# First-Order Methods for Smooth Convex Optimization in Isabelle/HOL

Feier Lyu

August 12, 2026

**Abstract**

This entry develops reusable Isabelle/HOL infrastructure for first-order methods in smooth convex optimization. The central application is projected-gradient descent, but the development is organized around general-purpose interfaces for gradients, first-order convexity certificates, smooth quadratic upper bounds, descent and telescoping arguments, projection geometry, projected-gradient mappings, residual certificates, strong convexity, and linear convergence rates. The main purpose of the entry is not only to formalize a single convergence proof, but to separate the analytic, geometric, and algorithmic components of first-order convergence arguments into reusable Isabelle/HOL layers. The resulting library can be used as a basis for future formalizations of constrained first-order optimization methods.

## Contents

# 1 Introduction

First-order methods are among the basic algorithmic tools of convex optimization. Their convergence proofs are usually built from a small number of reusable ingredients: first-order convexity inequalities, smooth quadratic upper bounds, one-step descent estimates, telescoping arguments, and, in the constrained case, projection inequalities. This entry formalizes these ingredients in Isabelle/HOL and combines them to obtain convergence guarantees for gradient descent and projected-gradient descent.

The focus of the development is smooth convex optimization. In the unconstrained case, the main algorithmic object is the gradient step. In the constrained case, the main object is the projected-gradient step, together with the associated projected-gradient mapping. The projected-gradient mapping plays the role of a constrained residual: its vanishing characterizes a projected-gradient fixed point and, under convexity assumptions, a global minimizer.

The entry also proves finite-horizon residual estimates, which give an explicit approximate-stationarity certificate for projected-gradient descent. The development is intended as a reusable library rather than a standalone formalization of one theorem. The theories are organized so that the algorithm-independent parts, such as descent and telescoping lemmas, are separated from gradient-specific and projection-specific arguments. This separation is meant to make the entry useful for later formalizations of other first-order methods.

The mathematical presentation follows standard references on convex optimization and first-order methods, such as [1, 2, 5, 6]. The entry also builds on the existing Isabelle/HOL infrastructure for analysis, convexity, and projections.

# 2 Main contributions

The main contributions of the entry are as follows.

- It provides reusable gradient and first-order convexity interfaces for smooth convex optimization in Isabelle/HOL.
- It separates analytic smoothness certificates from algorithmic convergence proofs through a quadratic smooth upper-bound interface and related line-descent formulations.

- It develops an algorithm-independent descent layer based on telescoping and averaging arguments, so that one-step progress estimates can be reused in convergence proofs for first-order methods.
- It formalizes convergence proofs for gradient descent and projected-gradient descent under smooth convex assumptions.
- It develops projection geometry and projected-gradient mapping residuals as reusable tools for constrained smooth convex optimization.
- It proves finite-horizon residual and epsilon-stationarity certificates for projected-gradient descent.
- It adds strong-convexity refinements, including distance-gap estimates and linear-rate convergence results.
- It provides concrete quadratic examples and a public theorem interface intended for downstream developments.

## 3 Structure of the development

The formalization is organized into several layers.

**Analytic and convexity interfaces.** The theories `Gradient_Preliminaries` and `Convex_Differentiable` contain the basic gradient and first-order convexity infrastructure used throughout the entry. They isolate the first-order inequalities needed in convergence proofs from the later algorithmic arguments.

**Smooth upper bounds and descent estimates.** The theories `Smooth_Convex` and `Lipschitz_Smoothness` develop the smoothness interface. The central object is the standard quadratic upper bound for smooth functions. This upper bound is the main tool used to derive one-step progress estimates for gradient descent and projected-gradient descent.

**Algorithm-independent descent tools.** The theory `Abstract_Descent` contains telescoping, summability, and averaging lemmas. These lemmas are deliberately separated from any particular algorithm. Once an algorithm supplies a suitable one-step progress inequality, the same abstract descent layer can be used to obtain finite-horizon convergence estimates.

**Gradient descent.** The theories `Gradient_Descent`, `Gradient_Descent_Rates`, and `Gradient_Descent_Convergence` instantiate the smoothness and descent infrastructure for the ordinary gradient method. They prove the standard sublinear function-value estimates for smooth convex minimization.

**Projection geometry and projected-gradient descent.** The theories `Projection_Geometry` and `Projection_Optimization` collect the projection inequalities and variational characterizations needed for constrained optimization. The theory `Projected_Gradient_Descent_Convergence` then proves the main convergence results for projected-gradient descent over a closed convex set.

**Projected-gradient mappings and residual certificates.** The theories `Projected_Gradient_Mapping`, `Projected_Gradient_Mapping_Rates`, and `Projected_Gradient_Descent_Residual_Convergence` develop the projected-gradient mapping as a constrained first-order residual. This part of the entry proves fixed-point and optimality characterizations, summability and averaging estimates for the residual norm, and finite-time epsilon-stationarity certificates.

**Strong convexity and linear convergence.** The theories `Strong_Convex` and `Projected_Gradient_Descent_Linear_Rate` formalize the strong-convexity refinements. These results upgrade the convex convergence statements to linear-rate estimates under the usual strong-convexity and smoothness assumptions.

**Examples and public interface.** The example theories instantiate the abstract interfaces on simple quadratic objectives. The theory `Main_Results` collects the most important theorem groups and is intended to be the main import target for downstream developments. The theory `Examples_Using_Main_Results` demonstrates how a client development can use the stable public theorem surface without importing the lower-level proof files directly. The final umbrella theory `Projected_Gradient_Descent` imports the complete entry.

# 4 Related work

The Isabelle/HOL libraries already provide substantial infrastructure for analysis, convex sets, convex functions, and metric projections. This entry builds on that background and develops an algorithmic layer for first-order methods in smooth convex optimization.

The AFP entry *Unconstrained Optimization* formalizes minimizers, strict and isolated local minimizers, and first- and second-order optimality conditions for unconstrained problems [3]. The present development is complementary: it focuses not on local optimality conditions for unconstrained problems, but on convergence proofs for gradient descent and projected-gradient descent, together with projection geometry, projected-gradient mappings, and residual certificates for constrained smooth convex optimization.

Outside Isabelle/HOL, Li et al. formalize convergence rates of several first-order algorithms for convex optimization in Lean4 [4]. Their work develops Lean infrastructure for gradients, subgradients, differentiable convex functions, and convergence proofs for a range of first-order algorithms. The present entry contributes complementary Isabelle/HOL/AFP infrastructure, with particular emphasis on projection geometry, projected-gradient descent, projected-gradient mappings, and residual certificates for constrained smooth convex optimization.

# 5 Conclusion

This entry formalizes a reusable Isabelle/HOL layer for smooth convex first-order optimization. Its main contributions are the separation of gradient, smoothness, descent, projection, and residual arguments; convergence proofs for gradient descent and projected-gradient descent; projected-gradient mapping optimality and residual estimates; and strong-convexity linear-rate refinements. The structure of the development is intended to support future extensions.

Natural next steps include proximal-gradient methods, conditional-gradient methods, accelerated methods, additional constrained examples such as box or polyhedral constraints, and sharper analytic bridges from Lipschitz-gradient assumptions to the quadratic upper-bound interface, for example via integral descent-lemma arguments along line segments.

```
theory Gradient_Preliminaries
  imports
    "HOL-Analysis.Convex_Euclidean_Space"
    "HOL-Analysis.Lipschitz"
begin
```

# 6 Gradient preliminaries

This theory provides a thin optimization-oriented wrapper around Isabelle/HOL's existing Fréchet derivative interface.

For a real-valued function on a real inner product space, saying that *f* has gradient *g* at *x* means that the Fréchet derivative of *f* at *x* is the linear map sending a direction *h* to *inner h g*. This is the usual gradient convention used in finite-dimensional optimization.

## 6.1 Pointwise gradients

```
definition has_gradient ::
  "('a::real_inner ⇒ real) ⇒ 'a ⇒ 'a ⇒ bool"
where
  "has_gradient f x g ⟷
```

```
    (f has_derivative (λh. inner h g)) (at x)"

definition has_gradient_within ::
  "('a::real_inner ⇒ real) ⇒ 'a set ⇒ 'a ⇒ 'a ⇒ bool"
where
  "has_gradient_within f S x g ⟷
    (f has_derivative (λh. inner h g)) (at x within S)"

lemma has_gradientD:
  assumes "has_gradient f x g"
  shows "(f has_derivative (λh. inner h g)) (at x)"
  ⟨proof⟩

lemma has_gradientI:
  assumes "(f has_derivative (λh. inner h g)) (at x)"
  shows "has_gradient f x g"
  ⟨proof⟩

lemma has_gradient_withinD:
  assumes "has_gradient_within f S x g"
  shows "(f has_derivative (λh. inner h g)) (at x within S)"
  ⟨proof⟩

lemma has_gradient_withinI:
  assumes "(f has_derivative (λh. inner h g)) (at x within S)"
  shows "has_gradient_within f S x g"
  ⟨proof⟩

lemma has_gradient_imp_differentiable:
  assumes "has_gradient f x g"
  shows "f differentiable (at x)"
  ⟨proof⟩

lemma has_gradient_within_imp_differentiable:
  assumes "has_gradient_within f S x g"
  shows "f differentiable (at x within S)"
  ⟨proof⟩
```

## 6.2 Uniqueness and the gradient operator

At an unrestricted point, the gradient is unique. This follows from uniqueness of the Fréchet derivative and non-degeneracy of the inner product.

```
lemma has_gradient_unique:
  fixes f :: "'a::real_inner ⇒ real"
  assumes g: "has_gradient f x g"
    and h: "has_gradient f x h"
  shows "g = h"
⟨proof⟩
```

**definition** *gradient ::*
  *"('a::real_inner ⇒ real) ⇒ 'a ⇒ 'a"*
**where**
  *"gradient f x = (THE g. has_gradient f x g)"*

**lemma** *gradient_eqI:*
  **assumes** *"has_gradient f x g"*
  **shows** *"gradient f x = g"*
  ⟨*proof*⟩

**lemma** *has_gradient_gradient:*
  **assumes** *"has_gradient f x g"*
  **shows** *"has_gradient f x (gradient f x)"*
  ⟨*proof*⟩

**lemma** *gradient_eq_iff:*
  **assumes** *"has_gradient f x g"*
  **shows** *"gradient f x = h ⟷ g = h"*
  ⟨*proof*⟩

## 6.3 Gradient fields on sets

For optimization proofs it is often cleaner to assume a named gradient field rather than repeatedly unpacking the gradient at every point.

**definition** *has_gradient_on ::*
  *"('a::real_inner ⇒ real) ⇒ 'a set ⇒ ('a ⇒ 'a) ⇒ bool"*
**where**
  *"has_gradient_on f S G ⟷ (∀x∈S. has_gradient f x (G x))"*

**definition** *has_gradient_within_on ::*
  *"('a::real_inner ⇒ real) ⇒ 'a set ⇒ ('a ⇒ 'a) ⇒ bool"*
**where**
  *"has_gradient_within_on f S G ⟷*
    *(∀x∈S. has_gradient_within f S x (G x))"*

**lemma** *has_gradient_onD:*
  **assumes** *"has_gradient_on f S G"*
    **and** *"x ∈ S"*
  **shows** *"has_gradient f x (G x)"*
  ⟨*proof*⟩

**lemma** *has_gradient_within_onD:*
  **assumes** *"has_gradient_within_on f S G"*
    **and** *"x ∈ S"*
  **shows** *"has_gradient_within f S x (G x)"*
  ⟨*proof*⟩

**lemma** *has_gradient_on_imp_differentiable:*
  **assumes** *"has_gradient_on f S G"*

and "x ∈ S"
shows "f differentiable (at x)"
⟨proof⟩

**lemma** *has_gradient_within_on_imp_differentiable:*
  **assumes** "has_gradient_within_on f S G"
    **and** "x ∈ S"
  **shows** "f differentiable (at x within S)"
  ⟨proof⟩

## 6.4 Elementary gradient rules

These lemmas are readable wrappers around standard derivative rules. They are included here so that later optimization proofs can be stated in gradient language rather than raw Fréchet derivative language.

**lemma** *has_gradient_const [simp]:*
  "has_gradient (λx. c) x 0"
  ⟨proof⟩

**lemma** *has_gradient_add:*
  **assumes** "has_gradient f x df"
    **and** "has_gradient g x dg"
  **shows** "has_gradient (λy. f y + g y) x (df + dg)"
⟨proof⟩

**lemma** *has_gradient_minus:*
  **assumes** "has_gradient f x df"
  **shows** "has_gradient (λy. - f y) x (- df)"
⟨proof⟩

**lemma** *has_gradient_diff:*
  **assumes** "has_gradient f x df"
    **and** "has_gradient g x dg"
  **shows** "has_gradient (λy. f y - g y) x (df - dg)"
⟨proof⟩

**lemma** *has_gradient_scale_const:*
  **assumes** "has_gradient f x df"
  **shows** "has_gradient (λy. c * f y) x (scaleR c df)"
⟨proof⟩

**lemma** *has_gradient_mult:*
  **assumes** "has_gradient f x df"
    **and** "has_gradient g x dg"
  **shows** "has_gradient (λy. f y * g y) x
    (scaleR (f x) dg + scaleR (g x) df)"
⟨proof⟩

**lemma** *has_gradient_neg_scale_const:*

```
  assumes "has_gradient f x df"
  shows "has_gradient (λy. - c * f y) x (scaleR (- c) df)"
  ⟨proof⟩
```

## 6.5 Linear functions and inner products

The following lemmas are useful for examples and for algebraic rewriting in projection proofs.

```
lemma has_gradient_inner_left [simp]:
  fixes a x :: "'a::real_inner"
  shows "has_gradient (λy. inner a y) x a"
  ⟨proof⟩

lemma has_gradient_inner_right [simp]:
  fixes a x :: "'a::real_inner"
  shows "has_gradient (λy. inner y a) x a"
  ⟨proof⟩

lemma has_gradient_inner_diff_left [simp]:
  fixes a b x :: "'a::real_inner"
  shows "has_gradient (λy. inner a (y - b)) x a"
⟨proof⟩

lemma has_gradient_inner_diff_right [simp]:
  fixes a b x :: "'a::real_inner"
  shows "has_gradient (λy. inner (y - b) a) x a"
⟨proof⟩
```

## 6.6 Affine lines and directional derivatives

Gradient descent and projected gradient descent proofs often restrict a multivariate function to an affine line, namely the map that sends `t` to `f (x + scaleR t d)`.

The derivative of this one-dimensional restriction is the directional derivative

inner d (gradient f (x + scaleR t d)).

```
definition affine_line ::
  "'a::real_vector ⇒ 'a ⇒ real ⇒ 'a"
where
  "affine_line x d t = x + scaleR t d"

definition restrict_to_line ::
  "('a::real_vector ⇒ real) ⇒ 'a ⇒ 'a ⇒ real ⇒ real"
where
  "restrict_to_line f x d t = f (affine_line x d t)"

lemma affine_line_0 [simp]:
  "affine_line x d 0 = x"
```

⟨*proof*⟩

**lemma** *affine_line_1 [simp]:*
  *"affine_line x d 1 = x + d"*
  ⟨*proof*⟩

**lemma** *affine_line_has_derivative:*
  *"((affine_line x d) has_derivative (λr. scaleR r d)) (at t)"*
  ⟨*proof*⟩

**lemma** *has_gradient_restrict_to_line:*
  **fixes** *f :: "'a::real_inner ⇒ real"*
  **assumes** *"has_gradient f (affine_line x d t) g"*
  **shows** *"((restrict_to_line f x d) has_field_derivative inner d g) (at t)"*
⟨*proof*⟩

**lemma** *has_gradient_restrict_to_line_at_0:*
  **fixes** *f :: "'a::real_inner ⇒ real"*
  **assumes** *"has_gradient f x g"*
  **shows** *"((λt::real. f (x + scaleR t d)) has_field_derivative inner d g) (at 0)"*
⟨*proof*⟩

## 6.7 Inner-product and norm algebra

These small algebraic lemmas are intentionally placed early. Later convergence proofs repeatedly expand squared norms and telescope inequalities.

**lemma** *inner_self_eq_norm_sq:*
  **fixes** *x :: "'a::real_inner"*
  **shows** *"inner x x = norm x ^ 2"*
  ⟨*proof*⟩

**lemma** *norm_sq_nonneg [simp]:*
  **fixes** *x :: "'a::real_inner"*
  **shows** *"0 ≤ norm x ^ 2"*
  ⟨*proof*⟩

**lemma** *norm_sq_add:*
  **fixes** *x y :: "'a::real_inner"*
  **shows** *"norm (x + y) ^ 2 =
    norm x ^ 2 + 2 * inner x y + norm y ^ 2"*
⟨*proof*⟩

**lemma** *norm_sq_diff:*
  **fixes** *x y :: "'a::real_inner"*
  **shows** *"norm (x - y) ^ 2 =
    norm x ^ 2 - 2 * inner x y + norm y ^ 2"*
⟨*proof*⟩

```
lemma norm_sq_diff_commute:
  fixes x y :: "'a::real_inner"
  shows "norm (x - y) ^ 2 = norm (y - x) ^ 2"
  ⟨proof⟩

lemma inner_diff_self:
  fixes x y :: "'a::real_inner"
  shows "inner (x - y) (x - y) = norm (x - y) ^ 2"
  ⟨proof⟩

lemma inner_le_norm:
  fixes x y :: "'a::real_inner"
  shows "inner x y ≤ norm x * norm y"
  ⟨proof⟩

lemma abs_inner_le_norm:
  fixes x y :: "'a::real_inner"
  shows "abs (inner x y) ≤ norm x * norm y"
  ⟨proof⟩
```

## 6.8 Optimization-oriented notation helpers

These are tiny wrappers, but they make later statements read like optimization rather than raw analysis.

```
definition zero_gradient_at ::
  "('a::real_inner ⇒ real) ⇒ 'a ⇒ bool"
where
  "zero_gradient_at f x ⟷ has_gradient f x 0"

definition stationary_at ::
  "('a::real_inner ⇒ real) ⇒ 'a ⇒ bool"
where
  "stationary_at f x ⟷ has_gradient f x 0"

lemma zero_gradient_at_iff_stationary_at:
  "zero_gradient_at f x ⟷ stationary_at f x"
  ⟨proof⟩

lemma stationary_atD:
  assumes "stationary_at f x"
  shows "has_gradient f x 0"
  ⟨proof⟩

lemma stationary_atI:
  assumes "has_gradient f x 0"
  shows "stationary_at f x"
  ⟨proof⟩
```

```
end
theory Convex_Differentiable
  imports Gradient_Preliminaries
begin
```

# 7 Convex differentiable functions and first-order certificates

This theory introduces the first-order certificate language used later for gradient descent and projected gradient descent.

The file isolates four basic notions:

global minimizers on a feasible set; first-order variational inequalities; supporting affine lower bounds; convex differentiable functions with a named gradient field.

The main result is the standard first-order supporting-hyperplane property: if f is convex and has gradient g at x, then

the affine first-order lower bound at x is below f y

for every feasible y.

Consequently, every convex differentiable function with a named gradient field satisfies a global gradient lower-bound property. Combined with the variational first-order condition, this gives a global optimality certificate.

## 7.1 Global minimizers

```
definition global_min_on ::
  "'a set ⇒ ('a ⇒ real) ⇒ 'a ⇒ bool"
where
  "global_min_on S f x ⟷ x ∈ S ∧ (∀y∈S. f x ≤ f y)"

lemma global_min_onI:
  assumes "x ∈ S"
    and "⋀y. y ∈ S ⟹ f x ≤ f y"
  shows "global_min_on S f x"
  ⟨proof⟩

lemma global_min_onD_mem:
  assumes "global_min_on S f x"
  shows "x ∈ S"
  ⟨proof⟩

lemma global_min_onD:
  assumes "global_min_on S f x"
    and "y ∈ S"
  shows "f x ≤ f y"
  ⟨proof⟩
```

```
lemma global_min_on_singleton [simp]:
  "global_min_on {x} f x"
  ⟨proof⟩

lemma global_min_on_UNIV_iff:
  "global_min_on UNIV f x ⟷ (∀y. f x ≤ f y)"
  ⟨proof⟩
```

## 7.2 First-order variational inequalities

For constrained convex minimization, the first-order condition at x with gradient g is

the directional inner product inner g (y - x) is nonnegative

for every feasible y.

This is the variational-inequality form of first-order optimality.

```
definition first_order_condition_at ::
  "'a::real_inner set ⇒ 'a ⇒ 'a ⇒ bool"
where
  "first_order_condition_at S g x ⟷
     x ∈ S ∧ (∀y∈S. 0 ≤ inner g (y - x))"

lemma first_order_condition_atI:
  assumes "x ∈ S"
    and "⋀y. y ∈ S ⟹ 0 ≤ inner g (y - x)"
  shows "first_order_condition_at S g x"
  ⟨proof⟩

lemma first_order_condition_atD_mem:
  assumes "first_order_condition_at S g x"
  shows "x ∈ S"
  ⟨proof⟩

lemma first_order_condition_atD:
  assumes "first_order_condition_at S g x"
    and "y ∈ S"
  shows "0 ≤ inner g (y - x)"
  ⟨proof⟩

lemma first_order_condition_zero_iff_mem [simp]:
  "first_order_condition_at S 0 x ⟷ x ∈ S"
  ⟨proof⟩
```

## 7.3 Supporting affine lower bounds

A vector g supports f at x on S if the corresponding affine first-order approximation is a global lower bound for f on S.

For differentiable convex functions, g will later be the gradient at x.

```
definition supports_at ::
```

```
  "'a::real_inner set ⇒ ('a ⇒ real) ⇒ 'a ⇒ 'a ⇒ bool"
where
  "supports_at S f x g ⟷
     x ∈ S ∧ (∀y∈S. f x + inner g (y - x) ≤ f y)"

definition supports_on ::
  "'a::real_inner set ⇒ ('a ⇒ real) ⇒ ('a ⇒ 'a) ⇒ bool"
where
  "supports_on S f G ⟷ (∀x∈S. supports_at S f x (G x))"

lemma supports_atI:
  assumes "x ∈ S"
    and "⋀y. y ∈ S ⟹ f x + inner g (y - x) ≤ f y"
  shows "supports_at S f x g"
  ⟨proof⟩

lemma supports_atD_mem:
  assumes "supports_at S f x g"
  shows "x ∈ S"
  ⟨proof⟩

lemma supports_atD:
  assumes "supports_at S f x g"
    and "y ∈ S"
  shows "f x + inner g (y - x) ≤ f y"
  ⟨proof⟩

lemma supports_onI:
  assumes "⋀x. x ∈ S ⟹ supports_at S f x (G x)"
  shows "supports_on S f G"
  ⟨proof⟩

lemma supports_onD:
  assumes "supports_on S f G"
    and "x ∈ S"
  shows "supports_at S f x (G x)"
  ⟨proof⟩

lemma supports_at_zero_iff_global_min_on:
  "supports_at S f x 0 ⟷ global_min_on S f x"
  ⟨proof⟩

lemma supports_at_zero_imp_global_min_on:
  assumes "supports_at S f x 0"
  shows "global_min_on S f x"
  ⟨proof⟩

lemma global_min_on_imp_supports_at_zero:
  assumes "global_min_on S f x"
```

```
  shows "supports_at S f x 0"
  ⟨proof⟩

lemma supports_at_and_first_order_condition_imp_global_min_on:
  assumes supp: "supports_at S f x g"
    and foc: "first_order_condition_at S g x"
  shows "global_min_on S f x"
⟨proof⟩

lemma supports_at_gradient_and_foc_imp_global_min_on:
  assumes supp: "supports_at S f x (gradient f x)"
    and foc: "first_order_condition_at S (gradient f x) x"
  shows "global_min_on S f x"
  ⟨proof⟩
```

## 7.4 Gradient lower-bound fields

This is the pointwise supporting-hyperplane property written with a named gradient field G.

Later, for convex differentiable functions, we will prove this property from convexity and differentiability.

```
definition gradient_lower_bound_on ::
  "'a::real_inner set ⇒ ('a ⇒ real) ⇒ ('a ⇒ 'a) ⇒ bool"
where
  "gradient_lower_bound_on S f G ⟷
     (∀x∈S. ∀y∈S. f x + inner (G x) (y - x) ≤ f y)"

lemma gradient_lower_bound_onI:
  assumes "⋀x y. x ∈ S ⟹ y ∈ S ⟹ f x + inner (G x) (y - x) ≤ f
y"
  shows "gradient_lower_bound_on S f G"
  ⟨proof⟩

lemma gradient_lower_bound_onD:
  assumes "gradient_lower_bound_on S f G"
    and "x ∈ S"
    and "y ∈ S"
  shows "f x + inner (G x) (y - x) ≤ f y"
  ⟨proof⟩

lemma gradient_lower_bound_on_imp_supports_on:
  assumes "gradient_lower_bound_on S f G"
  shows "supports_on S f G"
⟨proof⟩

lemma gradient_lower_bound_on_and_foc_imp_global_min_on:
  assumes lb: "gradient_lower_bound_on S f G"
    and foc: "first_order_condition_at S (G x) x"
  shows "global_min_on S f x"
```

⟨*proof*⟩

## 7.5 Convexity along feasible segments

The next lemmas convert the standard convex-combination statement into the optimization form x + t * (y - x).

```
lemma convex_contains_affine_line:
  fixes x y :: "'a::real_vector"
  assumes "convex S"
    and "x ∈ S"
    and "y ∈ S"
    and "0 ≤ t"
    and "t ≤ 1"
  shows "x + scaleR t (y - x) ∈ S"
⟨proof⟩

lemma convex_on_affine_lineD:
  fixes x y :: "'a::real_vector"
  assumes "convex_on S f"
    and "x ∈ S"
    and "y ∈ S"
    and "0 ≤ t"
    and "t ≤ 1"
  shows "f (x + scaleR t (y - x)) ≤ (1 - t) * f x + t * f y"
⟨proof⟩

lemma convex_on_affine_lineD_open01:
  fixes x y :: "'a::real_vector"
  assumes "convex_on S f"
    and "x ∈ S"
    and "y ∈ S"
    and "0 < t"
    and "t < 1"
  shows "f (x + scaleR t (y - x)) ≤ (1 - t) * f x + t * f y"
  ⟨proof⟩
```

## 7.6 First-order lower bound for convex differentiable functions

```
lemma convex_line_secant_slope_bound:
  fixes f :: "'a::real_inner ⇒ real"
  assumes conv: "convex_on S f"
    and x: "x ∈ S"
    and y: "y ∈ S"
    and tpos: "0 < t"
    and tle: "t ≤ 1"
  shows "(f (x + t *R (y - x)) - f x) / t ≤ f y - f x"
⟨proof⟩
```

**lemma** *has_gradient_line_slope_tendsto:*
  **fixes** *f :: "'a::real_inner ⇒ real"*
  **assumes** *grad: "has_gradient f x g"*
  **shows** *"((λt::real. (f (x + t *$_R$ d) - f x) / t) ⟶ inner d g) (at_right 0)"*
⟨*proof*⟩

**lemma** *convex_has_gradient_supports:*
  **fixes** *f :: "'a::real_inner ⇒ real"*
  **assumes** *conv: "convex_on S f"*
    **and** *grad: "has_gradient f x g"*
    **and** *x: "x ∈ S"*
    **and** *y: "y ∈ S"*
  **shows** *"f x + inner g (y - x) ≤ f y"*
⟨*proof*⟩

## 7.7 Convex differentiable functions with a named gradient field

**definition** *convex_differentiable_on ::*
  *"'a::real_inner set ⇒ ('a ⇒ real) ⇒ ('a ⇒ 'a) ⇒ bool"*
**where**
  *"convex_differentiable_on S f G ⟷*
    *convex_on S f ∧ has_gradient_on f S G"*

**lemma** *convex_differentiable_onI:*
  **assumes** *"convex_on S f"*
    **and** *"has_gradient_on f S G"*
  **shows** *"convex_differentiable_on S f G"*
  ⟨*proof*⟩

**lemma** *convex_differentiable_onD_convex_on:*
  **assumes** *"convex_differentiable_on S f G"*
  **shows** *"convex_on S f"*
  ⟨*proof*⟩

**lemma** *convex_differentiable_onD_has_gradient_on:*
  **assumes** *"convex_differentiable_on S f G"*
  **shows** *"has_gradient_on f S G"*
  ⟨*proof*⟩

**lemma** *convex_differentiable_on_convex:*
  **assumes** *"convex_differentiable_on S f G"*
  **shows** *"convex S"*
  ⟨*proof*⟩

**lemma** *convex_differentiable_on_gradientD:*
  **assumes** *"convex_differentiable_on S f G"*
    **and** *"x ∈ S"*

```
  shows "has_gradient f x (G x)"
  ⟨proof⟩

lemma convex_differentiable_on_differentiable:
  assumes "convex_differentiable_on S f G"
    and "x ∈ S"
  shows "f differentiable (at x)"
  ⟨proof⟩

lemma has_gradient_on_subset:
  assumes "has_gradient_on f T G"
    and "S ⊆ T"
  shows "has_gradient_on f S G"
  ⟨proof⟩

lemma convex_differentiable_on_subset:
  assumes "convex_differentiable_on T f G"
    and "S ⊆ T"
    and "convex S"
  shows "convex_differentiable_on S f G"
⟨proof⟩

lemma convex_differentiable_on_imp_gradient_lower_bound_on:
  assumes cd: "convex_differentiable_on S f G"
  shows "gradient_lower_bound_on S f G"
⟨proof⟩

lemma convex_differentiable_on_imp_supports_on:
  assumes cd: "convex_differentiable_on S f G"
  shows "supports_on S f G"
  ⟨proof⟩

lemma convex_differentiable_on_supports_at:
  assumes cd: "convex_differentiable_on S f G"
    and x: "x ∈ S"
  shows "supports_at S f x (G x)"
  ⟨proof⟩

lemma convex_differentiable_on_foc_imp_global_min_on:
  assumes cd: "convex_differentiable_on S f G"
    and foc: "first_order_condition_at S (G x) x"
  shows "global_min_on S f x"
  ⟨proof⟩
```

## 7.8 Main first-order consequences

```
theorem convex_differentiable_on_supporting_hyperplanes:
  assumes cd: "convex_differentiable_on S f G"
  shows "supports_on S f G"
```

⟨proof⟩

```
theorem convex_differentiable_on_first_order_sufficient:
  assumes cd: "convex_differentiable_on S f G"
    and foc: "first_order_condition_at S (G x) x"
  shows "global_min_on S f x"
  ⟨proof⟩
```

## 7.9 Locale form

```
locale convex_differentiable =
  fixes S :: "'a::real_inner set"
    and f :: "'a ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes convex_f: "convex_on S f"
    and gradient_f: "has_gradient_on f S G"
begin

lemma convex_set:
  "convex S"
  ⟨proof⟩

lemma has_gradient:
  assumes "x ∈ S"
  shows "has_gradient f x (G x)"
  ⟨proof⟩

lemma differentiable:
  assumes "x ∈ S"
  shows "f differentiable (at x)"
  ⟨proof⟩

lemma segment_mem:
  assumes "x ∈ S"
    and "y ∈ S"
    and "0 ≤ t"
    and "t ≤ 1"
  shows "x + scaleR t (y - x) ∈ S"
  ⟨proof⟩

lemma convex_bound_on_segment:
  assumes "x ∈ S"
    and "y ∈ S"
    and "0 ≤ t"
    and "t ≤ 1"
  shows "f (x + scaleR t (y - x)) ≤ (1 - t) * f x + t * f y"
  ⟨proof⟩

lemma gradient_lower_bound:
```

"gradient_lower_bound_on S f G"
⟨proof⟩

**lemma** *supports:*
  **assumes** "x ∈ S"
  **shows** "supports_at S f x (G x)"
  ⟨proof⟩

**lemma** *foc_imp_global_min:*
  **assumes** "first_order_condition_at S (G x) x"
  **shows** "global_min_on S f x"
  ⟨proof⟩

**end**

This completes the first-order convex certificate layer.

