

PEPit.jl tutorial: Accelerated inexact forward-backward

John Doe

December 14, 2025

1 Overview

This tutorial shows how to model and solve a **performance estimation problem (PEP)** for an **accelerated inexact forward-backward** method (AIFB), also known as an *inexact accelerated proximal gradient method*.

We consider the composite convex minimization problem

$$F_\star \triangleq \min_x \{F(x) \equiv f(x) + g(x)\}, \quad (1)$$

where:

- f is convex and L -smooth,
- g is closed, proper, and convex,
- $\nabla f(\cdot)$ is available exactly,
- $\text{prox}_{\gamma g}$ is available only **inexactly**, with a relative accuracy parameter $\zeta \in (0, 1)$.

Our goal is to compute the smallest possible $\tau(n, L, \zeta)$ such that the guarantee

$$F(x_n) - F_\star \leq \tau(n, L, \zeta) \|x_0 - x_\star\|^2, \quad (2)$$

is valid for the iterate x_n produced by AIFB, under the normalization $\|x_0 - x_\star\|^2 \leq 1$.

1.1 The AIFB algorithm

We implement a simplified instance of [1, Algorithm 3.1] (with the same parameter choices as in the reference example), namely

$$\begin{aligned} A_{t+1} &= A_t + \frac{\eta + \sqrt{\eta^2 + 4\eta A_t}}{2}, \\ y_t &= x_t + \frac{A_{t+1} - A_t}{A_{t+1}}(z_t - x_t), \\ (x_{t+1}, v_{t+1}) &\approx_{\varepsilon_t} \left(\text{prox}_{\gamma g}(y_t - \gamma \nabla f(y_t)), \text{prox}_{g^*/\gamma} \left(\frac{y_t - \gamma \nabla f(y_t)}{\gamma} \right) \right), \\ \varepsilon_t &= \frac{\zeta^2 \gamma^2}{2} \|v_{t+1} + \nabla f(y_t)\|^2, \\ z_{t+1} &= z_t - (A_{t+1} - A_t)(v_{t+1} + \nabla f(y_t)), \end{aligned} \quad (3)$$

with $\gamma = \frac{1}{L}$ and $\eta = (1 - \zeta^2)\gamma$.

1.2 Baseline theoretical guarantee

We will compare the PEPit-computed bound with the theoretical upper bound from [1, Corollary 3.5],

$$F(x_n) - F_\star \leq \frac{2L}{(1 - \zeta^2)n^2} \|x_0 - x_\star\|^2. \quad (4)$$

2 Setup

We start by loading PEPit.jl and OrderedCollections.jl.

- PEPit.jl provides the symbolic objects (Point, Expression, constraints) and the SDP builder/solver (solve!).
- OrderedCollections.jl provides OrderedDict, which this codebase uses for deterministic parameter passing.

```
using PEPit # Load PEPit.jl (PEP modeling primitives and solver endpoint).
using OrderedCollections # Use 'OrderedDict' for deterministic parameter dictionaries.
```

3 Step 1: Instantiate an empty PEP

We create an empty PEP() object that we will populate with:

- declared function classes (f and g),
- initial conditions (here $\|x_0 - x_\star\|^2 \leq 1$),
- algorithmic relations (the AIFB iteration),
- and the performance metric $F(x_n) - F_\star$.

```
problem = PEP() # Create an empty PEP container (lists of
↪ functions/points/constraints/metrics).
```

4 Step 2: Choose numerical parameters

We set:

- the smoothness constant L of f ,
- the relative inexactness parameter ζ ,
- the number of iterations n .

The reference example uses $L = 1.3$, $\zeta = 0.45$, and $n = 11$.

```

L = 1.3 # Smoothness constant L for the class of L-smooth convex functions.
ζ = 0.45 # Relative inexactness parameter ζ ∈ (0,1) controlling the proximal accuracy.
n = 11 # Number of AIFB iterations n.
verbose = true # Toggle verbose PEPit/solver output.

```

5 Step 3: Declare the function classes f and g

We declare:

- f as a smooth convex function with parameter L ,
- g as a (possibly nonsmooth) convex closed proper function.

Then we form the composite objective $F = f + g$.

```

f = declare_function!(problem, SmoothConvexFunction, OrderedDict("L" => L);
↳ reuse_gradient=true) # Declare f: convex and L-smooth (unique gradient at each point).
g = declare_function!(problem, ConvexFunction, OrderedDict()); reuse_gradient=false) # Declare
↳ g: closed, proper, convex (subgradients need not be unique).
F = f + g # Define the composite objective F(x) = f(x) + g(x).

```

6 Step 4: Model x_* and impose the initial condition

In a PEP, we do not search over points directly. Instead, we introduce **symbolic samples** of the oracles that must be consistent with the function class.

To represent an optimizer x_* , we record a *stationary* sample of the composite objective:

$$0 \in \partial F(x_*).$$

Then we introduce the initial point x_0 and impose the normalization $\|x_0 - x_*\|^2 \leq 1$.

```

xs = stationary_point!(F) # Create a symbolic minimizer x_* by enforcing 0 ∈ ∂F(x_*).
F_star = value!(F, xs) # Define the optimal value F_* = F(x_*).
x0 = set_initial_point!(problem) # Create the initial point x_0 (a new leaf 'Point').
set_initial_condition!(problem, (x0 - xs)^2 <= 1) # Impose ||x_0 - x_*||^2 ≤ 1 as the initial
↳ condition.

