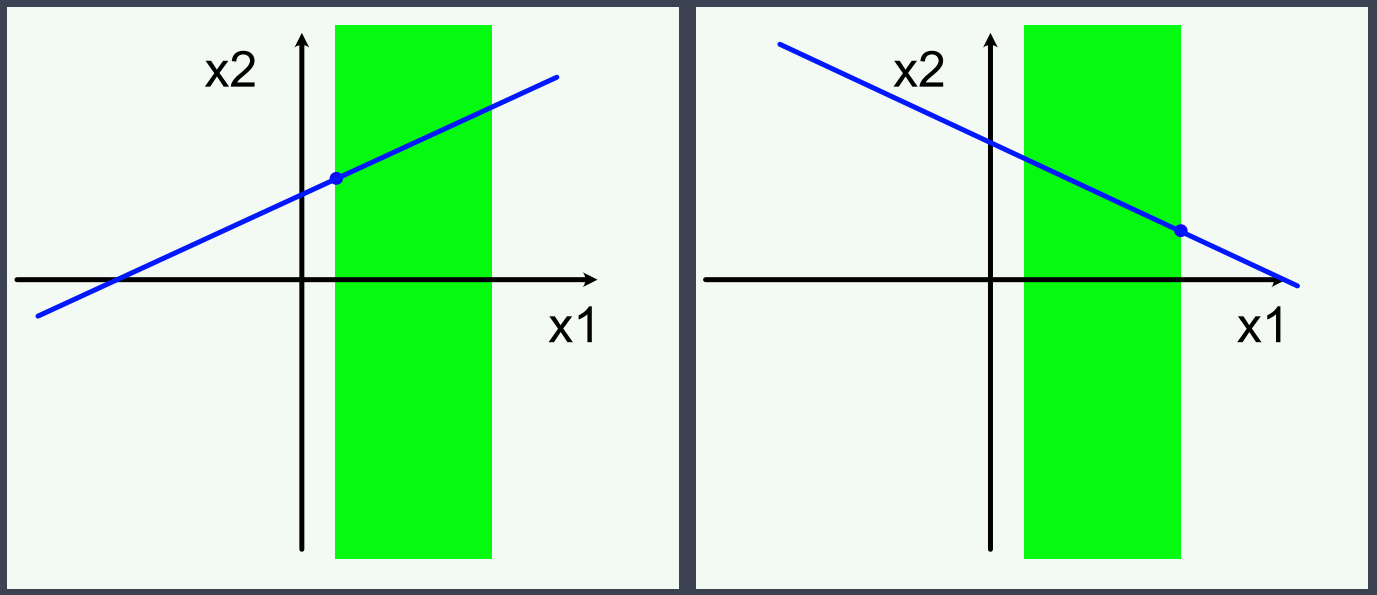
» Frank-Wolfe Algorithm

- Penalty method let's us handle constrained optimisation when projecting onto constrained set is hard/slow (so projected gradient descent is slow)
- *Frank-Wolfe* algorithm[1] uses another idea to avoid projections
- We start by going back to look at the special case where the cost function being minimised is *linear* ...

[1] https://en.wikipedia.org/wiki/Frank-Wolfe_algorithm

» Linear Cost Functions

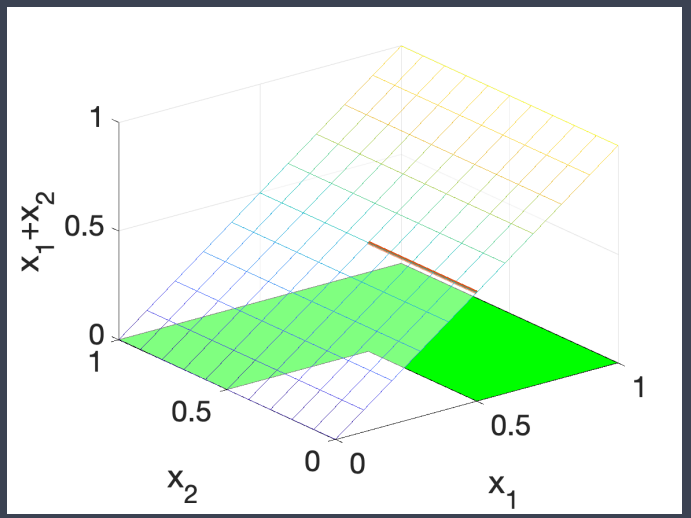
* Suppose the function we are trying to minimise is linear e.g $f(x) = -x$