The main bridge theorem is ⟦convex_on ?S ?f; has_gradient ?f ?x ?g; ?x ∈ ?S; ?y ∈ ?S⟧ ⟹ ?f ?x + ?g · (?y - ?x) ≤ ?f ?y: for a convex function that has a gradient at x, the gradient defines a supporting affine lower bound on the feasible set. The bundled consequence ⟦convex_differentiable_on ?S ?f ?G; first_order_condition_at ?S (?G ?x) ?x⟧ ⟹ global_min_on ?S ?f ?x gives the usual first-order sufficient condition for global optimality in convex optimization.

Later theories use these results as the first-order part of the analysis of smooth gradient descent and projected gradient descent.

**end**
**theory** *Smooth_Convex*
  **imports** *Convex_Differentiable*
**begin**

# 8 Smooth convex functions and quadratic upper bounds

This theory introduces the smoothness layer used later for gradient descent and projected gradient descent.

The central interface is the quadratic upper-bound property f y <= f x + inner (G x) (y - x) + (L / 2) * norm (y - x) ^ 2. This is the standard descent-lemma form of L-smoothness. For the purposes of the algorithmic development, it is useful to package this property directly as a reusable assumption. A later file can derive it from more primitive Lipschitz-gradient assumptions if needed.

## 8.1 Lipschitz gradient fields

**definition** *lipschitz_gradient_on ::*
  "real ⇒ 'a::real_normed_vector set ⇒ ('a ⇒ 'a) ⇒ bool"

```
where
  "lipschitz_gradient_on L S G ⟷
     0 ≤ L ∧ (∀x∈S. ∀y∈S. norm (G x - G y) ≤ L * norm (x - y))"

lemma lipschitz_gradient_onI:
  assumes "0 ≤ L"
    and "⋀x y. x ∈ S ⟹ y ∈ S ⟹
      norm (G x - G y) ≤ L * norm (x - y)"
  shows "lipschitz_gradient_on L S G"
  ⟨proof⟩

lemma lipschitz_gradient_onD_nonneg:
  assumes "lipschitz_gradient_on L S G"
  shows "0 ≤ L"
  ⟨proof⟩

lemma lipschitz_gradient_onD:
  assumes "lipschitz_gradient_on L S G"
    and "x ∈ S"
    and "y ∈ S"
  shows "norm (G x - G y) ≤ L * norm (x - y)"
  ⟨proof⟩

lemma lipschitz_gradient_on_subset:
  assumes lip: "lipschitz_gradient_on L T G"
    and subset: "S ⊆ T"
  shows "lipschitz_gradient_on L S G"
⟨proof⟩

lemma lipschitz_gradient_on_mono_L:
  assumes lip: "lipschitz_gradient_on L S G"
    and LM: "L ≤ M"
  shows "lipschitz_gradient_on M S G"
⟨proof⟩
```

## 8.2 Quadratic upper bounds

```
definition smooth_upper_bound_on ::
  "real ⇒ 'a::real_inner set ⇒ ('a ⇒ real) ⇒ ('a ⇒ 'a) ⇒ bool"
where
  "smooth_upper_bound_on L S f G ⟷
     0 ≤ L ∧
     (∀x∈S. ∀y∈S.
        f y ≤ f x + inner (G x) (y - x) + (L / 2) * norm (y - x) ^ 2)"

lemma smooth_upper_bound_onI:
  assumes "0 ≤ L"
    and "⋀x y. x ∈ S ⟹ y ∈ S ⟹
      f y ≤ f x + inner (G x) (y - x) + (L / 2) * norm (y - x) ^ 2"
```

```
  shows "smooth_upper_bound_on L S f G"
  ⟨proof⟩

lemma smooth_upper_bound_onD_nonneg:
  assumes "smooth_upper_bound_on L S f G"
  shows "0 ≤ L"
  ⟨proof⟩

lemma smooth_upper_bound_onD:
  assumes "smooth_upper_bound_on L S f G"
    and "x ∈ S"
    and "y ∈ S"
  shows "f y ≤ f x + inner (G x) (y - x) + (L / 2) * norm (y - x) ^ 2"
  ⟨proof⟩

lemma smooth_upper_bound_on_subset:
  assumes smooth: "smooth_upper_bound_on L T f G"
    and subset: "S ⊆ T"
  shows "smooth_upper_bound_on L S f G"
⟨proof⟩

lemma smooth_upper_bound_on_mono_L:
  assumes smooth: "smooth_upper_bound_on L S f G"
    and LM: "L ≤ M"
  shows "smooth_upper_bound_on M S f G"
⟨proof⟩
```

## 8.3 Smooth convex functions

```
definition smooth_convex_on ::
  "real ⇒ 'a::real_inner set ⇒ ('a ⇒ real) ⇒ ('a ⇒ 'a) ⇒ bool"
where
  "smooth_convex_on L S f G ⟷
     convex_differentiable_on S f G ∧ smooth_upper_bound_on L S f G"

lemma smooth_convex_onI:
  assumes "convex_differentiable_on S f G"
    and "smooth_upper_bound_on L S f G"
  shows "smooth_convex_on L S f G"
  ⟨proof⟩

lemma smooth_convex_onD_convex_differentiable:
  assumes "smooth_convex_on L S f G"
  shows "convex_differentiable_on S f G"
  ⟨proof⟩

lemma smooth_convex_onD_smooth_upper_bound:
  assumes "smooth_convex_on L S f G"
  shows "smooth_upper_bound_on L S f G"
```

⟨proof⟩

**lemma** *smooth_convex_onD_convex_on:*
  **assumes** *"smooth_convex_on L S f G"*
  **shows** *"convex_on S f"*
  ⟨proof⟩

**lemma** *smooth_convex_onD_has_gradient_on:*
  **assumes** *"smooth_convex_on L S f G"*
  **shows** *"has_gradient_on f S G"*
  ⟨proof⟩

**lemma** *smooth_convex_onD_smooth_nonneg:*
  **assumes** *"smooth_convex_on L S f G"*
  **shows** *"0 ≤ L"*
  ⟨proof⟩

**lemma** *smooth_convex_on_subset:*
  **assumes** *smooth: "smooth_convex_on L T f G"*
    **and** *subset: "S ⊆ T"*
    **and** *convex: "convex S"*
  **shows** *"smooth_convex_on L S f G"*
⟨proof⟩

## 8.4 Gradient steps

**definition** *gradient_step ::*
  *"real ⇒ ('a::real_vector ⇒ 'a) ⇒ 'a ⇒ 'a"*
**where**
  *"gradient_step alpha G x = x - scaleR alpha (G x)"*

**lemma** *gradient_step_zero_stepsize [simp]:*
  *"gradient_step 0 G x = x"*
  ⟨proof⟩

**lemma** *gradient_step_zero_gradient [simp]:*
  **assumes** *"G x = 0"*
  **shows** *"gradient_step alpha G x = x"*
  ⟨proof⟩

**lemma** *gradient_step_diff:*
  *"gradient_step alpha G x - x = - scaleR alpha (G x)"*
  ⟨proof⟩

**lemma** *gradient_step_inner:*
  **fixes** *G :: "'a::real_inner ⇒ 'a"*
  **shows** *"inner (G x) (gradient_step alpha G x - x) =*
    *- alpha * norm (G x) ^ 2"*
  ⟨proof⟩

```
lemma gradient_step_norm_sq:
  fixes G :: "'a::real_inner ⇒ 'a"
  shows "norm (gradient_step alpha G x - x) ^ 2 =
    alpha ^ 2 * norm (G x) ^ 2"
⟨proof⟩
```

## 8.5 One-step descent from the smooth upper bound

```
lemma smooth_upper_bound_gradient_step:
  assumes smooth: "smooth_upper_bound_on L S f G"
    and x: "x ∈ S"
    and step: "gradient_step alpha G x ∈ S"
  shows
    "f (gradient_step alpha G x)
      ≤ f x - alpha * norm (G x) ^ 2
          + (L / 2) * alpha ^ 2 * norm (G x) ^ 2"
⟨proof⟩

lemma descent_coefficient_bound:
  fixes alpha L :: real
  assumes alpha_nonneg: "0 ≤ alpha"
    and step_size: "alpha * L ≤ 1"
  shows "- alpha + (L / 2) * alpha ^ 2 ≤ - alpha / 2"
⟨proof⟩

lemma smooth_upper_bound_gradient_step_decrease:
  assumes smooth: "smooth_upper_bound_on L S f G"
    and x: "x ∈ S"
    and step: "gradient_step alpha G x ∈ S"
    and alpha_nonneg: "0 ≤ alpha"
    and step_size: "alpha * L ≤ 1"
  shows
    "f (gradient_step alpha G x)
      ≤ f x - (alpha / 2) * norm (G x) ^ 2"
⟨proof⟩

lemma smooth_upper_bound_gradient_step_decrease_inverse_L:
  assumes smooth: "smooth_upper_bound_on L S f G"
    and Lpos: "0 < L"
    and x: "x ∈ S"
    and step: "gradient_step (inverse L) G x ∈ S"
  shows
    "f (gradient_step (inverse L) G x)
      ≤ f x - (1 / (2 * L)) * norm (G x) ^ 2"
⟨proof⟩
```

## 8.6 Locale form

```
locale smooth_convex =
```

```
  convex_differentiable S f G
  for S :: "'a::real_inner set"
    and f :: "'a ⇒ real"
    and G :: "'a ⇒ 'a" +
  fixes L :: real
  assumes smooth_bound: "smooth_upper_bound_on L S f G"
begin

lemma smooth_nonneg:
  "0 ≤ L"
  ⟨proof⟩

lemma smooth_upper_bound:
  assumes "x ∈ S"
    and "y ∈ S"
  shows "f y ≤ f x + inner (G x) (y - x) + (L / 2) * norm (y - x) ^ 2"
  ⟨proof⟩

lemma gradient_step_bound:
  assumes "x ∈ S"
    and "gradient_step alpha G x ∈ S"
  shows
    "f (gradient_step alpha G x)
      ≤ f x - alpha * norm (G x) ^ 2
          + (L / 2) * alpha ^ 2 * norm (G x) ^ 2"
  ⟨proof⟩

lemma gradient_step_decrease:
  assumes "x ∈ S"
    and "gradient_step alpha G x ∈ S"
    and "0 ≤ alpha"
    and "alpha * L ≤ 1"
  shows
    "f (gradient_step alpha G x)
      ≤ f x - (alpha / 2) * norm (G x) ^ 2"
  ⟨proof⟩

lemma gradient_step_decrease_inverse_L:
  assumes "0 < L"
    and "x ∈ S"
    and "gradient_step (inverse L) G x ∈ S"
  shows
    "f (gradient_step (inverse L) G x)
      ≤ f x - (1 / (2 * L)) * norm (G x) ^ 2"
  ⟨proof⟩

end
```

This file packages the smooth quadratic upper-bound interface and derives the basic one-step decrease estimate for gradient steps.

The next theory can use these lemmas to prove descent and convergence properties for gradient descent sequences. For constrained problems, the same upper-bound interface will combine with projection variational inequalities.

```
end
theory Abstract_Descent
  imports "HOL-Analysis.Analysis"
begin
```

# 9 Abstract descent and telescoping lemmas

This theory contains abstract sequence and finite-sum lemmas used in descent analyses. These results are independent of gradients, projections, and particular optimization algorithms.

## 9.1 Finite-sum telescoping

If each local progress term is bounded by the decrease of a potential, then the sum of all progress terms is bounded by the total decrease of the potential.

```
lemma sum_progress_le_initial_gap:
  fixes a c :: "nat ⇒ real"
  assumes step: "⋀n. n < N ⟹ c n ≤ a n - a (Suc n)"
  shows "sum c {..<N} ≤ a 0 - a N"
  ⟨proof⟩
```

A version in which the terminal value of the potential is replaced by a lower bound.

```
lemma sum_progress_le_initial_minus_lower_bound:
  fixes a c :: "nat ⇒ real"
  assumes step: "⋀n. n < N ⟹ c n ≤ a n - a (Suc n)"
  assumes lower: "B ≤ a N"
  shows "sum c {..<N} ≤ a 0 - B"
⟨proof⟩
```

## 9.2 Average bounds

If a finite sum is bounded above, then at least one term is bounded by the corresponding average.

```
lemma exists_le_average_of_sum_bound:
  fixes a :: "nat ⇒ real"
  assumes N_pos: "N > 0"
  assumes nonneg: "⋀n. n < N ⟹ 0 ≤ a n"
  assumes sum_bound: "sum a {..<N} ≤ B"
  shows "∃n<N. a n ≤ B / real N"
⟨proof⟩
```

```
lemma exists_le_average_of_nonnegative_sum:
  fixes a :: "nat ⇒ real"
  assumes N_pos: "N > 0"
  assumes nonneg: "⋀n. n < N ⟹ 0 ≤ a n"
  shows "∃n<N. a n ≤ sum a {..<N} / real N"
⟨proof⟩
```

### 9.3 Linear recurrences

A one-step linear recurrence can be iterated to obtain a geometric bound.

```
lemma sequence_linear_rate_from_step:
  fixes a :: "nat ⇒ real"
  assumes q_nonneg: "0 ≤ q"
  assumes step: "⋀n. a (Suc n) ≤ q * a n"
  shows "a n ≤ q ^ n * a 0"
⟨proof⟩

end
theory Gradient_Descent
  imports Smooth_Convex
begin
```

## 10 Gradient descent sequences

This theory lifts the one-step descent estimates from `Smooth_Convex` to gradient descent sequences.

The purpose of this file is deliberately modest: it packages the recurrence

x (Suc n) = `gradient_step` alpha G (x n)

and proves the basic monotonicity consequences that follow from the smooth upper-bound interface.

### 10.1 Gradient descent recurrence

```
definition gradient_descent_iterates ::
  "real ⇒ ('a::real_vector ⇒ 'a) ⇒ (nat ⇒ 'a) ⇒ bool"
where
  "gradient_descent_iterates alpha G x ⟷
     (∀n. x (Suc n) = gradient_step alpha G (x n))"

lemma gradient_descent_iteratesI:
  assumes "⋀n. x (Suc n) = gradient_step alpha G (x n)"
  shows "gradient_descent_iterates alpha G x"
  ⟨proof⟩

lemma gradient_descent_iteratesD:
  assumes "gradient_descent_iterates alpha G x"
  shows "x (Suc n) = gradient_step alpha G (x n)"
```

⟨*proof*⟩

**lemma** *gradient_descent_iteratesE:*
**assumes** *"gradient_descent_iterates alpha G x"*
**obtains** *"x (Suc n) = gradient_step alpha G (x n)"*
⟨*proof*⟩

## 10.2 Feasible iterates

**definition** *feasible_iterates ::*
*"'a set ⇒ (nat ⇒ 'a) ⇒ bool"*
**where**
*"feasible_iterates S x ⟷ (∀n. x n ∈ S)"*

**lemma** *feasible_iteratesI:*
**assumes** *"⋀n. x n ∈ S"*
**shows** *"feasible_iterates S x"*
⟨*proof*⟩

**lemma** *feasible_iteratesD:*
**assumes** *"feasible_iterates S x"*
**shows** *"x n ∈ S"*
⟨*proof*⟩

**lemma** *feasible_iterates_subset:*
**assumes** *"feasible_iterates S x"*
**and** *"S ⊆ T"*
**shows** *"feasible_iterates T x"*
⟨*proof*⟩

**lemma** *gradient_descent_step_mem:*
**assumes** *gd: "gradient_descent_iterates alpha G x"*
**and** *feasible: "feasible_iterates S x"*
**shows** *"gradient_step alpha G (x n) ∈ S"*
⟨*proof*⟩

## 10.3 Objective value sequences

**definition** *objective_values ::*
*"('a ⇒ real) ⇒ (nat ⇒ 'a) ⇒ nat ⇒ real"*
**where**
*"objective_values f x n = f (x n)"*

**lemma** *objective_values_simp [simp]:*
*"objective_values f x n = f (x n)"*
⟨*proof*⟩

**definition** *nonincreasing_sequence ::*
*"(nat ⇒ real) ⇒ bool"*
**where**

```
  "nonincreasing_sequence a ⟷ (∀n. a (Suc n) ≤ a n)"

lemma nonincreasing_sequenceI:
  assumes "⋀n. a (Suc n) ≤ a n"
  shows "nonincreasing_sequence a"
  ⟨proof⟩

lemma nonincreasing_sequenceD:
  assumes "nonincreasing_sequence a"
  shows "a (Suc n) ≤ a n"
  ⟨proof⟩
```

## 10.4 One-step estimates along gradient descent iterates

```
lemma gradient_descent_one_step_bound:
  assumes smooth: "smooth_upper_bound_on L S f G"
    and gd: "gradient_descent_iterates alpha G x"
    and feasible: "feasible_iterates S x"
  shows
    "f (x (Suc n))
      ≤ f (x n) - alpha * norm (G (x n)) ^ 2
          + (L / 2) * alpha ^ 2 * norm (G (x n)) ^ 2"
⟨proof⟩

lemma gradient_descent_one_step_decrease:
  assumes smooth: "smooth_upper_bound_on L S f G"
    and gd: "gradient_descent_iterates alpha G x"
    and feasible: "feasible_iterates S x"
    and alpha_nonneg: "0 ≤ alpha"
    and step_size: "alpha * L ≤ 1"
  shows
    "f (x (Suc n))
      ≤ f (x n) - (alpha / 2) * norm (G (x n)) ^ 2"
⟨proof⟩

lemma gradient_descent_objective_mono_step:
  assumes smooth: "smooth_upper_bound_on L S f G"
    and gd: "gradient_descent_iterates alpha G x"
    and feasible: "feasible_iterates S x"
    and alpha_nonneg: "0 ≤ alpha"
    and step_size: "alpha * L ≤ 1"
  shows "f (x (Suc n)) ≤ f (x n)"
⟨proof⟩

lemma gradient_descent_objective_nonincreasing:
  assumes smooth: "smooth_upper_bound_on L S f G"
    and gd: "gradient_descent_iterates alpha G x"
    and feasible: "feasible_iterates S x"
    and alpha_nonneg: "0 ≤ alpha"
```

```
    and step_size: "alpha * L ≤ 1"
  shows "nonincreasing_sequence (objective_values f x)"
⟨proof⟩

lemma gradient_descent_step_progress:
  assumes smooth: "smooth_upper_bound_on L S f G"
    and gd: "gradient_descent_iterates alpha G x"
    and feasible: "feasible_iterates S x"
    and alpha_nonneg: "0 ≤ alpha"
    and step_size: "alpha * L ≤ 1"
  shows
    "(alpha / 2) * norm (G (x n)) ^ 2
      ≤ f (x n) - f (x (Suc n))"
⟨proof⟩
```

## 10.5 Locale form

```
locale gradient_descent =
  smooth_convex S f G L
  for S :: "'a::real_inner set"
    and f :: "'a ⇒ real"
    and G :: "'a ⇒ 'a"
    and L :: real +
  fixes alpha :: real
    and x :: "nat ⇒ 'a"
  assumes iterates: "gradient_descent_iterates alpha G x"
    and feasible: "feasible_iterates S x"
    and alpha_nonneg: "0 ≤ alpha"
    and step_size: "alpha * L ≤ 1"
begin

lemma iterate:
  "x (Suc n) = gradient_step alpha G (x n)"
  ⟨proof⟩

lemma iterate_mem:
  "x n ∈ S"
  ⟨proof⟩

lemma step_mem:
  "gradient_step alpha G (x n) ∈ S"
  ⟨proof⟩

lemma one_step_bound:
  "f (x (Suc n))
    ≤ f (x n) - alpha * norm (G (x n)) ^ 2
        + (L / 2) * alpha ^ 2 * norm (G (x n)) ^ 2"
  ⟨proof⟩
```

**lemma** *one_step_decrease:*
```
"f (x (Suc n))
  ≤ f (x n) - (alpha / 2) * norm (G (x n)) ^ 2"
⟨proof⟩
```

**lemma** *objective_mono_step:*
```
"f (x (Suc n)) ≤ f (x n)"
⟨proof⟩
```

**lemma** *objective_nonincreasing:*
```
"nonincreasing_sequence (objective_values f x)"
⟨proof⟩
```

**lemma** *step_progress:*
```
"(alpha / 2) * norm (G (x n)) ^ 2
  ≤ f (x n) - f (x (Suc n))"
⟨proof⟩
```

**end**

This file turns the pointwise gradient-step descent estimates into sequence statements for gradient descent.

The main consequence is that, under the smooth upper-bound assumption and the standard step-size condition that alpha * L is at most one, the objective values along a feasible gradient descent sequence form a nonincreasing sequence.

**end**
**theory** *Gradient_Descent_Rates*
  **imports**
    *Gradient_Descent*
    *Abstract_Descent*
**begin**

# 11 Rate infrastructure for gradient descent

This theory collects the first rate-style consequences of the gradient descent development.

The purpose of this file is intentionally modest. It does not yet prove the full O(1/N) function-value convergence theorem for smooth convex minimization. Instead, it packages the telescoping arguments that follow directly from the one-step progress estimate already proved in *Gradient_Descent*.

These lemmas are meant to be reusable later for projected gradient descent and other descent methods.

## 11.1 Finite-sum bounds for gradient descent

The main direct consequence of the one-step progress inequality is that the weighted sum of squared gradient norms is bounded by the total decrease in objective value.

```
lemma gradient_descent_sum_weighted_gradient_norm_sq_bound:
  fixes f :: "'a::real_inner ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_upper_bound_on L S f G"
  assumes gd: "gradient_descent_iterates alpha G x"
  assumes feasible: "feasible_iterates S x"
  assumes alpha_nonneg: "0 ≤ alpha"
  assumes step_size: "alpha * L ≤ 1"
  shows
    "sum (λn. (alpha / 2) * norm (G (x n)) ^ 2) {..<N}
      ≤ f (x 0) - f (x N)"
⟨proof⟩
```

A lower-bound version of the same result.

```
lemma gradient_descent_sum_weighted_gradient_norm_sq_bound_below:
  fixes f :: "'a::real_inner ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_upper_bound_on L S f G"
  assumes gd: "gradient_descent_iterates alpha G x"
  assumes feasible: "feasible_iterates S x"
  assumes alpha_nonneg: "0 ≤ alpha"
  assumes step_size: "alpha * L ≤ 1"
  assumes lower: "B ≤ f (x N)"
  shows
    "sum (λn. (alpha / 2) * norm (G (x n)) ^ 2) {..<N}
      ≤ f (x 0) - B"
⟨proof⟩
```

If the step size is strictly positive, the coefficient alpha / 2 can be divided out. This gives the standard finite-sum squared-gradient bound.

```
lemma gradient_descent_sum_gradient_norm_sq_bound:
  fixes f :: "'a::real_inner ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_upper_bound_on L S f G"
  assumes gd: "gradient_descent_iterates alpha G x"
  assumes feasible: "feasible_iterates S x"
  assumes alpha_pos: "0 < alpha"
  assumes step_size: "alpha * L ≤ 1"
  shows
    "sum (λn. norm (G (x n)) ^ 2) {..<N}
      ≤ (2 / alpha) * (f (x 0) - f (x N))"
⟨proof⟩
```

```
lemma gradient_descent_sum_gradient_norm_sq_bound_below:
```

```
  fixes f :: "'a::real_inner ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_upper_bound_on L S f G"
  assumes gd: "gradient_descent_iterates alpha G x"
  assumes feasible: "feasible_iterates S x"
  assumes alpha_pos: "0 < alpha"
  assumes step_size: "alpha * L ≤ 1"
  assumes lower: "B ≤ f (x N)"
  shows
    "sum (λn. norm (G (x n)) ^ 2) {..<N}
      ≤ (2 / alpha) * (f (x 0) - B)"
⟨proof⟩
```

## 11.2 Average and small-gradient consequences

The finite-sum bound immediately implies an average squared-gradient bound.

```
lemma gradient_descent_average_gradient_norm_sq_bound:
  fixes f :: "'a::real_inner ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_upper_bound_on L S f G"
  assumes gd: "gradient_descent_iterates alpha G x"
  assumes feasible: "feasible_iterates S x"
  assumes alpha_pos: "0 < alpha"
  assumes step_size: "alpha * L ≤ 1"
  assumes N_pos: "N > 0"
  shows
    "(1 / real N) * sum (λn. norm (G (x n)) ^ 2) {..<N}
      ≤ (2 / (alpha * real N)) * (f (x 0) - f (x N))"
⟨proof⟩
```

```
lemma gradient_descent_average_gradient_norm_sq_bound_below:
  fixes f :: "'a::real_inner ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_upper_bound_on L S f G"
  assumes gd: "gradient_descent_iterates alpha G x"
  assumes feasible: "feasible_iterates S x"
  assumes alpha_pos: "0 < alpha"
  assumes step_size: "alpha * L ≤ 1"
  assumes lower: "B ≤ f (x N)"
  assumes N_pos: "N > 0"
  shows
    "(1 / real N) * sum (λn. norm (G (x n)) ^ 2) {..<N}
      ≤ (2 / (alpha * real N)) * (f (x 0) - B)"
⟨proof⟩
```

The next theorem states the usual small-gradient consequence: among the first N iterates, at least one iterate has squared gradient norm bounded by the average finite-sum bound.

```
lemma gradient_descent_exists_small_gradient_norm_sq:
```

```
  fixes f :: "'a::real_inner ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_upper_bound_on L S f G"
  assumes gd: "gradient_descent_iterates alpha G x"
  assumes feasible: "feasible_iterates S x"
  assumes alpha_pos: "0 < alpha"
  assumes step_size: "alpha * L ≤ 1"
  assumes N_pos: "N > 0"
  shows
    "∃n<N. norm (G (x n)) ^ 2
      ≤ (2 / (alpha * real N)) * (f (x 0) - f (x N))"
⟨proof⟩
```

```
lemma gradient_descent_exists_small_gradient_norm_sq_below:
  fixes f :: "'a::real_inner ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_upper_bound_on L S f G"
  assumes gd: "gradient_descent_iterates alpha G x"
  assumes feasible: "feasible_iterates S x"
  assumes alpha_pos: "0 < alpha"
  assumes step_size: "alpha * L ≤ 1"
  assumes lower: "B ≤ f (x N)"
  assumes N_pos: "N > 0"
  shows
    "∃n<N. norm (G (x n)) ^ 2
      ≤ (2 / (alpha * real N)) * (f (x 0) - B)"
⟨proof⟩
```

## 11.3 Weighted small-gradient consequences

The weighted version does not require dividing by alpha. It is therefore available even for the degenerate case alpha = 0, although the resulting statement is mainly useful when alpha is positive.

```
lemma gradient_descent_exists_small_weighted_gradient_norm_sq:
  fixes f :: "'a::real_inner ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_upper_bound_on L S f G"
  assumes gd: "gradient_descent_iterates alpha G x"
  assumes feasible: "feasible_iterates S x"
  assumes alpha_nonneg: "0 ≤ alpha"
  assumes step_size: "alpha * L ≤ 1"
  assumes N_pos: "N > 0"
  shows
    "∃n<N. (alpha / 2) * norm (G (x n)) ^ 2
      ≤ (f (x 0) - f (x N)) / real N"
⟨proof⟩
```

## 11.4 Locale form

```
context gradient_descent
begin

lemma sum_weighted_gradient_norm_sq_bound:
  "sum (λn. (alpha / 2) * norm (G (x n)) ^ 2) {..<N}
    ≤ f (x 0) - f (x N)"
  ⟨proof⟩

lemma sum_weighted_gradient_norm_sq_bound_below:
  assumes lower: "B ≤ f (x N)"
  shows
    "sum (λn. (alpha / 2) * norm (G (x n)) ^ 2) {..<N}
      ≤ f (x 0) - B"
  ⟨proof⟩

lemma exists_small_weighted_gradient_norm_sq:
  assumes N_pos: "N > 0"
  shows
    "∃n<N. (alpha / 2) * norm (G (x n)) ^ 2
      ≤ (f (x 0) - f (x N)) / real N"
  ⟨proof⟩

lemma sum_gradient_norm_sq_bound:
  assumes alpha_pos: "0 < alpha"
  shows
    "sum (λn. norm (G (x n)) ^ 2) {..<N}
      ≤ (2 / alpha) * (f (x 0) - f (x N))"
  ⟨proof⟩

lemma sum_gradient_norm_sq_bound_below:
  assumes alpha_pos: "0 < alpha"
  assumes lower: "B ≤ f (x N)"
  shows
    "sum (λn. norm (G (x n)) ^ 2) {..<N}
      ≤ (2 / alpha) * (f (x 0) - B)"
  ⟨proof⟩

lemma average_gradient_norm_sq_bound:
  assumes alpha_pos: "0 < alpha"
  assumes N_pos: "N > 0"
  shows
    "(1 / real N) * sum (λn. norm (G (x n)) ^ 2) {..<N}
      ≤ (2 / (alpha * real N)) * (f (x 0) - f (x N))"
  ⟨proof⟩

lemma average_gradient_norm_sq_bound_below:
  assumes alpha_pos: "0 < alpha"
  assumes lower: "B ≤ f (x N)"
```

```
  assumes N_pos: "N > 0"
  shows
    "(1 / real N) * sum (λn. norm (G (x n)) ^ 2) {..<N}
      ≤ (2 / (alpha * real N)) * (f (x 0) - B)"
  ⟨proof⟩

lemma exists_small_gradient_norm_sq:
  assumes alpha_pos: "0 < alpha"
  assumes N_pos: "N > 0"
  shows
    "∃n<N. norm (G (x n)) ^ 2
      ≤ (2 / (alpha * real N)) * (f (x 0) - f (x N))"
  ⟨proof⟩

lemma exists_small_gradient_norm_sq_below:
  assumes alpha_pos: "0 < alpha"
  assumes lower: "B ≤ f (x N)"
  assumes N_pos: "N > 0"
  shows
    "∃n<N. norm (G (x n)) ^ 2
      ≤ (2 / (alpha * real N)) * (f (x 0) - B)"
  ⟨proof⟩

end
```

This file gives the first rate-style consequences of the descent estimate for gradient descent. The next natural step is to prove a stronger distance-to-solution one-step inequality under convexity, which can then be telescoped to obtain the standard O(1/N) function-value convergence rate.

```
end
theory Gradient_Descent_Convergence
  imports Gradient_Descent_Rates
begin
```

# 12 Function-value convergence for gradient descent

This theory proves the standard O(1/N) function-value convergence bound for fixed-step gradient descent on a smooth convex objective.

The proof is organized around the usual distance-to-comparator estimate: each gradient step decreases the squared distance potential enough to pay for the next function-value gap. The estimate is then telescoped.

