Hands-on Frank-Wolfe for constrained optimization

Mathieu Besançon - mathieu.besancon@inria.fr
Sébastien Designolle, Deborah Hendrych - Jannis Halbey
{designolle,hendrych,halbey}@zib.de

September 20, 2024

Inria, Université Grenoble Alpes, Zuse Institute Berlin

Outline

Frank-Wolfe algorithms

`FrankWolfe.jl`: a Julia implementation

Exercises

Table of Contents

Class of problems

Problem class considered:

$$(P) : \min_x \; f(x) \quad \text{s.t. } x \in \mathcal{C}$$

with:

- $\mathcal{C}$: compact convex set,
- f: continuously differentiable on $\mathcal{C}$.

Key requirement:
Optimizing a linear function over $\mathcal{C}$ much cheaper than (P) itself.
Linear Minimization Oracle (LMO):

$$d \to v \in \operatorname*{argmin}_{y \in \mathcal{C}} \langle y, d \rangle.$$

Frank-Wolfe template

Algorithm 1.1 Frank-Wolfe algorithm

Require: Point $x_0 \in \mathcal{C}$, function f.

Ensure: Iterates $x_1, \cdots \in \mathcal{C}$.

1: **for** $t = 0$ **to** $\ldots$ **do**

2: $\quad d_t \leftarrow \text{FirstOrderEstimate}(f, x_t)$

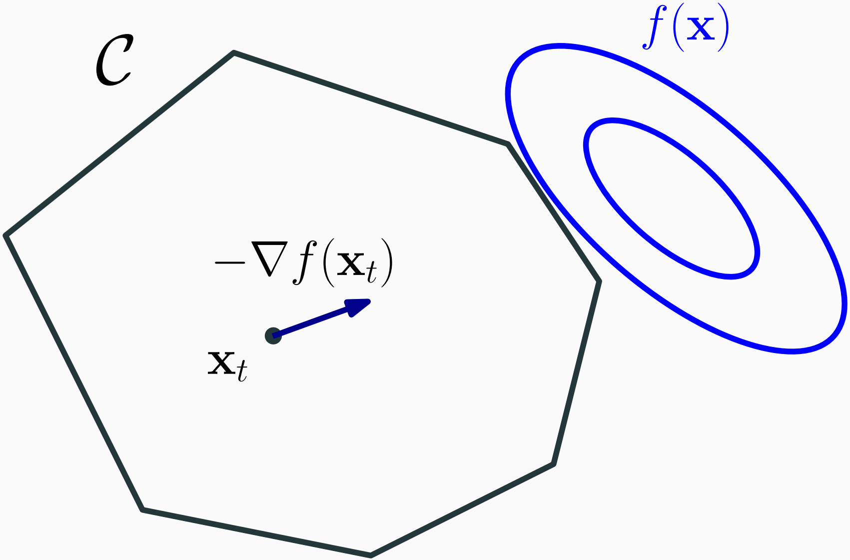
3: $\quad v_t \leftarrow \text{argmin}_{v \in \mathcal{C}} \langle d_t, v \rangle$

4: $\quad \gamma_t \leftarrow \text{StepSizeStrategy}(x_t, t, d_t, v_t)$

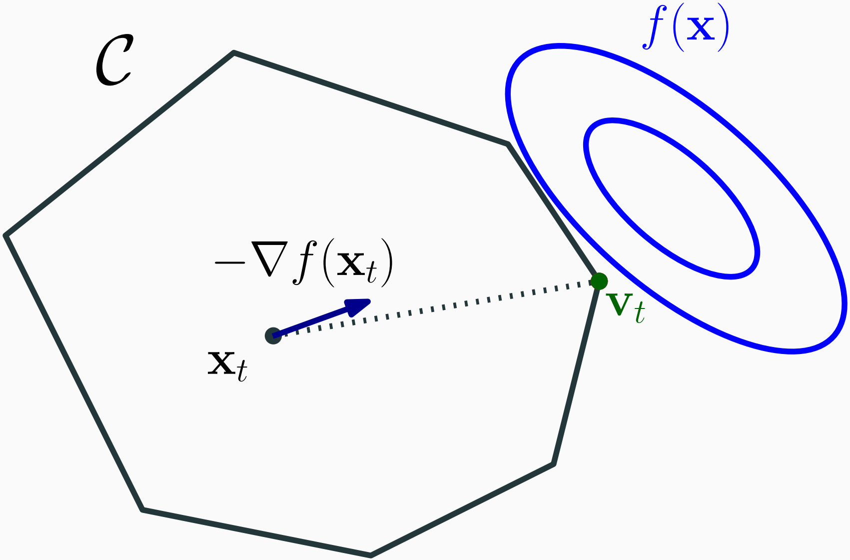
5: $\quad x_{t+1} \leftarrow x_t + \gamma_t (v_t - x_t)$