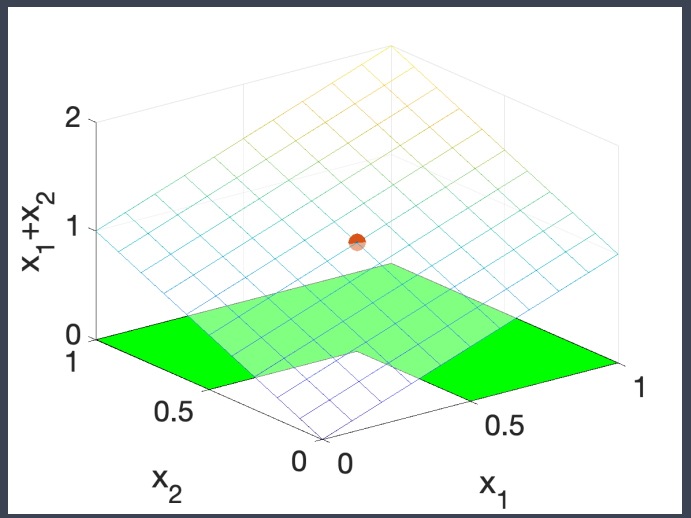
* The minimum will always lie on one of the constraints (one side or other of the green shaded region above)

» Linear Cost Functions

- The same is true when x is a vector. Suppose $x = [x_1, x_2]$ and we constrain $x_1 \geq 0.5$ and $x_2 \geq 0.5$
- Examples: $f(x) = x_1$ and $f(x) = x_1 + x_2$:



$f(x) = x_1$



$f(x) = x_1 + x_2$

- Red line/dot mark the minima
- The minimum will always lie on the constraints (the boundary of the green shaded region above)
- This fact can be v handy – often makes linear optimisation easier since know that solution lies on boundary of feasible set ...

» Linear Programmes

* General form of a linear function of a vector $x = [x_1, x_2, \ldots, x_n]$ is the weighted sum of the elements of x i.e.

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n$$

with weights $a_1, a_2, \ldots, a_n$. With vector $a = [a_1, a_2, \ldots, a_n]$, can be written in short form as $a^Tx = a_1x_1 + a_2x_2 + \cdots + a_nx_n$

* A *linear programme* is an optimisation where both the cost function and constraints are linear. Can be written as

$$\min_{x \in X} a^Tx$$

with $X = \{x \in \mathbb{R}^n : Cx \leq b\}$. Here C is a matrix, b is a vector:

$$Cx = \begin{bmatrix} c_{11} & c_{12} & \ldots & c_{1n} \\ c_{21} & c_{22} & \ldots & c_{2n} \\ \vdots & & & \\ c_{m1} & c_{m2} & \ldots & c_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \leq \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

$$c_{11}x_1 + c_{12}x_2 + \cdots + c_{1n}x_n \leq b_1$$

$$c_{21}x_1 + c_{22}x_2 + \cdots + c_{2n}x_n \leq b_2$$

$$\vdots$$

$$c_{m1}x_1 + c_{m2}x_2 + \cdots + c_{mn}x_n \leq b_m$$

» Linear Programmes

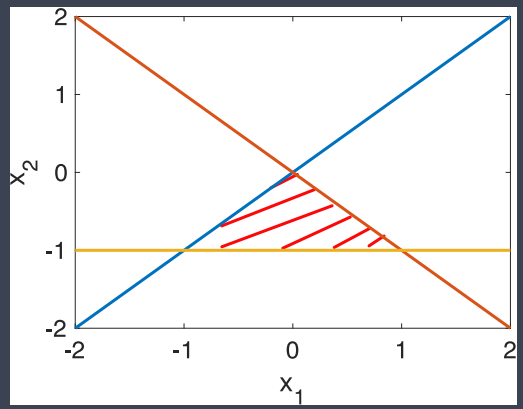
* Each linear constraint

$$c_{11}x_1 + c_{12}x_2 + \cdots + c_{1n}x_n \leq b_1$$

defines a line, sequence of constraints

$$\begin{aligned} c_{11}x_1 + c_{12}x_2 + \cdots + c_{1n}x_n &\leq b_1 \\ c_{21}x_1 + c_{22}x_2 + \cdots + c_{2n}x_n &\leq b_2 \\ &\vdots \\ c_{m1}x_1 + c_{m2}x_2 + \cdots + c_{mn}x_n &\leq b_m \end{aligned}$$