## 12.1 Elementary monotonicity lemmas

```
lemma nonincreasing_sequence_mono:
  fixes a :: "nat ⇒ real"
  assumes mono: "nonincreasing_sequence a"
```

**and** *mn: "m ≤ n"*
**shows** *"a n ≤ a m"*
⟨*proof*⟩

**lemma** *nonincreasing_sequence_shifted_sum_lower_bound:*
**fixes** *a :: "nat ⇒ real"*
**assumes** *mono: "nonincreasing_sequence a"*
**and** *N_pos: "N > 0"*
**shows** *"real N * (a N - b) ≤ (∑n<N. a (Suc n) - b)"*
⟨*proof*⟩

## 12.2 Distance identity for one gradient step

**lemma** *gradient_step_distance_decrease:*
**fixes** *x xstar :: "'a::real_inner"*
**shows**
*"norm (x - xstar) ^ 2 - norm (gradient_step alpha G x - xstar) ^ 2*
*=*
*2 * alpha * inner (G x) (x - xstar) - alpha ^ 2 * norm (G x) ^ 2"*
⟨*proof*⟩

**lemma** *gradient_step_distance_decrease_divided:*
**fixes** *x xstar :: "'a::real_inner"*
**assumes** *alpha_pos: "0 < alpha"*
**shows**
*"(norm (x - xstar) ^ 2 - norm (gradient_step alpha G x - xstar) ^*
*2)*
*/ (2 * alpha)*
*=*
*inner (G x) (x - xstar) - (alpha / 2) * norm (G x) ^ 2"*
⟨*proof*⟩

## 12.3 One-step function gap bound

**lemma** *gradient_descent_one_step_distance_bound_to_point:*
**fixes** *f :: "'a::real_inner ⇒ real"*
**and** *G :: "'a ⇒ 'a"*
**assumes** *smooth: "smooth_convex_on L S f G"*
**and** *gd: "gradient_descent_iterates alpha G x"*
**and** *feasible: "feasible_iterates S x"*
**and** *alpha_pos: "0 < alpha"*
**and** *step_size: "alpha * L ≤ 1"*
**and** *u_mem: "u ∈ S"*
**shows**
*"f (x (Suc n)) - f u*
*≤*
*(norm (x n - u) ^ 2 - norm (x (Suc n) - u) ^ 2) / (2 * alpha)"*
⟨*proof*⟩

**lemma** *gradient_descent_one_step_distance_bound_to_minimizer:*

**fixes** *f :: "'a::real_inner ⇒ real"*
**and** *G :: "'a ⇒ 'a"*
**assumes** *smooth: "smooth_convex_on L S f G"*
**and** *gd: "gradient_descent_iterates alpha G x"*
**and** *feasible: "feasible_iterates S x"*
**and** *alpha_pos: "0 < alpha"*
**and** *step_size: "alpha * L ≤ 1"*
**and** *minimizer: "global_min_on S f xstar"*
**shows**
*"f (x (Suc n)) - f xstar*
*≤*
*(norm (x n - xstar) ^ 2 - norm (x (Suc n) - xstar) ^ 2) / (2 * alpha)"*
⟨*proof*⟩

## 12.4 Telescoped function gap bounds

**lemma** *gradient_descent_sum_function_value_gaps_bound_to_point:*
**fixes** *f :: "'a::real_inner ⇒ real"*
**and** *G :: "'a ⇒ 'a"*
**assumes** *smooth: "smooth_convex_on L S f G"*
**and** *gd: "gradient_descent_iterates alpha G x"*
**and** *feasible: "feasible_iterates S x"*
**and** *alpha_pos: "0 < alpha"*
**and** *step_size: "alpha * L ≤ 1"*
**and** *u_mem: "u ∈ S"*
**shows**
*"(∑n<N. f (x (Suc n)) - f u)*
*≤*
*norm (x 0 - u) ^ 2 / (2 * alpha)*
*- norm (x N - u) ^ 2 / (2 * alpha)"*
⟨*proof*⟩

**lemma** *gradient_descent_sum_function_value_gaps_bound_to_minimizer:*
**fixes** *f :: "'a::real_inner ⇒ real"*
**and** *G :: "'a ⇒ 'a"*
**assumes** *smooth: "smooth_convex_on L S f G"*
**and** *gd: "gradient_descent_iterates alpha G x"*
**and** *feasible: "feasible_iterates S x"*
**and** *alpha_pos: "0 < alpha"*
**and** *step_size: "alpha * L ≤ 1"*
**and** *minimizer: "global_min_on S f xstar"*
**shows**
*"(∑n<N. f (x (Suc n)) - f xstar)*
*≤*
*norm (x 0 - xstar) ^ 2 / (2 * alpha)*
*- norm (x N - xstar) ^ 2 / (2 * alpha)"*
⟨*proof*⟩

**lemma** *gradient_descent_sum_function_value_gaps_bound_to_point_initial:*

```
  fixes f :: "'a::real_inner ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L S f G"
    and gd: "gradient_descent_iterates alpha G x"
    and feasible: "feasible_iterates S x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and u_mem: "u ∈ S"
  shows
    "(∑n<N. f (x (Suc n)) - f u)
      ≤ norm (x 0 - u) ^ 2 / (2 * alpha)"
⟨proof⟩
```

```
lemma gradient_descent_sum_function_value_gaps_bound_to_minimizer_initial:
  fixes f :: "'a::real_inner ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L S f G"
    and gd: "gradient_descent_iterates alpha G x"
    and feasible: "feasible_iterates S x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on S f xstar"
  shows
    "(∑n<N. f (x (Suc n)) - f xstar)
      ≤ norm (x 0 - xstar) ^ 2 / (2 * alpha)"
⟨proof⟩
```

## 12.5 The O(1/N) convergence rate

```
lemma gradient_descent_last_gap_times_N_bound:
  fixes f :: "'a::real_inner ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L S f G"
    and gd: "gradient_descent_iterates alpha G x"
    and feasible: "feasible_iterates S x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on S f xstar"
    and N_pos: "N > 0"
  shows
    "real N * (f (x N) - f xstar)
      ≤ norm (x 0 - xstar) ^ 2 / (2 * alpha)"
⟨proof⟩
```

```
theorem gradient_descent_function_value_gap_bound:
  fixes f :: "'a::real_inner ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L S f G"
    and gd: "gradient_descent_iterates alpha G x"
```

```
    and feasible: "feasible_iterates S x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on S f xstar"
    and N_pos: "N > 0"
  shows
    "f (x N) - f xstar
      ≤ norm (x 0 - xstar) ^ 2 / (2 * alpha * real N)"
⟨proof⟩
```

## 12.6 Locale form

```
context gradient_descent
begin

lemma smooth_convex_on_self:
  "smooth_convex_on L S f G"
⟨proof⟩

lemma one_step_distance_bound_to_point:
  assumes alpha_pos: "0 < alpha"
    and u_mem: "u ∈ S"
  shows
    "f (x (Suc n)) - f u
      ≤
     (norm (x n - u) ^ 2 - norm (x (Suc n) - u) ^ 2) / (2 * alpha)"
  ⟨proof⟩

lemma one_step_distance_bound_to_minimizer:
  assumes alpha_pos: "0 < alpha"
    and minimizer: "global_min_on S f xstar"
  shows
    "f (x (Suc n)) - f xstar
      ≤
     (norm (x n - xstar) ^ 2 - norm (x (Suc n) - xstar) ^ 2) / (2 * alpha)"
  ⟨proof⟩

lemma sum_function_value_gaps_bound_to_minimizer:
  assumes alpha_pos: "0 < alpha"
    and minimizer: "global_min_on S f xstar"
  shows
    "(∑n<N. f (x (Suc n)) - f xstar)
      ≤
     norm (x 0 - xstar) ^ 2 / (2 * alpha)
      - norm (x N - xstar) ^ 2 / (2 * alpha)"
  ⟨proof⟩

lemma sum_function_value_gaps_bound_to_minimizer_initial:
  assumes alpha_pos: "0 < alpha"
```

```
    and minimizer: "global_min_on S f xstar"
  shows
    "(∑n<N. f (x (Suc n)) - f xstar)
      ≤ norm (x 0 - xstar) ^ 2 / (2 * alpha)"
  ⟨proof⟩

lemma last_gap_times_N_bound:
  assumes alpha_pos: "0 < alpha"
    and minimizer: "global_min_on S f xstar"
    and N_pos: "N > 0"
  shows
    "real N * (f (x N) - f xstar)
      ≤ norm (x 0 - xstar) ^ 2 / (2 * alpha)"
  ⟨proof⟩

theorem function_value_gap_bound:
  assumes alpha_pos: "0 < alpha"
    and minimizer: "global_min_on S f xstar"
    and N_pos: "N > 0"
  shows
    "f (x N) - f xstar
      ≤ norm (x 0 - xstar) ^ 2 / (2 * alpha * real N)"
  ⟨proof⟩

end
```

The main theorem is ⟦*smooth_convex_on ?L ?S ?f ?G; gradient_descent_iterates ?alpha ?G ?x; feasible_iterates ?S ?x; 0 < ?alpha; ?alpha * ?L ≤ 1; global_min_on ?S ?f ?xstar; 0 < ?N*⟧ ⟹ *?f (?x ?N) - ?f ?xstar ≤ (norm (?x 0 - ?xstar))*$^2$ */ (2 * ?alpha * real ?N)*. It gives the classical fixed-step O(1/N) convergence estimate for smooth convex gradient descent, stated using the same gradient-descent recurrence and feasibility interface as the previous theories.

```
end
theory Projection_Geometry
  imports "HOL-Analysis.Analysis"
begin
```

# 13 Projection geometry

This theory contains the geometric and algebraic projection facts used by projected first-order methods. It is independent of any particular objective function or projected-gradient iteration.

## 13.1 Variational inequality for metric projections

```
lemma projection_variational_inequality:
  fixes C :: "'a::{real_inner,heine_borel} set"
```

```
  assumes convex: "convex C"
    and closed: "closed C"
    and u_mem: "u ∈ C"
  shows "inner (z - closest_point C z) (u - closest_point C z) ≤ 0"
  ⟨proof⟩
```

### 13.2 Three-point identity

```
lemma three_point_inner_identity:
  fixes x p u :: "'a::real_inner"
  shows "2 * inner (x - p) (p - u) =
    norm (x - u) ^ 2 - norm (p - u) ^ 2 - norm (p - x) ^ 2"
⟨proof⟩
```

### 13.3 Inner-product estimate from a projection inequality

The following lemma is the algebraic core behind the projected-gradient one-step estimate. It only uses a variational inequality and the three-point identity.

```
lemma projected_gradient_inner_bound_from_vi:
  fixes x p u g :: "'a::real_inner"
  assumes alpha_pos: "0 < alpha"
    and vi: "inner (x - scaleR alpha g - p) (u - p) ≤ 0"
  shows "inner g (p - u) ≤
    (norm (x - u) ^ 2 - norm (p - u) ^ 2 - norm (p - x) ^ 2) / (2 * alpha)"
⟨proof⟩

end
theory Projection_Optimization
  imports
    Gradient_Descent_Convergence
    Projection_Geometry
begin
```

## 14 Projection and projected gradient steps

This theory introduces the projection layer needed for projected gradient descent. The main results are the one-step distance inequality for a projected gradient step on a closed convex feasible set, and the associated one-step objective decrease in terms of the projected step length.

The projected-gradient mapping itself is introduced later, to avoid making this basic projection layer depend on the residual interface.

### 14.1 Projected gradient step

```
definition projected_gradient_step ::
  "'a::{real_inner,heine_borel} set ⇒ real ⇒ ('a ⇒ 'a) ⇒ 'a ⇒ 'a"
```

```
where
  "projected_gradient_step C alpha G x =
    closest_point C (gradient_step alpha G x)"

lemma projected_gradient_step_in_set:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes closed: "closed C"
    and nonempty: "C ≠ {}"
  shows "projected_gradient_step C alpha G x ∈ C"
  ⟨proof⟩

lemma projected_gradient_step_in_set_if_base_mem:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes closed: "closed C"
    and x_mem: "x ∈ C"
  shows "projected_gradient_step C alpha G y ∈ C"
⟨proof⟩

lemma projected_gradient_step_variational_inequality:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes convex: "convex C"
    and closed: "closed C"
    and u_mem: "u ∈ C"
  shows "inner
    (gradient_step alpha G x - projected_gradient_step C alpha G x)
    (u - projected_gradient_step C alpha G x) ≤ 0"
  ⟨proof⟩

lemma projected_gradient_inner_bound:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes convex: "convex C"
    and closed: "closed C"
    and u_mem: "u ∈ C"
    and alpha_pos: "0 < alpha"
  shows
    "inner (G x) (projected_gradient_step C alpha G x - u)
      ≤
     (norm (x - u) ^ 2
      - norm (projected_gradient_step C alpha G x - u) ^ 2
      - norm (projected_gradient_step C alpha G x - x) ^ 2)
      / (2 * alpha)"
⟨proof⟩
```

## 14.2 Projected gradient one-step bound

```
lemma projected_gradient_one_step_distance_bound_to_point:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
```

```
    and closed: "closed C"
    and convex: "convex C"
    and x_mem: "x ∈ C"
    and u_mem: "u ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
  defines "p ≡ projected_gradient_step C alpha G x"
  shows
    "f p - f u
      ≤
     (norm (x - u) ^ 2 - norm (p - u) ^ 2) / (2 * alpha)"
⟨proof⟩
```

## 14.3 Projected gradient step descent

Taking the comparison point u to be the current iterate x in the one-step distance inequality gives a descent estimate in terms of the projected step length. Later theories translate this step-length residual into the projected-gradient mapping norm.

```
lemma projected_gradient_step_progress_step_norm:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and x_mem: "x ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
  shows
    "(1 / (2 * alpha)) *
      norm (projected_gradient_step C alpha G x - x) ^ 2
      ≤ f x - f (projected_gradient_step C alpha G x)"
⟨proof⟩

lemma projected_gradient_step_descent_step_norm:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and x_mem: "x ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
  shows
    "f (projected_gradient_step C alpha G x)
      ≤ f x -
        (1 / (2 * alpha)) *
          norm (projected_gradient_step C alpha G x - x) ^ 2"
⟨proof⟩
```

```
lemma projected_gradient_step_descent_step_norm_add:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and x_mem: "x ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
  shows
    "f (projected_gradient_step C alpha G x)
      + (1 / (2 * alpha)) *
          norm (projected_gradient_step C alpha G x - x) ^ 2
      ≤ f x"
⟨proof⟩

lemma projected_gradient_one_step_distance_bound_to_minimizer:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and x_mem: "x ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
  defines "p ≡ projected_gradient_step C alpha G x"
  shows
    "f p - f xstar
      ≤
     (norm (x - xstar) ^ 2 - norm (p - xstar) ^ 2) / (2 * alpha)"
⟨proof⟩
```

## 14.4 Projected gradient descent iterates

```
definition projected_gradient_descent_iterates ::
  "'a::{real_inner,heine_borel} set ⇒ real ⇒ ('a ⇒ 'a) ⇒ (nat ⇒ 'a)
⇒ bool"
where
  "projected_gradient_descent_iterates C alpha G x ⟷
    (∀n. x (Suc n) = projected_gradient_step C alpha G (x n))"

lemma projected_gradient_descent_iteratesI:
  assumes "⋀n. x (Suc n) = projected_gradient_step C alpha G (x n)"
  shows "projected_gradient_descent_iterates C alpha G x"
  ⟨proof⟩

lemma projected_gradient_descent_iteratesD:
```

```
  assumes "projected_gradient_descent_iterates C alpha G x"
  shows "x (Suc n) = projected_gradient_step C alpha G (x n)"
  ⟨proof⟩

lemma projected_gradient_descent_iteratesE:
  assumes "projected_gradient_descent_iterates C alpha G x"
  obtains "x (Suc n) = projected_gradient_step C alpha G (x n)"
  ⟨proof⟩

lemma projected_gradient_descent_next_mem:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes pgd: "projected_gradient_descent_iterates C alpha G x"
    and closed: "closed C"
    and nonempty: "C ≠ {}"
  shows "x (Suc n) ∈ C"
⟨proof⟩

lemma projected_gradient_descent_feasible_from_initial:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes pgd: "projected_gradient_descent_iterates C alpha G x"
    and closed: "closed C"
    and x0_mem: "x 0 ∈ C"
  shows "feasible_iterates C x"
⟨proof⟩

lemma projected_gradient_descent_one_step_distance_bound_to_point:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and u_mem: "u ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
  shows
    "f (x (Suc n)) - f u
      ≤
     (norm (x n - u) ^ 2 - norm (x (Suc n) - u) ^ 2) / (2 * alpha)"
⟨proof⟩

lemma projected_gradient_descent_step_progress_step_norm:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
```

```
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
  shows
    "(1 / (2 * alpha)) * norm (x (Suc n) - x n) ^ 2
      ≤ f (x n) - f (x (Suc n))"
⟨proof⟩

lemma projected_gradient_descent_step_descent_step_norm:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
  shows
    "f (x (Suc n))
      ≤ f (x n) - (1 / (2 * alpha)) * norm (x (Suc n) - x n) ^ 2"
⟨proof⟩

lemma projected_gradient_descent_step_descent_step_norm_add:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
  shows
    "f (x (Suc n)) + (1 / (2 * alpha)) * norm (x (Suc n) - x n) ^ 2
      ≤ f (x n)"
⟨proof⟩

lemma projected_gradient_descent_one_step_distance_bound_to_minimizer:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
```

**shows**
```
"f (x (Suc n)) - f xstar
  ≤
 (norm (x n - xstar) ^ 2 - norm (x (Suc n) - xstar) ^ 2) / (2 * alpha)"
```
⟨*proof*⟩

The main one-step estimate of this file is ⟦*smooth_convex_on ?L ?C ?f ?G; closed ?C; convex ?C; ?x ∈ ?C; ?u ∈ ?C; 0 < ?alpha; ?alpha * ?L ≤ 1*⟧ ⟹ *?f (projected_gradient_step ?C ?alpha ?G ?x) - ?f ?u ≤ ((norm (?x - ?u))*$^2$ *- (norm (projected_gradient_step ?C ?alpha ?G ?x - ?u))*$^2$*) / (2 * ?alpha)*. It is the projected analogue of the distance-potential inequality used for the unconstrained gradient-descent convergence proof.

The descent estimate ⟦*smooth_convex_on ?L ?C ?f ?G; closed ?C; convex ?C; ?x ∈ ?C; 0 < ?alpha; ?alpha * ?L ≤ 1*⟧ ⟹ *1 / (2 * ?alpha) * (norm (projected_gradient_step ?C ?alpha ?G ?x - ?x))*$^2$ *≤ ?f ?x - ?f (projected_gradient_step ?C ?alpha ?G ?x)* is obtained by taking the comparison point to be the current iterate. It controls the objective decrease by the projected step length and is the step-level input for projected-gradient mapping residual-rate estimates.

**end**
**theory** *Projected_Gradient_Descent_Convergence*
**imports** *Projection_Optimization*
**begin**

# 15 Function-value convergence for projected gradient descent

This theory proves the standard O(1/N) function-value convergence bound for projected gradient descent on a closed convex feasible set.

The proof follows the same telescoping structure as the unconstrained gradient-descent convergence proof. The only new ingredient is the projected one-step distance estimate from the projection theory.

## 15.1 Monotonicity of projected gradient descent

**lemma** *projected_gradient_descent_objective_nonincreasing:*
**fixes** *f :: "'a::{real_inner,heine_borel} ⇒ real"*
**and** *G :: "'a ⇒ 'a"*
**assumes** *smooth: "smooth_convex_on L C f G"*
**and** *closed: "closed C"*
**and** *convex: "convex C"*
**and** *pgd: "projected_gradient_descent_iterates C alpha G x"*
**and** *feasible: "feasible_iterates C x"*
**and** *alpha_pos: "0 < alpha"*
**and** *step_size: "alpha * L ≤ 1"*
**shows** *"nonincreasing_sequence (objective_values f x)"*

⟨*proof*⟩

## 15.2 Summed function-value gaps

```
lemma projected_gradient_descent_sum_function_value_gaps_bound_to_point:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and u_mem: "u ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
  shows
    "(∑n<N. f (x (Suc n)) - f u)
      ≤
     norm (x 0 - u) ^ 2 / (2 * alpha)
      - norm (x N - u) ^ 2 / (2 * alpha)"
```
⟨*proof*⟩

```
lemma projected_gradient_descent_sum_function_value_gaps_bound_to_minimizer:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
  shows
    "(∑n<N. f (x (Suc n)) - f xstar)
      ≤
     norm (x 0 - xstar) ^ 2 / (2 * alpha)
      - norm (x N - xstar) ^ 2 / (2 * alpha)"
```
⟨*proof*⟩

```
lemma projected_gradient_descent_sum_function_value_gaps_bound_to_point_initial:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and u_mem: "u ∈ C"
```

```
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
  shows
    "(∑n<N. f (x (Suc n)) - f u)
      ≤ norm (x 0 - u) ^ 2 / (2 * alpha)"
⟨proof⟩

lemma projected_gradient_descent_sum_function_value_gaps_bound_to_minimizer_initial:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
  shows
    "(∑n<N. f (x (Suc n)) - f xstar)
      ≤ norm (x 0 - xstar) ^ 2 / (2 * alpha)"
⟨proof⟩
```

## 15.3 The O(1/N) convergence rate

```
lemma projected_gradient_descent_last_gap_times_N_bound:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
    and N_pos: "N > 0"
  shows
    "real N * (f (x N) - f xstar)
      ≤ norm (x 0 - xstar) ^ 2 / (2 * alpha)"
⟨proof⟩

theorem projected_gradient_descent_function_value_gap_bound_feasible:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
```

```
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
    and N_pos: "N > 0"
  shows
    "f (x N) - f xstar
      ≤ norm (x 0 - xstar) ^ 2 / (2 * alpha * real N)"
⟨proof⟩

theorem projected_gradient_descent_function_value_gap_bound:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
    and N_pos: "N > 0"
  shows
    "f (x N) - f xstar
      ≤ norm (x 0 - xstar) ^ 2 / (2 * alpha * real N)"
⟨proof⟩
```

## 15.4 Locale form

```
locale projected_gradient_descent =
  fixes C :: "'a::{real_inner,heine_borel} set"
    and L alpha :: real
    and f :: "'a ⇒ real"
    and G :: "'a ⇒ 'a"
    and x :: "nat ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and iterates: "projected_gradient_descent_iterates C alpha G x"
    and initial_feasible: "x 0 ∈ C"
    and step_size: "alpha * L ≤ 1"
begin

lemma feasible:
  "feasible_iterates C x"
  ⟨proof⟩

lemma objective_nonincreasing:
  assumes alpha_pos: "0 < alpha"
```

```
  shows "nonincreasing_sequence (objective_values f x)"
  ⟨proof⟩

lemma sum_function_value_gaps_bound_to_point:
  assumes alpha_pos: "0 < alpha"
    and u_mem: "u ∈ C"
  shows
    "(∑n<N. f (x (Suc n)) - f u)
      ≤
     norm (x 0 - u) ^ 2 / (2 * alpha)
      - norm (x N - u) ^ 2 / (2 * alpha)"
  ⟨proof⟩

lemma sum_function_value_gaps_bound_to_minimizer:
  assumes alpha_pos: "0 < alpha"
    and minimizer: "global_min_on C f xstar"
  shows
    "(∑n<N. f (x (Suc n)) - f xstar)
      ≤
     norm (x 0 - xstar) ^ 2 / (2 * alpha)
      - norm (x N - xstar) ^ 2 / (2 * alpha)"
  ⟨proof⟩

lemma last_gap_times_N_bound:
  assumes alpha_pos: "0 < alpha"
    and minimizer: "global_min_on C f xstar"
    and N_pos: "N > 0"
  shows
    "real N * (f (x N) - f xstar)
      ≤ norm (x 0 - xstar) ^ 2 / (2 * alpha)"
  ⟨proof⟩

theorem function_value_gap_bound:
  assumes alpha_pos: "0 < alpha"
    and minimizer: "global_min_on C f xstar"
    and N_pos: "N > 0"
  shows
    "f (x N) - f xstar
      ≤ norm (x 0 - xstar) ^ 2 / (2 * alpha * real N)"
  ⟨proof⟩

end
```

The main theorem is ⟦*smooth_convex_on ?L ?C ?f ?G; closed ?C; convex ?C; projected_gradient_descent_iterates ?C ?alpha ?G ?x; ?x 0 ∈ ?C; 0 < ?alpha; ?alpha * ?L ≤ 1; global_min_on ?C ?f ?xstar; 0 < ?N*⟧ ⟹ *?f (?x ?N) - ?f ?xstar ≤ (norm (?x 0 - ?xstar))*$^2$ */ (2 * ?alpha * real ?N)*. It states the classical O(1/N) function-value convergence estimate for fixed-step projected gradient descent on a closed convex feasible set.

```
end
theory Projected_Gradient_Mapping
  imports Projected_Gradient_Descent_Convergence
begin
```

# 16 Projected-gradient mappings and optimality certificates

This theory introduces the projected-gradient mapping associated with the projected-gradient step.

For a closed convex feasible set C and a stepsize alpha > 0, the projected gradient mapping is

(1 / alpha) * (x - `P_C` (x - alpha * G x)).

It measures the normalized residual of the projected-gradient fixed-point equation. Equivalently, its norm is the length of one projected-gradient step, divided by alpha. Thus the projected-gradient mapping is the constrained analogue of the gradient residual in unconstrained gradient descent.

The main purpose of this file is to connect four equivalent or closely related languages:

the projected-gradient step residual; zero projected-gradient mapping; fixed points of the projected-gradient step; first-order variational inequalities.

For convex differentiable objectives, the zero-residual condition gives global optimality.

## 16.1 Projected-gradient mappings

```
definition projected_gradient_mapping ::
  "'a::{real_inner,heine_borel} set ⇒ real ⇒ ('a ⇒ 'a) ⇒ 'a ⇒ 'a"
where
  "projected_gradient_mapping C alpha G x =
     (1 / alpha) *R (x - projected_gradient_step C alpha G x)"

lemma projected_gradient_mapping_step_relation:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes alpha_nonzero: "alpha ≠ 0"
  shows
    "alpha *R projected_gradient_mapping C alpha G x =
      x - projected_gradient_step C alpha G x"
  ⟨proof⟩

lemma projected_gradient_step_eq_sub_mapping:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes alpha_nonzero: "alpha ≠ 0"
  shows
    "projected_gradient_step C alpha G x =
```

```
    x - alpha *_R projected_gradient_mapping C alpha G x"
```
⟨proof⟩

## 16.2 Norm interpretation as a residual

The projected-gradient mapping is the projected step residual normalized by the stepsize. The following elementary identities are useful when translating descent in step length into descent in projected-gradient mapping norm.

```
lemma projected_gradient_mapping_norm_eq_step_distance_divide:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes alpha_pos: "0 < alpha"
  shows
    "norm (projected_gradient_mapping C alpha G x) =
      norm (x - projected_gradient_step C alpha G x) / alpha"
```
⟨proof⟩

```
lemma projected_gradient_base_step_distance_eq_alpha_mapping_norm:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes alpha_pos: "0 < alpha"
  shows
    "norm (x - projected_gradient_step C alpha G x) =
      alpha * norm (projected_gradient_mapping C alpha G x)"
```
⟨proof⟩

```
lemma projected_gradient_step_base_distance_eq_alpha_mapping_norm:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes alpha_pos: "0 < alpha"
  shows
    "norm (projected_gradient_step C alpha G x - x) =
      alpha * norm (projected_gradient_mapping C alpha G x)"
```
⟨proof⟩

```
lemma projected_gradient_step_distance_sq_eq_mapping_norm_sq:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes alpha_pos: "0 < alpha"
  shows
    "norm (projected_gradient_step C alpha G x - x) ^ 2 =
      alpha ^ 2 * norm (projected_gradient_mapping C alpha G x) ^ 2"
```
⟨proof⟩

```
lemma projected_gradient_mapping_norm_sq_eq_step_distance_sq:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes alpha_pos: "0 < alpha"
  shows
    "norm (projected_gradient_mapping C alpha G x) ^ 2 =
      norm (projected_gradient_step C alpha G x - x) ^ 2 / alpha ^ 2"
```
⟨proof⟩

```
lemma projected_gradient_mapping_zero_imp_step_eq:
```

**fixes** *C :: "'a::{real_inner,heine_borel} set"*
**assumes** *alpha_nonzero: "alpha ≠ 0"*
**and** *zero: "projected_gradient_mapping C alpha G x = 0"*
**shows** *"projected_gradient_step C alpha G x = x"*
⟨*proof*⟩

**lemma** *projected_gradient_step_eq_imp_mapping_zero:*
**fixes** *C :: "'a::{real_inner,heine_borel} set"*
**assumes** *step: "projected_gradient_step C alpha G x = x"*
**shows** *"projected_gradient_mapping C alpha G x = 0"*
⟨*proof*⟩

**lemma** *projected_gradient_mapping_zero_iff_fixed_point:*
**fixes** *C :: "'a::{real_inner,heine_borel} set"*
**assumes** *alpha_nonzero: "alpha ≠ 0"*
**shows**
*"projected_gradient_mapping C alpha G x = 0*
*⟷ projected_gradient_step C alpha G x = x"*
⟨*proof*⟩

**lemma** *projected_gradient_mapping_zero_iff_step_distance_zero:*
**fixes** *C :: "'a::{real_inner,heine_borel} set"*
**assumes** *alpha_nonzero: "alpha ≠ 0"*
**shows**
*"projected_gradient_mapping C alpha G x = 0*
*⟷ norm (projected_gradient_step C alpha G x - x) = 0"*
⟨*proof*⟩

## 16.3 Residual form of the projection variational inequality

**lemma** *gradient_step_minus_projected_gradient_step_eq_mapping_residual:*
**fixes** *C :: "'a::{real_inner,heine_borel} set"*
**assumes** *alpha_nonzero: "alpha ≠ 0"*
**shows**
*"gradient_step alpha G x - projected_gradient_step C alpha G x =*
*alpha $*_R$*
*(projected_gradient_mapping C alpha G x - G x)"*
⟨*proof*⟩

**lemma** *projected_gradient_mapping_variational_inequality:*
**fixes** *C :: "'a::{real_inner,heine_borel} set"*
**assumes** *convex: "convex C"*
**and** *closed: "closed C"*
**and** *u_mem: "u ∈ C"*
**and** *alpha_pos: "0 < alpha"*
**shows**
*"inner*
*(projected_gradient_mapping C alpha G x - G x)*
*(u - projected_gradient_step C alpha G x)*

≤ 0"
⟨proof⟩

## 16.4 Fixed points and first-order conditions

```
lemma projected_gradient_fixed_point_imp_first_order_condition:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes convex: "convex C"
    and closed: "closed C"
    and x_mem: "x ∈ C"
    and alpha_pos: "0 < alpha"
    and fixed: "projected_gradient_step C alpha G x = x"
  shows "first_order_condition_at C (G x) x"
⟨proof⟩

lemma first_order_condition_imp_projected_gradient_fixed_point:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes convex: "convex C"
    and closed: "closed C"
    and alpha_pos: "0 < alpha"
    and foc: "first_order_condition_at C (G x) x"
  shows "projected_gradient_step C alpha G x = x"
⟨proof⟩

theorem projected_gradient_fixed_point_iff_first_order_condition:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes convex: "convex C"
    and closed: "closed C"
    and x_mem: "x ∈ C"
    and alpha_pos: "0 < alpha"
  shows
    "projected_gradient_step C alpha G x = x
      ⟷ first_order_condition_at C (G x) x"
⟨proof⟩
```

