\{x|R(x) \leq c\}$ for some c.

Usually our optimal points will be some intersection with the smallest level sets and the feasible set.

Similar to neural networks, we can have an issue in optimization of local vs. global optimal points. A point z is *locally optimal* if there exists a value $R > 0$ such that z is optimal for problem

$$\min_x \; f_0(x) \text{ s.t. } f_i(x) \leq 0, \; i = 1, \ldots, m \text{ and } |x_i - z_i| \leq R, \; i = 1, \ldots, n.$$

A local minimizer x minimizes f_0, but only compared to nearby points on the feasible set. The value of the objective function at that point is *not* necessarily the (global) optimal value of the problem. Locally optimal points might be of no practical interest to the user. Visually, you could imagine that we could find ourselves in certain pits that locally seem like optima but globally are far from it.

Due to these problems (and others), often times even coming up with any minimizer can be difficult. However, there's a special class of problems called *convex problems* which have the nice property that *all* local optimums are also global optimums. Usually your objective function when plotted looks something like a bowl, i.e. there's a single point at the bottom which is the previously mentioned global optimum.

Figure 1: Convex and Non-convex functions (missing!)

Funny enough, even though most problems are non-convex, we could find a convex function that lower bounds our objective function and hope its global optimum is a decent solution to our original objective. Pushing this further, if we could find the tightest convex function which is a lower bound (think convex hulls), then its optimal point *is* the same as our original's! It looks like we just turned a hard problem into a much easier one, but we unfortunately have no idea how to find this convex-hull. Back to square one.

1.3 Course Outline

In this course, we shall deal specifically with convex optimization problems with special structure, such as:

- Least-Squares (LS)
- Linear Programs (LP)
- Convex Quadratic Program (QP)
- Second-order cone programs (SOCP)

For such specific models, very efficient solution algorithms exist with high quality implementations/software (CVX, for example). Most of the problems we come across can be categorized into one of the above structures, as we'll come to see.

A large part of our time will be spent talking about affine subspaces, and normal vectors to hyperplanes to geometrically construct feasible sets.

We'll also discuss a few non-convex problems that come up often in the real world:

- *Boolean/integer optimization*: $x_i \in \{0,1\}^n$; some variables are constrained to be Boolean or integers. Convex optimization can be used for getting (sometimes) good approximations.
- *Cardinality-constrained problems*: we seek to bound the number of non-zero elements in a vector variable. Convex optimization can be used for getting good approximations.
- *Non-linear programming*: usually non-convex problems with differentiable objective and functions. Algorithms provide only local minima.

It goes without saying that most (but not all) non-convex problems are *hard*!

For example, let's look at boolean optimization. We would like to solve $\min_{\boldsymbol{x}} c^\top \boldsymbol{x}$ subject to $A\boldsymbol{x} \leq b$ and $\boldsymbol{x} \in \{0,1\}^n$. One way of relaxing this into a problem that's easier to solve is to consider instead $\boldsymbol{x} \in [0,1]^n$, i.e. going from discrete to continuous feasible set. This is an important question because this allows us to do sparsity and feature-selection.

What this course is for:

- Learning to model and efficiently solve problems arising in Engineering, Management, Control, Finance, ML, etc.
- Learning to prototype small- to medium- sized problems on numerical computing platforms.
- Learning basics of applied linear algebra, convex optimization.

What this course is NOT:

- A course on mathematical convex analysis.
- A course on details of optimization algorithms.

Here's a high level overview of what we're talking about:

- Linear algebra models
  - Vectors, projection theorem, matrices, symmetric matrices.
  - Linear equation, least-squares and minimum-norm problems.
  - Singular value decomposition (SVD), PCA, and related optimization problems.
- Convex optimization models
  - Convex sets, convex functions, convex problems.
  - Optimality conditions, duality.
  - Special convex models: LP, QP, SOCP.
- Applications
  - Machine Learning.
  - Control.
  - Finance.
  - Engineering desisgn.

There will only be six homeworks in this course.