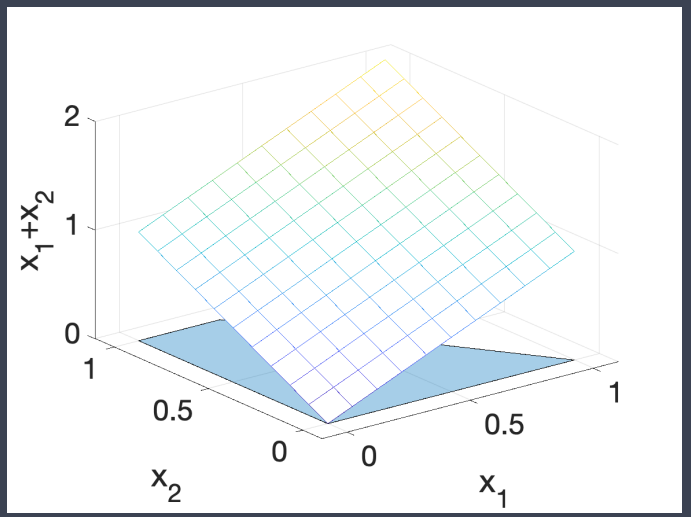
defines a polytope

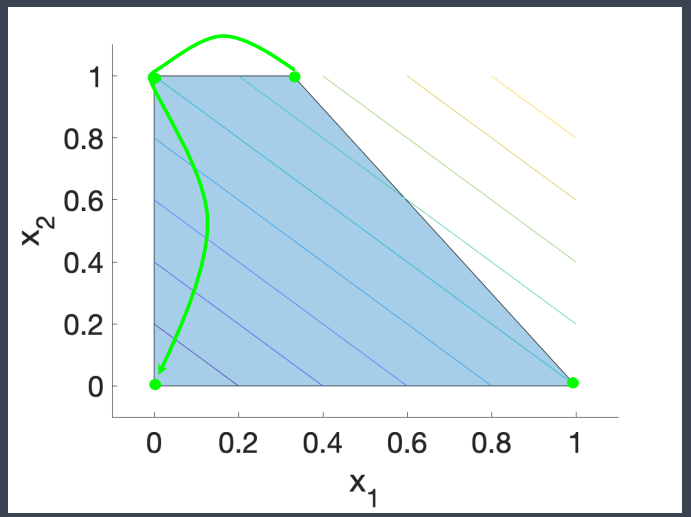


red line: $x_2 = -x_1$, blue line: $x_1 = x_2$, yellow line: $x_1 = -1$

» Linear Programmes

* Linear programme: $\min_{x \in X} a^T x$, $X = \{x \in \mathbb{R}^n : Cx \leq b\}$
* Minimise a linear cost function $a^T x$ with x constrained to lie in polytope X, e.g.