## 16.5 Zero projected-gradient mapping and first-order conditions

```
lemma projected_gradient_mapping_zero_imp_first_order_condition:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes convex: "convex C"
    and closed: "closed C"
    and x_mem: "x ∈ C"
    and alpha_pos: "0 < alpha"
    and zero: "projected_gradient_mapping C alpha G x = 0"
  shows "first_order_condition_at C (G x) x"
⟨proof⟩

lemma first_order_condition_imp_projected_gradient_mapping_zero:
```

```
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes convex: "convex C"
    and closed: "closed C"
    and alpha_pos: "0 < alpha"
    and foc: "first_order_condition_at C (G x) x"
  shows "projected_gradient_mapping C alpha G x = 0"
⟨proof⟩

theorem projected_gradient_mapping_zero_iff_first_order_condition:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes convex: "convex C"
    and closed: "closed C"
    and x_mem: "x ∈ C"
    and alpha_pos: "0 < alpha"
  shows
    "projected_gradient_mapping C alpha G x = 0
      ⟷ first_order_condition_at C (G x) x"
⟨proof⟩
```

## 16.6 Optimality consequences

```
lemma projected_gradient_fixed_point_imp_global_min_on:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and x_mem: "x ∈ C"
    and alpha_pos: "0 < alpha"
    and fixed: "projected_gradient_step C alpha G x = x"
  shows "global_min_on C f x"
⟨proof⟩

lemma projected_gradient_mapping_zero_imp_global_min_on:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and x_mem: "x ∈ C"
    and alpha_pos: "0 < alpha"
    and zero: "projected_gradient_mapping C alpha G x = 0"
  shows "global_min_on C f x"
⟨proof⟩

lemma first_order_condition_imp_global_min_on_smooth_convex:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
```

```
    and foc: "first_order_condition_at C (G x) x"
  shows "global_min_on C f x"
⟨proof⟩
```

## 16.7 Locale form

```
locale projected_gradient_mapping_context =
  fixes C :: "'a::{real_inner,heine_borel} set"
    and L alpha :: real
    and f :: "'a ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and alpha_pos: "0 < alpha"
begin

lemma alpha_nonzero:
  "alpha ≠ 0"
  ⟨proof⟩

lemma mapping_step_relation:
  "alpha *R projected_gradient_mapping C alpha G x =
    x - projected_gradient_step C alpha G x"
  ⟨proof⟩

lemma step_eq_sub_mapping:
  "projected_gradient_step C alpha G x =
    x - alpha *R projected_gradient_mapping C alpha G x"
  ⟨proof⟩

lemma mapping_norm_eq_step_distance_divide:
  "norm (projected_gradient_mapping C alpha G x) =
    norm (x - projected_gradient_step C alpha G x) / alpha"
  ⟨proof⟩

lemma base_step_distance_eq_alpha_mapping_norm:
  "norm (x - projected_gradient_step C alpha G x) =
    alpha * norm (projected_gradient_mapping C alpha G x)"
  ⟨proof⟩

lemma step_base_distance_eq_alpha_mapping_norm:
  "norm (projected_gradient_step C alpha G x - x) =
    alpha * norm (projected_gradient_mapping C alpha G x)"
  ⟨proof⟩

lemma step_distance_sq_eq_mapping_norm_sq:
  "norm (projected_gradient_step C alpha G x - x) ^ 2 =
    alpha ^ 2 * norm (projected_gradient_mapping C alpha G x) ^ 2"
```

⟨proof⟩

**lemma** *mapping_norm_sq_eq_step_distance_sq:*
  *"norm (projected_gradient_mapping C alpha G x) ^ 2 =*
    *norm (projected_gradient_step C alpha G x - x) ^ 2 / alpha ^ 2"*
  ⟨proof⟩

**lemma** *mapping_zero_iff_step_distance_zero:*
  *"projected_gradient_mapping C alpha G x = 0*
    *⟷ norm (projected_gradient_step C alpha G x - x) = 0"*
  ⟨proof⟩

**lemma** *mapping_zero_iff_fixed_point:*
  *"projected_gradient_mapping C alpha G x = 0*
    *⟷ projected_gradient_step C alpha G x = x"*
  ⟨proof⟩

**lemma** *mapping_variational_inequality:*
  **assumes** *u_mem: "u ∈ C"*
  **shows**
    *"inner*
      *(projected_gradient_mapping C alpha G x - G x)*
      *(u - projected_gradient_step C alpha G x)*
      *≤ 0"*
  ⟨proof⟩

**lemma** *fixed_point_imp_first_order_condition:*
  **assumes** *x_mem: "x ∈ C"*
    **and** *fixed: "projected_gradient_step C alpha G x = x"*
  **shows** *"first_order_condition_at C (G x) x"*
  ⟨proof⟩

**lemma** *first_order_condition_imp_fixed_point:*
  **assumes** *foc: "first_order_condition_at C (G x) x"*
  **shows** *"projected_gradient_step C alpha G x = x"*
  ⟨proof⟩

**lemma** *fixed_point_iff_first_order_condition:*
  **assumes** *x_mem: "x ∈ C"*
  **shows**
    *"projected_gradient_step C alpha G x = x*
      *⟷ first_order_condition_at C (G x) x"*
  ⟨proof⟩

**lemma** *mapping_zero_imp_first_order_condition:*
  **assumes** *x_mem: "x ∈ C"*
    **and** *zero: "projected_gradient_mapping C alpha G x = 0"*
  **shows** *"first_order_condition_at C (G x) x"*
  ⟨proof⟩

**lemma** *first_order_condition_imp_mapping_zero:*
  **assumes** *foc: "first_order_condition_at C (G x) x"*
  **shows** *"projected_gradient_mapping C alpha G x = 0"*
  ⟨*proof*⟩

**lemma** *mapping_zero_iff_first_order_condition:*
  **assumes** *x_mem: "x ∈ C"*
  **shows**
    *"projected_gradient_mapping C alpha G x = 0*
      *⟷ first_order_condition_at C (G x) x"*
  ⟨*proof*⟩

**lemma** *fixed_point_imp_global_min_on:*
  **assumes** *x_mem: "x ∈ C"*
    **and** *fixed: "projected_gradient_step C alpha G x = x"*
  **shows** *"global_min_on C f x"*
  ⟨*proof*⟩

**lemma** *mapping_zero_imp_global_min_on:*
  **assumes** *x_mem: "x ∈ C"*
    **and** *zero: "projected_gradient_mapping C alpha G x = 0"*
  **shows** *"global_min_on C f x"*
  ⟨*proof*⟩

**lemma** *first_order_condition_imp_global_min_on:*
  **assumes** *foc: "first_order_condition_at C (G x) x"*
  **shows** *"global_min_on C f x"*
  ⟨*proof*⟩

**end**

This file packages the projected-gradient mapping as an explicit optimization residual. The norm identities *0 < ?alpha ⟹ norm (projected_gradient_mapping ?C ?alpha ?G ?x) = norm (?x - projected_gradient_step ?C ?alpha ?G ?x) / ?alpha* and *0 < ?alpha ⟹ (norm (projected_gradient_step ?C ?alpha ?G ?x - ?x))*$^2$ *= ?alpha*$^2$ ** (norm (projected_gradient_mapping ?C ?alpha ?G ?x))*$^2$ show that the mapping norm is exactly the projected-step residual normalized by the stepsize.

The main bridge theorem is ⟦*convex ?C; closed ?C; ?x ∈ ?C; 0 < ?alpha*⟧ *⟹ (projected_gradient_mapping ?C ?alpha ?G ?x = 0) = first_order_condition_at ?C (?G ?x) ?x*: for a feasible point x and a positive stepsize, the projected-gradient mapping vanishes exactly when x satisfies the constrained first-order variational inequality.

Together with smooth convexity, this gives the optimality certificate ⟦*smooth_convex_on ?L ?C ?f ?G; closed ?C; convex ?C; ?x ∈ ?C; 0 < ?alpha; projected_gradient_mapping ?C ?alpha ?G ?x = 0*⟧ *⟹ global_min_on ?C ?f ?x*.

**end**
**theory** *Projected_Gradient_Mapping_Rates*
  **imports** *Projected_Gradient_Mapping*
**begin**

# 17 Projected-gradient mapping residual rates

This theory is the technical residual-rate engine for projected-gradient descent.

The previous theory introduces the projected-gradient mapping and proves its fixed-point and optimality properties. This theory proves finite-horizon estimates for the squared norm of that mapping along projected-gradient descent iterates.

The main proof pattern is the standard first-order complexity argument: a one-step progress inequality controls the squared projected-gradient mapping norm by the objective decrease, and an abstract telescoping argument converts this into finite-sum, average, and small-residual estimates.

This theory deliberately stays at the level of squared mapping norms. The next theory packages these estimates into user-facing residual convergence and epsilon-stationarity certificates.

## 17.1 Step length and projected-gradient mapping norm

```
lemma projected_gradient_step_distance_sq_eq_mapping_norm_sq:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes alpha_pos: "0 < alpha"
  shows
    "norm (projected_gradient_step C alpha G x - x) ^ 2 =
      alpha ^ 2 * norm (projected_gradient_mapping C alpha G x) ^ 2"
⟨proof⟩
```

```
lemma projected_gradient_mapping_norm_sq_eq_step_distance_sq:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes alpha_pos: "0 < alpha"
  shows
    "norm (projected_gradient_mapping C alpha G x) ^ 2 =
      norm (projected_gradient_step C alpha G x - x) ^ 2 / alpha ^ 2"
⟨proof⟩
```

## 17.2 One-step progress in terms of the mapping residual

```
lemma projected_gradient_step_progress_mapping:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
```

```
    and x_mem: "x ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
  shows
    "(alpha / 2) * norm (projected_gradient_mapping C alpha G x) ^ 2
      ≤ f x - f (projected_gradient_step C alpha G x)"
⟨proof⟩
```

```
lemma projected_gradient_descent_step_progress_mapping:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
  shows
    "(alpha / 2) * norm (projected_gradient_mapping C alpha G (x n)) ^
2
      ≤ f (x n) - f (x (Suc n))"
⟨proof⟩
```

## 17.3 Finite-sum mapping-residual bounds

The following estimates are the summability form of the residual-rate argument. The weighted bound is the direct telescoping consequence of the one-step progress inequality. The unweighted bounds divide out the positive stepsize factor.

```
lemma projected_gradient_descent_sum_weighted_mapping_norm_sq_bound_feasible:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
  shows
    "sum (λn. (alpha / 2) *
      norm (projected_gradient_mapping C alpha G (x n)) ^ 2) {..<N}
      ≤ f (x 0) - f (x N)"
⟨proof⟩
```

```
lemma projected_gradient_descent_sum_weighted_mapping_norm_sq_bound:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
```

**assumes** *smooth: "smooth_convex_on L C f G"*
  **and** *closed: "closed C"*
  **and** *convex: "convex C"*
  **and** *pgd: "projected_gradient_descent_iterates C alpha G x"*
  **and** *x0_mem: "x 0 ∈ C"*
  **and** *alpha_pos: "0 < alpha"*
  **and** *step_size: "alpha * L ≤ 1"*
**shows**
  *"sum (λn. (alpha / 2) **
    *norm (projected_gradient_mapping C alpha G (x n)) ^ 2) {..<N}*
    *≤ f (x 0) - f (x N)"*
⟨*proof*⟩

**lemma** *projected_gradient_descent_sum_weighted_mapping_norm_sq_bound_below_feasible:*
**fixes** *f :: "'a::{real_inner,heine_borel} ⇒ real"*
  **and** *G :: "'a ⇒ 'a"*
**assumes** *smooth: "smooth_convex_on L C f G"*
  **and** *closed: "closed C"*
  **and** *convex: "convex C"*
  **and** *pgd: "projected_gradient_descent_iterates C alpha G x"*
  **and** *feasible: "feasible_iterates C x"*
  **and** *alpha_pos: "0 < alpha"*
  **and** *step_size: "alpha * L ≤ 1"*
  **and** *lower: "B ≤ f (x N)"*
**shows**
  *"sum (λn. (alpha / 2) **
    *norm (projected_gradient_mapping C alpha G (x n)) ^ 2) {..<N}*
    *≤ f (x 0) - B"*
⟨*proof*⟩

**lemma** *projected_gradient_descent_sum_weighted_mapping_norm_sq_bound_below:*
**fixes** *f :: "'a::{real_inner,heine_borel} ⇒ real"*
  **and** *G :: "'a ⇒ 'a"*
**assumes** *smooth: "smooth_convex_on L C f G"*
  **and** *closed: "closed C"*
  **and** *convex: "convex C"*
  **and** *pgd: "projected_gradient_descent_iterates C alpha G x"*
  **and** *x0_mem: "x 0 ∈ C"*
  **and** *alpha_pos: "0 < alpha"*
  **and** *step_size: "alpha * L ≤ 1"*
  **and** *lower: "B ≤ f (x N)"*
**shows**
  *"sum (λn. (alpha / 2) **
    *norm (projected_gradient_mapping C alpha G (x n)) ^ 2) {..<N}*
    *≤ f (x 0) - B"*
⟨*proof*⟩

**lemma** *projected_gradient_descent_sum_mapping_norm_sq_bound_feasible:*
**fixes** *f :: "'a::{real_inner,heine_borel} ⇒ real"*

```
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
  shows
    "sum (λn. norm (projected_gradient_mapping C alpha G (x n)) ^ 2) {..<N}
      ≤ (2 / alpha) * (f (x 0) - f (x N))"
⟨proof⟩

lemma projected_gradient_descent_sum_mapping_norm_sq_bound:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
  shows
    "sum (λn. norm (projected_gradient_mapping C alpha G (x n)) ^ 2) {..<N}
      ≤ (2 / alpha) * (f (x 0) - f (x N))"
⟨proof⟩

lemma projected_gradient_descent_sum_mapping_norm_sq_bound_below_feasible:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and lower: "B ≤ f (x N)"
  shows
    "sum (λn. norm (projected_gradient_mapping C alpha G (x n)) ^ 2) {..<N}
      ≤ (2 / alpha) * (f (x 0) - B)"
⟨proof⟩

lemma projected_gradient_descent_sum_mapping_norm_sq_bound_below:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
```

```
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and lower: "B ≤ f (x N)"
  shows
    "sum (λn. norm (projected_gradient_mapping C alpha G (x n)) ^ 2) {..<N}
      ≤ (2 / alpha) * (f (x 0) - B)"
⟨proof⟩
```

## 17.4 Average mapping-residual bounds

Averaging the finite-sum estimates gives the usual O(1/N) bound on the average squared projected-gradient mapping norm.

```
lemma projected_gradient_descent_average_mapping_norm_sq_bound_feasible:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and N_pos: "N > 0"
  shows
    "(1 / real N) *
      sum (λn. norm (projected_gradient_mapping C alpha G (x n)) ^ 2)
{..<N}
      ≤ (2 / (alpha * real N)) * (f (x 0) - f (x N))"
⟨proof⟩

lemma projected_gradient_descent_average_mapping_norm_sq_bound:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and N_pos: "N > 0"
  shows
    "(1 / real N) *
      sum (λn. norm (projected_gradient_mapping C alpha G (x n)) ^ 2)
{..<N}
      ≤ (2 / (alpha * real N)) * (f (x 0) - f (x N))"
```

⟨proof⟩

**lemma** *projected_gradient_descent_average_mapping_norm_sq_bound_below_feasible:*
```
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and lower: "B ≤ f (x N)"
    and N_pos: "N > 0"
  shows
    "(1 / real N) *
      sum (λn. norm (projected_gradient_mapping C alpha G (x n)) ^ 2)
{..<N}
      ≤ (2 / (alpha * real N)) * (f (x 0) - B)"
```
⟨proof⟩

**lemma** *projected_gradient_descent_average_mapping_norm_sq_bound_below:*
```
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and lower: "B ≤ f (x N)"
    and N_pos: "N > 0"
  shows
    "(1 / real N) *
      sum (λn. norm (projected_gradient_mapping C alpha G (x n)) ^ 2)
{..<N}
      ≤ (2 / (alpha * real N)) * (f (x 0) - B)"
```
⟨proof⟩

## 17.5 Small mapping-residual consequences

The average bound implies that at least one of the first N iterates has squared projected-gradient mapping norm no larger than the average. These lemmas are the technical small-residual estimates later converted into epsilon-stationarity certificates.

**lemma** *projected_gradient_descent_exists_small_mapping_norm_sq_feasible:*
```
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
```

```
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and N_pos: "N > 0"
  shows
    "∃n<N.
      norm (projected_gradient_mapping C alpha G (x n)) ^ 2
      ≤ (2 / (alpha * real N)) * (f (x 0) - f (x N))"
⟨proof⟩

lemma projected_gradient_descent_exists_small_mapping_norm_sq:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and N_pos: "N > 0"
  shows
    "∃n<N.
      norm (projected_gradient_mapping C alpha G (x n)) ^ 2
      ≤ (2 / (alpha * real N)) * (f (x 0) - f (x N))"
⟨proof⟩

lemma projected_gradient_descent_exists_small_mapping_norm_sq_below_feasible:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and lower: "B ≤ f (x N)"
    and N_pos: "N > 0"
  shows
    "∃n<N.
      norm (projected_gradient_mapping C alpha G (x n)) ^ 2
      ≤ (2 / (alpha * real N)) * (f (x 0) - B)"
⟨proof⟩
```

```
lemma projected_gradient_descent_exists_small_mapping_norm_sq_below:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and lower: "B ≤ f (x N)"
    and N_pos: "N > 0"
  shows
    "∃n<N.
      norm (projected_gradient_mapping C alpha G (x n)) ^ 2
      ≤ (2 / (alpha * real N)) * (f (x 0) - B)"
⟨proof⟩
```

## 17.6 Minimizer-based mapping-residual bounds

When a global minimizer is available, the lower bound can be specialized to the optimal value. These versions are the most convenient form for complexity statements depending on the initial objective gap.

```
lemma projected_gradient_descent_sum_mapping_norm_sq_bound_to_minimizer_feasible:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
  shows
    "sum (λn. norm (projected_gradient_mapping C alpha G (x n)) ^ 2) {..<N}
      ≤ (2 / alpha) * (f (x 0) - f xstar)"
⟨proof⟩
```

```
lemma projected_gradient_descent_sum_mapping_norm_sq_bound_to_minimizer:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
```

```
    and minimizer: "global_min_on C f xstar"
  shows
    "sum (λn. norm (projected_gradient_mapping C alpha G (x n)) ^ 2) {..<N}
      ≤ (2 / alpha) * (f (x 0) - f xstar)"
⟨proof⟩
```

```
lemma projected_gradient_descent_exists_small_mapping_norm_sq_to_minimizer_feasible:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
    and N_pos: "N > 0"
  shows
    "∃n<N.
      norm (projected_gradient_mapping C alpha G (x n)) ^ 2
      ≤ (2 / (alpha * real N)) * (f (x 0) - f xstar)"
⟨proof⟩
```

```
lemma projected_gradient_descent_exists_small_mapping_norm_sq_to_minimizer:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
    and N_pos: "N > 0"
  shows
    "∃n<N.
      norm (projected_gradient_mapping C alpha G (x n)) ^ 2
      ≤ (2 / (alpha * real N)) * (f (x 0) - f xstar)"
⟨proof⟩
```

## 17.7 Recommended mapping-rate interface

The following theorem groups collect the main reusable consequences of this theory. They separate the technical rate engine from the later residual notation and epsilon-stationarity layer.

```
lemmas projected_gradient_mapping_step_length_results =
  projected_gradient_step_distance_sq_eq_mapping_norm_sq
```

```
  projected_gradient_mapping_norm_sq_eq_step_distance_sq

lemmas projected_gradient_mapping_progress_results =
  projected_gradient_step_progress_mapping
  projected_gradient_descent_step_progress_mapping

lemmas projected_gradient_mapping_weighted_sum_results =
  projected_gradient_descent_sum_weighted_mapping_norm_sq_bound_feasible
  projected_gradient_descent_sum_weighted_mapping_norm_sq_bound
  projected_gradient_descent_sum_weighted_mapping_norm_sq_bound_below_feasible
  projected_gradient_descent_sum_weighted_mapping_norm_sq_bound_below

lemmas projected_gradient_mapping_sum_results =
  projected_gradient_descent_sum_mapping_norm_sq_bound_feasible
  projected_gradient_descent_sum_mapping_norm_sq_bound
  projected_gradient_descent_sum_mapping_norm_sq_bound_below_feasible
  projected_gradient_descent_sum_mapping_norm_sq_bound_below
  projected_gradient_descent_sum_mapping_norm_sq_bound_to_minimizer_feasible
  projected_gradient_descent_sum_mapping_norm_sq_bound_to_minimizer

lemmas projected_gradient_mapping_average_results =
  projected_gradient_descent_average_mapping_norm_sq_bound_feasible
  projected_gradient_descent_average_mapping_norm_sq_bound
  projected_gradient_descent_average_mapping_norm_sq_bound_below_feasible
  projected_gradient_descent_average_mapping_norm_sq_bound_below

lemmas projected_gradient_mapping_small_residual_results =
  projected_gradient_descent_exists_small_mapping_norm_sq_feasible
  projected_gradient_descent_exists_small_mapping_norm_sq
  projected_gradient_descent_exists_small_mapping_norm_sq_below_feasible
  projected_gradient_descent_exists_small_mapping_norm_sq_below
  projected_gradient_descent_exists_small_mapping_norm_sq_to_minimizer_feasible
  projected_gradient_descent_exists_small_mapping_norm_sq_to_minimizer

lemmas projected_gradient_mapping_rate_engine =
  projected_gradient_mapping_step_length_results
  projected_gradient_mapping_progress_results
  projected_gradient_mapping_weighted_sum_results
  projected_gradient_mapping_sum_results
  projected_gradient_mapping_average_results
  projected_gradient_mapping_small_residual_results
```

## 17.8 Locale form

```
context projected_gradient_descent
begin

lemma step_progress_mapping:
  assumes alpha_pos: "0 < alpha"
```

```
  shows
    "(alpha / 2) * norm (projected_gradient_mapping C alpha G (x n)) ^
2
      ≤ f (x n) - f (x (Suc n))"
  ⟨proof⟩

lemma sum_weighted_mapping_norm_sq_bound:
  assumes alpha_pos: "0 < alpha"
  shows
    "sum (λn. (alpha / 2) *
      norm (projected_gradient_mapping C alpha G (x n)) ^ 2) {..<N}
      ≤ f (x 0) - f (x N)"
  ⟨proof⟩

lemma sum_weighted_mapping_norm_sq_bound_below:
  assumes alpha_pos: "0 < alpha"
    and lower: "B ≤ f (x N)"
  shows
    "sum (λn. (alpha / 2) *
      norm (projected_gradient_mapping C alpha G (x n)) ^ 2) {..<N}
      ≤ f (x 0) - B"
  ⟨proof⟩

lemma sum_mapping_norm_sq_bound:
  assumes alpha_pos: "0 < alpha"
  shows
    "sum (λn. norm (projected_gradient_mapping C alpha G (x n)) ^ 2) {..<N}
      ≤ (2 / alpha) * (f (x 0) - f (x N))"
  ⟨proof⟩

lemma sum_mapping_norm_sq_bound_below:
  assumes alpha_pos: "0 < alpha"
    and lower: "B ≤ f (x N)"
  shows
    "sum (λn. norm (projected_gradient_mapping C alpha G (x n)) ^ 2) {..<N}
      ≤ (2 / alpha) * (f (x 0) - B)"
  ⟨proof⟩

lemma average_mapping_norm_sq_bound:
  assumes alpha_pos: "0 < alpha"
    and N_pos: "N > 0"
  shows
    "(1 / real N) *
      sum (λn. norm (projected_gradient_mapping C alpha G (x n)) ^ 2)
{..<N}
      ≤ (2 / (alpha * real N)) * (f (x 0) - f (x N))"
  ⟨proof⟩

lemma average_mapping_norm_sq_bound_below:
```

```
  assumes alpha_pos: "0 < alpha"
    and lower: "B ≤ f (x N)"
    and N_pos: "N > 0"
  shows
    "(1 / real N) *
      sum (λn. norm (projected_gradient_mapping C alpha G (x n)) ^ 2)
{..<N}
      ≤ (2 / (alpha * real N)) * (f (x 0) - B)"
  ⟨proof⟩

lemma exists_small_mapping_norm_sq:
  assumes alpha_pos: "0 < alpha"
    and N_pos: "N > 0"
  shows
    "∃n<N.
      norm (projected_gradient_mapping C alpha G (x n)) ^ 2
      ≤ (2 / (alpha * real N)) * (f (x 0) - f (x N))"
  ⟨proof⟩

lemma exists_small_mapping_norm_sq_below:
  assumes alpha_pos: "0 < alpha"
    and lower: "B ≤ f (x N)"
    and N_pos: "N > 0"
  shows
    "∃n<N.
      norm (projected_gradient_mapping C alpha G (x n)) ^ 2
      ≤ (2 / (alpha * real N)) * (f (x 0) - B)"
  ⟨proof⟩

lemma sum_mapping_norm_sq_bound_to_minimizer:
  assumes alpha_pos: "0 < alpha"
    and minimizer: "global_min_on C f xstar"
  shows
    "sum (λn. norm (projected_gradient_mapping C alpha G (x n)) ^ 2) {..<N}
      ≤ (2 / alpha) * (f (x 0) - f xstar)"
  ⟨proof⟩

lemma exists_small_mapping_norm_sq_to_minimizer:
  assumes alpha_pos: "0 < alpha"
    and minimizer: "global_min_on C f xstar"
    and N_pos: "N > 0"
  shows
    "∃n<N.
      norm (projected_gradient_mapping C alpha G (x n)) ^ 2
      ≤ (2 / (alpha * real N)) * (f (x 0) - f xstar)"
  ⟨proof⟩

end
```

The main reusable consequences of this theory are:

*projected_gradient_descent_sum_mapping_norm_sq_bound*; *projected_gradient_descent_average*
*projected_gradient_descent_exists_small_mapping_norm_sq*; *projected_gradient_descent_exists_*

Together, these state that the squared projected-gradient mapping residual satisfies finite-sum, average, and finite-horizon small-residual bounds along projected-gradient descent.

The theorem group *projected_gradient_mapping_rate_engine* collects the main technical rate interface of this theory.

```
end
theory Projected_Gradient_Descent_Residual_Convergence
  imports Projected_Gradient_Mapping_Rates
begin
```

# 18 Residual convergence certificates for projected gradient descent

This theory is the residual-complexity layer of the projected-gradient descent development.

The previous theory proves finite-horizon estimates for the squared norm of the projected-gradient mapping. Here we package those estimates as user-facing residual convergence and epsilon-stationarity certificates.

The projected-gradient residual is the norm of the projected-gradient mapping. It is the constrained analogue of the gradient norm in unconstrained smooth optimization. Small residual therefore gives an approximate first-order stationarity certificate for constrained smooth convex optimization.

The main high-level result of this theory is a finite-horizon complexity statement: if the horizon N is large enough relative to the initial objective gap and the desired tolerance eps, then at least one of the first N projected-gradient iterates has projected-gradient residual at most eps.

The result is intentionally stated as a finite-horizon certificate rather than as a filter-limit statement. This is the form most commonly used in first-order complexity estimates, and it keeps the statement directly reusable for algorithmic convergence proofs.

## 18.1 Residual notation

```
definition projected_gradient_residual ::
  "'a::{real_inner,heine_borel} set ⇒ real ⇒ ('a ⇒ 'a) ⇒ 'a ⇒ real"
where
  "projected_gradient_residual C alpha G x =
    norm (projected_gradient_mapping C alpha G x)"

definition projected_gradient_residual_sq ::
  "'a::{real_inner,heine_borel} set ⇒ real ⇒ ('a ⇒ 'a) ⇒ 'a ⇒ real"
```

```
where
  "projected_gradient_residual_sq C alpha G x =
    norm (projected_gradient_mapping C alpha G x) ^ 2"

lemma projected_gradient_residual_nonneg:
  "0 ≤ projected_gradient_residual C alpha G x"
  ⟨proof⟩

lemma projected_gradient_residual_sq_nonneg:
  "0 ≤ projected_gradient_residual_sq C alpha G x"
  ⟨proof⟩

lemma projected_gradient_residual_sq_eq_residual_power2:
  "projected_gradient_residual_sq C alpha G x =
    projected_gradient_residual C alpha G x ^ 2"
  ⟨proof⟩

lemma projected_gradient_residual_eq_mapping_norm:
  "projected_gradient_residual C alpha G x =
    norm (projected_gradient_mapping C alpha G x)"
  ⟨proof⟩

lemma projected_gradient_residual_sq_eq_mapping_norm_sq:
  "projected_gradient_residual_sq C alpha G x =
    norm (projected_gradient_mapping C alpha G x) ^ 2"
  ⟨proof⟩

lemma projected_gradient_residual_le_of_sq_le:
  assumes sq_bound: "projected_gradient_residual_sq C alpha G x ≤ eps
^ 2"
    and eps_nonneg: "0 ≤ eps"
  shows "projected_gradient_residual C alpha G x ≤ eps"
⟨proof⟩

lemma projected_gradient_residual_sq_le_of_residual_le:
  assumes res_bound: "projected_gradient_residual C alpha G x ≤ eps"
    and eps_nonneg: "0 ≤ eps"
  shows "projected_gradient_residual_sq C alpha G x ≤ eps ^ 2"
⟨proof⟩
```

## 18.2 Finite-horizon squared-residual certificates

```
lemma projected_gradient_descent_exists_small_residual_sq_feasible:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
```

```
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and N_pos: "N > 0"
  shows
    "∃n<N.
      projected_gradient_residual_sq C alpha G (x n)
      ≤ (2 / (alpha * real N)) * (f (x 0) - f (x N))"
⟨proof⟩

lemma projected_gradient_descent_exists_small_residual_sq:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and N_pos: "N > 0"
  shows
    "∃n<N.
      projected_gradient_residual_sq C alpha G (x n)
      ≤ (2 / (alpha * real N)) * (f (x 0) - f (x N))"
⟨proof⟩

lemma projected_gradient_descent_exists_small_residual_sq_below_feasible:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and lower: "B ≤ f (x N)"
    and N_pos: "N > 0"
  shows
    "∃n<N.
      projected_gradient_residual_sq C alpha G (x n)
      ≤ (2 / (alpha * real N)) * (f (x 0) - B)"
⟨proof⟩

lemma projected_gradient_descent_exists_small_residual_sq_below:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
```

```
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and lower: "B ≤ f (x N)"
    and N_pos: "N > 0"
  shows
    "∃n<N.
      projected_gradient_residual_sq C alpha G (x n)
      ≤ (2 / (alpha * real N)) * (f (x 0) - B)"
⟨proof⟩
```

### 18.3 Finite-horizon residual certificates

```
lemma projected_gradient_descent_exists_epsilon_residual_feasible:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and N_pos: "N > 0"
    and eps_nonneg: "0 ≤ eps"
    and horizon:
      "(2 / (alpha * real N)) * (f (x 0) - f (x N)) ≤ eps ^ 2"
  shows
    "∃n<N. projected_gradient_residual C alpha G (x n) ≤ eps"
⟨proof⟩
```

```
lemma projected_gradient_descent_exists_epsilon_residual:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and N_pos: "N > 0"
    and eps_nonneg: "0 ≤ eps"
    and horizon:
      "(2 / (alpha * real N)) * (f (x 0) - f (x N)) ≤ eps ^ 2"
  shows
```

```
    "∃n<N. projected_gradient_residual C alpha G (x n) ≤ eps"
⟨proof⟩

lemma projected_gradient_descent_exists_epsilon_residual_below_feasible:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and lower: "B ≤ f (x N)"
    and N_pos: "N > 0"
    and eps_nonneg: "0 ≤ eps"
    and horizon:
      "(2 / (alpha * real N)) * (f (x 0) - B) ≤ eps ^ 2"
  shows
    "∃n<N. projected_gradient_residual C alpha G (x n) ≤ eps"
⟨proof⟩

lemma projected_gradient_descent_exists_epsilon_residual_below:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and lower: "B ≤ f (x N)"
    and N_pos: "N > 0"
    and eps_nonneg: "0 ≤ eps"
    and horizon:
      "(2 / (alpha * real N)) * (f (x 0) - B) ≤ eps ^ 2"
  shows
    "∃n<N. projected_gradient_residual C alpha G (x n) ≤ eps"
⟨proof⟩
```

## 18.4 Minimizer-based residual certificates

```
lemma projected_gradient_descent_exists_small_residual_sq_to_minimizer_feasible:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
```

```
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
    and N_pos: "N > 0"
  shows
    "∃n<N.
      projected_gradient_residual_sq C alpha G (x n)
      ≤ (2 / (alpha * real N)) * (f (x 0) - f xstar)"
⟨proof⟩

lemma projected_gradient_descent_exists_small_residual_sq_to_minimizer:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
    and N_pos: "N > 0"
  shows
    "∃n<N.
      projected_gradient_residual_sq C alpha G (x n)
      ≤ (2 / (alpha * real N)) * (f (x 0) - f xstar)"
⟨proof⟩

lemma projected_gradient_descent_exists_epsilon_residual_to_minimizer_feasible:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
    and N_pos: "N > 0"
    and eps_nonneg: "0 ≤ eps"
    and horizon:
      "(2 / (alpha * real N)) * (f (x 0) - f xstar) ≤ eps ^ 2"
  shows
    "∃n<N. projected_gradient_residual C alpha G (x n) ≤ eps"
⟨proof⟩
```

**lemma** *projected_gradient_descent_exists_epsilon_residual_to_minimizer:*
**fixes** *f :: "'a::{real_inner,heine_borel} ⇒ real"*
**and** *G :: "'a ⇒ 'a"*
**assumes** *smooth: "smooth_convex_on L C f G"*
**and** *closed: "closed C"*
**and** *convex: "convex C"*
**and** *pgd: "projected_gradient_descent_iterates C alpha G x"*
**and** *x0_mem: "x 0 ∈ C"*
**and** *alpha_pos: "0 < alpha"*
**and** *step_size: "alpha * L ≤ 1"*
**and** *minimizer: "global_min_on C f xstar"*
**and** *N_pos: "N > 0"*
**and** *eps_nonneg: "0 ≤ eps"*
**and** *horizon:*
*"(2 / (alpha * real N)) * (f (x 0) - f xstar) ≤ eps ^ 2"*
**shows**
*"∃n<N. projected_gradient_residual C alpha G (x n) ≤ eps"*
⟨*proof*⟩

## 18.5 Product-form epsilon-stationarity complexity

The following variants are the main finite-horizon epsilon-stationarity certificates of the theory.

