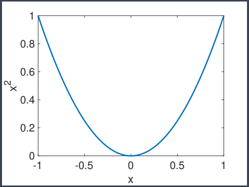
» Constrained Optimisation: Penalty Methods

* Suppose we want to solve $\min_{x \in X} x^2$ with $X = \{x \in \mathbb{R} : x \leq -0.5\}$



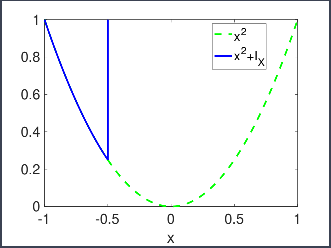
* Could use change of variables or projected gradient descent.

» Penalty Methods: Indicator Function

- *Another idea ...*
- Define penalty function $I_X(x) = \begin{cases} 0 & x \in X \\ +\infty & x \notin X \end{cases}$
- E.g. when $X = \{x \in \mathbb{R} : x \leq -0.5\}$ then $I_X(x) = \begin{cases} 0 & x \leq -0.5 \\ +\infty & x > 0.5 \end{cases}$
- Then the solution to optimisation

$$\min_x (x^2 + I_X(x))$$

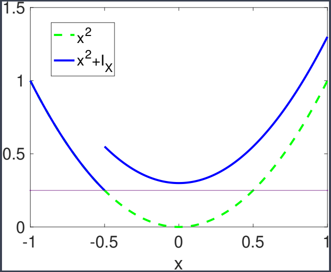
is the same as the solution to $\min_{x \in X} x^2$



» Penalty Methods: Indicator Function

- We don't have to se a penalty of $+\infty$ when x is outside admissible set X, the penalty value just has to be "big enough"
- E.g. $\min_{x \in X} x^2$ with $X = \{x \in \mathbb{R} : x \leq -0.5\}$. Try

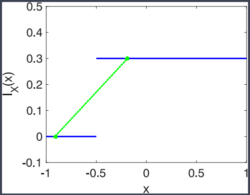
$$I_X(x) = \begin{cases} 0 & x \leq -0.5 \\ +0.3 & x > 0.5 \end{cases}$$



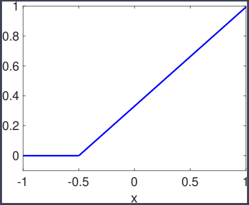
- Adding 0.3 in I_X is enough to ensure that the global minimum lies within set X
- Note: we'd better start with an initial x that is in set X, then won't "escape" from X. But if start outside X might converge to local minimum at $x = 0$ rather than global min at $x = -0.5$.

» Penalty Methods: Continuous Convex Penalty Functions

- When use $I_X(x)$ then:
  - Get a sharp "jump" in the cost function $f(x) + I_X(x)$ at the boundary of set X
  - *Convex* function: when draw a line between any two points on curve that line lies *above* the curve
  - Indicator penalty function $I_X(x)$ is *non-convex*, see green line:

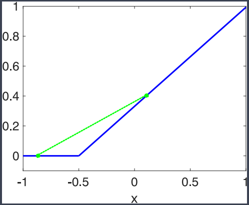


  $\rightarrow$ *function $f(x) + I_X(x)$ is non-convex even when $f(x)$ is convex*
- How about using a *continuous* and *convex* penalty? E.g.



» Penalty Methods: Continuous Convex Penalty Functions

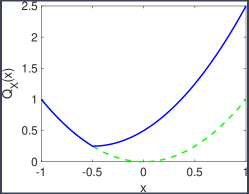
* *Continuous*: when plot penalty function vs x there are no jumps or gaps, instead have a continuous line.
* *Convex*: When draw a line between any two points on the curve that line lies *above* the curve, e.g. green line:



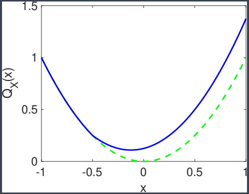
* Penalty function: $Q_X(x) = \begin{cases} 0 & x \in X \\ d(x,X) & x \notin X \end{cases}$ where $d(x,X)$ measures the distance between point x and set X, and:
  * *Continuous*: $d(x,X) \leq 0$ for all $x \in X$, $d(x,X) > 0$ for all $x \notin X$ and $d(x,X)$ is continuous
  * *Convex*: $d(x,X)$ is convex
* Can also write $Q_X(x) = \max(0, d(x,X))$. When $x \in X$, $d(x,X) \leq 0$ and so $Q_X(x) = 0$. When $x \notin X$, $d(x,X) > 0$ and so $Q_X(x) = d(x,X)$