```

7 Step 5: Encode the AIFB iteration (Equation @eq-aifb-iter)

We now implement the iteration in Equation (3).

7.1 About the inexact proximal step in PEPit.jl

We call `inexact_proximal_step!` with `opt="PD_gapI"` to model an **inexact** proximal computation of g at an input x_0 . The primitive returns an `Expression` `eps_var` that represents an upper bound on a primal-dual gap quantity (see `src/primitive_steps/inexact_proximal_step.jl`).

To match Equation (3), we add the extra (relative) accuracy constraint

$$\varepsilon_t \leq \frac{\zeta^2 \gamma^2}{2} \|v_{t+1} + \nabla f(y_t)\|^2. \quad (5)$$

This is the key modeling step that couples the inexactness level to the algorithm state.

```

γ = 1 / L # Set γ = 1/L (forward step-size).
η = (1 - ζ^2) * γ # Set η = (1 - ζ^2)γ (parameter used in A_{t+1}).
A = 0.0 # Initialize the acceleration scalar A_0 = 0.
x = x0 # Initialize the iterate x_0.
z = x0 # Initialize the auxiliary iterate z_0.
hx = nothing # Placeholder for the final value g(x_n) (filled inside the loop).

for t in 0:(n - 1) # Run n iterations, indexed as t = 0, ..., n - 1.
    A_next = A + (η + sqrt(η^2 + 4 * η * A)) / 2 # Update A_{t+1} = A_t + \frac{\eta + \sqrt{\eta^2 + 4\eta A_t}}{2}.
    y = x + (1 - A / A_next) * (z - x) # Extrapolation: y_t = x_t + \frac{A_{t+1} - A_t}{A_{t+1}}(z_t - x_t).
    gy = gradient!(f, y) # Gradient query: g_y = ∇f(y_t).
    x_next, _, hx_next, _, v_next, _, eps_var = inexact_proximal_step!(y - γ * gy, g, γ;
        ↪ opt="PD_gapI") # Inexact prox on g at y_t - γ∇f(y_t) (returns x_{t+1}, v_{t+1}, ε_t).
    add_constraint!(g, eps_var <= (ζ * γ)^2 / 2 * (v_next + gy)^2) # Enforce Equation
        ↪ @eq-aifb-accuracy: ε_t ≤ \frac{\zeta^2 \gamma^2}{2} \|v_{t+1} + ∇f(y_t)\|^2.
    z = z - (A_next - A) * (v_next + gy) # Update z_{t+1} = z_t - (A_{t+1} - A_t)(v_{t+1} + ∇f(y_t)).
    x = x_next # Commit the new iterate x_{t+1}.
    hx = hx_next # Store g(x_{t+1}) to later build F(x_n) = f(x_n) + g(x_n).
    A = A_next # Commit the scalar update A_t ← A_{t+1}.
end # End the AIFB loop.

```

8 Step 6: Define the performance metric and solve the PEP

The performance metric is the final objective error:

$$F(x_n) - F_\star = f(x_n) + g(x_n) - F_\star.$$

In PEPit, we encode this quantity as an Expression and pass it to `set_performance_metric!`. Then `solve!` constructs and solves the SDP and returns the worst-case value.

```

set_performance_metric!(problem, value!(f, x) + hx - F_star) # Set the performance metric to
    ↪ F(x_n) - F_\star.
pepit_tau = solve!(problem; verbose=verbose) # Solve the SDP and return the worst-case value
    ↪ τ(n, L, ζ).

```

9 Step 7: Compare with the theoretical upper bound (Equation @eq-aifb-theory)

The theory predicts the upper bound in Equation (4):

$$\tau_{\text{th}}(n, L, \zeta) = \frac{2L}{(1 - \zeta^2)n^2}.$$

We compute it and print both values.

```
theoretical_tau = 2 * L / (1 - ζ^2) / n^2 # Compute  $\tau_{\text{th}} = \frac{2L}{(1-\zeta^2)n^2}$  from Equation
↪ @eq-aifb-theory.

@info "Ⓜ Accelerated inexact forward-backward (AIFB) ***" # Print a short header.

@info "Ⓜ PEPit guarantee:\t F(x_n)-F_* <= $(round(pepit_tau, digits=7)) ||x_0 - x_*||^2" #
↪ Report the PEPit-computed value.

@info "Ⓜ Theoretical bound:\t F(x_n)-F_* <= $(round(theoretical_tau, digits=7)) ||x_0 -
↪ x_*||^2" # Report the theoretical comparison bound.
```

9.1 Expected output (for the reference parameters)

Assuming everything went well, with $L = 1.3$, $\zeta = 0.45$, and $n = 11$, the reference run reports approximately:

```
PEPit guarantee:      F(x_n)-F_* <= 0.0187341 ||x_0 - x_*||^2
Theoretical bound:    F(x_n)-F_* <= 0.0269437 ||x_0 - x_*||^2
```

Our numerical values may differ slightly depending on the SDP solver and tolerances.

10 Reference

[1] M. Barre, A. Taylor, F. Bach (2021). *A note on approximate accelerated forward-backward methods with absolute and relative errors, and possibly strongly convex objectives.* arXiv:2106.15536v2. URL: <https://arxiv.org/pdf/2106.15536v2.pdf>