They use a product-form horizon assumption, which is often the most readable form in optimization statements. Instead of requiring a bound on a quotient, the assumption states directly that the horizon is large enough relative to the initial objective gap and eps squared.

**lemma** *projected_gradient_descent_exists_epsilon_residual_to_minimizer_product_feasible:*
**fixes** *f :: "'a::{real_inner,heine_borel} ⇒ real"*
**and** *G :: "'a ⇒ 'a"*
**assumes** *smooth: "smooth_convex_on L C f G"*
**and** *closed: "closed C"*
**and** *convex: "convex C"*
**and** *pgd: "projected_gradient_descent_iterates C alpha G x"*
**and** *feasible: "feasible_iterates C x"*
**and** *alpha_pos: "0 < alpha"*
**and** *step_size: "alpha * L ≤ 1"*
**and** *minimizer: "global_min_on C f xstar"*
**and** *N_pos: "N > 0"*
**and** *eps_pos: "0 < eps"*
**and** *horizon:*
*"2 * (f (x 0) - f xstar) ≤ alpha * real N * eps ^ 2"*
**shows**
*"∃n<N. projected_gradient_residual C alpha G (x n) ≤ eps"*
⟨*proof*⟩

**lemma** *projected_gradient_descent_exists_epsilon_residual_to_minimizer_product:*
**fixes** *f :: "'a::{real_inner,heine_borel} ⇒ real"*

```
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
    and N_pos: "N > 0"
    and eps_pos: "0 < eps"
    and horizon:
      "2 * (f (x 0) - f xstar) ≤ alpha * real N * eps ^ 2"
  shows
    "∃n<N. projected_gradient_residual C alpha G (x n) ≤ eps"
⟨proof⟩
```

### 18.6 Residual-zero consequences

A zero residual is exactly the projected-gradient fixed-point condition, and therefore gives the usual constrained first-order optimality certificate. These lemmas simply rephrase the existing projected-gradient mapping certificates using the residual notation introduced above.

```
lemma projected_gradient_residual_zero_iff_mapping_zero:
  "projected_gradient_residual C alpha G x = 0
    ⟷ projected_gradient_mapping C alpha G x = 0"
  ⟨proof⟩

lemma projected_gradient_residual_zero_iff_fixed_point:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes alpha_nonzero: "alpha ≠ 0"
  shows
    "projected_gradient_residual C alpha G x = 0
      ⟷ projected_gradient_step C alpha G x = x"
  ⟨proof⟩

lemma projected_gradient_residual_zero_iff_first_order_condition:
  fixes C :: "'a::{real_inner,heine_borel} set"
  assumes convex: "convex C"
    and closed: "closed C"
    and x_mem: "x ∈ C"
    and alpha_pos: "0 < alpha"
  shows
    "projected_gradient_residual C alpha G x = 0
      ⟷ first_order_condition_at C (G x) x"
  ⟨proof⟩

lemma projected_gradient_residual_zero_imp_global_min_on:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
```

```
    and G :: "'a ⇒ 'a"
  assumes smooth: "smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and x_mem: "x ∈ C"
    and alpha_pos: "0 < alpha"
    and zero: "projected_gradient_residual C alpha G x = 0"
  shows "global_min_on C f x"
⟨proof⟩
```

## 18.7 Recommended public residual-complexity interface

The following aliases give short, stable names to the main user-facing results of this theory. They are intended for use in the AFP document, README, and downstream developments.

The long theorem names above preserve the exact assumptions used in the proofs. The aliases below identify the main residual-complexity and optimality certificates exposed by this theory.

```
lemmas projected_gradient_descent_residual_sq_complexity =
  projected_gradient_descent_exists_small_residual_sq_to_minimizer

lemmas projected_gradient_descent_feasible_residual_sq_complexity =
  projected_gradient_descent_exists_small_residual_sq_to_minimizer_feasible

lemmas projected_gradient_descent_epsilon_stationarity_complexity =
  projected_gradient_descent_exists_epsilon_residual_to_minimizer_product

lemmas projected_gradient_descent_feasible_epsilon_stationarity_complexity
=
  projected_gradient_descent_exists_epsilon_residual_to_minimizer_product_feasible

lemmas projected_gradient_residual_fixed_point_certificate =
  projected_gradient_residual_zero_iff_fixed_point

lemmas projected_gradient_residual_optimality_certificate =
  projected_gradient_residual_zero_iff_first_order_condition

lemmas projected_gradient_residual_global_min_certificate =
  projected_gradient_residual_zero_imp_global_min_on

lemmas projected_gradient_residual_public_interface =
  projected_gradient_residual_eq_mapping_norm
  projected_gradient_residual_sq_eq_mapping_norm_sq
  projected_gradient_residual_sq_eq_residual_power2
  projected_gradient_residual_le_of_sq_le
  projected_gradient_residual_sq_le_of_residual_le
  projected_gradient_descent_residual_sq_complexity
  projected_gradient_descent_epsilon_stationarity_complexity
```

```
projected_gradient_residual_fixed_point_certificate
projected_gradient_residual_optimality_certificate
projected_gradient_residual_global_min_certificate
```

## 18.8 Locale form

```
context projected_gradient_descent
begin

lemma exists_small_residual_sq:
  assumes alpha_pos: "0 < alpha"
    and N_pos: "N > 0"
  shows
    "∃n<N.
      projected_gradient_residual_sq C alpha G (x n)
      ≤ (2 / (alpha * real N)) * (f (x 0) - f (x N))"
  ⟨proof⟩

lemma exists_small_residual_sq_below:
  assumes alpha_pos: "0 < alpha"
    and lower: "B ≤ f (x N)"
    and N_pos: "N > 0"
  shows
    "∃n<N.
      projected_gradient_residual_sq C alpha G (x n)
      ≤ (2 / (alpha * real N)) * (f (x 0) - B)"
  ⟨proof⟩

lemma exists_epsilon_residual:
  assumes alpha_pos: "0 < alpha"
    and N_pos: "N > 0"
    and eps_nonneg: "0 ≤ eps"
    and horizon:
      "(2 / (alpha * real N)) * (f (x 0) - f (x N)) ≤ eps ^ 2"
  shows
    "∃n<N. projected_gradient_residual C alpha G (x n) ≤ eps"
  ⟨proof⟩

lemma exists_epsilon_residual_below:
  assumes alpha_pos: "0 < alpha"
    and lower: "B ≤ f (x N)"
    and N_pos: "N > 0"
    and eps_nonneg: "0 ≤ eps"
    and horizon:
      "(2 / (alpha * real N)) * (f (x 0) - B) ≤ eps ^ 2"
  shows
    "∃n<N. projected_gradient_residual C alpha G (x n) ≤ eps"
  ⟨proof⟩
```

```
lemma exists_small_residual_sq_to_minimizer:
  assumes alpha_pos: "0 < alpha"
    and minimizer: "global_min_on C f xstar"
    and N_pos: "N > 0"
  shows
    "∃n<N.
      projected_gradient_residual_sq C alpha G (x n)
      ≤ (2 / (alpha * real N)) * (f (x 0) - f xstar)"
  ⟨proof⟩

lemma exists_epsilon_residual_to_minimizer:
  assumes alpha_pos: "0 < alpha"
    and minimizer: "global_min_on C f xstar"
    and N_pos: "N > 0"
    and eps_nonneg: "0 ≤ eps"
    and horizon:
      "(2 / (alpha * real N)) * (f (x 0) - f xstar) ≤ eps ^ 2"
  shows
    "∃n<N. projected_gradient_residual C alpha G (x n) ≤ eps"
  ⟨proof⟩

lemma exists_epsilon_residual_to_minimizer_product:
  assumes alpha_pos: "0 < alpha"
    and minimizer: "global_min_on C f xstar"
    and N_pos: "N > 0"
    and eps_pos: "0 < eps"
    and horizon:
      "2 * (f (x 0) - f xstar) ≤ alpha * real N * eps ^ 2"
  shows
    "∃n<N. projected_gradient_residual C alpha G (x n) ≤ eps"
  ⟨proof⟩

end
```

The main public theorem in this file is *projected_gradient_descent_epsilon_stationarity_comple*

It is an alias for *projected_gradient_descent_exists_epsilon_residual_to_minimizer_product*. It states that, if the horizon N is large enough in the usual product-form complexity bound, then one of the first N projected-gradient iterates has projected-gradient residual at most eps.

The theorem group *projected_gradient_residual_public_interface* collects the main residual notation, residual-complexity, and residual-optimality facts provided by this theory.

```
end
theory Strong_Convex
  imports Projected_Gradient_Mapping
begin
```

# 19 Strong convexity for smooth first-order methods

This theory introduces a first-order lower-bound interface for strong convexity.

The main interface is *f y >= f x + inner (G x) (y - x) + (mu / 2) * norm (y - x) ^ 2*. This formulation is well suited to the existing development because it uses the same named gradient field *G* as the smooth-convex and projected-gradient layers.

The file proves three kinds of consequences: strong convexity implies the ordinary convex first-order lower bound; at a first-order optimal point, the function-value gap controls squared distance; under positive strong convexity, global minimizers are unique.

These results are intended to feed into later linear-convergence proofs for gradient descent and projected gradient descent.

## 19.1 First-order strong convexity lower bounds

```
definition strong_convex_lower_bound_on ::
  "real ⇒ 'a::real_inner set ⇒ ('a ⇒ real) ⇒ ('a ⇒ 'a) ⇒ bool"
where
  "strong_convex_lower_bound_on mu S f G ⟷
     0 ≤ mu ∧
     (∀x∈S. ∀y∈S.
        f x + inner (G x) (y - x) + (mu / 2) * norm (y - x) ^ 2 ≤ f y)"

lemma strong_convex_lower_bound_onI:
  assumes "0 ≤ mu"
    and "⋀x y. x ∈ S ⟹ y ∈ S ⟹
      f x + inner (G x) (y - x) + (mu / 2) * norm (y - x) ^ 2 ≤ f y"
  shows "strong_convex_lower_bound_on mu S f G"
  ⟨proof⟩

lemma strong_convex_lower_bound_onD_nonneg:
  assumes "strong_convex_lower_bound_on mu S f G"
  shows "0 ≤ mu"
  ⟨proof⟩

lemma strong_convex_lower_bound_onD:
  assumes "strong_convex_lower_bound_on mu S f G"
    and "x ∈ S"
    and "y ∈ S"
  shows
    "f x + inner (G x) (y - x) + (mu / 2) * norm (y - x) ^ 2 ≤ f y"
  ⟨proof⟩

lemma strong_convex_lower_bound_on_subset:
```

**assumes** *strong: "strong_convex_lower_bound_on mu T f G"*
**and** *subset: "S ⊆ T"*
**shows** *"strong_convex_lower_bound_on mu S f G"*
⟨*proof*⟩

**lemma** *strong_convex_lower_bound_on_mono_mu:*
**assumes** *strong: "strong_convex_lower_bound_on mu S f G"*
**and** *nu_nonneg: "0 ≤ nu"*
**and** *nu_le_mu: "nu ≤ mu"*
**shows** *"strong_convex_lower_bound_on nu S f G"*
⟨*proof*⟩

**lemma** *strong_convex_lower_bound_on_imp_gradient_lower_bound_on:*
**assumes** *strong: "strong_convex_lower_bound_on mu S f G"*
**shows** *"gradient_lower_bound_on S f G"*
⟨*proof*⟩

**lemma** *strong_convex_lower_bound_on_imp_supports_on:*
**assumes** *strong: "strong_convex_lower_bound_on mu S f G"*
**shows** *"supports_on S f G"*
⟨*proof*⟩

## 19.2 Strongly convex differentiable functions

**definition** *strongly_convex_differentiable_on ::*
*"real ⇒ 'a::real_inner set ⇒ ('a ⇒ real) ⇒ ('a ⇒ 'a) ⇒ bool"*
**where**
*"strongly_convex_differentiable_on mu S f G ⟷*
*convex_differentiable_on S f G ∧ strong_convex_lower_bound_on mu*
*S f G"*

**lemma** *strongly_convex_differentiable_onI:*
**assumes** *"convex_differentiable_on S f G"*
**and** *"strong_convex_lower_bound_on mu S f G"*
**shows** *"strongly_convex_differentiable_on mu S f G"*
⟨*proof*⟩

**lemma** *strongly_convex_differentiable_onD_convex_differentiable:*
**assumes** *"strongly_convex_differentiable_on mu S f G"*
**shows** *"convex_differentiable_on S f G"*
⟨*proof*⟩

**lemma** *strongly_convex_differentiable_onD_strong:*
**assumes** *"strongly_convex_differentiable_on mu S f G"*
**shows** *"strong_convex_lower_bound_on mu S f G"*
⟨*proof*⟩

**lemma** *strongly_convex_differentiable_onD_nonneg:*
**assumes** *"strongly_convex_differentiable_on mu S f G"*

**shows** *"0 ≤ mu"*
⟨*proof*⟩

**lemma** *strongly_convex_differentiable_onD_convex_on:*
**assumes** *"strongly_convex_differentiable_on mu S f G"*
**shows** *"convex_on S f"*
⟨*proof*⟩

**lemma** *strongly_convex_differentiable_onD_has_gradient_on:*
**assumes** *"strongly_convex_differentiable_on mu S f G"*
**shows** *"has_gradient_on f S G"*
⟨*proof*⟩

**lemma** *strongly_convex_differentiable_on_subset:*
**assumes** *strong: "strongly_convex_differentiable_on mu T f G"*
**and** *subset: "S ⊆ T"*
**and** *convex: "convex S"*
**shows** *"strongly_convex_differentiable_on mu S f G"*
⟨*proof*⟩

**lemma** *strongly_convex_differentiable_on_imp_gradient_lower_bound_on:*
**assumes** *strong: "strongly_convex_differentiable_on mu S f G"*
**shows** *"gradient_lower_bound_on S f G"*
⟨*proof*⟩

**lemma** *strongly_convex_differentiable_on_imp_supports_on:*
**assumes** *strong: "strongly_convex_differentiable_on mu S f G"*
**shows** *"supports_on S f G"*
⟨*proof*⟩

## 19.3 Strongly smooth convex functions

**definition** *strongly_smooth_convex_on ::*
*"real ⇒ real ⇒ 'a::real_inner set ⇒ ('a ⇒ real) ⇒ ('a ⇒ 'a) ⇒ bool"*
**where**
*"strongly_smooth_convex_on L mu S f G ⟷*
*smooth_convex_on L S f G ∧ strong_convex_lower_bound_on mu S f G"*

**lemma** *strongly_smooth_convex_onI:*
**assumes** *"smooth_convex_on L S f G"*
**and** *"strong_convex_lower_bound_on mu S f G"*
**shows** *"strongly_smooth_convex_on L mu S f G"*
⟨*proof*⟩

**lemma** *strongly_smooth_convex_onD_smooth:*
**assumes** *"strongly_smooth_convex_on L mu S f G"*
**shows** *"smooth_convex_on L S f G"*
⟨*proof*⟩

**lemma** *strongly_smooth_convex_onD_strong:*
  **assumes** *"strongly_smooth_convex_on L mu S f G"*
  **shows** *"strong_convex_lower_bound_on mu S f G"*
  ⟨*proof*⟩

**lemma** *strongly_smooth_convex_onD_convex_differentiable:*
  **assumes** *"strongly_smooth_convex_on L mu S f G"*
  **shows** *"convex_differentiable_on S f G"*
  ⟨*proof*⟩

**lemma** *strongly_smooth_convex_onD_smooth_upper_bound:*
  **assumes** *"strongly_smooth_convex_on L mu S f G"*
  **shows** *"smooth_upper_bound_on L S f G"*
  ⟨*proof*⟩

**lemma** *strongly_smooth_convex_onD_strongly_convex_differentiable:*
  **assumes** *"strongly_smooth_convex_on L mu S f G"*
  **shows** *"strongly_convex_differentiable_on mu S f G"*
⟨*proof*⟩

**lemma** *strongly_smooth_convex_on_subset:*
  **assumes** *strong: "strongly_smooth_convex_on L mu T f G"*
    **and** *subset: "S ⊆ T"*
    **and** *convex: "convex S"*
  **shows** *"strongly_smooth_convex_on L mu S f G"*
⟨*proof*⟩

## 19.4 Global minimizers and first-order conditions

**lemma** *global_min_on_value_eq:*
  **assumes** *min_x: "global_min_on S f x"*
    **and** *min_y: "global_min_on S f y"*
  **shows** *"f x = f y"*
⟨*proof*⟩

**lemma** *global_min_on_has_gradient_imp_first_order_condition:*
  **fixes** *f :: "'a::real_inner ⇒ real"*
  **assumes** *convex: "convex S"*
    **and** *minimizer: "global_min_on S f x"*
    **and** *grad: "has_gradient f x g"*
  **shows** *"first_order_condition_at S g x"*
⟨*proof*⟩

**lemma** *convex_differentiable_on_global_min_imp_first_order_condition:*
  **assumes** *cd: "convex_differentiable_on S f G"*
    **and** *minimizer: "global_min_on S f x"*
  **shows** *"first_order_condition_at S (G x) x"*
⟨*proof*⟩

## 19.5 Distance-gap lower bounds

**lemma** *strong_convex_foc_distance_gap:*
  **fixes** *f :: "'a::real_inner ⇒ real"*
  **assumes** *strong: "strong_convex_lower_bound_on mu S f G"*
    **and** *foc: "first_order_condition_at S (G xstar) xstar"*
    **and** *x_mem: "x ∈ S"*
  **shows**
    *"(mu / 2) * norm (x - xstar) ^ 2 ≤ f x - f xstar"*
⟨*proof*⟩

**lemma** *strongly_convex_global_min_distance_gap:*
  **fixes** *f :: "'a::real_inner ⇒ real"*
  **assumes** *strong: "strongly_convex_differentiable_on mu S f G"*
    **and** *minimizer: "global_min_on S f xstar"*
    **and** *x_mem: "x ∈ S"*
  **shows**
    *"(mu / 2) * norm (x - xstar) ^ 2 ≤ f x - f xstar"*
⟨*proof*⟩

**lemma** *strongly_smooth_convex_global_min_distance_gap:*
  **fixes** *f :: "'a::real_inner ⇒ real"*
  **assumes** *strong: "strongly_smooth_convex_on L mu S f G"*
    **and** *minimizer: "global_min_on S f xstar"*
    **and** *x_mem: "x ∈ S"*
  **shows**
    *"(mu / 2) * norm (x - xstar) ^ 2 ≤ f x - f xstar"*
⟨*proof*⟩

**lemma** *strong_convex_foc_distance_gap_nonnegative:*
  **fixes** *f :: "'a::real_inner ⇒ real"*
  **assumes** *strong: "strong_convex_lower_bound_on mu S f G"*
    **and** *foc: "first_order_condition_at S (G xstar) xstar"*
    **and** *x_mem: "x ∈ S"*
  **shows** *"0 ≤ f x - f xstar"*
⟨*proof*⟩

## 19.6 Uniqueness of minimizers

**lemma** *strongly_convex_global_min_unique:*
  **fixes** *f :: "'a::real_inner ⇒ real"*
  **assumes** *strong: "strongly_convex_differentiable_on mu S f G"*
    **and** *mu_pos: "0 < mu"*
    **and** *min_x: "global_min_on S f x"*
    **and** *min_y: "global_min_on S f y"*
  **shows** *"x = y"*
⟨*proof*⟩

**lemma** *strongly_smooth_convex_global_min_unique:*
  **fixes** *f :: "'a::real_inner ⇒ real"*

```
  assumes strong: "strongly_smooth_convex_on L mu S f G"
    and mu_pos: "0 < mu"
    and min_x: "global_min_on S f x"
    and min_y: "global_min_on S f y"
  shows "x = y"
⟨proof⟩
```

## 19.7 Consequences for projected-gradient optimality

```
lemma strongly_smooth_projected_mapping_zero_distance_gap:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
  assumes strong: "strongly_smooth_convex_on L mu C f G"
    and closed: "closed C"
    and convex: "convex C"
    and xstar_mem: "xstar ∈ C"
    and alpha_pos: "0 < alpha"
    and zero: "projected_gradient_mapping C alpha G xstar = 0"
    and x_mem: "x ∈ C"
  shows
    "(mu / 2) * norm (x - xstar) ^ 2 ≤ f x - f xstar"
⟨proof⟩
```

```
lemma strongly_smooth_projected_mapping_zero_unique_global_min:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
  assumes strong: "strongly_smooth_convex_on L mu C f G"
    and mu_pos: "0 < mu"
    and closed: "closed C"
    and convex: "convex C"
    and x_mem: "x ∈ C"
    and alpha_pos: "0 < alpha"
    and zero: "projected_gradient_mapping C alpha G x = 0"
    and minimizer: "global_min_on C f y"
  shows "x = y"
⟨proof⟩
```

## 19.8 Locale form: strongly convex differentiable functions

```
locale strongly_convex_differentiable =
  convex_differentiable S f G
  for S :: "'a::real_inner set"
    and f :: "'a ⇒ real"
    and G :: "'a ⇒ 'a" +
  fixes mu :: real
  assumes strong_lower_bound: "strong_convex_lower_bound_on mu S f G"
begin
```

```
lemma strong_nonneg:
  "0 ≤ mu"
  ⟨proof⟩
```

```
lemma strong_lower:
  assumes "x ∈ S"
    and "y ∈ S"
  shows
    "f x + inner (G x) (y - x) + (mu / 2) * norm (y - x) ^ 2 ≤ f y"
  ⟨proof⟩

lemma strongly_convex_differentiable_on_self:
  "strongly_convex_differentiable_on mu S f G"
⟨proof⟩

lemma gradient_lower_bound:
  "gradient_lower_bound_on S f G"
  ⟨proof⟩

lemma supports:
  "supports_on S f G"
  ⟨proof⟩

lemma global_min_imp_first_order_condition:
  assumes minimizer: "global_min_on S f x"
  shows "first_order_condition_at S (G x) x"
⟨proof⟩

lemma foc_distance_gap:
  assumes foc: "first_order_condition_at S (G xstar) xstar"
    and x_mem: "x ∈ S"
  shows
    "(mu / 2) * norm (x - xstar) ^ 2 ≤ f x - f xstar"
  ⟨proof⟩

lemma global_min_distance_gap:
  assumes minimizer: "global_min_on S f xstar"
    and x_mem: "x ∈ S"
  shows
    "(mu / 2) * norm (x - xstar) ^ 2 ≤ f x - f xstar"
  ⟨proof⟩

lemma global_min_unique:
  assumes mu_pos: "0 < mu"
    and min_x: "global_min_on S f x"
    and min_y: "global_min_on S f y"
  shows "x = y"
  ⟨proof⟩

end
```

## 19.9 Locale form: strongly smooth convex functions

```
locale strongly_smooth_convex =
  smooth_convex S f G L
  for S :: "'a::real_inner set"
    and f :: "'a ⇒ real"
    and G :: "'a ⇒ 'a"
    and L :: real +
  fixes mu :: real
  assumes strong_lower_bound: "strong_convex_lower_bound_on mu S f G"
begin

lemma strong_nonneg:
  "0 ≤ mu"
  ⟨proof⟩

lemma strongly_smooth_convex_on_self:
  "strongly_smooth_convex_on L mu S f G"
⟨proof⟩

lemma strongly_convex_differentiable_on_self:
  "strongly_convex_differentiable_on mu S f G"
  ⟨proof⟩

lemma strong_lower:
  assumes "x ∈ S"
    and "y ∈ S"
  shows
    "f x + inner (G x) (y - x) + (mu / 2) * norm (y - x) ^ 2 ≤ f y"
  ⟨proof⟩

lemma global_min_distance_gap:
  assumes minimizer: "global_min_on S f xstar"
    and x_mem: "x ∈ S"
  shows
    "(mu / 2) * norm (x - xstar) ^ 2 ≤ f x - f xstar"
  ⟨proof⟩

lemma global_min_unique:
  assumes mu_pos: "0 < mu"
    and min_x: "global_min_on S f x"
    and min_y: "global_min_on S f y"
  shows "x = y"
  ⟨proof⟩

end
```

The main reusable theorem for later convergence arguments is ⟦*strongly_smooth_convex_on ?L ?mu ?S ?f ?G; global_min_on ?S ?f ?xstar; ?x ∈ ?S*⟧ ⟹ *?mu / 2 * (norm (?x - ?xstar))*$^2$ ≤ *?f ?x - ?f ?xstar*. It says that, for a strongly smooth convex objective, the function-value gap to a global minimizer controls the squared distance to that minimizer.

This is the key ingredient needed to turn the existing O(1/N) convergence arguments into linear-convergence arguments under positive strong convexity.

**end**
**theory** *Projected_Gradient_Descent_Linear_Rate*
**imports**
*Strong_Convex*
*Abstract_Descent*
**begin**

# 20 Linear convergence of projected gradient descent

This theory proves a robust linear convergence estimate for projected gradient descent under a first-order strong-convexity lower-bound assumption.

The proof uses the projected one-step distance inequality from the projected gradient descent development and the strong-convexity distance-gap lower bound.

The resulting contraction is *(1 + alpha * mu) * norm (x (Suc n) - xstar) ^ 2 <= norm (x n - xstar) ^ 2*. Equivalently, with *q = 1 / (1 + alpha * mu)*, we obtain *norm (x N - xstar) ^ 2 <= q^N * norm (x 0 - xstar) ^ 2*.

This is not intended to be the sharpest possible textbook contraction factor. Its purpose is to provide a stable Isabelle-friendly linear rate that follows directly from the existing projected one-step inequality.

