Constrained Optimisation

* So far we’ve looked at unconstrained optimisation, e.g. minimise \( f(x) = x^2 \):

\[ x = 0 \] minimises \( f(x) \)

* What if the allowed choices of \( x \) are constrained e.g. \( 0.25 \leq x \leq 1 \):

now \( x = 0.25 \) minimises \( f(x) \) for \( x \geq 0.25 \)

* Adding constraints can change the value of \( x \) that is the minimiser
Constrained Optimisation

Notation

* Unconstrained optimisation:

\[
\min_x f(x)
\]

* Constrained optimisation

\[
\min_{x \in X} f(x)
\]

with \(X\) the set of allowed values for vector \(x\) e.g.

\[X = \{x \in \mathbb{R} : x \geq 0.25\}\] or \(X = \{x \in \mathbb{R}^2 : 0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1\}\)

* Curly brackets \(\{\}\) indicate its a set
* \(\mathbb{R}^2\) superscript 2 indicates that \(x\) is a vector with 2 elements, \(\mathbb{R}\) means the elements are real valued
* : reads as “such that”
* So first example read: \(x\) is real-valued such that \(x \geq 0.25\)
* Second example reads: \(x\) is a vector with 2 elements such that \(0 \leq x_1 \leq 1\) and \(0 \leq x_2 \leq 1\)
* See e.g. https://en.wikipedia.org/wiki/Set-builder_notation for set notation

* Special case is \(X = \mathbb{R}^n\). The \(n\) superscript means its a vector with \(n\) elements, \(\mathbb{R}\) means the elements are real-valued. Then we’re back to an unconstrained optimisation and usually just drop \(X\) i.e. write

\[
\min_{x \in \mathbb{R}^n} f(x) \text{ or } \min_x f(x)
\]
» Change of Variables

Sometimes (if we’re lucky) we can directly convert a constrained optimisation into an unconstrained optimisation

* Example: Suppose $f(x) = (x + 2)^2$ and we require $x$ to be non-negative i.e. $X = \{x \in \mathbb{R} : x \geq 0\}$. Make a change of variable:

  * Define $x = e^z$ and new function $g(z) = (e^z + 2)^2$. As $z$ varies between $-\infty$ and $+\infty$, $x = e^z$ varies between 0 and $+\infty$ i.e. $x \in X$.

* Solving unconstrained optimisation

$$\min_z g(z) = (e^z + 2)^2$$

is now the same as solving constrained optimisation

$$\min_{x \geq 0} f(x) = (x + 2)^2$$
∗ \( f(x) = (x + 2)^2, \ g(z) = (e^z + 2)^2 \). Gradient descent, constant step size \( \alpha = 0.05 \):

\[ \begin{array}{c}
\text{Left-hand plot: see that } z \text{ updates so as to decrease } g(z). \ g(z) \text{ is minimised by } z \to -\infty \text{ since } g(-\infty) = 2^2 = 4.
\end{array} \]

\[ \begin{array}{c}
\text{Right-hand plot: see that } x = e^z \text{ heads to 0, but never goes negative (so stays within admissible set } X). \text{ Function } f(x) \to 2^2 = 4.
\end{array} \]

∗ If no constraints then minimum would be \( f(0) = 0 \) when \( x = -2 \), but \( x = -2 \) lies outside set \( X \).
Suppose we require $x$ to be between 0 and 1 i.e. $X = \{x \in \mathbb{R} : 0 \leq x \leq 1\}$. Make change of variable: $x = \frac{1}{1 + e^{-z}}$

As $z$ varies between $-\infty$ and $+\infty$, $x = \frac{1}{1 + e^{-z}}$ varies between 0 and 1 i.e. $x \in X$.