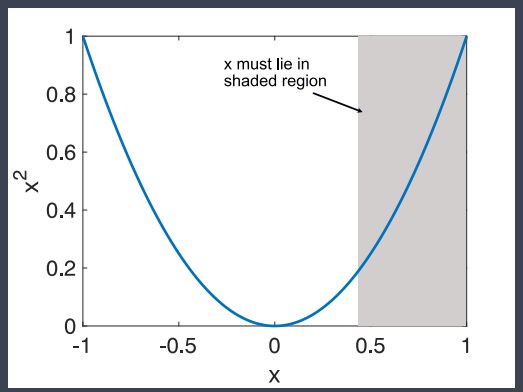


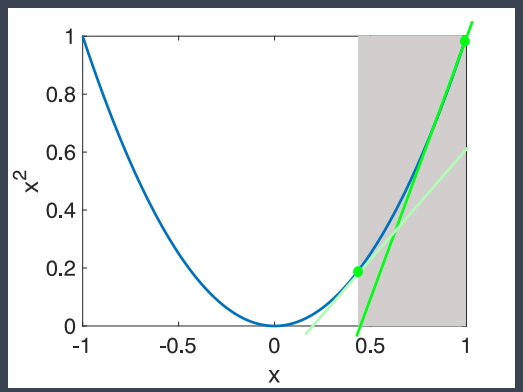
* Solution will lie on boundary of polytope, at one of vertices.
* To find minimiser, walk along polytope vertices $\rightarrow$ *Simplex algorithm*[2]
* Note: not same as Nelder-Mead Simplex algo for nonlinear optimsation, sometimes called Dantzig Simplex algo to distinguish
* E.g. see scipy.optimize.linprog https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linprog.html

[2] https://en.wikipedia.org/wiki/Simplex_algorithm

» NonLinear Cost Functions

* *Key idea*: Suppose linear optimisation over constraint set X can be done quickly. Can we minimise a nonlinear function over set X by repeatedly minimising linear functions on X?
* Maybe ...
* Example: suppose we want to minimise $f(x) = x^2$ subject to constraint that $x \in X = \{x : 0.5 \leq x \leq 1\}$. *Solution is $x = 0.5$, on boundary of set X.*

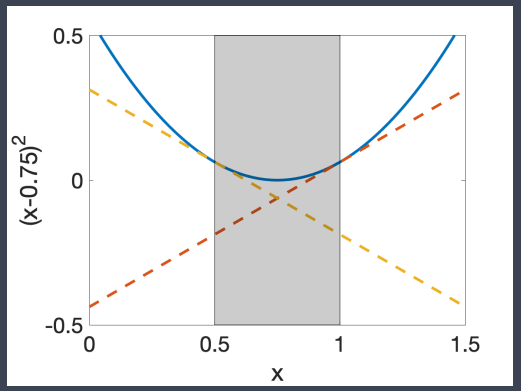




* $\frac{df}{dx}(x) = 2x$
* Suppose we start at $x_0 = 1$ and $\frac{df}{dx}(1) = 2$. Now solve $\min_{x \in X} \frac{df}{dx}(x_0)x = \min_{x \in X} 2x$. Solution is $x = 0.5$.
* Repeat. Now $x_1 = 0.5$ and $\frac{df}{dx}(0.5) = 1$. Solve $\min_{x \in X} \frac{df}{dx}(x_1)x = \min_{x \in X} x$. Solution is $x = 0.5$, and stop.
* *We've managed to solve a nonlinear constrained optimisation by solving multiple linear constrained optimisations.*

» NonLinear Cost Functions

* Suppose now we want to minimise $f(x) = (x - 0.75)^2$ such that $x \in X = \{x : 0.5 \leq x \leq 1\}$. *Solution is $x = 0.75$, in interior of set X.*
  * $\frac{df}{dx}(x) = 2(x - 0.75)$
  * Suppose we start at $x_0 = 1$. Now solve $\min_{x \in X} \frac{df}{dx}(x_0)x = \min_{x \in X} 0.5x$. Solution is $x = 0.5$.



  * Repeat. Now $x_1 = 0.5$. Solve $\min_{x \in X} \frac{df}{dx}(x_1)x = \min_{x \in X} -0.5x$. Solution is $x = 1$.
  * *We repeatedly alternate between 0.5 and 1, spending equal time at each value*
  * *On average $\frac{1}{T}\sum_{t=1}^{T} x_t = 0.75$ i.e. solves the nonlinear optimisation*

» Frank-Wolfe Algorithm

* To solve $\min_{x \in X} f(x)$ with $x = [x_1, \ldots, x_n]$:

  initialise $x_0 = 0, t = 0, 0 < \beta < 1$
  Repeat:
  $z_t \in \arg\min_{x \in X} \nabla f(x_t) x$
  $x_{t+1} = \beta x_t + (1 - \beta) z_t$
  $t = t + 1$

  and recall $\nabla f(x) = [\frac{\partial f}{\partial x_1}(x), \frac{\partial f}{\partial x_2}(x), \ldots, \frac{\partial f}{\partial x_n}(x)]$