6: **end for**

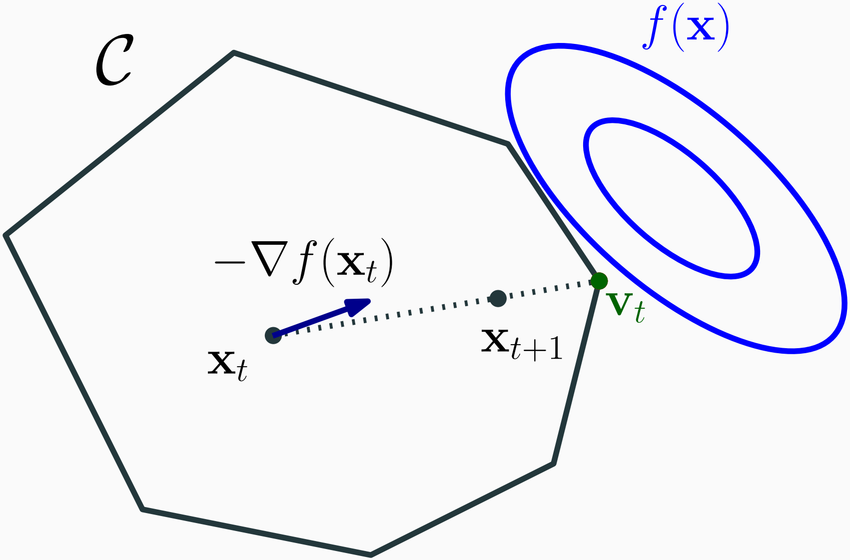
FW, visualized



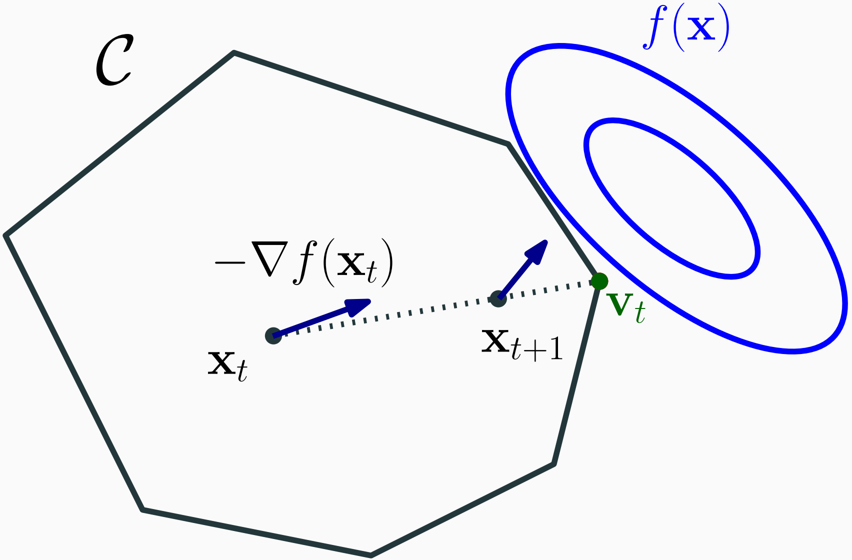
FW, visualized



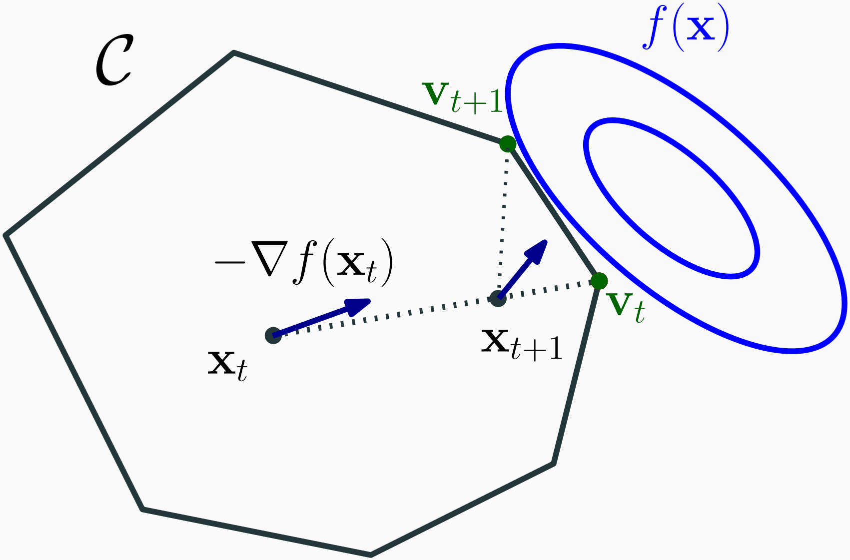
FW, visualized



FW, visualized



FW, visualized



Upper and lower bounds

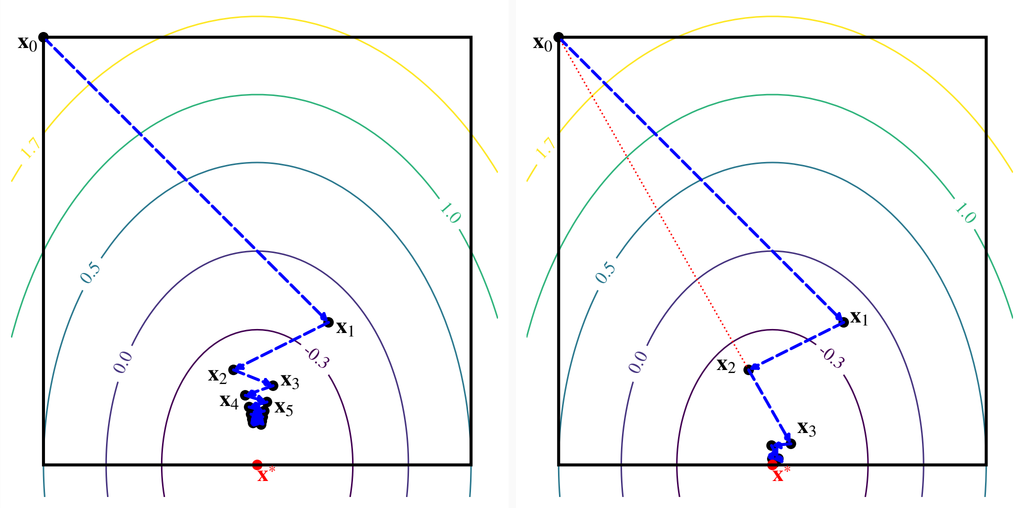
All x_t are feasible by convexity. Furthermore:

$$
\begin{aligned}
f(x) - f(x^*) &\leq \langle \nabla f(x), x - x^* \rangle && \text{(convexity)} \\
&\leq \langle \nabla f(x), x - v \rangle + \langle \nabla f(x), v \rangle - \langle \nabla f(x), x^* \rangle \\
&\leq \langle \nabla f(x), x - v \rangle := \text{FW gap} && (\text{since } v \in \operatorname{argmin} \langle \nabla f(x), v \rangle)
\end{aligned}
$$

Lower bound on optimality as a by-product at each iteration.

Active set representations

Away FW, Blended (Pairwise) Conditional Gradients, ...



Linear Minimization Oracle

How hard is optimizing a linear function over $\mathcal{C}$?