Example: suppose $f(x) = (x + 2)^2$ then $g(z) = \left(\frac{1}{1 + e^{-z}} + 2\right)^2$. Solving unconstrained optimisation

$$\min_z g(z) = \left(\frac{1}{1 + e^{-z}} + 2\right)^2$$

is now the same as solving constrained optimisation

$$\min_{0 \leq x \leq 1} f(x) = (x + 2)^2$$
Change of Variables

- Usually there no free lunch in optimisation ....
- By adding constraints we might expect that we make the optimisation problem “harder” i.e. it will take longer to find minimiser

\[ f(x) = (x + 2)^2 \] is minimised by \( x = -2 \), \( g(z) = (e^z + 2)^2 \) is minimised by \( z = -\infty \).

- Recall quadratic-like cost functions (strongly-convex cost functions) like \( f(x) = (x + 2)^2 \) are easy/fast to minimise
- But see that \( g(z) \) has a large flat section on left-hand side of plot where gradient is getting smaller and smaller \( \rightarrow \) gradient descent algos will tend to converge slowly in this region.

- Our change of variables has converted a strongly-convex optimisation into a harder one which is not strongly-convex
Change of variables can also make a convex (“easy”) optimisation into a non-convex (“hard”) one ...

E.g. suppose \( f(x) = 0.02x^2 - x - 1 \) and we change variable so \( x = e^z \). Then \( g(z) = 0.02(e^z)^2 - e^z - 1 \):

See that \( g(z) \) is non-convex even though \( f(x) \) is convex. The non-convexity is benign in this example (still just one global minimum, gradient descent will find it), but needn’t always be ...
Another example. Suppose \( f(x) = 0.02x^2 - x - 1 \) again but now we change variable to \( x = z^2 \).

As \( z \) varies between \(-\infty\) and \(+\infty\), \( z^2 \geq 0 \).

\[ g(z) = 0.02z^4 - z^2 - 1: \]

\[ \Rightarrow \]

See that \( g(z) \) has two minima even though \( f(x) \) only has one \( \rightarrow \) that’s because \( 5^2 = (-5)^2 \)

This is why tend to prefer \( e^z \) rather than \( z^2 \) as change of var to ensure \( x \geq 0 \)
Projected Gradient Descent

* Usually we’re not so lucky and can’t just make a change of vars.

* Recall iterative gradient descent algorithm to minimise function $f(x)$:

  ```python
  for k in range(num_iters):
      step_t = $\alpha \left[ \frac{\partial f}{\partial x_1}(x_t), \frac{\partial f}{\partial x_2}(x_t), \ldots, \frac{\partial f}{\partial x_n}(x_t) \right]$
      x_{t+1} = x_t - step_t
  ```

* **Projected gradient descent**: changes $x_{t+1} = x_t - \text{step}_t$ to:

  $$z_{t+1} = x_t - \text{step}_t$$

  $$x_{t+1} \in \arg \min_{x \in X} d(z_{t+1}, x)$$

* Here $\arg \min_{x \in X} d(z_{t+1}, x)$ is the set of $x$ values (there might be more than one) that minimise function $d(z_{t+1}, x)$.

* Function $d(z, x)$ measures the distance between $z$ and $x$ e.g. Euclidean distance

  $$d(z, x) = \sum_{i=1}^{n} (z_i - x_i)^2$$
**Projected Gradient Descent**

* **Projected gradient descent**: changes $x_{t+1} = x_t - step_t$ to:

$$
\begin{align*}
z_{t+1} & = x_t - step_t \\
x_{t+1} & \in \text{arg min}_{x \in X} d(z_{t+1}, x)
\end{align*}
$$

* $\text{arg min}_{x \in X} d(z_{t+1}, x)$ is the set of $x$ values (there might be more than one) that minimise $d(z_{t+1}, x)$

* $d(z, x)$ measures distance between $z$ and $x$ e.g. $d(z, x) = \sum_{i=1}^{n} (z_i - x_i)^2$

* $z_{t+1}$ might lie outside set $X$ is allowed values, so we choose $x_{t+1}$ to be the value in $X$ that is closest to $z_{t+1}$, e.g.