2 Thursday, January 21st

Today's lecture went pretty fast and sped through sections, so likewise these notes are pretty rough. Also I don't have time to add pictures (unless I rip them off from somewhere), so I hope you have Beth Harmon level visualization powers.

2.1 Introduction

We usually write vectors in column format, i.e.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

Element x_i is the ith component, and the number n of components is the *dimension* of x. Importantly, in math, we always 1-index, not 0-index.

We can use vectors for example in bag of words frequency matching.

Example 2.1 (Time series). We can model a time series, i.e. the evolution of a physical or economical quantity. We can represent it like $x = [x(1)\ x(2)\ \dots\ x(T)]^\top \in \mathbb{R}^T$ where each $x(k)$ is the value of the quantity at time k.

One example of a model for time series is an "auto-regressive" model, where we assume that the output depends linearly on certain previous terms and some stochasticity (i.e. error). For example, if we think it only depends on the previous two days, we could write that

$$x_t \approx \alpha_1 x_{t-1} + \alpha_2 x_{t-2} + \text{error}.$$

Now for a list of definitions...

Vectors have a few special properties, in that the operations of sum, difference, and scalar multiplication all hold and are closed, i.e. that they return other vectors. These properties define a *vector space*. The simplest example is $\mathcal{X} = \mathbb{R}^n$.

From a vector space $\mathcal{X}$, a nonempty subset $\mathcal{V}$ of $\mathcal{X}$ is a *subspace* if, for any scalars α, β,

$$x, y \in \mathcal{V} \implies \alpha x + \beta y \in \mathcal{V}.$$

For example, the set of all possible linear combinations of vectors in $S = \{x^{(1)}, \dots, x^{(m)}\}$ forms a subspace called the *span* of S, denoted as $\text{span}(S)$.

Finally, given two subspaces $\mathcal{X}, \mathcal{Y}$ in $\mathbb{R}^n$, the *direct sum* of $\mathcal{X}, \mathcal{Y}$, denoted by $\mathcal{X} \oplus \mathcal{Y}$, is the set of vectors of the form $x + y$ with $x \in \mathcal{X}, y \in \mathcal{Y}$. One can easily check that $\mathcal{X} \oplus \mathcal{Y}$ is a subspace.

We also have the familiar concepts of linear independence and hence the *basis* for a subspace, whose cardinality defines the dimension of the subspace.

Definition 1 (Affine sets). An affine set is a set of the form

$$\mathcal{A} = \{x \in \mathcal{X} \mid x = v + x^{(0)}, v \in \mathcal{V}\},$$

where $x^{(0)}$ is a given point and $\mathcal{V}$ is a given subspace of $\mathcal{X}$. Subspaces are affine spaces containing the origin (i.e. $x^{(0)}$ is the origin).

Naturally, a *line* is a one-dimensional affine set. We start with some point x_0 which hinges the line, and then some vector v which determines the direction (i.e. including all multiples of it).

Quick quiz: how many values do we need to uniquely determine lines in 2D? You might think it's 2, but we need to identify the vertical line as well, so it's 3. Think of $ax + by + c = 0$.

One question we might have is how do we measure the *size* of a vector; norms are the answer to this question. In short, a *norm* simply assigns real numbers to vectors in a consistent manner, i.e. it satisfies the following properties:

Definition 2 (Norm).

We will primarily use three norms:

- For $p = 2$, we obtain the standard Euclidean length
$$\|x\|_2 = \sqrt{\sum_{k=1}^{n} x_k^2},$$
- or $p = 1$ which gives the sum-of-absolute-values length (or Manhattan distance)
$$\|x\|_1 = \sum_{k=1}^{n} |x_k|.$$
- When $p = \infty$ defines the ℓ_∞ norm (max absolute value norm)
$$\|x\|_\infty = \max_{k=1,\ldots,n} |x_k|.$$
- The cardinality of a vector x is often called the ℓ_0 (pseudo-) norm, which

2.2 Inner product, orthogonality

Inner products are interesting since we can think of vectors now as functions acting on other vectors, i.e. like $\langle x, \cdot \rangle$.