## 20.1 The linear-rate contraction factor

```
definition projected_gradient_linear_rate_factor :: "real ⇒ real ⇒ real"
where
  "projected_gradient_linear_rate_factor alpha mu =
     inverse (1 + alpha * mu)"

lemma projected_gradient_linear_rate_denominator_pos:
  fixes alpha mu :: real
  assumes alpha_nonneg: "0 ≤ alpha"
    and mu_nonneg: "0 ≤ mu"
  shows "0 < 1 + alpha * mu"
⟨proof⟩

lemma projected_gradient_linear_rate_factor_nonnegative:
  assumes alpha_nonneg: "0 ≤ alpha"
    and mu_nonneg: "0 ≤ mu"
  shows "0 ≤ projected_gradient_linear_rate_factor alpha mu"
⟨proof⟩
```

```
lemma projected_gradient_linear_rate_factor_positive:
  assumes alpha_nonneg: "0 ≤ alpha"
    and mu_nonneg: "0 ≤ mu"
  shows "0 < projected_gradient_linear_rate_factor alpha mu"
⟨proof⟩

lemma projected_gradient_linear_rate_factor_le_one:
  assumes alpha_nonneg: "0 ≤ alpha"
    and mu_nonneg: "0 ≤ mu"
  shows "projected_gradient_linear_rate_factor alpha mu ≤ 1"
⟨proof⟩

lemma projected_gradient_linear_rate_factor_lt_one:
  assumes alpha_pos: "0 < alpha"
    and mu_pos: "0 < mu"
  shows "projected_gradient_linear_rate_factor alpha mu < 1"
⟨proof⟩
```

## 20.2 One-step distance contraction

```
lemma projected_gradient_descent_strong_one_step_distance_contract:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes strong: "strongly_smooth_convex_on L mu C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
  shows
    "(1 + alpha * mu) * norm (x (Suc n) - xstar) ^ 2
      ≤ norm (x n - xstar) ^ 2"
⟨proof⟩

lemma projected_gradient_descent_strong_one_step_distance_contract_factor:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes strong: "strongly_smooth_convex_on L mu C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
  shows
```

```
    "norm (x (Suc n) - xstar) ^ 2
      ≤ projected_gradient_linear_rate_factor alpha mu
          * norm (x n - xstar) ^ 2"
```
⟨*proof*⟩

## 20.3 Distance linear convergence

**lemma** *projected_gradient_descent_distance_sq_linear_rate_feasible:*
```
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes strong: "strongly_smooth_convex_on L mu C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
  shows
    "norm (x N - xstar) ^ 2
      ≤ projected_gradient_linear_rate_factor alpha mu ^ N
          * norm (x 0 - xstar) ^ 2"
```
⟨*proof*⟩

**lemma** *projected_gradient_descent_distance_sq_linear_rate:*
```
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes strong: "strongly_smooth_convex_on L mu C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
  shows
    "norm (x N - xstar) ^ 2
      ≤ projected_gradient_linear_rate_factor alpha mu ^ N
          * norm (x 0 - xstar) ^ 2"
```
⟨*proof*⟩

## 20.4 Function-value linear convergence

**lemma** *projected_gradient_descent_function_value_linear_rate_Suc_feasible:*
```
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes strong: "strongly_smooth_convex_on L mu C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
```

```
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
  shows
    "f (x (Suc N)) - f xstar
      ≤ projected_gradient_linear_rate_factor alpha mu ^ N
          * norm (x 0 - xstar) ^ 2 / (2 * alpha)"
⟨proof⟩

lemma projected_gradient_descent_function_value_linear_rate_Suc:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes strong: "strongly_smooth_convex_on L mu C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
  shows
    "f (x (Suc N)) - f xstar
      ≤ projected_gradient_linear_rate_factor alpha mu ^ N
          * norm (x 0 - xstar) ^ 2 / (2 * alpha)"
⟨proof⟩

lemma projected_gradient_descent_function_value_linear_rate_feasible:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes strong: "strongly_smooth_convex_on L mu C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and feasible: "feasible_iterates C x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
    and N_pos: "N > 0"
  shows
    "f (x N) - f xstar
      ≤ projected_gradient_linear_rate_factor alpha mu ^ (N - 1)
          * norm (x 0 - xstar) ^ 2 / (2 * alpha)"
⟨proof⟩

lemma projected_gradient_descent_function_value_linear_rate:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes strong: "strongly_smooth_convex_on L mu C f G"
```

```
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
    and N_pos: "N > 0"
  shows
    "f (x N) - f xstar
      ≤ projected_gradient_linear_rate_factor alpha mu ^ (N - 1)
          * norm (x 0 - xstar) ^ 2 / (2 * alpha)"
⟨proof⟩
```

## 20.5 Strict contraction under positive strong convexity

```
lemma projected_gradient_descent_strict_rate_factor:
  assumes alpha_pos: "0 < alpha"
    and mu_pos: "0 < mu"
  shows "projected_gradient_linear_rate_factor alpha mu < 1"
  ⟨proof⟩

lemma projected_gradient_descent_rate_factor_nonnegative_from_strong:
  assumes strong: "strongly_smooth_convex_on L mu C f G"
    and alpha_pos: "0 < alpha"
  shows "0 ≤ projected_gradient_linear_rate_factor alpha mu"
⟨proof⟩
```

## 20.6 Locale form

```
locale projected_gradient_descent_linear_rate =
  projected_gradient_descent C L alpha f G x
  for C :: "'a::{real_inner,heine_borel} set"
    and L alpha :: real
    and f :: "'a ⇒ real"
    and G :: "'a ⇒ 'a"
    and x :: "nat ⇒ 'a" +
  fixes mu :: real
  assumes strong_lower_bound: "strong_convex_lower_bound_on mu C f G"
begin

lemma strongly_smooth_convex_on_self:
  "strongly_smooth_convex_on L mu C f G"
⟨proof⟩

lemma strong_nonneg:
  "0 ≤ mu"
  ⟨proof⟩

lemma rate_factor_nonnegative:
```

```
  assumes alpha_pos: "0 < alpha"
  shows "0 ≤ projected_gradient_linear_rate_factor alpha mu"
⟨proof⟩

lemma rate_factor_positive:
  assumes alpha_pos: "0 < alpha"
  shows "0 < projected_gradient_linear_rate_factor alpha mu"
⟨proof⟩

lemma rate_factor_le_one:
  assumes alpha_pos: "0 < alpha"
  shows "projected_gradient_linear_rate_factor alpha mu ≤ 1"
⟨proof⟩

lemma rate_factor_lt_one:
  assumes alpha_pos: "0 < alpha"
    and mu_pos: "0 < mu"
  shows "projected_gradient_linear_rate_factor alpha mu < 1"
  ⟨proof⟩

lemma one_step_distance_contract:
  assumes alpha_pos: "0 < alpha"
    and minimizer: "global_min_on C f xstar"
  shows
    "(1 + alpha * mu) * norm (x (Suc n) - xstar) ^ 2
      ≤ norm (x n - xstar) ^ 2"
  ⟨proof⟩

lemma one_step_distance_contract_factor:
  assumes alpha_pos: "0 < alpha"
    and minimizer: "global_min_on C f xstar"
  shows
    "norm (x (Suc n) - xstar) ^ 2
      ≤ projected_gradient_linear_rate_factor alpha mu
          * norm (x n - xstar) ^ 2"
  ⟨proof⟩

lemma distance_sq_linear_rate:
  assumes alpha_pos: "0 < alpha"
    and minimizer: "global_min_on C f xstar"
  shows
    "norm (x N - xstar) ^ 2
      ≤ projected_gradient_linear_rate_factor alpha mu ^ N
          * norm (x 0 - xstar) ^ 2"
  ⟨proof⟩

lemma function_value_linear_rate_Suc:
  assumes alpha_pos: "0 < alpha"
    and minimizer: "global_min_on C f xstar"
```

**shows**
*"f (x (Suc N)) - f xstar*
*≤ projected_gradient_linear_rate_factor alpha mu ^ N*
** norm (x 0 - xstar) ^ 2 / (2 * alpha)"*
⟨*proof*⟩

**lemma** *function_value_linear_rate:*
**assumes** *alpha_pos: "0 < alpha"*
**and** *minimizer: "global_min_on C f xstar"*
**and** *N_pos: "N > 0"*
**shows**
*"f (x N) - f xstar*
*≤ projected_gradient_linear_rate_factor alpha mu ^ (N - 1)*
** norm (x 0 - xstar) ^ 2 / (2 * alpha)"*
⟨*proof*⟩

**end**

The main distance-rate theorem is ⟦*strongly_smooth_convex_on ?L ?mu ?C ?f ?G; closed ?C; convex ?C; projected_gradient_descent_iterates ?C ?alpha ?G ?x; ?x 0 ∈ ?C; 0 < ?alpha; ?alpha * ?L ≤ 1; global_min_on ?C ?f ?xstar*⟧ ⟹ *(norm (?x ?N - ?xstar))*$^2$ ≤ *(projected_gradient_linear_rate_factor ?alpha ?mu)*$^{?N}$ ** (norm (?x 0 - ?xstar))*$^2$. It gives an exponential decay estimate for squared distance to a global minimizer, with factor *projected_gradient_linear_rate_fact*

The main function-value version is ⟦*strongly_smooth_convex_on ?L ?mu ?C ?f ?G; closed ?C; convex ?C; projected_gradient_descent_iterates ?C ?alpha ?G ?x; ?x 0 ∈ ?C; 0 < ?alpha; ?alpha * ?L ≤ 1; global_min_on ?C ?f ?xstar*⟧ ⟹ *?f (?x (Suc ?N)) - ?f ?xstar* ≤ *(projected_gradient_linear_rate_factor ?alpha ?mu)*$^{?N}$ ** (norm (?x 0 - ?xstar))*$^2$ */ (2 * ?alpha)*. It bounds the function-value gap at the next iterate by the same linear factor applied to the initial squared distance.

**end**
**theory** *Lipschitz_Smoothness*
**imports** *Projected_Gradient_Descent_Linear_Rate*
**begin**

# 21 Lipschitz gradients and smoothness interfaces

This theory connects the reusable smooth quadratic upper-bound interface with a more primitive Lipschitz-gradient assumption.

The main bridge is the standard descent lemma along line segments: *f y <= f x + inner (G x) (y - x) + (L / 2) * norm (y - x) ^ 2*. The theory packages this bridge so that later convergence proofs can use the same smoothness interface regardless of whether smoothness is assumed directly or derived from a Lipschitz-gradient condition.

## 21.1 Line-segment descent-lemma certificates

```
definition line_descent_bound_on ::
  "real ⇒ 'a::real_inner set ⇒ ('a ⇒ real) ⇒ ('a ⇒ 'a) ⇒ bool"
where
  "line_descent_bound_on L S f G ⟷
     0 ≤ L ∧
     (∀x∈S. ∀y∈S.
        f y ≤ f x + inner (G x) (y - x) + (L / 2) * norm (y - x) ^ 2)"

lemma line_descent_bound_onI:
  assumes "0 ≤ L"
    and "⋀x y. x ∈ S ⟹ y ∈ S ⟹
      f y ≤ f x + inner (G x) (y - x) + (L / 2) * norm (y - x) ^ 2"
  shows "line_descent_bound_on L S f G"
  ⟨proof⟩

lemma line_descent_bound_onD_nonneg:
  assumes "line_descent_bound_on L S f G"
  shows "0 ≤ L"
  ⟨proof⟩

lemma line_descent_bound_onD:
  assumes "line_descent_bound_on L S f G"
    and "x ∈ S"
    and "y ∈ S"
  shows "f y ≤ f x + inner (G x) (y - x) + (L / 2) * norm (y - x) ^ 2"
  ⟨proof⟩

lemma line_descent_bound_on_imp_smooth_upper_bound_on:
  assumes line: "line_descent_bound_on L S f G"
  shows "smooth_upper_bound_on L S f G"
⟨proof⟩

lemma smooth_upper_bound_on_imp_line_descent_bound_on:
  assumes smooth: "smooth_upper_bound_on L S f G"
  shows "line_descent_bound_on L S f G"
⟨proof⟩

lemma line_descent_bound_on_iff_smooth_upper_bound_on:
  "line_descent_bound_on L S f G ⟷ smooth_upper_bound_on L S f G"
⟨proof⟩

lemma line_descent_bound_on_subset:
  assumes line: "line_descent_bound_on L T f G"
    and subset: "S ⊆ T"
  shows "line_descent_bound_on L S f G"
⟨proof⟩

lemma line_descent_bound_on_mono_L:
```

**assumes** *line: "line_descent_bound_on L S f G"*
**and** *LM: "L ≤ M"*
**shows** *"line_descent_bound_on M S f G"*
⟨*proof*⟩

## 21.2 Lipschitz-supported smoothness

**definition** *lipschitz_smooth_on ::*
*"real ⇒ 'a::real_inner set ⇒ ('a ⇒ real) ⇒ ('a ⇒ 'a) ⇒ bool"*
**where**
*"lipschitz_smooth_on L S f G ⟷*
*has_gradient_on f S G ∧*
*lipschitz_gradient_on L S G ∧*
*line_descent_bound_on L S f G"*

**lemma** *lipschitz_smooth_onI:*
**assumes** *"has_gradient_on f S G"*
**and** *"lipschitz_gradient_on L S G"*
**and** *"line_descent_bound_on L S f G"*
**shows** *"lipschitz_smooth_on L S f G"*
⟨*proof*⟩

**lemma** *lipschitz_smooth_onD_has_gradient_on:*
**assumes** *"lipschitz_smooth_on L S f G"*
**shows** *"has_gradient_on f S G"*
⟨*proof*⟩

**lemma** *lipschitz_smooth_onD_lipschitz_gradient:*
**assumes** *"lipschitz_smooth_on L S f G"*
**shows** *"lipschitz_gradient_on L S G"*
⟨*proof*⟩

**lemma** *lipschitz_smooth_onD_line_descent:*
**assumes** *"lipschitz_smooth_on L S f G"*
**shows** *"line_descent_bound_on L S f G"*
⟨*proof*⟩

**lemma** *lipschitz_smooth_onD_smooth_upper_bound:*
**assumes** *"lipschitz_smooth_on L S f G"*
**shows** *"smooth_upper_bound_on L S f G"*
⟨*proof*⟩

**lemma** *lipschitz_smooth_onD_nonneg:*
**assumes** *"lipschitz_smooth_on L S f G"*
**shows** *"0 ≤ L"*
⟨*proof*⟩

**lemma** *lipschitz_smooth_on_subset:*
**assumes** *smooth: "lipschitz_smooth_on L T f G"*

**and** *subset: "S ⊆ T"*
**shows** *"lipschitz_smooth_on L S f G"*
⟨*proof*⟩

**lemma** *lipschitz_smooth_on_mono_L:*
**assumes** *smooth: "lipschitz_smooth_on L S f G"*
**and** *LM: "L ≤ M"*
**shows** *"lipschitz_smooth_on M S f G"*
⟨*proof*⟩

## 21.3 Connection with smooth convexity

**definition** *lipschitz_smooth_convex_on ::*
*"real ⇒ 'a::real_inner set ⇒ ('a ⇒ real) ⇒ ('a ⇒ 'a) ⇒ bool"*
**where**
*"lipschitz_smooth_convex_on L S f G ⟷*
*convex_on S f ∧ lipschitz_smooth_on L S f G"*

**lemma** *lipschitz_smooth_convex_onI:*
**assumes** *"convex_on S f"*
**and** *"lipschitz_smooth_on L S f G"*
**shows** *"lipschitz_smooth_convex_on L S f G"*
⟨*proof*⟩

**lemma** *lipschitz_smooth_convex_onD_convex_on:*
**assumes** *"lipschitz_smooth_convex_on L S f G"*
**shows** *"convex_on S f"*
⟨*proof*⟩

**lemma** *lipschitz_smooth_convex_onD_lipschitz_smooth:*
**assumes** *"lipschitz_smooth_convex_on L S f G"*
**shows** *"lipschitz_smooth_on L S f G"*
⟨*proof*⟩

**lemma** *lipschitz_smooth_convex_onD_has_gradient_on:*
**assumes** *"lipschitz_smooth_convex_on L S f G"*
**shows** *"has_gradient_on f S G"*
⟨*proof*⟩

**lemma** *lipschitz_smooth_convex_onD_lipschitz_gradient:*
**assumes** *"lipschitz_smooth_convex_on L S f G"*
**shows** *"lipschitz_gradient_on L S G"*
⟨*proof*⟩

**lemma** *lipschitz_smooth_convex_onD_smooth_upper_bound:*
**assumes** *"lipschitz_smooth_convex_on L S f G"*
**shows** *"smooth_upper_bound_on L S f G"*
⟨*proof*⟩

**lemma** *lipschitz_smooth_convex_on_imp_convex_differentiable_on:*
  **assumes** *smooth: "lipschitz_smooth_convex_on L S f G"*
  **shows** *"convex_differentiable_on S f G"*
⟨*proof*⟩

**lemma** *lipschitz_smooth_convex_on_imp_smooth_convex_on:*
  **assumes** *smooth: "lipschitz_smooth_convex_on L S f G"*
  **shows** *"smooth_convex_on L S f G"*
⟨*proof*⟩

**lemma** *smooth_convex_on_and_lipschitz_gradient_imp_lipschitz_smooth_convex_on:*
  **assumes** *smooth: "smooth_convex_on L S f G"*
    **and** *lip: "lipschitz_gradient_on L S G"*
  **shows** *"lipschitz_smooth_convex_on L S f G"*
⟨*proof*⟩

## 21.4 Elementary consequences of Lipschitz gradients

**lemma** *lipschitz_gradient_on_dist:*
  **fixes** *G :: "'a::real_inner ⇒ 'a"*
  **assumes** *lip: "lipschitz_gradient_on L S G"*
    **and** *x: "x ∈ S"*
    **and** *y: "y ∈ S"*
  **shows** *"norm (G y - G x) ≤ L * norm (y - x)"*
⟨*proof*⟩

**lemma** *lipschitz_gradient_on_inner_difference_bound:*
  **fixes** *G :: "'a::real_inner ⇒ 'a"*
  **assumes** *lip: "lipschitz_gradient_on L S G"*
    **and** *x: "x ∈ S"*
    **and** *y: "y ∈ S"*
  **shows** *"inner (G y - G x) (y - x) ≤ L * norm (y - x) ^ 2"*
⟨*proof*⟩

**lemma** *lipschitz_gradient_on_inner_difference_abs_bound:*
  **fixes** *G :: "'a::real_inner ⇒ 'a"*
  **assumes** *lip: "lipschitz_gradient_on L S G"*
    **and** *x: "x ∈ S"*
    **and** *y: "y ∈ S"*
  **shows** *"abs (inner (G y - G x) (y - x)) ≤ L * norm (y - x) ^ 2"*
⟨*proof*⟩

**lemma** *lipschitz_gradient_on_segment_dist:*
  **fixes** *G :: "'a::real_inner ⇒ 'a"*
  **assumes** *lip: "lipschitz_gradient_on L S G"*
    **and** *convex: "convex S"*
    **and** *x: "x ∈ S"*
    **and** *y: "y ∈ S"*
    **and** *t_nonneg: "0 ≤ t"*

```
    and t_le: "t ≤ 1"
  shows
    "norm (G (x + t *R (y - x)) - G x)
      ≤ L * t * norm (y - x)"
⟨proof⟩

lemma lipschitz_gradient_on_segment_inner_bound:
  fixes G :: "'a::real_inner ⇒ 'a"
  assumes lip: "lipschitz_gradient_on L S G"
    and convex: "convex S"
    and x: "x ∈ S"
    and y: "y ∈ S"
    and t_nonneg: "0 ≤ t"
    and t_le: "t ≤ 1"
  shows
    "inner (G (x + t *R (y - x)) - G x) (y - x)
      ≤ L * t * norm (y - x) ^ 2"
⟨proof⟩
```

## 21.5 A mean-value bridge from Lipschitz gradients

The following interface records the one-dimensional mean-value identity along each feasible line segment. It is intentionally separated from `has_gradient_on`: different developments may prove this certificate from their preferred differentiability infrastructure.

Together with a Lipschitz gradient, this certificate gives a quadratic upper-bound with constant 2 * L. This is not the sharp descent lemma, but it is already enough to connect primitive Lipschitz-gradient assumptions to the algorithmic convergence theorems in this entry.

```
definition line_mean_value_gradient_on ::
  "'a::real_inner set ⇒ ('a ⇒ real) ⇒ ('a ⇒ 'a) ⇒ bool"
where
  "line_mean_value_gradient_on S f G ⟷
    (∀x∈S. ∀y∈S.
       ∃t. 0 ≤ t ∧ t ≤ 1 ∧
         f y - f x =
           inner (G (x + t *R (y - x))) (y - x))"

lemma line_mean_value_gradient_onI:
  assumes "⋀x y. x ∈ S ⟹ y ∈ S ⟹
    ∃t. 0 ≤ t ∧ t ≤ 1 ∧
      f y - f x =
        inner (G (x + t *R (y - x))) (y - x)"
  shows "line_mean_value_gradient_on S f G"
  ⟨proof⟩

lemma line_mean_value_gradient_onD:
  assumes mvt: "line_mean_value_gradient_on S f G"
```

```
    and x: "x ∈ S"
    and y: "y ∈ S"
  obtains t where
    "0 ≤ t"
    "t ≤ 1"
    "f y - f x =
      inner (G (x + t *R (y - x))) (y - x)"
  ⟨proof⟩

lemma line_mean_value_and_lipschitz_gradient_imp_line_descent_bound_on_twice:
  fixes f :: "'a::real_inner ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes lip: "lipschitz_gradient_on L S G"
    and convex: "convex S"
    and mvt: "line_mean_value_gradient_on S f G"
  shows "line_descent_bound_on (2 * L) S f G"
⟨proof⟩

lemma line_mean_value_and_lipschitz_gradient_imp_smooth_upper_bound_on_twice:
  fixes f :: "'a::real_inner ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes lip: "lipschitz_gradient_on L S G"
    and convex: "convex S"
    and mvt: "line_mean_value_gradient_on S f G"
  shows "smooth_upper_bound_on (2 * L) S f G"
⟨proof⟩

lemma line_mean_value_and_lipschitz_gradient_imp_lipschitz_smooth_on_twice:
  fixes f :: "'a::real_inner ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes grad: "has_gradient_on f S G"
    and lip: "lipschitz_gradient_on L S G"
    and convex: "convex S"
    and mvt: "line_mean_value_gradient_on S f G"
  shows "lipschitz_smooth_on (2 * L) S f G"
⟨proof⟩

lemma line_mean_value_and_lipschitz_gradient_imp_lipschitz_smooth_convex_on_twice:
  fixes f :: "'a::real_inner ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes convex_f: "convex_on S f"
    and grad: "has_gradient_on f S G"
    and lip: "lipschitz_gradient_on L S G"
    and convex: "convex S"
    and mvt: "line_mean_value_gradient_on S f G"
  shows "lipschitz_smooth_convex_on (2 * L) S f G"
⟨proof⟩

lemma line_mean_value_and_lipschitz_gradient_imp_smooth_convex_on_twice:
```

**fixes** *f :: "'a::real_inner ⇒ real"*
  **and** *G :: "'a ⇒ 'a"*
**assumes** *convex_f: "convex_on S f"*
  **and** *grad: "has_gradient_on f S G"*
  **and** *lip: "lipschitz_gradient_on L S G"*
  **and** *convex: "convex S"*
  **and** *mvt: "line_mean_value_gradient_on S f G"*
**shows** *"smooth_convex_on (2 * L) S f G"*
⟨*proof*⟩

## 21.6 Algorithmic consequences through smoothness

**lemma** *lipschitz_smooth_convex_gradient_step_decrease:*
**assumes** *lsc: "lipschitz_smooth_convex_on L S f G"*
  **and** *x: "x ∈ S"*
  **and** *step: "gradient_step alpha G x ∈ S"*
  **and** *alpha_nonneg: "0 ≤ alpha"*
  **and** *step_size: "alpha * L ≤ 1"*
**shows**
  *"f (gradient_step alpha G x)*
  *≤ f x - (alpha / 2) * norm (G x) ^ 2"*
⟨*proof*⟩

**lemma** *lipschitz_smooth_convex_gradient_descent_objective_nonincreasing:*
**assumes** *lsc: "lipschitz_smooth_convex_on L S f G"*
  **and** *gd: "gradient_descent_iterates alpha G x"*
  **and** *feasible: "feasible_iterates S x"*
  **and** *alpha_nonneg: "0 ≤ alpha"*
  **and** *step_size: "alpha * L ≤ 1"*
**shows** *"nonincreasing_sequence (objective_values f x)"*
⟨*proof*⟩

**lemma** *lipschitz_smooth_convex_gradient_descent_function_value_gap_bound:*
**fixes** *f :: "'a::real_inner ⇒ real"*
  **and** *G :: "'a ⇒ 'a"*
**assumes** *lsc: "lipschitz_smooth_convex_on L S f G"*
  **and** *gd: "gradient_descent_iterates alpha G x"*
  **and** *feasible: "feasible_iterates S x"*
  **and** *alpha_pos: "0 < alpha"*
  **and** *step_size: "alpha * L ≤ 1"*
  **and** *minimizer: "global_min_on S f xstar"*
  **and** *N_pos: "N > 0"*
**shows**
  *"f (x N) - f xstar*
  *≤ norm (x 0 - xstar) ^ 2 / (2 * alpha * real N)"*
⟨*proof*⟩

**lemma** *lipschitz_smooth_convex_projected_gradient_descent_function_value_gap_bound:*
**fixes** *f :: "'a::{real_inner,heine_borel} ⇒ real"*

```
    and G :: "'a ⇒ 'a"
  assumes lsc: "lipschitz_smooth_convex_on L C f G"
    and closed: "closed C"
    and convex: "convex C"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on C f xstar"
    and N_pos: "N > 0"
  shows
    "f (x N) - f xstar
      ≤ norm (x 0 - xstar) ^ 2 / (2 * alpha * real N)"
⟨proof⟩

lemma line_mean_value_lipschitz_projected_gradient_descent_function_value_gap_bound:
  fixes f :: "'a::{real_inner,heine_borel} ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes convex_f: "convex_on C f"
    and grad: "has_gradient_on f C G"
    and lip: "lipschitz_gradient_on L C G"
    and closed: "closed C"
    and convex: "convex C"
    and mvt: "line_mean_value_gradient_on C f G"
    and pgd: "projected_gradient_descent_iterates C alpha G x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * (2 * L) ≤ 1"
    and minimizer: "global_min_on C f xstar"
    and N_pos: "N > 0"
  shows
    "f (x N) - f xstar
      ≤ norm (x 0 - xstar) ^ 2 / (2 * alpha * real N)"
⟨proof⟩
```

## 21.7 Locale form

```
locale lipschitz_smooth =
  fixes S :: "'a::real_inner set"
    and L :: real
    and f :: "'a ⇒ real"
    and G :: "'a ⇒ 'a"
  assumes gradient_f: "has_gradient_on f S G"
    and lipschitz_G: "lipschitz_gradient_on L S G"
    and line_descent: "line_descent_bound_on L S f G"
begin

lemma lipschitz_smooth_on_self:
  "lipschitz_smooth_on L S f G"
```

⟨proof⟩

```
lemma smooth_upper_bound:
  "smooth_upper_bound_on L S f G"
  ⟨proof⟩

lemma L_nonneg:
  "0 ≤ L"
  ⟨proof⟩

lemma line_descent_bound:
  assumes "x ∈ S"
    and "y ∈ S"
  shows "f y ≤ f x + inner (G x) (y - x) + (L / 2) * norm (y - x) ^ 2"
  ⟨proof⟩

lemma lipschitz_dist:
  assumes "x ∈ S"
    and "y ∈ S"
  shows "norm (G y - G x) ≤ L * norm (y - x)"
  ⟨proof⟩

lemma inner_difference_bound:
  assumes "x ∈ S"
    and "y ∈ S"
  shows "inner (G y - G x) (y - x) ≤ L * norm (y - x) ^ 2"
  ⟨proof⟩

end

locale lipschitz_smooth_convex =
  lipschitz_smooth S L f G
  for S :: "'a::real_inner set"
    and L :: real
    and f :: "'a ⇒ real"
    and G :: "'a ⇒ 'a" +
  assumes convex_f: "convex_on S f"
begin

lemma lipschitz_smooth_convex_on_self:
  "lipschitz_smooth_convex_on L S f G"
⟨proof⟩

lemma smooth_convex_on_self:
  "smooth_convex_on L S f G"
  ⟨proof⟩

lemma convex_differentiable_on_self:
  "convex_differentiable_on S f G"
```

⟨*proof*⟩

```
lemma gradient_step_decrease:
  assumes "x ∈ S"
    and "gradient_step alpha G x ∈ S"
    and "0 ≤ alpha"
    and "alpha * L ≤ 1"
  shows
    "f (gradient_step alpha G x)
      ≤ f x - (alpha / 2) * norm (G x) ^ 2"
  ⟨proof⟩

lemma gradient_descent_function_value_gap_bound:
  assumes gd: "gradient_descent_iterates alpha G x"
    and feasible: "feasible_iterates S x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha * L ≤ 1"
    and minimizer: "global_min_on S f xstar"
    and N_pos: "N > 0"
  shows
    "f (x N) - f xstar
      ≤ norm (x 0 - xstar) ^ 2 / (2 * alpha * real N)"
  ⟨proof⟩

end
```

The main bridge theorem for the sharp certificate interface is *lipschitz_smooth_convex_on ?L ?S ?f ?G* $\Longrightarrow$ *smooth_convex_on ?L ?S ?f ?G*. It allows any development stated in terms of *smooth_convex_on* to be used with the more informative *lipschitz_smooth_convex_on* interface.

The mean-value bridge ⟦*convex_on ?S ?f; has_gradient_on ?f ?S ?G; lipschitz_gradient_on ?L ?S ?G; convex ?S; line_mean_value_gradient_on ?S ?f ?G*⟧ $\Longrightarrow$ *smooth_convex_on (2 * ?L) ?S ?f ?G* connects primitive Lipschitz-gradient assumptions to the same convergence framework, with constant 2 * L. This loses the sharp textbook constant but avoids committing the algorithmic library to a particular integration formalization. A future refinement can recover the sharp constant L by proving the standard integral descent lemma along line segments.

```
end
theory Examples_Quadratic
  imports Lipschitz_Smoothness
begin
```

## 22 Quadratic examples

This theory gives a basic one-dimensional quadratic example.

The objective is

*f x = x^2 / 2*,

with gradient field

G x = x.

This is the simplest nontrivial example satisfying the smooth convex, Lipschitz-gradient, and strong-convexity interfaces developed in the previous theories.

The purpose of this file is not to develop a large collection of examples, but to demonstrate that the abstract assumptions used in the gradient-descent and projected-gradient-descent theorems can be instantiated concretely.