» Penalty Methods: Continuous Convex Penalty Functions

* E.g. $f(x) = x^2$, $X = \{x \in \mathbb{R} : x \leq -0.5\}$, penalty $Q_X(x) = \max(0, x + 0.5)$

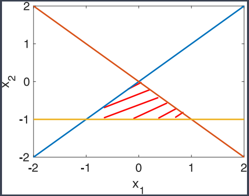


* Now $f(x) + Q_X(x)$ is continuous and convex. But non-smooth (has a "kink" at $x = -0.5$)
* Note: we need $Q_X(x)$ to increase quickly enough that global min is in set X. E.g. $Q_X(x) = 0.25 \max(0, x + 0.5)$ *is too small*:



» Penalty Methods: Continuous Convex Penalty Functions

- *Multiple constraints*: often admissible set X is defined as the intersection of multiple sets e.g. a polytope



- 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.
- Useful trick: Rewrite each constraint into the standard form $g(x) \leq 0$ for a suitable function $g(\cdot)$.
  - $x_2 \leq -x_1 \rightarrow x_2 + x_1 \leq 0$
  - $x_1 \leq x_2 \rightarrow x_1 - x_2 \leq 0$
  - $x_1 \geq -1 \rightarrow -x_1 \leq 1 \rightarrow -x_1 - 1 \leq 0$
- Then use $Q_X(x) = \max(0, g_1(x)) + \max(0, g_2(x)) + \max(0, g_3(x))$ with $g_1(x) = x_2 + x_1$, $g_2(x) = x_1 - x_2$, $g_3(x) = -x_1 - 1$.
- Another example: $X = \{x \in \mathbb{R} : 0 \leq x \leq 1\}$.
  - Rewrite as $X = \{x : -x \leq 0, x - 1 \leq 0\}$ and use $Q_X(x) = \max(0, -x) + \max(0, x - 1)$

» Penalty Methods: Continuous Convex Penalty Functions

Convex constraints:

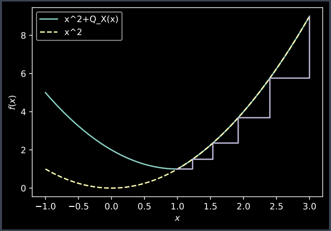
* When $X = \{x : g_1(x) \le 0, g_2(x) \le 0, \dots, g_m(x) \le 0\}$ and functions $g_1(\cdot), \dots, g_m(\cdot)$ are convex then use $Q_X(x) = \max(0, g_1(x)) + \max(0, g_2(x)) + \dots + \max(0, g_m(x))$
* Note: (i) convex functions are always continuous, (ii) max is a convex function, (iii) adding convex functions gives a convex function $\rightarrow$ *$Q_X(x)$ is convex and continuous*
* Note: linear functions are convex e.g. $g_1(x) = x - 1$

Non-convex constraints:

* A reasonable choice for penalty function $Q_X(x)$ is often clear from the context. But will often be non-convex.

» Example

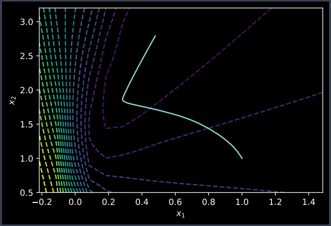
* $f(x) = x^2$ and we require x to be greater than +1 i.e. $X = \{x \in \mathbb{R} : x \geq 1\}$. Use $Q_X(x) = 2\max(0, -x+1)$. Gradient descent with constant step size $\alpha = 0.1$, initial $x = +3$:



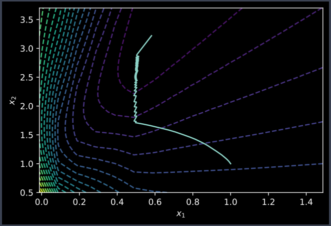
* Yellow dashed line is $f(x) = x^2$, blue line is $f(x) + Q_X(x)$
* See that x decreases until it hits constraint at $x = +1$, and then stops there. $x = +1$ minimises $f(x) + Q_X(x)$
* Note need to scale penalty to $2\max(0, -x+1)$, using $\max(0, -x+1)$ is not enough to move global minimum of $f(x) + Q_X(x)$ to $x = +1$