Definition 3 (Inner product). An *inner product* on a (real) vector space $\mathcal{X}$ is a real-valued function which maps pairs $x, y \in \mathcal{X}$ into a scalar denoted by $\langle x, y \rangle$. The inner product satisfies the following axioms (for $x, y, z \in \mathcal{X}$ and scalar α):
$$\langle x, x \rangle \geq 0;$$

A vector space with an inner product is called an *inner product space*, and the standard inner product in $\mathbb{R}^n$ is the row-column product of two vectors,
$$\langle x, y \rangle = x^\top y = \sum_{k=1}^{n} x_k y_k.$$

Finally, inner products also induce a norm given by $\|x\| = \sqrt{\langle x, x \rangle}$ (which you may notice is identical to the ℓ_2-norm in $\mathbb{R}^n$).

With inner products, we can define the angle θ between two vectors x, y by
$$\cos\theta = \frac{x^\top y}{\|x\|_2 \|y\|_2},$$
which results from doing some easy geometry. This lets us characterize when x, y are *orthogonal* (i.e. $\theta = \pm 90°$), and when x, y are *parallel* (i.e. $\theta = 0°, \pm 180°$).

Since $|\cos\theta| \leq 1$, we easily get the following inequality.

Theorem 4 (Cauchy-Schwarz Inequality)**.** For vectors $x, y \in \mathbb{R}^n$,

$$|x^\top y| \leq \|x\|_2 \|y\|_2$$

where inequality holds when $x = \alpha y$ for scalar α (i.e. when $|\cos\theta| = 1$).

We also have the following generalization with ℓ_p norms.

Theorem 5 (Holder's Inequality)**.** For any vectors $x, y \in \mathbb{R}^n$ and for any $p, q \geq 1$ such that $1/p + 1/q = 1$, it holds that

$$|x^\top y| \leq \sum_{k=1}^{n} |x_k y_k| \leq \|x\|_p \|y\|_q.$$

As a quick aside, let's discuss maximizing the inner product over norm balls. In other words, we're trying to solve the optimization problem

$$\max_{\|x\|_p \leq 1} x^\top y.$$

For $p = 2$, ...

For generic inner product spaces, we say that two vectors x, y in an inner product space $\mathcal{X}$ are *orthogonal* if $\langle x, y \rangle = 0$, which we denote by $x \perp y$. Nonzero vectors are *mutually orthogonal* if they are pairwise orthogonal.

We have an important theorem concerning projections.

Theorem 6 (Projection Theorem)**.** Let $\mathcal{X}$ be an inner product space, x be a given element in $\mathcal{X}$, and $\mathcal{S}$ a subspace of $\mathcal{X}$. Then, there exists a unique vector $x^* \in \mathcal{S}$ which is the solution to the problem

$$\min_{y \in \mathcal{S}} \|y - x\|.$$

Moreover, a necessary and sufficient condition for x^* being the optimal solution for this problem is that $x^* \in \mathcal{S},\ (x - x^*) \perp \mathcal{S}$.

Let $p \in \mathbb{R}^n$ be a given point. We want to compute the Euclidean projection p^* of p onto a line $L = \{x_0 + \text{span}(u)\}$, $\|u\|_2 = 1$.

Now say we want to solve the harder problem of projecting onto a span of vectors, say $\mathcal{S} = \text{span}(x^{(1)}, \ldots, x^{(d)}) \subseteq \mathcal{X}$. By the projection theorem, we can convert this into a system of equations and solve.

2.3 Functions and maps

In this class, we usually reserve the term *function* to denote $f : \mathbb{R}^n \to \mathbb{R}$, and use the term *map* when $f : \mathbb{R}^n \to \mathbb{R}^m$ and $m > 1$.