## 22.1 The one-dimensional quadratic objective

```
definition quadratic_real :: "real ⇒ real"
where
  "quadratic_real x = x ^ 2 / 2"

definition quadratic_real_gradient :: "real ⇒ real"
where
  "quadratic_real_gradient x = x"
```

## 22.2 Elementary algebra

```
lemma quadratic_real_nonnegative:
  "0 ≤ quadratic_real x"
  ⟨proof⟩

lemma quadratic_real_zero [simp]:
  "quadratic_real 0 = 0"
  ⟨proof⟩

lemma quadratic_real_gradient_zero [simp]:
  "quadratic_real_gradient 0 = 0"
  ⟨proof⟩

lemma quadratic_real_expansion:
  "quadratic_real y =
    quadratic_real x
    + inner (quadratic_real_gradient x) (y - x)
    + (1 / 2) * norm (y - x) ^ 2"
  ⟨proof⟩

lemma quadratic_real_convex_identity:
  fixes u v x y :: real
  assumes sum_one: "u + v = 1"
  shows
    "u * quadratic_real x + v * quadratic_real y
      - quadratic_real (u * x + v * y)
```

```
    = (u * v * (x - y) ^ 2) / 2"
⟨proof⟩

lemma quadratic_real_convex_ineq:
  fixes u v x y :: real
  assumes u_nonneg: "0 ≤ u"
    and v_nonneg: "0 ≤ v"
    and sum_one: "u + v = 1"
  shows
    "quadratic_real (u * x + v * y)
      ≤ u * quadratic_real x + v * quadratic_real y"
⟨proof⟩

lemma quadratic_real_convex_on_UNIV:
  "convex_on UNIV quadratic_real"
  ⟨proof⟩
```

## 22.3 Gradient and convexity

```
lemma quadratic_real_has_gradient:
  "has_gradient quadratic_real x (quadratic_real_gradient x)"
  ⟨proof⟩

lemma quadratic_real_has_gradient_on_UNIV:
  "has_gradient_on quadratic_real UNIV quadratic_real_gradient"
  ⟨proof⟩

lemma quadratic_real_convex_differentiable_on_UNIV:
  "convex_differentiable_on UNIV quadratic_real quadratic_real_gradient"
⟨proof⟩

lemma quadratic_real_convex_differentiable_on:
  assumes convex: "convex C"
  shows "convex_differentiable_on C quadratic_real quadratic_real_gradient"
⟨proof⟩
```

## 22.4 Smoothness and Lipschitz gradients

```
lemma quadratic_real_smooth_upper_bound_on_UNIV:
  "smooth_upper_bound_on 1 UNIV quadratic_real quadratic_real_gradient"
⟨proof⟩

lemma quadratic_real_smooth_upper_bound_on:
  assumes "C ⊆ UNIV"
  shows "smooth_upper_bound_on 1 C quadratic_real quadratic_real_gradient"
  ⟨proof⟩

lemma quadratic_real_line_descent_bound_on_UNIV:
  "line_descent_bound_on 1 UNIV quadratic_real quadratic_real_gradient"
  ⟨proof⟩
```

**lemma** *quadratic_real_lipschitz_gradient_on_UNIV:*
*"lipschitz_gradient_on 1 UNIV quadratic_real_gradient"*
⟨*proof*⟩

**lemma** *quadratic_real_lipschitz_gradient_on:*
**assumes** *"C ⊆ UNIV"*
**shows** *"lipschitz_gradient_on 1 C quadratic_real_gradient"*
⟨*proof*⟩

**lemma** *quadratic_real_smooth_convex_on_UNIV:*
*"smooth_convex_on 1 UNIV quadratic_real quadratic_real_gradient"*
⟨*proof*⟩

**lemma** *quadratic_real_smooth_convex_on:*
**assumes** *convex: "convex C"*
**shows** *"smooth_convex_on 1 C quadratic_real quadratic_real_gradient"*
⟨*proof*⟩

**lemma** *quadratic_real_lipschitz_smooth_on_UNIV:*
*"lipschitz_smooth_on 1 UNIV quadratic_real quadratic_real_gradient"*
⟨*proof*⟩

**lemma** *quadratic_real_lipschitz_smooth_convex_on_UNIV:*
*"lipschitz_smooth_convex_on 1 UNIV quadratic_real quadratic_real_gradient"*
⟨*proof*⟩

## 22.5 Strong convexity

**lemma** *quadratic_real_strong_lower_bound_on_UNIV:*
*"strong_convex_lower_bound_on 1 UNIV quadratic_real quadratic_real_gradient"*
⟨*proof*⟩

**lemma** *quadratic_real_strong_lower_bound_on:*
**assumes** *"C ⊆ UNIV"*
**shows** *"strong_convex_lower_bound_on 1 C quadratic_real quadratic_real_gradient"*
⟨*proof*⟩

**lemma** *quadratic_real_strongly_convex_differentiable_on_UNIV:*
*"strongly_convex_differentiable_on 1 UNIV quadratic_real quadratic_real_gradient"*
⟨*proof*⟩

**lemma** *quadratic_real_strongly_smooth_convex_on_UNIV:*
*"strongly_smooth_convex_on 1 1 UNIV quadratic_real quadratic_real_gradient"*
⟨*proof*⟩

**lemma** *quadratic_real_strongly_smooth_convex_on:*
**assumes** *convex: "convex C"*
**shows** *"strongly_smooth_convex_on 1 1 C quadratic_real quadratic_real_gradient"*

⟨proof⟩

## 22.6 Global minimizers

```
lemma quadratic_real_global_min_on_zero:
  assumes zero_mem: "0 ∈ C"
  shows "global_min_on C quadratic_real 0"
⟨proof⟩

lemma quadratic_real_global_min_UNIV:
  "global_min_on UNIV quadratic_real 0"
  ⟨proof⟩

lemma quadratic_real_unique_global_min_on:
  assumes convex: "convex C"
    and zero_mem: "0 ∈ C"
    and minimizer: "global_min_on C quadratic_real x"
  shows "x = 0"
⟨proof⟩
```

## 22.7 Unconstrained gradient descent example

```
lemma quadratic_real_gradient_descent_function_value_gap_bound:
  assumes gd: "gradient_descent_iterates alpha quadratic_real_gradient
x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha ≤ 1"
    and N_pos: "N > 0"
  shows
    "quadratic_real (x N) - quadratic_real 0
      ≤ norm (x 0 - 0) ^ 2 / (2 * alpha * real N)"
⟨proof⟩

lemma quadratic_real_gradient_descent_function_value_gap_bound_simplified:
  assumes gd: "gradient_descent_iterates alpha quadratic_real_gradient
x"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha ≤ 1"
    and N_pos: "N > 0"
  shows
    "quadratic_real (x N)
      ≤ norm (x 0) ^ 2 / (2 * alpha * real N)"
  ⟨proof⟩
```

## 22.8 Projected gradient descent example

```
lemma quadratic_real_projected_gradient_descent_function_value_gap_bound:
  assumes closed: "closed C"
    and convex: "convex C"
    and zero_mem: "0 ∈ C"
```

```
    and pgd: "projected_gradient_descent_iterates C alpha quadratic_real_gradient
x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha ≤ 1"
    and N_pos: "N > 0"
  shows
    "quadratic_real (x N) - quadratic_real 0
      ≤ norm (x 0 - 0) ^ 2 / (2 * alpha * real N)"
⟨proof⟩

lemma quadratic_real_projected_gradient_descent_function_value_gap_bound_simplified:
  assumes closed: "closed C"
    and convex: "convex C"
    and zero_mem: "0 ∈ C"
    and pgd: "projected_gradient_descent_iterates C alpha quadratic_real_gradient
x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha ≤ 1"
    and N_pos: "N > 0"
  shows
    "quadratic_real (x N)
      ≤ norm (x 0) ^ 2 / (2 * alpha * real N)"
  ⟨proof⟩

lemma quadratic_real_projected_gradient_descent_distance_linear_rate:
  assumes closed: "closed C"
    and convex: "convex C"
    and zero_mem: "0 ∈ C"
    and pgd: "projected_gradient_descent_iterates C alpha quadratic_real_gradient
x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha ≤ 1"
  shows
    "norm (x N - 0) ^ 2
      ≤ projected_gradient_linear_rate_factor alpha 1 ^ N
          * norm (x 0 - 0) ^ 2"
⟨proof⟩

lemma quadratic_real_projected_gradient_descent_distance_linear_rate_simplified:
  assumes closed: "closed C"
    and convex: "convex C"
    and zero_mem: "0 ∈ C"
    and pgd: "projected_gradient_descent_iterates C alpha quadratic_real_gradient
x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
```

```
    and step_size: "alpha ≤ 1"
  shows
    "norm (x N) ^ 2
      ≤ projected_gradient_linear_rate_factor alpha 1 ^ N
          * norm (x 0) ^ 2"
⟨proof⟩
```

This file demonstrates that the abstract assumptions used throughout the development are non-vacuous. The one-dimensional quadratic objective satisfies the gradient, convexity, smoothness, Lipschitz-gradient, and strong-convexity interfaces with constants L = 1 and mu = 1.

The final lemmas instantiate both the O(1/N) projected-gradient theorem and the linear distance-rate theorem for this concrete objective.

```
end
theory Examples_Projected_Quadratic
  imports
    Examples_Quadratic
    Projected_Gradient_Descent_Residual_Convergence
begin
```

# 23 Projected quadratic examples

This theory gives concrete constrained examples for the projected-gradient descent library.

The objective is the one-dimensional quadratic `f x = x^2 / 2` with gradient field G x = x. The previous example theory instantiated the smoothness, convexity, and strong-convexity interfaces for this objective. Here we use those facts to instantiate the projected-gradient mapping residual bounds and the finite-horizon epsilon-stationarity certificates.

## 23.1 A reusable constrained quadratic template

The first group of lemmas works for any closed convex feasible set containing the unconstrained minimizer 0. This gives a small reusable template for instantiating the abstract projected-gradient descent theorems on concrete quadratic constrained problems.

```
lemma quadratic_real_projected_gradient_descent_exists_small_residual_sq:
  assumes closed: "closed C"
    and convex: "convex C"
    and zero_mem: "0 ∈ C"
    and pgd: "projected_gradient_descent_iterates C alpha quadratic_real_gradient
x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha ≤ 1"
    and N_pos: "N > 0"
```

```
  shows
    "∃n<N.
      projected_gradient_residual_sq C alpha quadratic_real_gradient (x
n)
        ≤ (2 / (alpha * real N)) * quadratic_real (x 0)"
```
⟨proof⟩

```
lemma quadratic_real_projected_gradient_descent_exists_epsilon_residual:
  assumes closed: "closed C"
    and convex: "convex C"
    and zero_mem: "0 ∈ C"
    and pgd: "projected_gradient_descent_iterates C alpha quadratic_real_gradient
x"
    and x0_mem: "x 0 ∈ C"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha ≤ 1"
    and N_pos: "N > 0"
    and eps_pos: "0 < eps"
    and horizon:
      "2 * quadratic_real (x 0) ≤ alpha * real N * eps ^ 2"
  shows
    "∃n<N. projected_gradient_residual C alpha quadratic_real_gradient
(x n) ≤ eps"
```
⟨proof⟩

```
lemma quadratic_real_projected_gradient_residual_zero_imp_zero:
  assumes closed: "closed C"
    and convex: "convex C"
    and zero_mem: "0 ∈ C"
    and x_mem: "x ∈ C"
    and alpha_pos: "0 < alpha"
    and zero:
      "projected_gradient_residual C alpha quadratic_real_gradient x =
0"
  shows "x = 0"
```
⟨proof⟩

## 23.2 The nonnegative half-line

We now specialize the previous template to the concrete closed convex feasible set [0,). This is a simple constrained problem whose minimizer lies on the feasible set and whose projected-gradient residual certificates follow directly from the abstract theory.

```
definition nonnegative_real :: "real set"
where
  "nonnegative_real = {0..}"

lemma nonnegative_real_iff [simp]:
  "x ∈ nonnegative_real ⟷ 0 ≤ x"
```

⟨proof⟩

**lemma** *zero_mem_nonnegative_real [simp]:*
  *"0 ∈ nonnegative_real"*
  ⟨proof⟩

**lemma** *closed_nonnegative_real:*
  *"closed nonnegative_real"*
  ⟨proof⟩

**lemma** *convex_nonnegative_real:*
  *"convex nonnegative_real"*
  ⟨proof⟩

**lemma** *quadratic_real_smooth_convex_on_nonnegative:*
  *"smooth_convex_on 1 nonnegative_real quadratic_real quadratic_real_gradient"*
  ⟨proof⟩

**lemma** *quadratic_real_strongly_smooth_convex_on_nonnegative:*
  *"strongly_smooth_convex_on 1 1 nonnegative_real*
    *quadratic_real quadratic_real_gradient"*
  ⟨proof⟩

**lemma** *quadratic_real_global_min_on_nonnegative_zero:*
  *"global_min_on nonnegative_real quadratic_real 0"*
  ⟨proof⟩

**lemma** *quadratic_real_unique_global_min_on_nonnegative:*
  **assumes** *minimizer: "global_min_on nonnegative_real quadratic_real x"*
  **shows** *"x = 0"*
  ⟨proof⟩

## 23.3 Projected-gradient step on the nonnegative half-line

The projected-gradient step for the quadratic objective can be unfolded to the projection of the scalar point (1 - alpha) * x onto the half-line. We do not need a closed-form formula for this projection in order to instantiate the convergence theory.

**lemma** *nonnegative_quadratic_projected_gradient_step_unfold:*
  *"projected_gradient_step nonnegative_real alpha quadratic_real_gradient*
*x =*
    *closest_point nonnegative_real ((1 - alpha) * x)"*
  ⟨proof⟩

**lemma** *nonnegative_quadratic_projected_gradient_mapping_unfold:*
  *"projected_gradient_mapping nonnegative_real alpha quadratic_real_gradient*
*x =*
    *(1 / alpha) * (x - closest_point nonnegative_real ((1 - alpha) * x))"*
  ⟨proof⟩

**lemma** *nonnegative_quadratic_projected_gradient_residual_unfold:*
*"projected_gradient_residual nonnegative_real alpha quadratic_real_gradient x =*
*norm ((1 / alpha) * (x - closest_point nonnegative_real ((1 - alpha) * x)))"*
⟨*proof*⟩

**lemma** *nonnegative_quadratic_projected_gradient_residual_sq_unfold:*
*"projected_gradient_residual_sq nonnegative_real alpha quadratic_real_gradient x =*
*norm ((1 / alpha) * (x - closest_point nonnegative_real ((1 - alpha) * x))) ^ 2"*
⟨*proof*⟩

## 23.4 Function-value convergence on the half-line

**lemma** *nonnegative_quadratic_projected_gradient_descent_function_value_gap_bound:*
**assumes** *pgd:*
*"projected_gradient_descent_iterates nonnegative_real alpha*
*quadratic_real_gradient x"*
**and** *x0_nonneg: "0 ≤ x 0"*
**and** *alpha_pos: "0 < alpha"*
**and** *step_size: "alpha ≤ 1"*
**and** *N_pos: "N > 0"*
**shows**
*"quadratic_real (x N)*
*≤ norm (x 0) ^ 2 / (2 * alpha * real N)"*
⟨*proof*⟩

**lemma** *nonnegative_quadratic_projected_gradient_descent_distance_linear_rate:*
**assumes** *pgd:*
*"projected_gradient_descent_iterates nonnegative_real alpha*
*quadratic_real_gradient x"*
**and** *x0_nonneg: "0 ≤ x 0"*
**and** *alpha_pos: "0 < alpha"*
**and** *step_size: "alpha ≤ 1"*
**shows**
*"norm (x N) ^ 2*
*≤ projected_gradient_linear_rate_factor alpha 1 ^ N * norm (x 0) ^ 2"*
⟨*proof*⟩

## 23.5 Projected-gradient residual bounds on the half-line

**lemma** *nonnegative_quadratic_projected_gradient_descent_exists_small_residual_sq:*
**assumes** *pgd:*
*"projected_gradient_descent_iterates nonnegative_real alpha*
*quadratic_real_gradient x"*
**and** *x0_nonneg: "0 ≤ x 0"*

```
    and alpha_pos: "0 < alpha"
    and step_size: "alpha ≤ 1"
    and N_pos: "N > 0"
  shows
    "∃n<N.
      projected_gradient_residual_sq nonnegative_real alpha
        quadratic_real_gradient (x n)
        ≤ (2 / (alpha * real N)) * quadratic_real (x 0)"
⟨proof⟩

lemma nonnegative_quadratic_projected_gradient_descent_exists_epsilon_residual:
  assumes pgd:
      "projected_gradient_descent_iterates nonnegative_real alpha
        quadratic_real_gradient x"
    and x0_nonneg: "0 ≤ x 0"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha ≤ 1"
    and N_pos: "N > 0"
    and eps_pos: "0 < eps"
    and horizon:
      "2 * quadratic_real (x 0) ≤ alpha * real N * eps ^ 2"
  shows
    "∃n<N.
      projected_gradient_residual nonnegative_real alpha
        quadratic_real_gradient (x n) ≤ eps"
⟨proof⟩

lemma nonnegative_quadratic_projected_gradient_descent_exists_epsilon_residual_product_for
  assumes pgd:
      "projected_gradient_descent_iterates nonnegative_real alpha
        quadratic_real_gradient x"
    and x0_nonneg: "0 ≤ x 0"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha ≤ 1"
    and N_pos: "N > 0"
    and eps_pos: "0 < eps"
    and horizon:
      "quadratic_real (x 0) ≤ (alpha * real N * eps ^ 2) / 2"
  shows
    "∃n<N.
      projected_gradient_residual nonnegative_real alpha
        quadratic_real_gradient (x n) ≤ eps"
⟨proof⟩
```

## 23.6 Residual-zero certificate on the half-line

```
lemma nonnegative_quadratic_projected_gradient_residual_zero_imp_zero:
  assumes x_nonneg: "0 ≤ x"
    and alpha_pos: "0 < alpha"
```

```
    and zero:
      "projected_gradient_residual nonnegative_real alpha
        quadratic_real_gradient x = 0"
  shows "x = 0"
⟨proof⟩

lemma nonnegative_quadratic_projected_gradient_residual_zero_imp_global_min:
  assumes x_nonneg: "0 ≤ x"
    and alpha_pos: "0 < alpha"
    and zero:
      "projected_gradient_residual nonnegative_real alpha
        quadratic_real_gradient x = 0"
  shows "global_min_on nonnegative_real quadratic_real x"
⟨proof⟩
```

## 23.7 A bounded interval constraint

We also instantiate the projected-gradient library on a bounded interval constraint [a,b] containing the unconstrained minimizer 0. This gives a more concrete constrained example than the nonnegative half-line, while still using the same abstract projected-gradient convergence and residual machinery.

The point of this example is that no closed-form formula for the projection is needed. The metric projection and its variational inequality are supplied by the general projection-geometry layer.

```
definition bounded_interval_real :: "real ⇒ real ⇒ real set"
where
  "bounded_interval_real a b = {a..b}"

lemma bounded_interval_real_iff [simp]:
  "x ∈ bounded_interval_real a b ⟷ a ≤ x ∧ x ≤ b"
  ⟨proof⟩

lemma zero_mem_bounded_interval_real [simp]:
  assumes "a ≤ 0"
    and "0 ≤ b"
  shows "0 ∈ bounded_interval_real a b"
  ⟨proof⟩

lemma closed_bounded_interval_real:
  "closed (bounded_interval_real a b)"
  ⟨proof⟩

lemma convex_bounded_interval_real:
  "convex (bounded_interval_real a b)"
  ⟨proof⟩

lemma quadratic_real_smooth_convex_on_bounded_interval:
```

```
  "smooth_convex_on 1 (bounded_interval_real a b)
    quadratic_real quadratic_real_gradient"
  ⟨proof⟩

lemma quadratic_real_strongly_smooth_convex_on_bounded_interval:
  "strongly_smooth_convex_on 1 1 (bounded_interval_real a b)
    quadratic_real quadratic_real_gradient"
  ⟨proof⟩

lemma quadratic_real_global_min_on_bounded_interval_zero:
  assumes a0: "a ≤ 0"
    and b0: "0 ≤ b"
  shows "global_min_on (bounded_interval_real a b) quadratic_real 0"
  ⟨proof⟩

lemma quadratic_real_unique_global_min_on_bounded_interval:
  assumes a0: "a ≤ 0"
    and b0: "0 ≤ b"
    and minimizer:
      "global_min_on (bounded_interval_real a b) quadratic_real x"
  shows "x = 0"
  ⟨proof⟩
```

## 23.8 Projected-gradient step on a bounded interval

For the quadratic objective, the projected-gradient step is the projection of (1 - alpha) * x onto the interval [a,b].

```
lemma bounded_interval_quadratic_projected_gradient_step_unfold:
  "projected_gradient_step (bounded_interval_real a b) alpha
      quadratic_real_gradient x =
    closest_point (bounded_interval_real a b) ((1 - alpha) * x)"
  ⟨proof⟩

lemma bounded_interval_quadratic_projected_gradient_mapping_unfold:
  "projected_gradient_mapping (bounded_interval_real a b) alpha
      quadratic_real_gradient x =
    (1 / alpha) *
      (x - closest_point (bounded_interval_real a b) ((1 - alpha) * x))"
  ⟨proof⟩

lemma bounded_interval_quadratic_projected_gradient_residual_unfold:
  "projected_gradient_residual (bounded_interval_real a b) alpha
      quadratic_real_gradient x =
    norm ((1 / alpha) *
      (x - closest_point (bounded_interval_real a b) ((1 - alpha) * x)))"
  ⟨proof⟩

lemma bounded_interval_quadratic_projected_gradient_residual_sq_unfold:
  "projected_gradient_residual_sq (bounded_interval_real a b) alpha
```

```
    quadratic_real_gradient x =
  norm ((1 / alpha) *
    (x - closest_point (bounded_interval_real a b) ((1 - alpha) * x)))
^ 2"
```
⟨proof⟩

## 23.9 Function-value convergence on a bounded interval

**lemma** *bounded_interval_quadratic_projected_gradient_descent_function_value_gap_bound:*
```
  assumes a0: "a ≤ 0"
    and b0: "0 ≤ b"
    and pgd:
      "projected_gradient_descent_iterates (bounded_interval_real a b)
alpha
        quadratic_real_gradient x"
    and x0_lower: "a ≤ x 0"
    and x0_upper: "x 0 ≤ b"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha ≤ 1"
    and N_pos: "N > 0"
  shows
    "quadratic_real (x N)
      ≤ norm (x 0) ^ 2 / (2 * alpha * real N)"
```
⟨proof⟩

**lemma** *bounded_interval_quadratic_projected_gradient_descent_distance_linear_rate:*
```
  assumes a0: "a ≤ 0"
    and b0: "0 ≤ b"
    and pgd:
      "projected_gradient_descent_iterates (bounded_interval_real a b)
alpha
        quadratic_real_gradient x"
    and x0_lower: "a ≤ x 0"
    and x0_upper: "x 0 ≤ b"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha ≤ 1"
  shows
    "norm (x N) ^ 2
      ≤ projected_gradient_linear_rate_factor alpha 1 ^ N * norm (x 0)
^ 2"
```
⟨proof⟩

## 23.10 Projected-gradient residual bounds on a bounded interval

**lemma** *bounded_interval_quadratic_projected_gradient_descent_exists_small_residual_sq:*
```
  assumes a0: "a ≤ 0"
    and b0: "0 ≤ b"
    and pgd:
```

```
      "projected_gradient_descent_iterates (bounded_interval_real a b)
alpha
        quadratic_real_gradient x"
    and x0_lower: "a ≤ x 0"
    and x0_upper: "x 0 ≤ b"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha ≤ 1"
    and N_pos: "N > 0"
  shows
    "∃n<N.
      projected_gradient_residual_sq (bounded_interval_real a b) alpha
        quadratic_real_gradient (x n)
        ≤ (2 / (alpha * real N)) * quadratic_real (x 0)"
⟨proof⟩

lemma bounded_interval_quadratic_projected_gradient_descent_exists_epsilon_residual:
  assumes a0: "a ≤ 0"
    and b0: "0 ≤ b"
    and pgd:
      "projected_gradient_descent_iterates (bounded_interval_real a b)
alpha
        quadratic_real_gradient x"
    and x0_lower: "a ≤ x 0"
    and x0_upper: "x 0 ≤ b"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha ≤ 1"
    and N_pos: "N > 0"
    and eps_pos: "0 < eps"
    and horizon:
      "2 * quadratic_real (x 0) ≤ alpha * real N * eps ^ 2"
  shows
    "∃n<N.
      projected_gradient_residual (bounded_interval_real a b) alpha
        quadratic_real_gradient (x n) ≤ eps"
⟨proof⟩

lemma bounded_interval_quadratic_projected_gradient_descent_exists_epsilon_residual_produc
  assumes a0: "a ≤ 0"
    and b0: "0 ≤ b"
    and pgd:
      "projected_gradient_descent_iterates (bounded_interval_real a b)
alpha
        quadratic_real_gradient x"
    and x0_lower: "a ≤ x 0"
    and x0_upper: "x 0 ≤ b"
    and alpha_pos: "0 < alpha"
    and step_size: "alpha ≤ 1"
    and N_pos: "N > 0"
    and eps_pos: "0 < eps"
```

```
    and horizon:
      "quadratic_real (x 0) ≤ (alpha * real N * eps ^ 2) / 2"
  shows
    "∃n<N.
      projected_gradient_residual (bounded_interval_real a b) alpha
        quadratic_real_gradient (x n) ≤ eps"
⟨proof⟩
```

### 23.11 Residual-zero certificate on a bounded interval

```
lemma bounded_interval_quadratic_projected_gradient_residual_zero_imp_zero:
  assumes a0: "a ≤ 0"
    and b0: "0 ≤ b"
    and x_lower: "a ≤ x"
    and x_upper: "x ≤ b"
    and alpha_pos: "0 < alpha"
    and zero:
      "projected_gradient_residual (bounded_interval_real a b) alpha
        quadratic_real_gradient x = 0"
  shows "x = 0"
⟨proof⟩
```

```
lemma bounded_interval_quadratic_projected_gradient_residual_zero_imp_global_min:
  assumes x_lower: "a ≤ x"
    and x_upper: "x ≤ b"
    and alpha_pos: "0 < alpha"
    and zero:
      "projected_gradient_residual (bounded_interval_real a b) alpha
        quadratic_real_gradient x = 0"
  shows "global_min_on (bounded_interval_real a b) quadratic_real x"
⟨proof⟩
```

The most important concrete consequences in this file are: ⟦projected_gradient_descent_iterates nonnegative_real ?alpha quadratic_real_gradient ?x; 0 ≤ ?x 0; 0 < ?alpha; ?alpha ≤ 1; 0 < ?N⟧ ⟹ quadratic_real (?x ?N) ≤ (norm (?x 0))$^2$ / (2 * ?alpha * real ?N), ⟦projected_gradient_descent_iterates nonnegative_real ?alpha quadratic_real_gradient ?x; 0 ≤ ?x 0; 0 < ?alpha; ?alpha ≤ 1; 0 < ?N⟧ ⟹ ∃n<?N. projected_gradient_residual_sq nonnegative_real ?alpha quadratic_real_gradient (?x n) ≤ 2 / (?alpha * real ?N) * quadratic_real (?x 0), ⟦projected_gradient_descent_iterates nonnegative_real ?alpha quadratic_real_gradient ?x; 0 ≤ ?x 0; 0 < ?alpha; ?alpha ≤ 1; 0 < ?N; 0 < ?eps; 2 * quadratic_real (?x 0) ≤ ?alpha * real ?N * ?eps$^2$⟧ ⟹ ∃n<?N. projected_gradient_residual nonnegative_real ?alpha quadratic_real_gradient (?x n) ≤ ?eps, ⟦0 ≤ ?x; 0 < ?alpha; projected_gradient_residual nonnegative_real ?alpha quadratic_real_gradient ?x = 0⟧ ⟹ ?x = 0, ⟦?a ≤ 0; 0 ≤ ?b; projected_gradient_descent_iterates (bounded_interval_real ?a ?b) ?alpha quadratic_real_gradient ?x; ?a ≤ ?x 0; ?x 0 ≤ ?b; 0 < ?alpha; ?alpha ≤ 1; 0 < ?N⟧ ⟹ quadratic_real (?x ?N) ≤ (norm (?x 0))$^2$ / (2 * ?alpha * real ?N), ⟦?a ≤ 0; 0 ≤ ?b; projected_gradient_descent_ *(bounded_interval_real ?a ?b) ?alpha quadratic_real_gradient ?x; ?a* $\leq$ *?x 0; ?x 0* $\leq$ *?b; 0 < ?alpha; ?alpha* $\leq$ *1; 0 < ?N; 0 < ?eps; 2 * quadratic_real (?x 0)* $\leq$ *?alpha * real ?N * ?eps*$^2$$]\!] \Longrightarrow \exists$ *n<?N. projected_gradient_residual (bounded_interval_real ?a ?b) ?alpha quadratic_real_gradient (?x n)* $\leq$ *?eps*, and $[\![$*?a* $\leq$ *0; 0* $\leq$ *?b; ?a* $\leq$ *?x; ?x* $\leq$ *?b; 0 < ?alpha; projected_gradient_residual (bounded_interval_real ?a ?b) ?alpha quadratic_real_gradient ?x = 0*$]\!] \Longrightarrow$ *?x = 0*.

Together, they show that the abstract projected-gradient convergence and residual-stationarity theorems can be instantiated on simple constrained smooth strongly convex quadratic problems, including both an unbounded half-line constraint and a bounded interval constraint.

**end**
**theory** *Main_Results*
  **imports**
    *Lipschitz_Smoothness*
    *Examples_Projected_Quadratic*
**begin**

# 24 Main reusable theorem interface

This theory is the recommended public import target of the entry.

It collects the main reusable theorem groups for smooth convex first-order optimization, while hiding the internal proof structure of the individual theory files. Downstream developments should normally import this theory instead of importing the lower-level proof files directly.

The entry is organized as a small library layer for smooth convex first-order optimization. The main reusable components are:

1. gradient and convex first-order interfaces; 2. smooth upper-bound and descent interfaces; 3. abstract telescoping lemmas for convergence proofs; 4. gradient descent and projected-gradient descent convergence theorems; 5. projection geometry and projected-gradient step rules; 6. projected-gradient mapping optimality and residual certificates; 7. strong-convexity and linear-rate results; 8. concrete quadratic examples.

The theorem groups below are organized from low-level analytic interfaces to high-level convergence and residual-complexity statements. The final section collects the recommended citation surface of the entry.

## 24.1 Gradient interfaces

These theorem groups provide the basic gradient-like interface used throughout the entry. The pointwise rules introduce and eliminate gradient facts at a single point, while the set-based rules are used to work on feasible regions. The algebraic rules are useful for instantiating the framework on concrete objectives.

**lemmas** *gradient_pointwise_interface* =
*has_gradientI*
*has_gradientD*
*has_gradient_unique*
*gradient_eqI*
*has_gradient_gradient*

**lemmas** *gradient_field_interface* =
*has_gradient_onD*
*has_gradient_within_onD*
*has_gradient_on_subset*

**lemmas** *gradient_rule_interface* =
*has_gradient_const*
*has_gradient_add*
*has_gradient_minus*
*has_gradient_diff*
*has_gradient_scale_const*
*has_gradient_mult*
*has_gradient_inner_left*
*has_gradient_inner_right*

## 24.2 Convex first-order certificates

These facts connect convexity, gradients, and first-order optimality. They are the main bridge between an analytic certificate, such as a variational inequality, and global optimality for a convex objective.

**lemmas** *global_min_interface* =
*global_min_onI*
*global_min_onD_mem*
*global_min_onD*

**lemmas** *first_order_condition_interface* =
*first_order_condition_atI*
*first_order_condition_atD_mem*
*first_order_condition_atD*

**lemmas** *convex_first_order_results* =
*convex_has_gradient_supports*
*convex_differentiable_on_imp_gradient_lower_bound_on*
*convex_differentiable_on_supporting_hyperplanes*
*convex_differentiable_on_first_order_sufficient*

## 24.3 Smooth convex interfaces and descent lemmas

The convergence proofs are parameterized by a quadratic smooth upper-bound certificate. This isolates the algorithmic descent argument from the analytic source of smoothness. Later theories can instantiate this certificate from Lipschitz-gradient assumptions, line-descent bounds, or concrete examples.

**lemmas** *smooth_upper_bound_interface =*
  *smooth_upper_bound_onI*
  *smooth_upper_bound_onD_nonneg*
  *smooth_upper_bound_onD*
  *smooth_upper_bound_on_subset*
  *smooth_upper_bound_on_mono_L*

**lemmas** *smooth_convex_interface =*
  *smooth_convex_onI*
  *smooth_convex_onD_convex_differentiable*
  *smooth_convex_onD_smooth_upper_bound*
  *smooth_convex_onD_convex_on*
  *smooth_convex_onD_has_gradient_on*
  *smooth_convex_on_subset*

**lemmas** *gradient_step_interface =*
  *gradient_step_diff*
  *gradient_step_inner*
  *gradient_step_norm_sq*

**lemmas** *gradient_step_descent_results =*
  *smooth_upper_bound_gradient_step*
  *smooth_upper_bound_gradient_step_decrease*
  *smooth_upper_bound_gradient_step_decrease_inverse_L*

## 24.4 Abstract descent and telescoping

These lemmas are independent of gradients and projections. They package the finite-sum and averaging arguments used by the convergence proofs. They are intended to be reusable for other first-order algorithms with one-step progress estimates.

**lemmas** *abstract_descent_results =*
  *sum_progress_le_initial_gap*
  *sum_progress_le_initial_minus_lower_bound*
  *exists_le_average_of_sum_bound*
  *exists_le_average_of_nonnegative_sum*
  *sequence_linear_rate_from_step*

## 24.5 Gradient descent

The following groups expose the unconstrained gradient-descent layer. They include the iteration predicate, one-step descent estimates, gradient-residual rate bounds, and the standard O(1/N) function-value convergence theorem.