» Example

- Toy neural network and we require x_1 to be greater than 0.5 i.e. $X = \{x \in \mathbb{R}^2 : x_1 \geq 0.5\}$. Use $Q_X(x) = 0.02\,\mathsf{max}(0, -x + 0.5)$. Gradient descent with constant step size $\alpha = 0.75$, initial $x = [1, 1]$:

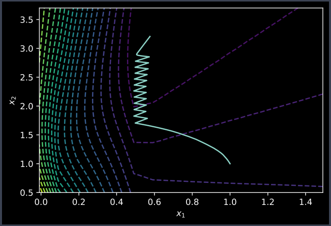


unconstrained



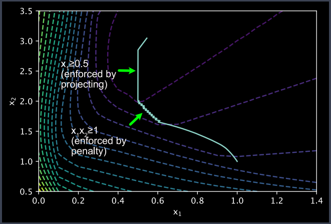
constrained

- Note scaled penalty to $0.02\,\mathsf{max}(0, -x + 0.5)$. If smaller than 0.02 then penalty is not enough to enforce constraint, if larger then need to reduce step size or get oscillations e.g. with $0.1\,\mathsf{max}(0, -x + 0.5)$:



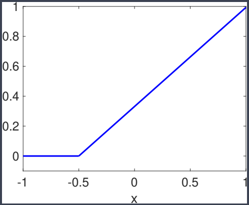
» Combining Penalty Method With Projected Gradient Descent

* With penalty method we convert constrained optimisation to unconstrained optimisation $\rightarrow$ Polyak, Adagrad, RMSpop, Nesterov, Adam can all be used as before
* Can also combine penalty approach with projected gradient descent, i.e. use penalty method to handle the constraints which are not easy to project and use projected gradient descent for the rest
    * Toy neural net, $x = [x_1, x_2]$ and $X = \{x : x_1 \geq 0.5, x_1 x_2 \geq 1\}$.
    * Projecting onto set $\{x_1 : x_1 \geq 0.5\}$ is easy, just use $x_1 \leftarrow \max(0.5, x_1)$. Projecting onto $\{x : x_1 x_2 \geq 1\}$ is harder, so use penalty $Q_X(x) = \max(0, -x_1 x_2 + 1)$.
    * Mixed penalty/projected gradient descent, initial $x = [1, 1]$, constant step size $\alpha = 0.75$:



» Non-Smooth Penalty vs Smooth Penalty

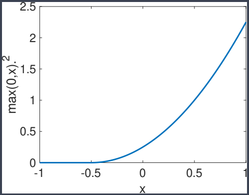
* Penalty $Q_X(x)$ is continuous and convex, but it has a“kink” at the boundary of set $X \rightarrow$ it’s *non-smooth*
* E.g. $Q_X(x) = \max(0, x + 0.5)$:



* So $f(x) + Q_X(x)$ is also non-smooth. We know that non-smooth functions can be harder/slower to minimise ...
* ... can we not use a smooth penalty?

» Non-Smooth Penalty vs Smooth Penalty

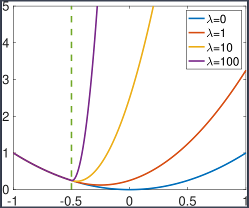
- ... can we not use a smooth penalty?
- How about $P_X(x) = \mathtt{max}(0, x + 0.5)^2$



- Now no “kink” at $x = -0.5$

» Non-Smooth Penalty vs Smooth Penalty

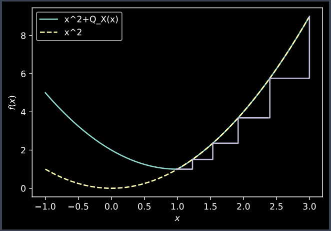
- There's a problem though. Suppose use $P_X(x) = \lambda \max(0, x + 0.5)^2$. To force x to lie in set X we need λ to be $+\infty$!
- E.g. $f(x) = x^2$, $P_X(x) = \lambda \max(0, x + 0.5)^2$, vary λ:



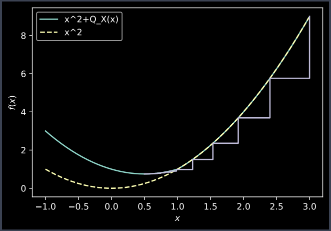
- Large λ means $f(x) + P_X(x)$ has large gradients and we need to use a small step size in gradient descent $\rightarrow$ slower convergence
- Remember: no free lunch ... generally expect adding constraints to make our optimisation harder, and there's no way to escape that

» How To Ensure Penalty Is Large Enough?