* **Notation**: usually simplified to: $x_{t+1} = P_X(x_t - step_t)$, called the *projection* of $x_t - step_t$ onto $X$. 
Examples Of Fast Projections

* In general, calculating $P_X(x_t - step_t)$ means solving an optimisation problem $\rightarrow$ computationally expensive and slow

* But in some common special cases we can write the answer directly. E.g:

* $X = \{x \in \mathbb{R} : x \geq 0\}$, $d(x, z)$ is Euclidean distance. Then

$$P_X(z) = \begin{cases} 
  z & z \geq 0 \\
  0 & z < 0 
\end{cases}$$

$\rightarrow$ projection onto the set of positive values

* $X = \{x \in \mathbb{R} : a \leq x \leq b\}$, $d(x, z)$ is Euclidean distance. Then

$$P_X(z) = \begin{cases} 
  z & a \leq z \leq b \\
  a & z < a \\
  b & z > b 
\end{cases}$$

$\rightarrow$ projection onto interval $[a, b]$
Examples Of Fast Projections

- $x$ is a vector $x = [x_1, x_2, \ldots, x_n]$
- Have element-wise constraints $a_1 \leq x_1 \leq b_1$, $a_2 \leq x_2 \leq b_2, \ldots, a_n \leq x_n \leq b_n$.
- $d(x, z)$ is Euclidean distance.
- Then just separately project each element $x_i$ onto interval $[a_i, b_i]$ i.e
  
  $$[x_1, x_2, \ldots, x_n] = P_X([z_1, z_2, \ldots, z_n])$$

  means

  $$x_i = \begin{cases} 
  z_i & a_i \leq z_i \leq b_i \\
  a_i & z_i < a_i \\
  b_i & z_i > b_i 
  \end{cases}$$

  for $i = 1, 2, \ldots, n$

  $\rightarrow$ projection onto a (hyper)cube
Examples Of Fast Projections

- *projection onto a (hyper)cube*
- e.g. $x = [x_1, x_2, x_3]$ and $a_i = -5$, $b_i = +5$, $i = 1, 2, 3$:

![Diagram showing projection onto a cube]

- Blue dot is projected onto the closest point (green dot) on face of cube that is nearest to it
Examples Of Fast Projections

- What if sides of cube are not aligned with the axes?
- Simpler case first: suppose we restrict $x$ to be on one side of a line, e.g. $x_1 \leq x_2$:

  - Blue dot is projected onto closest point on boundary (marked by green dot) → *Projection Onto Half-Plane*
Examples Of Fast Projections

* When \( x = [x_1, x_2, \ldots, x_n] \) the general equation of a boundary line/plane:
  \[
  a^T x \leq b
  \]
  where \( a \) is some vector and \( b \) is a scalar

* E.g.
  * \( x = [x_1, x_2], a = [1, -1], b = 0 \) corresponds to \( x_1 - x_2 \leq 0 \) i.e. \( x_1 \leq x_2 \)
  * \( x = [x_1, x_2], a = [1, 0], b = 1 \) corresponds to \( x_1 \leq 1 \)
  * \( x = [x_1, x_2], a = [-1, 0], b = -1 \) corresponds to \( -x_1 \leq -1 \) i.e. \( x_1 \geq 1 \)

* \( X = \{ x \in \mathbb{R}^n : a^T x \leq b \} \), \( d(x, z) \) is Euclidean distance. Then

\[
P_X(z) = \begin{cases} 
  z & a^T z \leq b \\
  z - \frac{a^T z - b}{a^T a} a & a^T z > b
\end{cases}
\]

and recall \( a^T a = \sum_{i=1}^{n} a_i^2 \), \( a^T z = \sum_{i=1}^{n} a_i z_i \)

→ Projection Onto Half-Plane
Examples Of Fast Projections

- \( X = \{ x \in \mathbb{R}^n : a^T x \leq b \} \), \( d(x, z) \) is Euclidean distance. Then

\[
P_X(z) =\begin{cases} 
  z & a^T z \leq b \\
  z - \frac{a^T z - b}{a^T a} a & a^T z > b
\end{cases}
\]

and recall \( a^T a = \sum_{i=1}^n a_i^2 \), \( a^T z = \sum_{i=1}^n a_i z_i \)

- E.g. \( x = [x_1, x_2] \), \( a = [1, 0] \), \( b = 1 \) then \( a^T x = [1, 0]^T [x_1, x_2] = x_1 \) and constraint is \( x_1 \leq 0 \).