**lemmas** *gradient_descent_iteration_interface =*
  *gradient_descent_iteratesI*

*gradient_descent_iteratesD*
*gradient_descent_iteratesE*
*feasible_iteratesI*
*feasible_iteratesD*
*feasible_iterates_subset*

**lemmas** *gradient_descent_one_step_results =*
*gradient_descent_one_step_bound*
*gradient_descent_one_step_decrease*
*gradient_descent_objective_mono_step*
*gradient_descent_objective_nonincreasing*
*gradient_descent_step_progress*

**lemmas** *gradient_descent_rate_results =*
*gradient_descent_sum_weighted_gradient_norm_sq_bound*
*gradient_descent_sum_gradient_norm_sq_bound*
*gradient_descent_average_gradient_norm_sq_bound*
*gradient_descent_exists_small_gradient_norm_sq*
*gradient_descent_exists_small_weighted_gradient_norm_sq*

**lemmas** *gradient_descent_convergence_results =*
*gradient_descent_one_step_distance_bound_to_point*
*gradient_descent_one_step_distance_bound_to_minimizer*
*gradient_descent_sum_function_value_gaps_bound_to_point*
*gradient_descent_sum_function_value_gaps_bound_to_minimizer*
*gradient_descent_last_gap_times_N_bound*
*gradient_descent_function_value_gap_bound*

## 24.6 Projection geometry

This group provides the metric-projection geometry used by the projected method. The variational inequality for closest points is the central geometric input for projected-gradient descent.

**lemmas** *projection_geometry_results =*
*projection_variational_inequality*
*three_point_inner_identity*
*projected_gradient_inner_bound_from_vi*

## 24.7 Projected gradient steps

These rules describe a single projected-gradient step. They expose feasibility of the projected point, the projection variational inequality, and the one-step distance estimates used in the convergence proof.

**lemmas** *projected_gradient_step_interface =*
*projected_gradient_step_in_set*
*projected_gradient_step_in_set_if_base_mem*
*projected_gradient_step_variational_inequality*
*projected_gradient_inner_bound*

**lemmas** *projected_gradient_step_descent_results =*
*projected_gradient_one_step_distance_bound_to_point*
*projected_gradient_one_step_distance_bound_to_minimizer*

## 24.8 Projected gradient descent

These theorem groups give the core projected-gradient descent convergence layer. They prove feasibility preservation, monotonicity of the objective, and the O(1/N) function-value convergence bound for smooth convex objectives over a closed convex feasible set.

**lemmas** *projected_gradient_descent_iteration_interface =*
*projected_gradient_descent_iteratesI*
*projected_gradient_descent_iteratesD*
*projected_gradient_descent_iteratesE*
*projected_gradient_descent_next_mem*
*projected_gradient_descent_feasible_from_initial*

**lemmas** *projected_gradient_descent_one_step_results =*
*projected_gradient_descent_one_step_distance_bound_to_point*
*projected_gradient_descent_one_step_distance_bound_to_minimizer*
*projected_gradient_descent_objective_nonincreasing*

**lemmas** *projected_gradient_descent_convergence_results =*
*projected_gradient_descent_sum_function_value_gaps_bound_to_point*
*projected_gradient_descent_sum_function_value_gaps_bound_to_minimizer*
*projected_gradient_descent_last_gap_times_N_bound*
*projected_gradient_descent_function_value_gap_bound_feasible*
*projected_gradient_descent_function_value_gap_bound*

## 24.9 Projected-gradient mappings

The projected-gradient mapping is the constrained analogue of the gradient residual. A zero projected-gradient mapping is equivalent to a projected fixed point and, over a closed convex feasible set, to the first-order variational inequality. For smooth convex objectives, this gives a global optimality certificate.

**lemmas** *projected_gradient_mapping_interface =*
*projected_gradient_mapping_step_relation*
*projected_gradient_step_eq_sub_mapping*
*projected_gradient_mapping_zero_iff_fixed_point*
*projected_gradient_mapping_zero_imp_step_eq*
*projected_gradient_step_eq_imp_mapping_zero*
*gradient_step_minus_projected_gradient_step_eq_mapping_residual*
*projected_gradient_mapping_variational_inequality*

**lemmas** *projected_gradient_mapping_optimality_results =*
*projected_gradient_fixed_point_iff_first_order_condition*

*projected_gradient_mapping_zero_iff_first_order_condition*
*projected_gradient_fixed_point_imp_global_min_on*
*projected_gradient_mapping_zero_imp_global_min_on*
*first_order_condition_imp_global_min_on_smooth_convex*

### 24.10 Projected-gradient mapping residual rates

These results strengthen projected-gradient descent from function-value convergence to residual convergence. They show that the projected-gradient mapping residual has a finite O(1/N) small-residual certificate along the iterates.

**lemmas** *projected_gradient_mapping_residual_norm_results =*
*projected_gradient_mapping_norm_eq_step_distance_divide*
*projected_gradient_base_step_distance_eq_alpha_mapping_norm*
*projected_gradient_step_base_distance_eq_alpha_mapping_norm*
*projected_gradient_step_distance_sq_eq_mapping_norm_sq*
*projected_gradient_mapping_norm_sq_eq_step_distance_sq*
*projected_gradient_mapping_zero_iff_step_distance_zero*

**lemmas** *projected_gradient_mapping_step_progress_results =*
*projected_gradient_step_progress_step_norm*
*projected_gradient_step_progress_mapping*
*projected_gradient_descent_step_progress_mapping*

**lemmas** *projected_gradient_mapping_sum_rate_results =*
*projected_gradient_descent_sum_weighted_mapping_norm_sq_bound_feasible*
*projected_gradient_descent_sum_weighted_mapping_norm_sq_bound*
*projected_gradient_descent_sum_mapping_norm_sq_bound_feasible*
*projected_gradient_descent_sum_mapping_norm_sq_bound*
*projected_gradient_descent_sum_mapping_norm_sq_bound_to_minimizer_feasible*
*projected_gradient_descent_sum_mapping_norm_sq_bound_to_minimizer*

**lemmas** *projected_gradient_mapping_average_rate_results =*
*projected_gradient_descent_average_mapping_norm_sq_bound_feasible*
*projected_gradient_descent_average_mapping_norm_sq_bound*
*projected_gradient_descent_average_mapping_norm_sq_bound_below_feasible*
*projected_gradient_descent_average_mapping_norm_sq_bound_below*

**lemmas** *projected_gradient_mapping_small_residual_results =*
*projected_gradient_descent_exists_small_mapping_norm_sq_feasible*
*projected_gradient_descent_exists_small_mapping_norm_sq*
*projected_gradient_descent_exists_small_mapping_norm_sq_below_feasible*
*projected_gradient_descent_exists_small_mapping_norm_sq_below*
*projected_gradient_descent_exists_small_mapping_norm_sq_to_minimizer_feasible*
*projected_gradient_descent_exists_small_mapping_norm_sq_to_minimizer*

## 24.11 Residual convergence and epsilon-stationarity

This group rephrases projected-gradient mapping bounds as residual convergence certificates. The main user-facing statement is an epsilon-stationarity complexity result: if the finite horizon is large enough, then one of the first N iterates has projected-gradient residual at most eps.

**lemmas** *projected_gradient_residual_interface =*
  *projected_gradient_residual_nonneg*
  *projected_gradient_residual_sq_nonneg*
  *projected_gradient_residual_sq_eq_residual_power2*
  *projected_gradient_residual_le_of_sq_le*
  *projected_gradient_residual_sq_le_of_residual_le*

**lemmas** *projected_gradient_residual_small_sq_results =*
  *projected_gradient_descent_exists_small_residual_sq_feasible*
  *projected_gradient_descent_exists_small_residual_sq*
  *projected_gradient_descent_exists_small_residual_sq_below_feasible*
  *projected_gradient_descent_exists_small_residual_sq_below*
  *projected_gradient_descent_exists_small_residual_sq_to_minimizer_feasible*
  *projected_gradient_descent_exists_small_residual_sq_to_minimizer*

**lemmas** *projected_gradient_epsilon_stationarity_results =*
  *projected_gradient_descent_exists_epsilon_residual_feasible*
  *projected_gradient_descent_exists_epsilon_residual*
  *projected_gradient_descent_exists_epsilon_residual_below_feasible*
  *projected_gradient_descent_exists_epsilon_residual_below*
  *projected_gradient_descent_exists_epsilon_residual_to_minimizer_feasible*
  *projected_gradient_descent_exists_epsilon_residual_to_minimizer*
  *projected_gradient_descent_exists_epsilon_residual_to_minimizer_product_feasible*
  *projected_gradient_descent_exists_epsilon_residual_to_minimizer_product*

**lemmas** *projected_gradient_residual_zero_results =*
  *projected_gradient_residual_zero_iff_mapping_zero*
  *projected_gradient_residual_zero_iff_fixed_point*
  *projected_gradient_residual_zero_iff_first_order_condition*
  *projected_gradient_residual_zero_imp_global_min_on*

## 24.12 Strong convexity

The strong-convexity layer provides distance-gap estimates, uniqueness of global minimizers, and sharper certificates for projected-gradient stationary points.

**lemmas** *strong_convex_interface =*
  *strong_convex_lower_bound_onI*
  *strong_convex_lower_bound_onD_nonneg*
  *strong_convex_lower_bound_onD*
  *strong_convex_lower_bound_on_subset*
  *strong_convex_lower_bound_on_mono_mu*

**lemmas** *strongly_convex_interface =*
  *strongly_convex_differentiable_onI*
  *strongly_convex_differentiable_onD_convex_differentiable*
  *strongly_convex_differentiable_onD_strong*
  *strongly_smooth_convex_onI*
  *strongly_smooth_convex_onD_smooth*
  *strongly_smooth_convex_onD_strong*

**lemmas** *strong_convex_optimality_results =*
  *convex_differentiable_on_global_min_imp_first_order_condition*
  *strong_convex_foc_distance_gap*
  *strongly_convex_global_min_distance_gap*
  *strongly_smooth_convex_global_min_distance_gap*
  *strongly_convex_global_min_unique*
  *strongly_smooth_convex_global_min_unique*
  *strongly_smooth_projected_mapping_zero_distance_gap*
  *strongly_smooth_projected_mapping_zero_unique_global_min*

## 24.13 Linear convergence for projected gradient descent

Under strong convexity, projected-gradient descent admits a linear convergence rate. The first group collects elementary facts about the contraction factor, and the second group gives the distance and function-value linear-rate theorems.

**lemmas** *projected_gradient_linear_rate_factor_results =*
  *projected_gradient_linear_rate_denominator_pos*
  *projected_gradient_linear_rate_factor_nonnegative*
  *projected_gradient_linear_rate_factor_positive*
  *projected_gradient_linear_rate_factor_le_one*
  *projected_gradient_linear_rate_factor_lt_one*

**lemmas** *projected_gradient_descent_linear_rate_results =*
  *projected_gradient_descent_strong_one_step_distance_contract*
  *projected_gradient_descent_strong_one_step_distance_contract_factor*
  *projected_gradient_descent_distance_sq_linear_rate*
  *projected_gradient_descent_function_value_linear_rate_Suc*
  *projected_gradient_descent_function_value_linear_rate*
  *projected_gradient_descent_strict_rate_factor*

## 24.14 Lipschitz smoothness and mean-value bridge

This interface separates Lipschitz-gradient-style assumptions from the algorithmic convergence layer. The basic certificate interface records the equivalence between line-descent bounds and the smooth upper-bound property.

The mean-value bridge connects primitive Lipschitz-gradient assumptions to the same convergence framework with constant 2 * L. This gives a reusable route from standard differentiability-style assumptions to the projected-gradient descent convergence theorems, without committing the algorithmic layer to a particular integration formalization.

```
lemmas lipschitz_smoothness_interface =
  line_descent_bound_onI
  line_descent_bound_onD_nonneg
  line_descent_bound_onD
  line_descent_bound_on_imp_smooth_upper_bound_on
  smooth_upper_bound_on_imp_line_descent_bound_on
  line_descent_bound_on_iff_smooth_upper_bound_on
  lipschitz_smooth_onI
  lipschitz_smooth_onD_has_gradient_on
  lipschitz_smooth_onD_lipschitz_gradient
  lipschitz_smooth_onD_line_descent
  lipschitz_smooth_onD_smooth_upper_bound
  lipschitz_smooth_convex_onI
  lipschitz_smooth_convex_on_imp_smooth_convex_on
  smooth_convex_on_and_lipschitz_gradient_imp_lipschitz_smooth_convex_on

lemmas lipschitz_mean_value_bridge_interface =
  line_mean_value_gradient_onI
  line_mean_value_gradient_onD
  line_mean_value_and_lipschitz_gradient_imp_line_descent_bound_on_twice
  line_mean_value_and_lipschitz_gradient_imp_smooth_upper_bound_on_twice
  line_mean_value_and_lipschitz_gradient_imp_lipschitz_smooth_on_twice
  line_mean_value_and_lipschitz_gradient_imp_lipschitz_smooth_convex_on_twice
  line_mean_value_and_lipschitz_gradient_imp_smooth_convex_on_twice

lemmas lipschitz_mean_value_algorithmic_results =
  line_mean_value_lipschitz_projected_gradient_descent_function_value_gap_bound
```

## 24.15 Quadratic examples

The quadratic examples instantiate the abstract interfaces on the one-dimensional quadratic objective. They are useful both as regression tests for the library and as compact demonstrations of how to use the public API.

```
lemmas quadratic_real_example_results =
  quadratic_real_has_gradient
  quadratic_real_has_gradient_on_UNIV
  quadratic_real_convex_on_UNIV
  quadratic_real_convex_differentiable_on_UNIV
  quadratic_real_smooth_upper_bound_on_UNIV
  quadratic_real_lipschitz_gradient_on_UNIV
  quadratic_real_smooth_convex_on_UNIV
  quadratic_real_lipschitz_smooth_on_UNIV
  quadratic_real_lipschitz_smooth_convex_on_UNIV
  quadratic_real_strong_lower_bound_on_UNIV
  quadratic_real_strongly_smooth_convex_on_UNIV
  quadratic_real_global_min_on_zero
```

**lemmas** *projected_quadratic_template_results =*
*quadratic_real_projected_gradient_descent_exists_small_residual_sq*
*quadratic_real_projected_gradient_descent_exists_epsilon_residual*
*quadratic_real_projected_gradient_residual_zero_imp_zero*

**lemmas** *nonnegative_quadratic_example_results =*
*closed_nonnegative_real*
*convex_nonnegative_real*
*quadratic_real_smooth_convex_on_nonnegative*
*quadratic_real_strongly_smooth_convex_on_nonnegative*
*quadratic_real_global_min_on_nonnegative_zero*
*quadratic_real_unique_global_min_on_nonnegative*
*nonnegative_quadratic_projected_gradient_step_unfold*
*nonnegative_quadratic_projected_gradient_mapping_unfold*
*nonnegative_quadratic_projected_gradient_residual_unfold*
*nonnegative_quadratic_projected_gradient_descent_function_value_gap_bound*
*nonnegative_quadratic_projected_gradient_descent_distance_linear_rate*
*nonnegative_quadratic_projected_gradient_descent_exists_small_residual_sq*
*nonnegative_quadratic_projected_gradient_descent_exists_epsilon_residual*
*nonnegative_quadratic_projected_gradient_residual_zero_imp_zero*

## 24.16 Recommended public API

The following theorem groups are the recommended public surface of the entry. The larger groups provide a stable overview for downstream developments, while the final citation surface gives short aliases for the main high-level theorems.

The lower-level groups above remain available for specialized uses, but downstream developments should prefer the stable aliases below when referring to the main convergence, residual, optimality, and linear-rate results.

### 24.16.1 Core infrastructure

**lemmas** *main_first_order_infrastructure =*
*gradient_pointwise_interface*
*gradient_field_interface*
*gradient_rule_interface*
*global_min_interface*
*first_order_condition_interface*
*convex_first_order_results*

**lemmas** *main_smooth_descent_infrastructure =*
*smooth_upper_bound_interface*
*smooth_convex_interface*
*gradient_step_interface*
*gradient_step_descent_results*
*abstract_descent_results*

**lemmas** *main_projection_infrastructure =*
*projection_geometry_results*
*projected_gradient_step_interface*
*projected_gradient_step_descent_results*

### 24.16.2 Main convergence theorems

**lemmas** *main_gradient_descent_theorems =*
*gradient_descent_function_value_gap_bound*
*gradient_descent_sum_gradient_norm_sq_bound*
*gradient_descent_average_gradient_norm_sq_bound*
*gradient_descent_exists_small_gradient_norm_sq*

**lemmas** *main_projected_gradient_descent_theorems =*
*projected_gradient_descent_function_value_gap_bound*
*projected_gradient_descent_sum_function_value_gaps_bound_to_minimizer*
*projected_gradient_descent_last_gap_times_N_bound*
*projected_gradient_descent_objective_nonincreasing*

**lemmas** *main_lipschitz_bridge_theorems =*
*line_mean_value_and_lipschitz_gradient_imp_smooth_convex_on_twice*
*line_mean_value_lipschitz_projected_gradient_descent_function_value_gap_bound*

**lemmas** *main_projected_gradient_mapping_theorems =*
*projected_gradient_mapping_zero_iff_fixed_point*
*projected_gradient_mapping_zero_iff_first_order_condition*
*projected_gradient_mapping_zero_imp_global_min_on*
*projected_gradient_descent_sum_mapping_norm_sq_bound*
*projected_gradient_descent_average_mapping_norm_sq_bound*
*projected_gradient_descent_exists_small_mapping_norm_sq*
*projected_gradient_descent_exists_small_mapping_norm_sq_to_minimizer*

**lemmas** *main_epsilon_stationarity_theorems =*
*projected_gradient_residual_zero_iff_first_order_condition*
*projected_gradient_residual_zero_imp_global_min_on*
*projected_gradient_descent_exists_small_residual_sq_to_minimizer*
*projected_gradient_descent_exists_epsilon_residual_to_minimizer*
*projected_gradient_descent_exists_epsilon_residual_to_minimizer_product*

**lemmas** *main_strong_convexity_and_linear_rate_theorems =*
*strongly_smooth_convex_global_min_distance_gap*
*strongly_smooth_convex_global_min_unique*
*projected_gradient_descent_distance_sq_linear_rate*
*projected_gradient_descent_function_value_linear_rate*
*projected_gradient_descent_strict_rate_factor*

**lemmas** *main_example_theorems =*
*quadratic_real_smooth_convex_on_UNIV*

*quadratic_real_strongly_smooth_convex_on_UNIV*
*quadratic_real_global_min_on_zero*
*nonnegative_quadratic_projected_gradient_descent_function_value_gap_bound*
*nonnegative_quadratic_projected_gradient_descent_exists_epsilon_residual*
*nonnegative_quadratic_projected_gradient_residual_zero_imp_zero*
*bounded_interval_quadratic_projected_gradient_descent_function_value_gap_bound*
*bounded_interval_quadratic_projected_gradient_descent_exists_epsilon_residual*
*bounded_interval_quadratic_projected_gradient_residual_zero_imp_zero*

### 24.16.3 Stable aliases for citation

The following aliases give short, stable names to the main high-level statements of the entry. They are intended for use in the AFP document, README, and downstream developments.

These aliases are the part of the public surface that should remain stable under later internal refactorings of the proof files.

**lemmas** *gradient_descent_sublinear_complexity =*
*gradient_descent_function_value_gap_bound*

**lemmas** *gradient_descent_gradient_residual_complexity =*
*gradient_descent_exists_small_gradient_norm_sq*

**lemmas** *projected_gradient_descent_sublinear_complexity =*
*projected_gradient_descent_function_value_gap_bound*

**lemmas** *projected_gradient_descent_mapping_residual_complexity =*
*projected_gradient_descent_exists_small_mapping_norm_sq_to_minimizer*

**lemmas** *projected_gradient_descent_epsilon_stationarity_complexity =*
*projected_gradient_descent_exists_epsilon_residual_to_minimizer_product*

**lemmas** *projected_gradient_mapping_optimality_certificate =*
*projected_gradient_mapping_zero_iff_first_order_condition*

**lemmas** *projected_gradient_mapping_global_min_certificate =*
*projected_gradient_mapping_zero_imp_global_min_on*

**lemmas** *projected_gradient_residual_optimality_certificate =*
*projected_gradient_residual_zero_iff_first_order_condition*

**lemmas** *projected_gradient_residual_global_min_certificate =*
*projected_gradient_residual_zero_imp_global_min_on*

**lemmas** *lipschitz_mean_value_smooth_convex_bridge =*
*line_mean_value_and_lipschitz_gradient_imp_smooth_convex_on_twice*

**lemmas** *lipschitz_mean_value_projected_gradient_descent_sublinear_complexity*
*=*

line_mean_value_lipschitz_projected_gradient_descent_function_value_gap_bound

**lemmas** *strongly_convex_distance_gap_certificate =*
*strongly_smooth_convex_global_min_distance_gap*

**lemmas** *projected_gradient_descent_linear_distance_complexity =*
*projected_gradient_descent_distance_sq_linear_rate*

**lemmas** *projected_gradient_descent_linear_function_value_complexity =*
*projected_gradient_descent_function_value_linear_rate*

### 24.16.4 Complete recommended theorem surface

This final group collects the recommended high-level API of the entry. It is useful as a compact overview of the main reusable results.

**lemmas** *first_order_methods_public_api =*
*main_first_order_infrastructure*
*main_smooth_descent_infrastructure*
*lipschitz_smoothness_interface*
*lipschitz_mean_value_bridge_interface*
*lipschitz_mean_value_algorithmic_results*
*main_projection_infrastructure*
*main_gradient_descent_theorems*
*main_projected_gradient_descent_theorems*
*main_lipschitz_bridge_theorems*
*main_projected_gradient_mapping_theorems*
*main_epsilon_stationarity_theorems*
*main_strong_convexity_and_linear_rate_theorems*
*main_example_theorems*

**lemmas** *first_order_methods_citation_surface =*
*gradient_descent_sublinear_complexity*
*gradient_descent_gradient_residual_complexity*
*projected_gradient_descent_sublinear_complexity*
*projected_gradient_descent_mapping_residual_complexity*
*projected_gradient_descent_epsilon_stationarity_complexity*
*projected_gradient_mapping_optimality_certificate*
*projected_gradient_mapping_global_min_certificate*
*projected_gradient_residual_optimality_certificate*
*projected_gradient_residual_global_min_certificate*
*lipschitz_mean_value_smooth_convex_bridge*
*lipschitz_mean_value_projected_gradient_descent_sublinear_complexity*
*strongly_convex_distance_gap_certificate*
*projected_gradient_descent_linear_distance_complexity*
*projected_gradient_descent_linear_function_value_complexity*

**lemmas** *bounded_interval_quadratic_example_results =*
*bounded_interval_real_iff*
*zero_mem_bounded_interval_real*

```
closed_bounded_interval_real
convex_bounded_interval_real
quadratic_real_smooth_convex_on_bounded_interval
quadratic_real_strongly_smooth_convex_on_bounded_interval
quadratic_real_global_min_on_bounded_interval_zero
quadratic_real_unique_global_min_on_bounded_interval
bounded_interval_quadratic_projected_gradient_step_unfold
bounded_interval_quadratic_projected_gradient_mapping_unfold
bounded_interval_quadratic_projected_gradient_residual_unfold
bounded_interval_quadratic_projected_gradient_descent_function_value_gap_bound
bounded_interval_quadratic_projected_gradient_descent_distance_linear_rate
bounded_interval_quadratic_projected_gradient_descent_exists_small_residual_sq
bounded_interval_quadratic_projected_gradient_descent_exists_epsilon_residual
bounded_interval_quadratic_projected_gradient_residual_zero_imp_zero
```

Recommended theorem groups for downstream users:

*first_order_methods_public_api first_order_methods_citation_surface*

The individual aliases in *first_order_methods_citation_surface* provide stable names for the main convergence, residual, optimality, Lipschitz-bridge, and linear-rate results of the entry.

**end**
**theory** *Examples_Using_Main_Results*
  **imports** *Main_Results*
**begin**

# 25 Using the public theorem interface

This theory illustrates how a downstream development can use the public interface collected in *Main_Results*.

The point of this file is deliberately modest: it does not import any of the internal proof layers directly. Instead, it treats *Main_Results* as the public entry point of the library and shows how client developments can refer to the stable theorem groups and aliases collected there.

This is useful as a regression test for the public API: if the internal proof files are later reorganized, the facts used below should remain available from *Main_Results*.

## 25.1 Accessing the recommended public surface

A client theory can first collect the complete recommended public API. This is mostly useful as a compact overview of the entry.

**lemmas** *client_public_api =*
  *first_order_methods_public_api*

For citation and downstream reuse, the smaller citation surface is usually the more appropriate entry point.

**lemmas** *client_citation_surface =*
*first_order_methods_citation_surface*

## 25.2 Smooth convex first-order infrastructure

The following groups demonstrate that the analytic and descent infrastructure is available without importing the lower-level theories directly.

**lemmas** *client_first_order_infrastructure =*
*main_first_order_infrastructure*

**lemmas** *client_smooth_descent_infrastructure =*
*main_smooth_descent_infrastructure*

**lemmas** *client_projection_infrastructure =*
*main_projection_infrastructure*

## 25.3 Gradient descent results

A downstream user interested in unconstrained gradient descent can reuse the stable sublinear convergence and gradient-residual complexity aliases.

**lemmas** *client_gradient_descent_sublinear_rate =*
*gradient_descent_sublinear_complexity*

**lemmas** *client_gradient_descent_residual_rate =*
*gradient_descent_gradient_residual_complexity*

**lemmas** *client_gradient_descent_results =*
*main_gradient_descent_theorems*

## 25.4 Projected-gradient descent results

For constrained smooth convex optimization, the projected-gradient descent layer exposes the standard function-value convergence theorem.

**lemmas** *client_projected_gradient_descent_sublinear_rate =*
*projected_gradient_descent_sublinear_complexity*

**lemmas** *client_projected_gradient_descent_results =*
*main_projected_gradient_descent_theorems*

## 25.5 Projected-gradient mapping and residual certificates

The projected-gradient mapping is the constrained stationarity residual used by this entry. The following aliases expose the main optimality and residual complexity certificates from the public interface.

**lemmas** *client_projected_gradient_mapping_optimality =*
*projected_gradient_mapping_optimality_certificate*

**lemmas** *client_projected_gradient_mapping_global_min =*
*projected_gradient_mapping_global_min_certificate*

**lemmas** *client_projected_gradient_residual_optimality =*
*projected_gradient_residual_optimality_certificate*

**lemmas** *client_projected_gradient_residual_global_min =*
*projected_gradient_residual_global_min_certificate*

**lemmas** *client_projected_gradient_mapping_residual_complexity =*
*projected_gradient_descent_mapping_residual_complexity*

**lemmas** *client_projected_gradient_epsilon_stationarity =*
*projected_gradient_descent_epsilon_stationarity_complexity*

**lemmas** *client_projected_gradient_mapping_results =*
*main_projected_gradient_mapping_theorems*

**lemmas** *client_projected_gradient_epsilon_stationarity_results =*
*main_epsilon_stationarity_theorems*

## 25.6 Strong convexity and linear-rate results

The strongly convex layer provides distance-gap certificates and linear convergence estimates for projected-gradient descent.

**lemmas** *client_strongly_convex_distance_gap =*
*strongly_convex_distance_gap_certificate*

**lemmas** *client_projected_gradient_linear_distance_rate =*
*projected_gradient_descent_linear_distance_complexity*

**lemmas** *client_projected_gradient_linear_function_value_rate =*
*projected_gradient_descent_linear_function_value_complexity*

**lemmas** *client_strong_convexity_and_linear_rate_results =*
*main_strong_convexity_and_linear_rate_theorems*

## 25.7 Concrete quadratic examples

The example layer can also be accessed from *Main_Results*. These facts show how the abstract theorem surface is instantiated on a simple one-dimensional quadratic objective and on the nonnegative half-line constraint.

**lemmas** *client_quadratic_examples =*
*main_example_theorems*

**lemmas** *client_quadratic_smooth_convex_example =*
*quadratic_real_smooth_convex_on_UNIV*

**lemmas** *client_quadratic_strongly_smooth_convex_example =*
*quadratic_real_strongly_smooth_convex_on_UNIV*

**lemmas** *client_quadratic_global_min_example =*
*quadratic_real_global_min_on_zero*

**lemmas** *client_nonnegative_quadratic_pgd_rate_example =*
*nonnegative_quadratic_projected_gradient_descent_function_value_gap_bound*

**lemmas** *client_nonnegative_quadratic_epsilon_stationarity_example =*
*nonnegative_quadratic_projected_gradient_descent_exists_epsilon_residual*

**lemmas** *client_nonnegative_quadratic_residual_optimality_example =*
*nonnegative_quadratic_projected_gradient_residual_zero_imp_zero*

## 25.8 A compact downstream package

A downstream project may collect only the parts of the public API that it needs. The following bundle is an example of a small client-facing package for smooth convex projected-gradient descent.

**lemmas** *client_projected_gradient_descent_package =*
*client_smooth_descent_infrastructure*
*client_projection_infrastructure*
*client_projected_gradient_descent_sublinear_rate*
*client_projected_gradient_mapping_optimality*
*client_projected_gradient_mapping_global_min*
*client_projected_gradient_mapping_residual_complexity*
*client_projected_gradient_epsilon_stationarity*
*client_strongly_convex_distance_gap*
*client_projected_gradient_linear_distance_rate*
*client_projected_gradient_linear_function_value_rate*

This file intentionally proves no new mathematical theorem. Its purpose is to document and test the public theorem surface exposed by *Main_Results*. In particular, it shows that client developments can work with the stable aliases and recommended theorem groups without depending on the internal organization of the proof files.

## 25.9 Lipschitz-gradient bridge

The mean-value bridge shows how a client can connect primitive Lipschitz-gradient assumptions to the smooth-convex interface used by the algorithmic convergence theorems.

**lemmas** *client_lipschitz_mean_value_smooth_convex_bridge =*
*lipschitz_mean_value_smooth_convex_bridge*

**lemmas** *client_lipschitz_mean_value_projected_gradient_rate =*

*lipschitz_mean_value_projected_gradient_descent_sublinear_complexity*

**end**
**theory** *Projected_Gradient_Descent*
**imports** *Main_Results*
**begin**

# 26 Projected Gradient Descent

This is the umbrella theory of the entry.

It imports the complete public development for smooth convex first-order optimization through *Main_Results*. The imported material includes reusable interfaces for gradients, convex first-order reasoning, smooth upper bounds, abstract descent arguments, projection geometry, projected-gradient descent, projected-gradient mappings, residual convergence certificates, strong convexity, linear-rate refinements, Lipschitz-smoothness bridges, and quadratic examples.

The recommended public theorem surface is collected in *Main_Results*. Downstream developments should normally import *Main_Results* when they want to reuse the library, rather than depending on the internal organization of the individual proof files.

**end**

# References


[1] A. Beck. *First-Order Methods in Optimization.* SIAM, 2017.

[2] D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, 1999.

[3] D. Bryant. Unconstrained optimization. *Archive of Formal Proofs*, July 2025. `https://isa-afp.org/entries/Unconstrained_Optimization.html`, Formal proof development.

[4] C. Li, Z. Wang, W. He, Y. Wu, S. Xu, and Z. Wen. Formalization of convergence rates of four first-order algorithms for convex optimization. *Journal of Automated Reasoning*, 69(28), 2025.

[5] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course.* Kluwer Academic Publishers, 2004.

[6] Y. Nesterov. *Lectures on Convex Optimization.* Springer, 2018.