A *linear* function is simply a function that perserves scaling and additivity, i.e. that

$$f(\alpha x) = \alpha f(x) \quad \forall x \in \mathbb{R}^n, \alpha \in \mathbb{R}$$
$$f(x + y) = f(x) + f(y) \quad \forall x, y \in \mathbb{R}^n.$$

The gradient of a function can be intepreted in the context of level sets. Geometrically, the gradient of f at a point x_0 is a vector $\nabla f(x_0)$ perpendicular to the contour line of f at level $\alpha = f(x_0)$, pointing from x_0 outwards the α-sublevel set (i.e. points towards higher values of the function).

3 Tuesday, January 26th: Matrices and Linear Maps

3.1 Matrix Basics

In this class, we will use the convention that if A is a data matrix, then every column of A is a single data point. Also, note that $x^\top Ay = y^\top A^\top x$, since the inner product is symmetric and the result is a scalar. This will be useful later on when we discuss duality.

There's many ways we can think of a matrix-vector product. Let $A \in \mathbb{R}^{m,n}$ with columns $a_1, \dots, a_n \in \mathbb{R}^m$, and $b \in \mathbb{R}^n$ a vector. Then we can define the product by

$$Ab = \sum_{k=1}^{n} a_k b_k,$$

which is essentially just a weighted sum of the columns of A with the elements of b as coefficients.

We can do the same definition for multiplication on the right by A. Let $\alpha_1, \dots, \alpha_m$ be the rows of A. If instead we had some $c \in \mathbb{R}^m$, we can multiply on the left by it's transpose to get

$$c^\top A = \sum_{k=1}^{m} c_k \alpha_k^\top,$$

which says that this is a weighted sum of the rows α_k of A using the elements of c as coefficients.

From matrix-vector products, we can then define matrix products. We can view a matrix $A \in \mathbb{R}^{m,n}$ in two different ways; either as a collection of columns or rows.

$$A = \begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix}, \text{ or } A = \begin{bmatrix} \alpha_1^\top \\ \alpha_2^\top \\ \vdots \\ \alpha_m^\top \end{bmatrix},$$

where $a_1, \dots, a_n \in \mathbb{R}^m$ denote the columns of A and $\alpha_1^\top, \dots, \alpha_m^\top \in \mathbb{R}^n$ denote the rows of A.

We have three main interpretations of matrix-matrix products. One is obtained by letting A act as a linear map on the columns of B, where $B = \mathbb{R}^{n,p}$ and $B = [b_1 \ \dots \ b_p]$. This gives

$$AB = A \begin{bmatrix} b_1 & \dots & b_p \end{bmatrix} = \begin{bmatrix} Ab_1 & \dots & Ab_p \end{bmatrix}.$$

Similarly, we can also view the product as the linear map B operating on the rows of A, which gives

$$AB = \begin{bmatrix} \alpha_1^\top \\ \vdots \\ \alpha_m^\top \end{bmatrix} B = \begin{bmatrix} \alpha_1^\top B \\ \vdots \\ \alpha_m^\top B \end{bmatrix}.$$

Finally, we can also write AB as a sum of *dyadic* matrices (i.e. rank one matrices of the form $uv^\top$). Letting $\beta_i^\top$ denote the rows of B, we can write

$$AB = \sum_{i=1}^{n} a_i \beta_i^\top.$$

3.2 Matrices as linear maps

Assume that A is an $m \times n$ matrix unless stated otherwise. Recall that we can interpret matrices as *linear maps*, i.e. maps where $f(ax + by) = af(x) + bf(y)$ for scalars a, b. Affine maps are simply linear functions plus a constant, i.e. $f(x) = Ax + b$.

Also recall that we have the familiar concepts of range, rank, and nullspace. Specifically, the set of linear combinations of the columns a_i's of a matrix A are of the form Ax, for $x \in \mathbb{R}^n$. We call this the *range* of A, and denote it as

$$\mathcal{R}(A) = \{Ax | x \in \mathbb{R}^n\}.$$

The dimension of $\mathcal{R}(A)$ (i.e. cardinality of a basis for A or the number of lin. ind. rows of A) is called the *rank* of A. We also have that $\text{rank}(A) = \text{rank}(A^\top)$.