- Recall example: $f(x) = x^2$, $X = \{x \in \mathbb{R} : x \geq 1\}$. Gradient descent with constant step size $\alpha = 0.1$, initial $x = +3$:



$Q_X(x) = 2\max(0, -x+1)$



$Q_X(x) = \max(0, -x+1)$

- Need to scale penalty to $2\max(0, -x+1)$, using $\max(0, -x+1)$ is not enough to move global minimum of $f(x) + Q_X(x)$ to $x = +1$
- Write $P_X(x) = \lambda \max(0, -x+1)$. How to choose λ?
- *One idea: Run gradient descent for a few iterations, check if $x \in X$, if not then increase λ. Repeat.*
- Note: intermediate values of x may not be in $X \rightarrow$ often this is ok, but not always
- Works fine, but can we do something more elegant?

» Multiplier Update

* Suppose we want to minimise $f(x)$ subject to constraint that $x \in X = \{x \in \mathbb{R}^n : g(x) \leq 0\}$
* Gradient descent algorithm with penalty method is:

  $x_{t+1} = x_t - \alpha \nabla F_\lambda(x)$

  with step size α,

  $$F_\lambda(x) = f(x) + \lambda Q_X(x) = f(x) + \lambda \max(0, g(x))$$

  and

  $$\nabla F_\lambda(x) = [\frac{\partial F_\lambda}{\partial x_1}(x), \frac{\partial F_\lambda}{\partial x_2}(x), \ldots, \frac{\partial F_\lambda}{\partial x_n}(x)]$$

* When $x \notin X$ then $g(x) > 0$, so why not use this to adapt λ e.g.

  $x_{t+1} = x_t - \alpha \nabla F_{\lambda_t}(x)$
  $\lambda_{t+1} = \lambda_t + \beta \max(0, g(x_t))$

  where $\beta > 0$ is the learning rate for λ. When $x \notin X$ we increase λ.
* λ will keep increasing while x is persistently outside set X

» A Variation

- A variation on this idea is to use the update:

  $$x_{t+1} = x_t - \alpha \nabla F_{\lambda_t}(x)$$
  $$\lambda_{t+1} = [\lambda_t + \beta g(x_t)]_+$$

  where $[z]_+ = z$ when $z \geq 0$ and otherwise 0 i.e. $[z]_+$ projects z onto the set of non-negative values $\rightarrow$ makes sure that λ_{t+1} doesn't become negative
- What is the effect of this change?
  - Suppose the minimum of $f(x)$ lies inside set X, e.g. $f(x) = x^2$ and $X = \{x : x \leq 1\}$. Then solution to constrained and unconstrained optimisations is the same.
  - Once x_t gets near to minimum, $g(x_t) < 0$ forever and so λ_t *decreases* until eventually $\lambda_{t+1} = 0$.
  - In contrast, when $\lambda_{t+1} = \lambda_t + \beta \max(0, g(x_t))$ then λ_t can never decrease
- When the constraint is inactive at optimum the multiplier λ becomes zero i.e. if x^* minimises $f(x)$ on set X and $g(x^*) < 0$ then $\lambda \rightarrow 0$.
- Otherwise when constraint is active at the optimum the multipler λ might be non-zero i.e. when $g(x^*) = 0$.
- Note: we can't have $g(x^*) > 0$ since then $x^* \notin X$

» Primal-Dual Update

We can simplify this update ....

* When $g(x^*) < 0$ the multiplier $\lambda \to 0$ and so $\lambda g(x^*) = 0$.
* When the constraint is active at optimum $g(x^*) = 0$ and so $\lambda g(x^*) = 0$ (even if $\lambda > 0$).
* So we always have $\lambda g(x^*) = 0 \to$ *complementary slackness*
* Replace $F_{\lambda_t}(x) = f(x) + \lambda \max(0, g(x))$ with

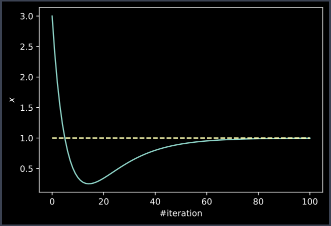
$$G_\lambda(x) = f(x) + \lambda g(x))$$

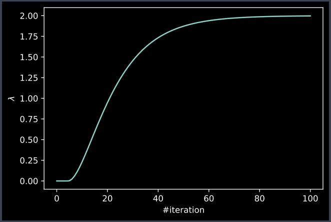
to get *primal-dual* update:

$$\begin{aligned} x_{t+1} &= x_t - \alpha \nabla G_{\lambda_t}(x) \\ \lambda_{t+1} &= [\lambda_t + \beta g(x_t)]_+ \end{aligned}$$

