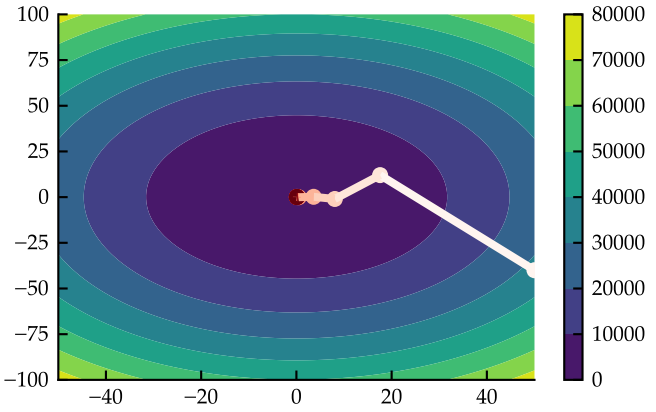

Figure 8.2. Gradient descent on f_2.

8.2 Constrained Linear Optimization

DJ

Consider a linear objective function $f : \mathbb{R}^n \to \mathbb{R}$ with variables x_i in vector $\boldsymbol{x}$ and coefficients c_i in vector $\boldsymbol{c}$:

$$f(\boldsymbol{x}) = \boldsymbol{c} \cdot \boldsymbol{x}$$

subject to the linear **constraints**—restrictions on x_i—

$$A\boldsymbol{x} \leq \boldsymbol{a}, \tag{8.3}$$

$$B\boldsymbol{x} = \boldsymbol{b}, \text{ and} \tag{8.4}$$

$$\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \tag{8.5}$$

where A and B are constant matrices and $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{l}, \boldsymbol{u}$ are n-vectors. This is one formulation of what is called a **linear programming problem**. Usually we want to **maximize** f over the constraints. Such problems frequently arise throughout engineering, for instance in manufacturing, transportation, operations, etc. They are called **constrained** because there are constraints on $\boldsymbol{x}$; they are called **linear** because the objective function and the constraints are linear.

We call a pair $(\boldsymbol{x}, f(\boldsymbol{x}))$ for which $\boldsymbol{x}$ satisfies equation (8.3) a **feasible solution**. Of course, not every feasible solution is **optimal**: a feasible solution is optimal iff there exists no other feasible solution for which f is greater (assuming we're maximizing). We call the vector subspace of feasible solutions $S \subset \mathbb{R}^n$.

8.2.1 Feasible Solutions Form a Polytope

Consider the effect of the constraints. Each of the equalities and inequalities defines a linear **hyperplane** in $\mathbb{R}^n$ (i.e. a linear subspace of dimension $n-1$): either as a boundary of S (inequality) or as a restriction of S to the hyperplane. When joined, these hyperplanes are the boundary of S (equalities restrict S to lower dimension). So we see that each of the boundaries of S is **flat**, which makes S a **polytope** (in $\mathbb{R}^2$, a polygon). What makes this especially interesting is that polytopes have **vertices** where the hyperplanes intersect. Solutions at the vertices are called **basic feasible solutions**.

8.2.2 Only the Vertices Matter

Our objective function f is linear, so for some constant h, $f(\boldsymbol{x}) = h$ defines a **level set** that is itself a hyperplane H in $\mathbb{R}^n$. If this hyperplane intersects S at a point $\boldsymbol{x}$, $(\boldsymbol{x}, f(\boldsymbol{x}) = h)$ is the corresponding solution. There are three possibilities when H intersects S:

1. $H \cap S$ is a vertex of S,
2. $H \cap S$ is a boundary hyperplane of S, or
3. $H \cap S$ slices through the interior of S.

However, this third option implies that there exists a level set G corresponding to $f(\boldsymbol{x}) = g$ such that G intersects S and $g > h$, so solutions on $H \cap S$ are *not optimal*. (We have not proven this, but it may be clear from our progression.) We conclude that either the first or second case must be true for optimal solutions. And notice that in both cases, a (potentially optimal) solution occurs at at least one vertex. The key insight, then, is that **an optimal solution occurs at a vertex of** S.

This means we don't need to search all of S, or even its boundary: we need only search the vertices. Helpful as this is, it restricts us down to $\binom{n}{\#\text{ constraints}}$ potentially optimal solutions—usually still too many to search in a naïve way. In (**lec:the_simplex_algorithm**), this is mitigated by introducing a powerful searching method.

8.3 The Simplex Algorithm

The **simplex algorithm** (or "method") is an iterative technique for finding an optimal solution of the linear programming problem of section 8.2. The details of the algorithm are somewhat involved, but the basic idea is to start at a vertex of the feasible solution space S and traverse an edge of the polytope that leads to another vertex with a greater value of f. Then, repeat this process until there is no neighboring vertex with a greater value of f, at which point the solution is guaranteed to be optimal.

Rather than present the details of the algorithm, we choose to show an example using Python. There have been some improvements on the original algorithm that have been implemented into many standard software packages, including Python's `scipy` package (Virtanen et al. 2019) module `scipy.optimize`.[1]

Example 8.2

Maximize the objective function

$$f(\boldsymbol{x}) = \boldsymbol{c} \cdot \boldsymbol{x} \tag{8.6}$$

for $\boldsymbol{x} \in \mathbb{R}^2$ and

$$\boldsymbol{c} = \begin{bmatrix} 5 & 2 \end{bmatrix}^\top \tag{8.7}$$

subject to constraints

$$0 \leq x_1 \leq 10 \tag{8.8}$$

$$-5 \leq x_2 \leq 15 \tag{8.9}$$

$$4x_1 + x_2 \leq 40 \tag{8.10}$$

$$x_1 + 3x_2 \leq 35 \tag{8.11}$$

$$-8x_1 - x_2 \geq -75. \tag{8.12}$$

First, load some Python packages.

```
from scipy.optimize import linprog
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
```

Ecoding the Problem Before we can use `linprog`, we must first encode equations (8.6) and (8.8) into a form `linprog` will recognize. We begin with f, which we can write as $\boldsymbol{c} \cdot \boldsymbol{x}$ with the coefficients of $\boldsymbol{c}$ as follows.

1. Another Python package `pulp` (PuLP) is probably more popular for linear programming; however, we choose `scipy.optimize` because it has applications beyond linear programming.