On the other hand, the null space of A gives the vectors which are mapped to zero by A (i.e. the *kernel*) and is denoted by

$$\mathcal{N}(A) = \{x \in \mathbb{R}^n | Ax = 0\}.$$

The null space of a matrix gives you the "ambiguity" of the solutions to $Ax = 0$. For any two solutions $Ax = 0$ and $Ay = 0$, $A(x + y) = A(x - y) = 0$, so $x + y, x - y \in \mathcal{N}(A)$ for example.

Importantly, $\mathcal{R}(A^\top)$ and $\mathcal{N}(A)$ are mutually orthogonal subspaces, i.e. $\mathcal{N}(A) \perp \mathcal{R}(A^\top)$, meaning that every vector in one space is orthogonal to every vector in the other. This leads us to the following fundamental theorem.

Theorem 7 (Fundamental Theorem of Linear Algebra)**.** For any given matrix $A \in \mathbb{R}^{m,n}$, it holds that $\mathcal{N}(A) \perp \mathcal{R}(A^\top)$ and $\mathcal{R}(A) \perp \mathcal{N}(A^\top)$, hence

$$\mathcal{N}(A) \oplus \mathcal{R}(A^\top) = \mathbb{R}^n$$
$$\mathcal{R}(A) \oplus \mathcal{N}(A^\top) = \mathbb{R}^m.$$

Consequently, we can decompose any vector $x \in \mathbb{R}^n$ into a vector in the range of $A^\top$ and another in the nullspace of A:

$$x = A^\top \xi + z, \quad z \in \mathcal{N}(A).$$

Similarly, we can decompose any vector $w \in \mathbb{R}^m$ into a vector in the range of A and another in the nullspace of $A^\top$:

$$w = A\varphi + \zeta, \quad \zeta \in \mathcal{N}(A^\top).$$

This theorem makes much more sense geometrically. We did a derivation of least squares using the fundamental theorem which is cool, and I might put in when I find time.

3.3 Matrix Concepts

Things we didn't cover in lecture that I need to review and note on (for my own sake): determinants, inverses, similar matrices, eigenvalues and diagonalizability, ...

Some matrices with special structure:

- Square, diagonal, triangular (upper or lower).
- Symmetric: a square matrix A such that $A = A^\top$.
- Orthogonal: a square matrix A such that $AA^\top = A^\top A = I$. These are interesting since this implies $A^{-1} = A^\top$. So for any vector x, $\|Ax\|_2^2 = x^\top A^\top A x = x^\top x = \|x\|_2^2$, so orthogonal matrices don't change the norm of x. Hence, they actually represent rotations.
- Dyads (i.e. "outer products"): matrices of the form $uv^\top$. Let $A = uv^\top$. Then the (i, j)-th entry of A is
$$A_{ij} = e_i^\top (uv^\top) e_j = (e_i^\top u)(v_j^\top e_j) = u_i y_j,$$
(where e_i, e_j are the elementary vectors) which is reminiscent of the inner product.

Outer products are quite useful in practice, for multiple reasons. If you have a data matrix like a time series, and observe that it is close to an outer product, then you can tell that you have factors which are scaled versions of each other.

We can also find what the rank of an outer product is. If $A = xy^\top$, then an element of its range looks like $Az = (xy^\top)z = x(y^\top z)$. This last term is a scalar (as it's an inner product), so all outputs are scaled versions of x, and thus A is rank one. In fact, all dyads are rank one.

4 Thursday, January 28th

4.1 Orthogonalization: Gram-Schmidt

Recall that a basis $(u_i)_{i=1}^n$ is said to be *orthogonal* if $u_i^\top u_j = 0$ for $i \neq j$. If $\|u_i\|_2 = 1$, the basis is said to be *orthonormal.*

Orthogonalization is the process where we find an orthonormal basis of the span of given vectors (which could be linearly dependent!). Specifically, given vectors $a_1, \ldots, a_k \in \mathbb{R}^n$, an orthogonalization procedures computes vectors $q_1, \ldots, q_n \in \mathbb{R}^n$ such that

$$S := \text{span}\{a_1, \ldots, a_k\} = \text{span}\{q_1, \ldots, q_r\},$$

where $r = \dim S$ and $(q_1, \ldots, q_r)$ is an orthonormal basis for S.