Problem	Feasible set $\mathcal{C}$	Oracle
Generic (P)	Polytope	LP solver: p/d simplex
Sparse regression, discrete measures	$\ell_{1,2,\infty}$ norm balls, simplex	Closed form
Optimal Transport	Transportation polytope	Network simplex
Matching, Markov chains	Birkhoff polytope	Hungarian algorithm
Image segmentation, PCA	Nuclear norm	Greatest singular vector pair
Combinatorics, covariance estimation	Spectraplex	Greatest eigenvector
Discrete problems	conv$(\{x_1 \ldots x_k\})$	Enumeration

Requirement: black box computation of primal value only
$\rightarrow$ e.g. convex hull of Mixed-Integer Problems.

Table of Contents

`FrankWolfe.jl`: yet another implementation?

FW algorithms: deceptive simplicity but no de-facto implementation.
Consequence: reinventing the wheel, potential bugs, performance variability, etc.
Goal: one central toolbox for:

1. **Practitioners** solving optimization problems fitting the form (P),
2. **Researchers** on Frank-Wolfe-type algorithms developing new methods.

One-slide package summary:

Implemented in Julia:

- Compiled to native code, reaches C-like performance;
- Highly generic thanks to multiple dispatch;
- Generic numeric types: reduced (16, 32, 64 bits) and extended (128, GNU MP) precision, rationals;
- Memory-saving mode, in-place gradient computations;
- Switchable components - bring your own LMO / gradient / step size;
- MathOptInterface & JuMP compatibility.

Also a Python wrapper at `ZIB-IOL/frankwolfe-py`.

Main variants

Variant	**Convergence** Progress/iter.	Time/iter.	**Sparsity**	**Numerical Stability**	**Active Set?**	**Lazy?**
FW	Low	Low	Low	High	No	Yes
Away FW	Medium	Medium-High	Medium	Medium-High	Yes	Yes
Stoch. FW	Low	Low	Low	High	No	No
Blended CG	High	Medium	High	Medium	Yes	By design
B-Pair. CG	Medium+	Low	High	High	Yes	By design

More variants

Bring your own component:

1: $d_t \leftarrow \text{FirstOrderEstimate}(f, x_t)$

2: $v_t \leftarrow \text{argmin}_{v \in C} \langle d_t, v \rangle$

3: $\gamma_t \leftarrow \text{StepSizeStrategy}(x_t, t, d_t, v_t)$

Each component has predefined types (simplex LMO, agnostic step size...) and a way to define your own.

Example usage

$$\min_{x \in \Delta} \frac{1}{2} \|x - y\|^2$$

```
using FrankWolfe
using LinearAlgebra
n = 1000
y = randn(n)
function f(x)
    return 1/2 * norm(x - y)^2
end
function grad!(storage, x)
    # perform entrywise without temporary allocation
    @. storage = x - y
end
```

Example usage (2)

```
lmo = FrankWolfe.ProbabilitySimplexOracle(1.0)
x0 = FrankWolfe.compute_extreme_point(lmo, zeros(n))
xfinal, vfinal, primal_value, _ = FrankWolfe.frank_wolfe(
    f, grad!, lmo, x0,
    max_iterations=1000, epsilon=10^-8, # other options
)
```

FrankWolfe.jl, how and where?

Registered on the official Julia package registry:

```
using Pkg
Pkg.add("FrankWolfe")
using FrankWolfe
```

Open-source license (MIT), available on GitHub: `ZIB-IOL/FrankWolfe.jl`
FrankWolfe.jl: a high-performance and flexible toolbox for Frank-Wolfe algorithms and Conditional Gradients
MB, A. Carderera, S. Pokutta, INFORMS Journal on Computing, 2021.

Table of Contents

The exercise repository

1. Clone the repo: `https://github.com/matbesancon/frankwolfe_co_at_work`
2. Open Pluto, see: `https://plutojl.org/install`

Important things on Pluto:

- **Reactive**: Change the value of $x \rightarrow$ change all results using x
- One value only for each variable
- One statement per cell $\rightarrow$ use `begin...` `end` blocks when needed.

The exercises

1. Introduction with a simple problem on the simplex
2. Interactive queries with callbacks
3. Active set Frank-Wolfe methods