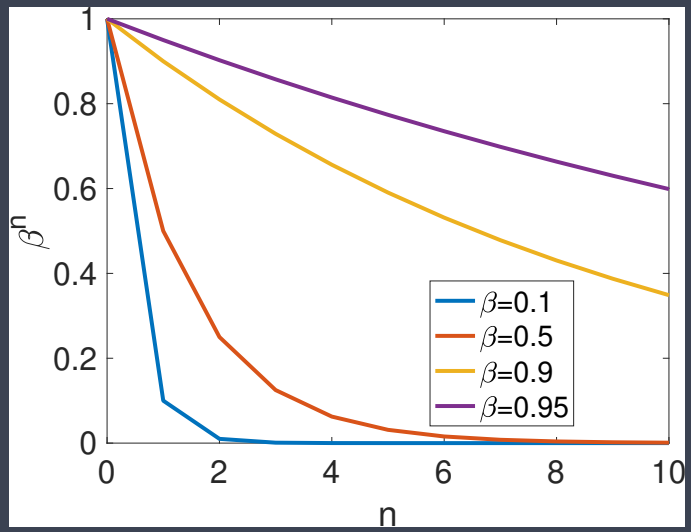
* $\nabla f(x_t) x$ is linear cost function ($\nabla f(x_t)$ is constant, min is wrt x)
  $\rightarrow \min_{x \in X} \nabla f(x_t) x$ is linear optimisation over set X
* Solution z_t to linear optimisation will lie on boundary of set X
* x_{t+1} is running average of $z_1, z_2, \ldots, z_{t-1}$. Parameter β controls this averaging
* Note: no step size α.

» Running Average

* $x_{t+1} = \beta x_t + (1-\beta) z_t$
* Expanding this out:

$$
\begin{aligned}
x_1 &= (1-\beta) z_0 \\
x_2 &= \beta x_1 + (1-\beta) z_1 = \beta(1-\beta) z_0 + (1-\beta) z_1 \\
&= (1-\beta)(\beta z_0 + z_1) \\
x_3 &= \beta x_2 + (1-\beta) z_2 = \beta^2 (1-\beta) z_0 + \beta(1-\beta) z_1 + (1-\beta) z_2 \\
&= (1-\beta)(\beta^2 z_0 + \beta z_1 + z_2)
\end{aligned}
$$

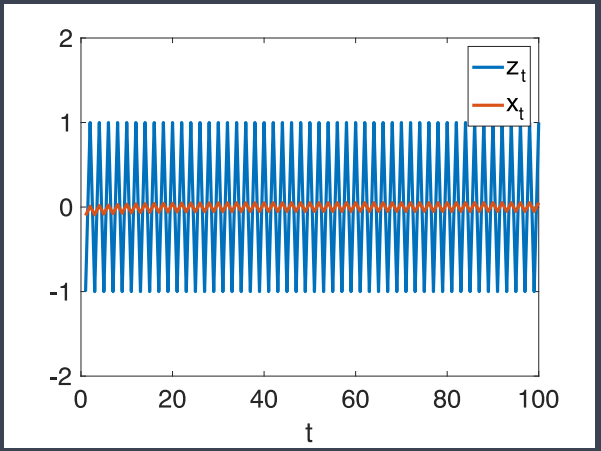
so x_t is weighted sum of $z_0, z_1, \ldots, z_{t-1}$. E.g. with $\beta = 0.9$ then $\beta = 0.9$, $\beta^2 = 0.81$, $\beta^3 = 0.729$ etc



* Sum $\beta^n + \beta^{n-1} + \cdots + \beta + 1 = \frac{1-\beta^n}{1-\beta} \approx \frac{1}{1-\beta}$ for large n

» Running Average

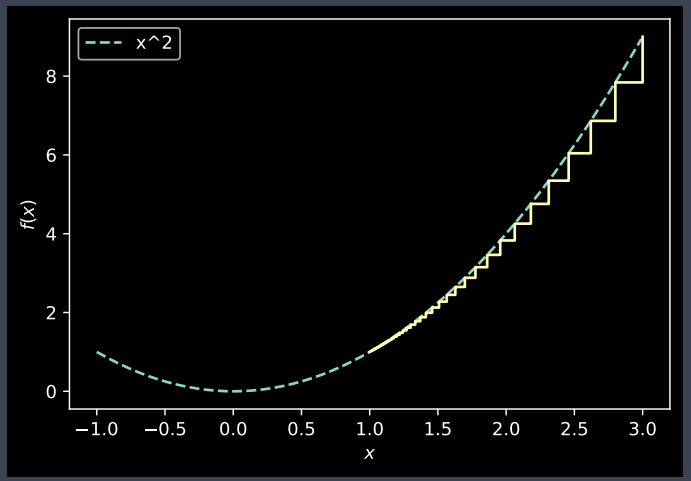
* Example: $\beta = 0.9$, $z_t = (-1)^t$ i.e. $z_0 = 0, z_1 = 1, z_2 = +1, z_3 = -1, \ldots$
* $x_{t+1} = \beta x_t + (1 - \beta) z_t$:

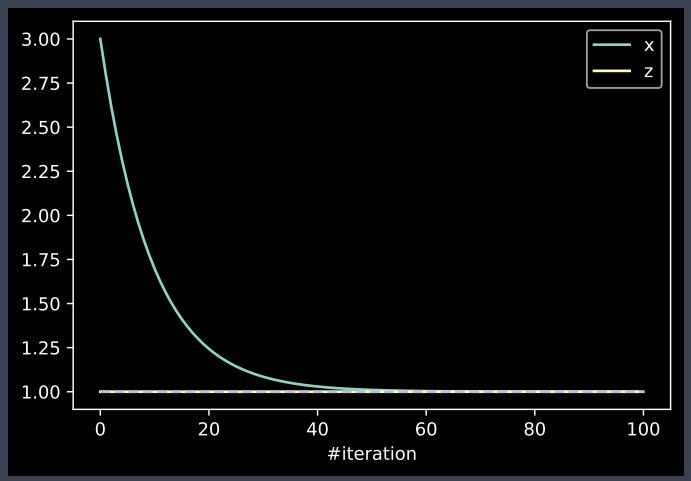


* See that x_t smooths the z_t values and is roughly equal to their average

» Example

* $f(x) = x^2$, $X = \{x \in \mathbb{R} : 1 \leq x \leq 5\}$, initial $x = 3$. Minimiser is $x = +1$ i.e on boundary of X. Frank-Wolfe, $\beta = 0.9$:

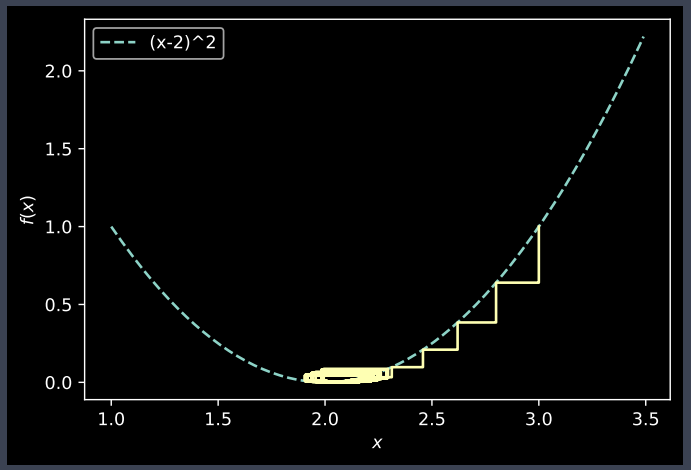


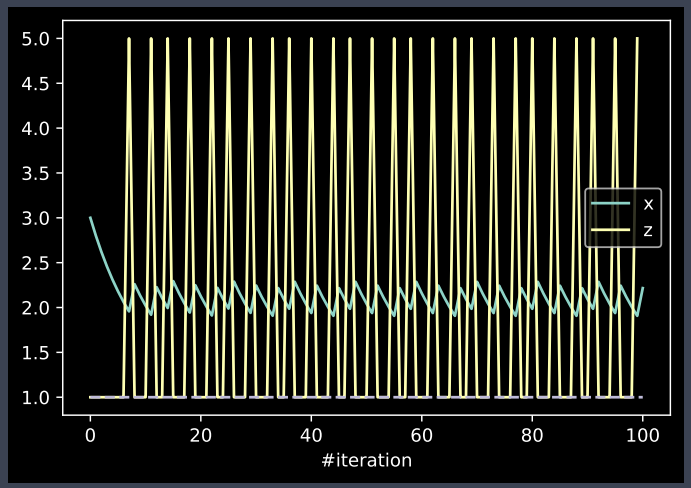


* See that z goes straight to $+1$, but running average x takes a while to "catch up"