One useful concept for this process is that of a projection onto a line. We have a point $a \in \mathbb{R}^n$ which we want to project onto a line $L(q) := \{tq : t \in \mathbb{R}\}$, where q is a unit vector in $\mathbb{R}^n$. This amounts to solving the optimization problem

$$\min_{t\in\mathbb{R}} \|a - tq\|_2$$

for the closest point on $L(q)$ to a. The resulting vector $a_{\text{proj}} := t^* q$ (t^* being the optimal value) is this projection onto the line $L(q)$. This optimization problem has a simple solution (by geometry):

$$a_{\text{proj}} = (q^\top a)q.$$

A simple interpretation of this is that we are *removing the component* of a along q, since $a - a_{\text{proj}}$ and a_{proj} are orthogonal. Gram-Schmidt uses this idea to then create a set of vectors which are orthogonal to each other. The big idea is to first orthogonalize each vector w.r.t. the previous ones then normalize it to have norm one.

Theorem 8 (Gram-Schmidt Procedure). Let $a_1, \ldots, a_n$ be linearly independent. The Gram-Schmidt procedure below will output an orthonormal basis for $\text{span}\{a_1, \ldots, a_n\}$.

- Set $\tilde{q}_1 = a_1$.
- Normalize: set $q_1 = \tilde{q}_1/\|\tilde{q}_1\|_2$.
- Remove component of q_1 in a_2:: set $\tilde{q}_2 = a_2 - (a_2^\top q_1)q_1$.
- Normalize: set $q_2 = \tilde{q}_2/\|\tilde{q}_2\|_2$.
- Remove component of q_1, q_2 in a_3: set $\tilde{q}_3 = a_3 - (a_3^\top q_1)q_1 - (a_3^\top q_2)q_2$.
- Normalize: set $q_3 = \tilde{q}_3/\|\tilde{q}_3\|_2$.
- etc.

We start by projecting each a_i onto the previous normalized basis vectors, which makes it orthogonal to them. Then we normalize so that our basis becomes orthonormal.

This is well-defined, since if $\tilde{q}_i$ is ever 0, this means that the vectors are linearly dependent, which we assumed against. This means we can easily modify the above to produce a basis in the linearly dependent case; simply skip a_i if the result $\tilde{q}_i$ turns out to be 0.

4.2 QR Decomposition

The basic goal of QR decomposition is to factor a matrix as a product of two matrices which hopefully have simple structure to exploit. This will make it easier to solve equations $Ax = y$, since we can instead analyze $Cx = z$ and then $Bz = y$ as

$$Ax = B(Cx) = Bz = y.$$

For example, if we could factor $A = BC$ where B is some rotation, then we could easily solve this by taking its inverse rotation.

The QR decomposition is just the Gram-Schmidt procedure applied to the columns of the matrix, with the result applied in matrix form.

Consider an $m \times n$ matrix $A = (a_1, \dots, a_n)$ with columns a_i, and assume the columns are linearly independent. Then A is full column-rank, so each step of G-S can be written as

$$a_i = (a_i^\top q_1)q_1 + \cdots + (a_i^\top q_{i-1})q_{i-1} + \|\tilde{q}_i\|_2 q_i, \quad i = 1, \dots, n$$

where recall that $\tilde{q}_i = a_i - (a_i^\top q_1)q_1 - \cdots - (a_i^\top q_{i-1})q_{i-1}$ (the remaining components of a_i, per se). We write this as

$$a_1 = r_{i1}q_1 + \cdots + r_{i,i-1}q_{i-1} + r_{ii}q_i, \quad i = 1, \dots, n,$$

where $r_{ij} = (a_i^\top q_j)$ for $1 \le j \le i-1$ and $r_{ii} = \|\tilde{q}_i\|_2$.