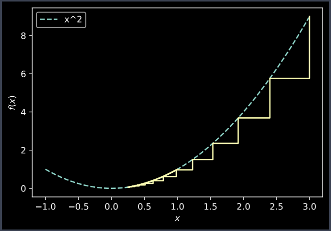
» Primal-Dual Update: Example

- $f(x) = x^2$ and we require x to be greater than +1 i.e. $X = \{x \in \mathbb{R} : x \geq 1\}$.
- Use $g(x) = -x + 1$ i.e. $G_\lambda(x) = x^2 + \lambda(-x + 1)$.
- Primal-dual update, constant step size $\alpha = 0.1 = \beta$, initial $x = +3$:



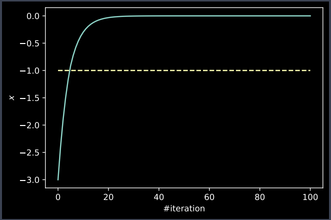


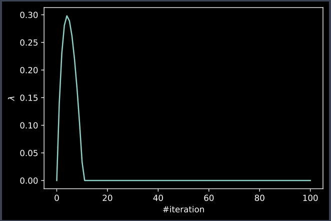
- See that x decreases until it violates constraint $x \geq 1$, then λ starts increasing until $x \to 1$ i.e until $x \in X$



» Primal-Dual Update: Example

- Now change constraint to $X = \{x \in \mathbb{R} : x \geq -1\}$. Since x^2 is minimised at $x = 0$, $\min_x f(x) = \min_{x \in X} f(x)$
- Use $g(x) = -x - 1$ i.e. $F_\lambda(x) = x^2 + \lambda(-x - 1)$.
- Primal-dual update, constant step size $\alpha = 0.1 = \beta$, initial $x = -3$:





- Initially x violates constraint and so λ increases, but once $x \geq -1$ then λ starts decreasing until it becomes 0.

» Primal-Dual Update: Multiple Constraints

- So far we used $X = \{x \in \mathbb{R}^n : g(x) \leq 0\}$ i.e. just one constraint function $g(\cdot)$.
- Can easily extend to multiple constraint functions i.e. $X = \{x : g_1(x) \leq 0, g_2(x) \leq 0, \ldots, g_m(x) \leq 0\}$
- Now
$$G_\lambda(x) = f(x_t) + \lambda_1 g_1(x_t) + \lambda_2 g_2(x_t) + \cdots + \lambda_m g_m(x_t)$$
i.e. we associate a different multiplier with each constraint and so have a vector of multipliers
$$\lambda_t = [\lambda_{1,t}, \lambda_{2,t}, \ldots, \lambda_{m,t}]$$
- Primal-dual update then becomes
$$\begin{aligned} x_{t+1} &= x_t - \alpha \nabla G_{\lambda_t}(x) \\ \lambda_{1,t+1} &= [\lambda_{1,t} + \beta g_1(x_t)]_+ \\ \lambda_{2,t+1} &= [\lambda_{2,t} + \beta g_2(x_t)]_+ \\ &\vdots \\ \lambda_{m,t+1} &= [\lambda_{m,t} + \beta g_m(x_t)]_+ \end{aligned}$$
- For those constraints which are inactive at optimum the multiplier will converge to 0 e.g. if x^* minimises $f(x)$ on set X and $g_1(x^*) < 0$ then $\lambda_1 \to 0$. Only constraints which are active at optimum have non-zero multiplier.

» Summary

* Can convert constrained optimisation to an unconstrained optimisation by adding a penalty function.
* Indicator penalty function is conceptually simple but numerically bad (non-smooth, non-convex)
* Linear penalty is much better numerically (continuous, convex although non-smooth)
* Smooth penalty means have to use infinite weight to ensure solution converges to a point in admissible set X (no free lunch)
* Need to scale the penalty to make it big enough …
* … easy enough to learn the right scaling either by (i) trial and error, (ii) progressively increasing penalty, (iii) primal-dual update
* Can combine penalty method with projected gradient descent