» Example

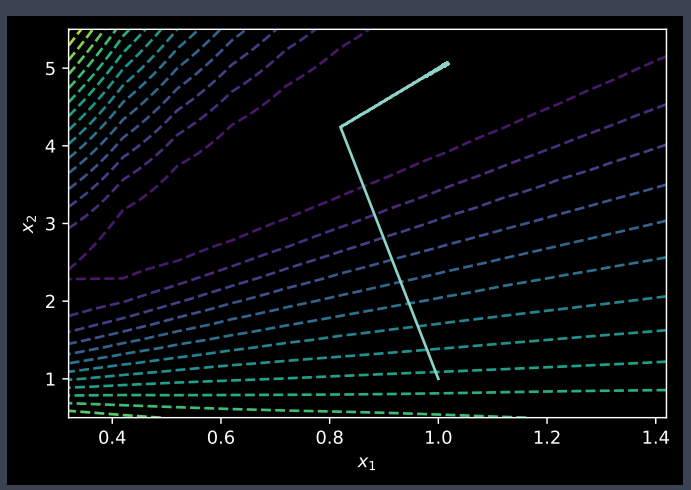
* $f(x) = (x-2)^2$, $X = \{x \in \mathbb{R} : 1 \leq x \leq 5\}$, initial $x = 3$. Minimiser is $x = +2$ i.e in interior of $X \rightarrow$ *so expect z to oscillate between +1 and +5 (boundaries of X) but have average equal to* $+2$.
* Frank-Wolfe, $\beta = 0.9$:

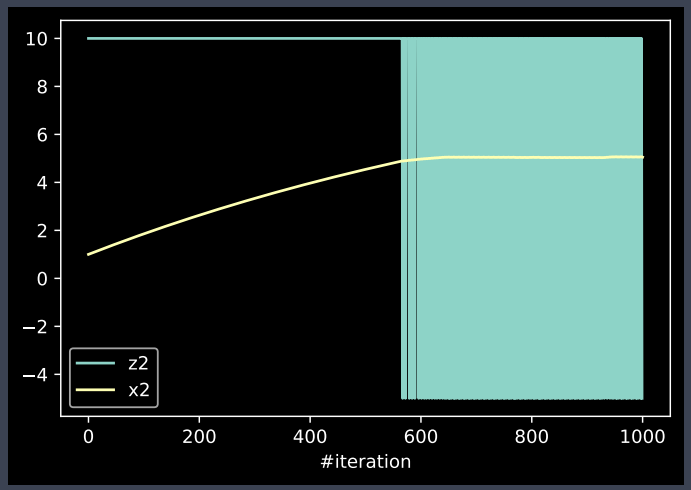




» Example

- Toy neural net with minimiser $x = [1, 5]$, constraint $x_1 \geq 0.5$. But to use Frank-Wofle also need to constrain max value of x_1 and max/min values of x_2 (since linear programme step will find points on boundary of X, so *need all values in X to be bounded*).
- $X = \{x \in \mathbb{R}^2 : 0.5 \leq x_1 \leq 5, -5 \leq x_2 \leq 10\}$, initial $x = [1, 1]$
- Frank-Wolfe, $\beta = 0.99$:

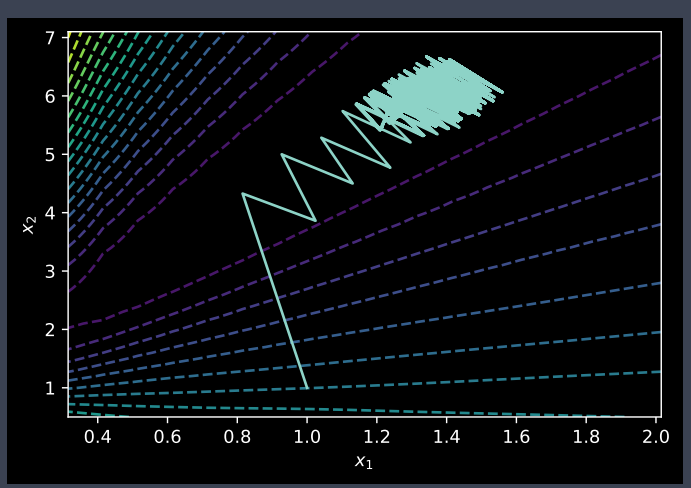


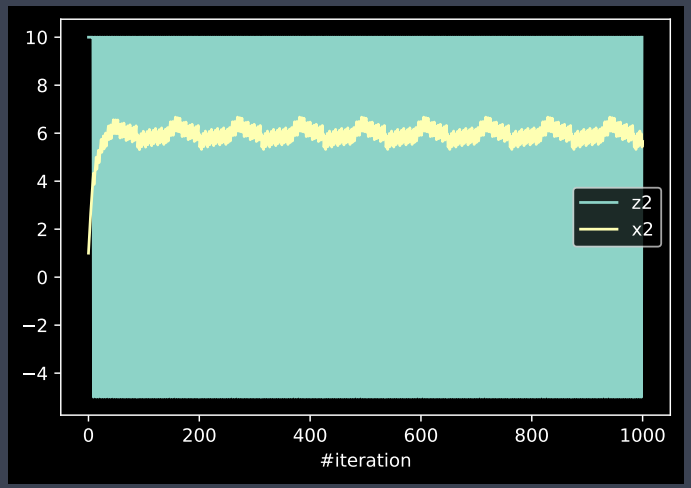


- Note: larger value of $\beta = 0.999$ used so as to reduce size of oscillations in x and obtain convergence to minimum. When smaller β used, e.g. $\beta = 0.995$, then converges to sub-optimal point ...

» Example

- Toy neural net with minimiser $x = [1, 5]$,
  $X = \{x \in \mathbb{R}^2 : 0.5 \leq x_1 \leq 5, -5 \leq x_2 \leq 10\}$, initial $x = [1, 1]$
- When smaller β used, e.g. $\beta = 0.95$, then converges to sub-optimal point and x jumps around quite a bit ...
- Frank-Wolfe, $\beta = 0.95$:





» Interpretation As Actions

- Frank-Wolfe avoids projections, and so do penalty approaches.
- Penalty approaches are generally computationally cheaper - no need to solve a linear programme at each iteration.
- But Frank-Wolfe has an interesting connection with "actions":
  - *Suppose we can only choose between n actions* e.g. between two routes to travel to work. We repeat these actions e.g. we travel to work every day.
  - *Associate a vector with each action* e.g. $u_1 = [1, 0]$ and $u_2 = [0, 1]$ for the two routes
  - *Define X to be a polytope with vertex corresponding to an action* e.g. for two routes $X = \{x = \alpha u_1 + (1 - \alpha)u_2 : 0 \leq \alpha \leq 1\}$. Can think of α as the probability of choosing route 1 each day and $1 - \alpha$ as probability of choosing route 2.
  - *Define cost function $f(x)$* e.g. travel time.
  - Run Frank-Wolfe algo to minimise $f(x)$ such that $x \in X$.
  - *At each iteration the FW algorithm selects a vertex of X as z i.e. selects an action*. E.g. selects a route each day.
  - *The running average of the actions converges to a minimiser of $f(x)$.*
- That leads us into online learning and optimisation ...

» What Next?

- This module is just a first course on optimisation, with a focus on practical understanding and use in machine learning.
- Optimisation itself is a huge subject … possible next steps include:
  - *Theoretical analysis of convergence rates of algorithms.*
    - Plenty on the web about this
    - Almost entirely focussed on convex optimisation (convex cost function and constraints). But neural nets etc are all non-convex
    - Almost entirely focussed on worst-case analysis. But we're usually more interested in algos that work well on our specific problem
  - *Optimality conditions and duality.*
    - KKT conditions that need to be satisfied by an optimum - sometimes allow solution to be obtained directly e.g. our projection examples
    - Duality. Can be used to reformulate optimisation problem.
  - *Privacy of gradients*
    - Federated learning (already being used by Google)
    - Sharing gradients can reveal information e.g. words typed on phone keyboard, image viewed by camera
    - An area of active research …
  - *Online learning*
    - Multi-arm bandits, online convex optimisation
    - Reinforcement learning
  - *Integer optimisation methods*
    - Perhaps main topic in optimisation outside of ML
    - Decision variables x can only take integer values e.g. 0 or 1.