Since we got this output from G-S, we know the q_i's form an orthonormal basis, so the matrix $Q = (q_1, \dots, q_n)$ satisfies $Q^\top Q = \mathbf{I}_n$ (i.e. is an *orthonormal* matrix). The QR decomposition thus allows us to write the matrix in a *factored* form

$$A = QR, \quad Q = \begin{bmatrix} q_1 & \dots & q_n \end{bmatrix}, \quad R = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & & r_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & & 0 & r_{nn} \end{pmatrix}$$

where Q is an $m \times n$ matrix with $Q^\top Q = \mathbf{I}_n$ and R is $n \times n$ upper-triangular.

With this decomposition, this makes it very efficient to solve linear equations, since any equation of the form $Ax = y$ can be written as $QRx = y$. We can then multiply by $Q^\top$ to get $Rx = Q^\top y$, and R is upper-triangular, so we can efficiently solve this using back-substitution.

When the columns of A are not independent, G-S simply skips over any zero vectors $\tilde{q}_j$ that we encounter. In matrix form, this means we obtain a factorization $A = QR$, with $Q \in \mathbb{R}^{m \times r}$, $r = \text{rank}(A)$, and R having an upper staircase form like

$$R = \begin{pmatrix} * & * & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & 0 & 0 & * \end{pmatrix}$$

This is just as simple for us to solve with back-substitution, but we can also permute the columns of R so that we bring all the pivot entries to thhe front.

4.3 Kernels in Machine Learning

Recall that in least-squares, our training problem is solving the optimization problem

$$\min_{w,b} \mathcal{L}(X^\top w + b \cdot 1, y),$$

where the pairs of poinits (X_i, y_i) with $X_i \in \mathbb{R}^n, y_i \in \mathbb{R}$ form our training set.

Our predictions are then given by $\hat{y}(x) = w^\top x + b$ once we find these optimal w, b. This means we can compute an error between the *actual* measured output y_i with our predictions $\hat{y}_i$, which forms our loss function that we try to optimize:

$$\min_{w,b} \sum_{i=1}^{n} (y_i - (w^\top x_i + b))^2.$$

The shape of our data is $X = \begin{bmatrix} x_1 & \dots & x_m \end{bmatrix}$, i.e. an $n \times m$ matrix, so

$$X^\top w + b \cdot 1 = \begin{bmatrix} x_1^\top \cdot w + b \\ \vdots \\ x_m^\top \cdot w + b \end{bmatrix}.$$

Analyzing the term $X^\top w$ is the key to understanding the impact of kernels.

Suppose we simplify our problem to

$$\min_{w} \mathcal{L}(X^\top w) + \lambda \|w\|_2^2$$

where $\lambda > 0$ is a penalty/regularization term on our weights trying to force our training process to find a more "robust example for weights w." It helps account for noise in your dataset, so that small changes in the x_i won't result in large changes in our loss.

Let's analyze the simple case $m = 2$ (# of data points) and $n = 3$ (dimension of data points). Invoking the fundamental theorem of linear algebra, we can write our weight vector in terms of a vector in the span of our x_1, x_2, i.e. $w = Xv + r$ where $v \in \text{span}\{x_1, \dots, x_m\}$ and $r \in \mathcal{N}(X^\top)$, so $X^\top r = 0$. So $X^\top w = X^\top X v + X^\top r = X^\top X v$, which let's us rewrite the optimization problem as

$$\min_{w,v} \mathcal{L}[X^\top X v, y] + \lambda v^\top X^\top X v.$$

In other words, we've converted the optimization in terms of X instead into an optimization in terms of the matrix $X^\top X$. We refer to this matrix as the *kernel* matrix $K = X^\top X$, or the Gram matrix of our data. Note that

$$K_{ij} = e_i^\top K e_j = e_i^\top X^\top X e_j = x_i^\top x_j = \langle x_i, x_j \rangle.$$

This inspires us to think of other inner products that aren't just the usual scalar product $\langle x_i, x_j \rangle = x_i^\top x_j$. We could use nonlinear products instead which let use use richer models on transformed data instead.