- Projection is

\[
P_X(z) =\begin{cases} 
  z & z_1 \leq 1 \\
  z - \frac{z_1 - 1}{1} [1, 0] = [1, z_2] & z_1 > 1
\end{cases}
\]

i.e. if first element of \( z \) is greater than 1 we set it equal to 1

- Formula above also works when line/plane is not aligned with the axes e.g. when \( x_1 \leq x_2 \). Then \( a = [1, -1] \), \( b = 0 \). Will leave that to you to try out ...
Examples Of Fast Projections

* A polytope is created from several intersecting lines e.g.

* red line: $x_2 = -x_1$, blue line: $x_1 = x_2$, yellow line: $x_1 = -1$

* Intersection of constraints $x_2 \leq -x_1$, $x_1 \leq x_2$, $x_1 \geq -1$ is the shaded triangle in the middle.

* By combining more lines we can make other shapes e.g. hexagon.

* To project onto polytope just repeatedly use previous projection on half-plane formula (once for each boundary line of polytope)
Special case: *Projection Onto Simplex*

\[ X = \{ x \in \mathbb{R}^n : \sum_{i=1}^{n} x_1 \leq 1, x_1 \geq 0, x_2 \geq 0, \ldots, x_n \geq 0 \} \]

- Vectors in simplex $X$ can be thought of as probability vectors (non-negative and elements sum to 1)
- See [https://lcondat.github.io/publis/Condat_simplexproj.pdf](https://lcondat.github.io/publis/Condat_simplexproj.pdf) for a fast algo
Example

* $f(x) = x^2$ and we require $x$ to be less than $-1$ i.e. $X = \{x \in \mathbb{R} : x \leq -1\}$. Projected gradient with constant step size $\alpha = 0.1$, initial $x = -3$:

See that $x$ increases until $z_t = x_t - \text{step}_t$ starts to become bigger than $-1$, then $x_{t+1}$ is snapped back to $x_{t+1} = -1$ by the projection, and thereafter stays there.

* And with $X = \{x \in \mathbb{R} : x \geq +1\}$, initial $x = +3$:
* Toy neural network and we require $x_1$ to be greater than 0.5 i.e. 
$X = \{ x \in \mathbb{R}^2 : x_1 \geq 0.5 \}$. Projected gradient with constant step size 
$\alpha = 0.75$, initial $x = [1, 1]$:
Summary: Projected Gradient Descent

* If can efficiently compute projection onto feasible set $X$, then can use projected gradient descent to solve constrained optimisations

* We just looked at constant step sizes, what about using Polyak, Adagrad, RMSprop, Heavy Ball, Nesterov Acceleration, Adam with projected gradient descent?

* Adagrad can be used directly\(^1\) with projected gradient descent

* Nesterov acceleration also carries over directly. Recall with gradient descent we used:

$$
z_{t+1} = \beta z_t - \alpha \nabla f(x_t + \beta z_t), \quad x_{t+1} = x_t + z_{t+1}
$$

See that $z_t = x_t - x_{t-1}$ and define $y_t = x_t + \beta z_t$. Then equivalently

$$
y_t = x_t + \beta(x_t - x_{t-1}), \quad x_{t+1} = y_t - \alpha \nabla f(y_t)
$$

With projected gradient descent use:

$$
y_t = x_t + \beta(x_t - x_{t-1}), \quad x_{t+1} = P_X(y_t - \alpha \nabla f(y_t))
$$

(sometimes called FISTA “Fast Iterative Shrinkage-Thresholding Algorithm” due to paper where originally proposed)

* V little (no?) work on use of Polyak step size, RMSprop, Heavy Ball, Adam with projected gradient descent

\(^1\)https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf