Classification of Smooth Unconstrained Optimization Problems

Daniel Henderson

May 24, 2025

Abstract

We study the local convexity properties of a benchmark suite of smooth, unconstrained minimization problems drawn from the `OptimizationProblems.jl` [MOS] Julia package. For each problem, we review its origin, present the analytic form of the objective $f : \mathbb{R}^n \to \mathbb{R}$ and the standard starting point $\boldsymbol{x}_0$. We introduce a sampling-based procedure to classify the critical-point structure and verify the positive definiteness of the Hessian in a neighborhood of a strict local minimizer. Numerical experiments confirm that while some test functions exhibit strong local convexity, others contain narrow regions of non-convexity that can slow down standard schemes, e.g. (TODO: add "tacking" ref. here and complete results statement) **Keywords:** benchmarking, numerical optimization, stationary points, saddle points,

Contents

1 Introduction

Understanding the local convexity of an objective function near strict local minimizers is fundamental to the design and convergence analysis of smooth optimization algorithms. Strong local convexity near a minimizer often guarantees rapid convergence of gradient-based and Newton-type methods. In contrast, narrow or non-convex regions can result in slow progress or convergence to saddle points. We select benchmark problems from the *Constrained and Unconstrained Testing Environment* SIF (CUTE) [**?**]. We sample the objective of each problem to reveal long, narrow valleys of weak/zero curvature surrounding strict minimizers. Additinally, we analyze the spectral pattern of the Hessian of f at critical points obtained from seeding standard schemes from randomly sampled starting locations surrounding each problems initial iterate. Moreover, we replicate the findings of Kok et al. for the generalized Rosenbrock problem [KS09] and extend their analysis to a larger set of benchmark problems.

Our contributions are:

- literature survey of known techniques for classifying critical points.
- a sampling-based classification algorithm for sampling the curvature within a neighborhood of ϵ-first-order stationary points.
- systematically compares the local convexity profiles of standard benchmark problems in continuous optimization. (ref artificial article)
- supplemental material containing analytical expressions f, and the accepted initial iterate $\boldsymbol{x}_0$, as well as a discussion of each problem's provenance and known properties from the literature.

Introduction overview. Section 1.1 establishes the notation adopted throughout the manuscript. Section 1.2 formulates the smooth unconstrained optimization problem, while Section 1.3 recasts it as a gradient-flow dynamical system, proves well-posedness, and shows that its equilibria coincide with the stationary points of the objective. Further phase-space analysis yields a constructive method for solving the original problem. The concept of an *optimization scheme* is then formalized, accompanied by illustrative examples that recover classical results for gradient descent. Finally, Section 1.5 enumerates the benchmark test functions considered in this study.

Organization of the paper. Section 2 develops a theoretical framework for classifying critical points using dynamical systems, Morse theory, and spectral analysis of $\nabla^2 f(\boldsymbol{x}^*)$. Section 3 details our problem selection criteria and introduces the benchmark problems under study. In Section 4, we present numerical experiments for each problem, describe our methodology for determining strict local minimizers $\boldsymbol{x}^*$, and outline our sampling-based convexity classification algorithm. Finally, Section 5 summarizes our findings, offers practical recommendations, and discusses directions for future work.

1.1 Preliminaries

We use the notation defined in the Appendix 6 as used by Nocedal and Wright [NW06].

Assumption 1.1 (Standing Assumptions)**.**

1. **Domain.** $\Omega \subset \mathbb{R}^n$ is path-connected, bounded, and open, and its boundary $\partial\Omega$ is a C^k $(k \geq 1)$ embedded hypersurface.
2. **Manifold structure.** The closure $\overline{\Omega} = \Omega \cup \partial\Omega$ is compact and carries the structure of a C^k manifold with boundary $\partial\Omega$.
3. **Objective function.** The objective satisfies $f \in C^k(\overline{\Omega})$ with $k \geq 2$, so, f, ∇f, and $\nabla^2 f$ extend continuously to $\partial\Omega$.

TODO - Add ρ-Hessian Lipschitz and ℓ-smooth assumptions? - Critical Point Non-Degeneracy? - Path-Connectedness of Ω? Allows us to assume that f is convex of on sublevel sets... needed for morse theory and further results in dynamical systems, e.g., gradient flow retracts sub-level sets onto unstable manifolds..

1.2 Unconstrained Optimization Problem

Consider the general form of a **smooth unconstrained optimization problem**

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x}), \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the optimization variable and $f : \mathbb{R}^n \to \mathbb{R}$ is a twice continuously differentiable objective function.
If a solution of (1) exists, then we denote the **optimal value** and **optimal solution** as

$$\boldsymbol{x}^* \in \arg\min_{\boldsymbol{x}} f(\boldsymbol{x}) \quad \text{and} \quad f^* = \min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x}),$$

respectively. The optimal value f^* is the minimum value of f over the entire domain $\mathbb{R}^n$.

In practice, we choose Ω in accordance with our standing assumptions 1.1 to contain all iterates and gradient-flow trajectories under consideration. By Heine–Borel, the closed and bounded set $\overline{\Omega}$ is compact, so the extreme-value theorem guarantees that the continuous real-valued function f attains its maximum and minimum on $\overline{\Omega}$; where continuity of f on $\overline{\Omega}$ is guaranteed by the standing C^k assumption of f. So by our choice of Ω, we can make global optimality claims about the optimal value within our chosen domain.

A point satisfying the first-order optimality condition is called a *critical point*. Such a point can be a local minimizer, local maximizer, or saddle point. We classify critical points as follows (cf. [NW06]):

- A *local minimizer* if there exists a neighborhood N of $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in N$.
- A *strict local minimizer* if there exists a neighborhood N of $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}^*) < f(\boldsymbol{x})$ for all $\boldsymbol{x} \in N \setminus \{\boldsymbol{x}^*\}$.

- An *isolated local minimizer* if there exists a neighborhood N of $\boldsymbol{x}^*$ such that $\boldsymbol{x}^*$ is the unique local minimizer in N.
- A *global minimizer* if $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathbb{R}^n$.

Since we focus on nonlinear objective functions, global optimality generally cannot be guaranteed. However, local optimality of a critical point $\boldsymbol{x}^*$ is verified using the following conditions:

$$\begin{aligned} \text{Necessary:} \quad & \nabla f(\boldsymbol{x}^*) = 0 \ \text{ and } \ \nabla^2 f(\boldsymbol{x}^*) \succeq 0 \\ \text{Sufficient:} \quad & \nabla^2 f(\boldsymbol{x}^*) \succ 0. \end{aligned}$$

The necessary condition for $\boldsymbol{x}^*$ to be a local minimizer of f is that $\boldsymbol{x}^*$ satisfies the first-order and second-order optimality conditions. The *second-order* optimality condition states that the Hessian of f is positive semidefinite at $\boldsymbol{x}^*$. If the Hessian is strictly positive definite at $\boldsymbol{x}^*$, it is both a necessary and a sufficient condition that $\boldsymbol{x}^*$ is a strict local minimizer. It holds that strict local minimizers are isolated, corresponding precisely to points where f is strictly convex.

1.3 Continuous Model

We reinterpret the minimization problem (1) as the search for equilibria of the *gradient flow* dynamical system. Interpreting our problem as a dynamical system allows us to:

- exploit geometric structure when analyzing the critical-point landscapes;
- design time-discretisations of the gradient flow ODE (GF) and construct error bounds using the continuous model;
- leverage the stability theory of autonomous ODEs on compact manifolds.

In this section, we establish a link between the minimization of f and the trajectories of a dynamical system.

Definition 1. The *gradient-flow* dynamical system is defined by the IVP

$$\boldsymbol{\gamma}'(t) = -\nabla f\big(\boldsymbol{\gamma}(t)\big) \ \text{ subject to } \boldsymbol{\gamma}(0) = \boldsymbol{x}_0 \in \overline{\Omega} \tag{GF}$$

An integral curve $\boldsymbol{\gamma} : [0, \infty) \to \overline{\Omega}$ satisfying the *gradient-flow* IVP is a gradient *flow-line*, or, *trajectory*.

Suppose the initial point $\boldsymbol{x}_0$ is a critical point, then the gradient flow ODE is satisfied by the constant trajectory $\boldsymbol{\gamma}(t) = \boldsymbol{x}_0$ for any time t. Notice that constant $\boldsymbol{\gamma}(t)$ implies $\boldsymbol{\gamma}'(t) = \mathbf{0}$, substituting into the gradient-flow ODE asserts that $-\nabla f(\boldsymbol{x}_0) = \mathbf{0}$, which holds true since $\boldsymbol{x}_0$ is a critical point (i.e., $\nabla f(\boldsymbol{x}_0) = \mathbf{0}$). Consequently, the existence of a solution holds when $\boldsymbol{x}_0$ is a critical point and such points correspond to stationary equilibria in the gradient flow ODE phase space. Now we show equation (GF) is well-posed for all $\boldsymbol{x}_0 \in \overline{\Omega}$.

Let $\boldsymbol{\gamma}_{\boldsymbol{x}_0}$ be any gradient *flow-line* starting at some point $\boldsymbol{x}_0$ in $\overline{\Omega}$. Note that the trajectory $\boldsymbol{\gamma}_{\boldsymbol{x}_0}$ is driven by the steepest descent direction of the objective function f. But f is bounded below by the minimum value f^*, so the trajectory $\boldsymbol{\gamma}_{\boldsymbol{x}_0}$ cannot escape the compact set $\overline{\Omega}$ in finite time. Indeed, we will show that

the trajectory $\boldsymbol{\gamma}_{\boldsymbol{x}_0}$ is guaranteed to converge to a critical point of f in $\overline{\Omega}$. But first we must establish the IVP (GF) is well-posed for all $\boldsymbol{x}_0 \in \overline{\Omega}$.

theorem (Existence)**.** For every $\boldsymbol{x}_0 \in \overline{\Omega}$ the IVP (GF) admits a unique solution $\boldsymbol{\gamma}_{\boldsymbol{x}_0} \in C^1\big([0,\infty);\overline{\Omega}\big)$ that exists for all time $t \geq 0$.

Sketch. Our standing assumptions imply that $-\nabla f$ is globally Lipschitz on the compact set $\overline{\Omega}$, i.e., $-f$ is ℓ-smooth since ∇f is continuous on $\overline{\Omega}$. (ref appendix for ℓ-smoothness definition). The Picard-Lindelöf theorem guarantees the existence of a unique solution $\boldsymbol{\gamma}_{\boldsymbol{x}_0}$ to the IVP (GF) for some finite amount of time. The following Theorem shows that the solution exists for all time $t \geq 0$, namely, that the trajectory $\boldsymbol{\gamma}_{\boldsymbol{x}_0}(t)$ and remains in the compact set $\overline{\Omega}$ for all $t \geq 0$. □

Remark. In our proof of Theorem we assumed f and $-\nabla f$ are defined on all of $\mathbb{R}^n$, so that trajectories are well defined globally; we only observe them on our chosen domain $\overline{\Omega}$. We commit such theoretical shortcuts knowingly, as our shortcut is pathed when we introduce the assumption that Ω is path-connected, which ensures convexity of f on the sublevel set $L^-_{f(\boldsymbol{x}_0)}$, so that the gradient flow ODE is well-defined on the entire sublevel set.

The following lemma characterizes the monotonicity of f along the trajectory $\boldsymbol{\gamma}_{\boldsymbol{x}_0}$.

Lemma A. *Along any trajectory $\boldsymbol{\gamma}_{\boldsymbol{x}_0}(t)$ one has*

$$\frac{d}{dt} f\big(\boldsymbol{\gamma}_{\boldsymbol{x}_0}(t)\big) = -\|\nabla f\big(\boldsymbol{\gamma}_{\boldsymbol{x}_0}(t)\big)\|^2 \;\leq\; 0, \quad t \geq 0. \tag{2}$$

Hence $f \circ \boldsymbol{\gamma}_{x_0}$ is non-increasing in t, and strictly decreasing whenever $\nabla f(\boldsymbol{\gamma}_{\boldsymbol{x}_0}(t)) \neq 0$.

Proof. Differentiating the composition $f \circ \boldsymbol{\gamma}_{\boldsymbol{x}_0}$ w.r.t. t using the chain rule yields

$$\begin{aligned}
\frac{d}{dt} f\big(\boldsymbol{\gamma}_{\boldsymbol{x}_0}(t)\big) &= \nabla f\big(\boldsymbol{\gamma}_{\boldsymbol{x}_0}(t)\big) \cdot \frac{d}{dt}\boldsymbol{\gamma}_{\boldsymbol{x}_0}(t) \\
&= \nabla f\big(\boldsymbol{\gamma}_{\boldsymbol{x}_0}(t)\big) \cdot \boldsymbol{\gamma}'_{\boldsymbol{x}_0}(t) \\
&= \nabla f\big(\boldsymbol{\gamma}_{\boldsymbol{x}_0}(t)\big) \cdot \big(-\nabla f\big(\boldsymbol{\gamma}_{\boldsymbol{x}_0}(t)\big)\big) \\
&= \langle \nabla f\big(\boldsymbol{\gamma}_{\boldsymbol{x}_0}(t)\big), -\nabla f\big(\boldsymbol{\gamma}_{\boldsymbol{x}_0}(t)\big)\rangle \\
&= -\langle \nabla f\big(\boldsymbol{\gamma}_{\boldsymbol{x}_0}(t)\big), \nabla f\big(\boldsymbol{\gamma}_{\boldsymbol{x}_0}(t)\big)\rangle \\
&= -\|\nabla f\big(\boldsymbol{\gamma}_{\boldsymbol{x}_0}(t)\big)\|^2 \\
&\leq 0.
\end{aligned}$$

A direct consequence of the positive-definiteness property of a norm is that the final inequality is strict for all points $\boldsymbol{\gamma}_{\boldsymbol{x}_0}(t)$ that aren't critical. □

theorem (Properties of the gradient flow)**.** Let $\boldsymbol{x}_0 \in \overline{\Omega}$ and let $\boldsymbol{\gamma}_{\boldsymbol{x}_0} : [0,\infty) \to \overline{\Omega}$ be the unique solution of (GF). Then

1. The flow map $(\boldsymbol{x}_0, t) \mapsto \boldsymbol{\gamma}_{\boldsymbol{x}_0}(t)$ is continuous in both variables.
2. Every accumulation point of $\boldsymbol{\gamma}_{\boldsymbol{x}_0}(t)$ as $t \to \infty$ belongs to $\mathrm{crit}(f)$.

Proof. Suppose, to the contrary, that some accumulation point $\boldsymbol{x}^*$ satisfies $\nabla f(\boldsymbol{x}^*) \neq 0$. FIXME: After re-structuring assumptions; reference prior remark. □

1.4 Optimization Schemes

We introduce a *scheme* for solving a *unconstrained optimization problem* based on the *gradient flow* dynamical system 1.

Definition 2. An **optimization scheme** is a one-parameter family of iteration operators $T_h : \overline{\Omega} \to \overline{\Omega}$, indexed by a step size $h \in (0, h_0]$ where h_0 is constant, that generates an iterative sequence using the rule

$$\boldsymbol{x}_{k+1} = T_h(\boldsymbol{x}_k) \text{ for } k = 0, 1, 2, \dots \tag{3}$$

starting from an initial point $\boldsymbol{x}_0 \in \overline{\Omega}$. The scheme is well-defined such that the triplet $(\boldsymbol{x}_0, h, T_h)$ satisfy:

- Consistency: T_h is *consistent of order* $p \geq 1$ with the flow (GF) if

  $$\big\| T_h(\boldsymbol{x}) - \boldsymbol{x} + h\,\nabla f(\boldsymbol{x}) \big\| \;=\; \mathcal{O}\big(h^{p+1}\big) \quad \text{as } h \to 0,\ \forall\, \boldsymbol{x} \in \overline{\Omega}.$$

  A consistent scheme approximates the continuous gradient flow w/ a local error of $\mathcal{O}(h^{p+1})$ committed at each step k; where p is the global order. Consistent schemes reproduce the exact optimality condition in the limit as $h \to 0$.

- Stability: Let $\boldsymbol{x}^* \in \operatorname{crit}(f)$ be a *strict* local minimizer. The family $\{T_h\}$ is *stable* at $\boldsymbol{x}^*$ if

  $$\exists\, c > 0,\ h_{\max} > 0,\ \forall\, h \in (0, h_{\max}] : \qquad \rho\big(DT_h(\boldsymbol{x}^*)\big) \leq 1 - ch,$$

  where $\rho(\cdot)$ denotes the spectral radius. Equivalently,

  $$\|T_h(\boldsymbol{x}) - T_h(\boldsymbol{y})\| \leq (1 - ch)\|\boldsymbol{x} - \boldsymbol{y}\|$$

  for all $\boldsymbol{x}, \boldsymbol{y}$ in some neighborhood of $\boldsymbol{x}^*$. A stable scheme is contractive in a neighborhood of $\boldsymbol{x}^*$, meaning that the distance between iterates shrinks by at least a factor of $1 - ch$ at each step k.

If a scheme is order-p consistent and locally contractive at a strict minimizer $\boldsymbol{x}^*$, then for any fixed $h \in (0, h_{\max}]$ the iterates satisfy

$$\|\boldsymbol{x}_k - \boldsymbol{x}^*\| \leq (1 - ch)^k \|\boldsymbol{x}_0 - \boldsymbol{x}^*\|,$$

and hence $\boldsymbol{x}_k \to \boldsymbol{x}^*$ as $k \to \infty$.

Table 1 summarises four standard choices for T_h, stating their consistency order and the conditions under which local contractivity holds; see, e.g., Nocedal-Wright [NW06] for detailed discussions of these schemes.

Scheme	**Iteration operator** $T_h(\boldsymbol{x})$	**Order** p	**Stability near** $\boldsymbol{x}^*$
Gradient descent (GD)	$\boldsymbol{x} - h\,\nabla f(\boldsymbol{x})$	1	$\nabla^2 f(\boldsymbol{x}^*) \succeq \mu I \succ 0,\ 0 < h \leq 1/\ell$
Newton (NM)	$\boldsymbol{x} - h\,\nabla^2 f(\boldsymbol{x})^{-1} \nabla f(\boldsymbol{x})$	2	$\nabla^2 f(\boldsymbol{x}^*) \succ 0,\ 0 < h \leq 1$
Quasi–Newton (QN)	$\boldsymbol{x} - h\,B_k \nabla f(\boldsymbol{x}) \quad (B_k \to \nabla^2 f(\boldsymbol{x}^*)^{-1})$	1	Same as NM once $B_k \succ 0$ and $\|B_k\|$ is bounde
Trust region (TR)	$\boldsymbol{x} + \arg\min_{\|\boldsymbol{\tau}\| \leq \Delta} m_{\boldsymbol{x}}(\boldsymbol{\tau})$	2	Same as NM for sufficiently small Δ

Table 1: Representative optimization schemes for problem (1). (ℓ denotes the Lipschitz constant of ∇f on $\overline{\Omega}$).

Analysis of Gradient Descent (GD)

theorem. Assume f is ℓ-smooth and α-strongly convex and that $\epsilon > 0$. If we iterate the gradient descent *scheme* with $h = h_0 = \frac{1}{\ell}$ held fixed, i.e.,

$$T_h(\boldsymbol{x}_k) = \boldsymbol{x}_k - \frac{1}{\ell}\nabla f(\boldsymbol{x}_k),$$

then $d(x_k, x^*) \leq \epsilon$ for all $k > K$ where K is chosen to satisfy

$$\frac{2\ell}{\alpha} \cdot \log\left(\frac{d(x_0, x^*)}{\epsilon}\right) \leq K.$$

Remark. Under *ℓ-smoothness* and *α-strong convexity* assumptions in a neighborhood Ω about $\boldsymbol{x}^*$, it may be shown directly from the above theorem above that the *GD scheme* converges linearly to the optimal solution $\boldsymbol{x}^*$ at a rate of

$$\frac{d(\boldsymbol{x}_k, \boldsymbol{x}^*)}{d(\boldsymbol{x}_{k-1}, \boldsymbol{x}^*)} \leq 1 - \frac{\alpha}{\ell}$$

where $d(\boldsymbol{x}_k, \boldsymbol{x}^*)$ is the distance between the current iterate $\boldsymbol{x}_k$ and the optimal solution $\boldsymbol{x}^*$. The convergence rate is linear in the sense that the distance between the current iterate and the optimal solution decreases by a factor of $1 - \frac{\alpha}{\ell}$ at each iteration. (Ref: TODO)

Remark. Convergence to a first-order stationary point trivally implies convergence to a ϵ-first-order stationary point. Similarly, convergence to a second-order stationary point trivially implies convergence to a ϵ-second-order stationary point.

theorem. Assume f is ℓ-smooth, then for any $\epsilon > 0$, if we iterate the GD scheme with $h = h_0 = \frac{1}{\ell}$ held fixed starting from $\boldsymbol{x}_0 \in \Omega$ where Ω is a neighborhood of $\boldsymbol{x}^*$, then the number of iterations K required to achieve the stopping condition $\|\nabla f(\boldsymbol{x}_k)\| \leq \epsilon$ is at most

$$\left\lceil \frac{\ell}{\epsilon^2}\left(f(\boldsymbol{x}_0) - f(\boldsymbol{x}^*)\right) \right\rceil$$

Remark. **TODO and Questions**

- State how we use theorems in when performing analysis from the results of our experimewts.
- What is the relationship between ℓ and α?
- In practice do we know how to compute ℓ and α?
- What is the relationship between ℓ and ρ?
- In practice do we know how to compute ℓ and ρ?

1.5 Benchmark Problems

We select a subset of the `OptimizationProblems.jl` [MOS] Julia package that support automatic differentiation (AD) natively through operator overloading. (TODO: Cite ForwardDiff.jl, Flux.jl, etc.) Each problem

is implemented as `ADNLPModel` instance, which is a wrapper around the `NLPModel` interface whose backend AD engine is configurable to support forward-mode or reverse-mode.

todo: - ref overleaf document and introduction section - enumerate the problems

2 Morse Theory

Note that Ω is a bounded subset of $\mathbb{R}^n$, so its closure $\overline{\Omega} = \Omega \cup \partial\Omega$ is a compact subset of $\mathbb{R}^n$, by the Heine-Borel Theorem. Also, the boundary $\partial\Omega$ is sufficiently smooth, so we can apply the theory of smooth manifolds. The closure $\overline{\Omega}$ is a compact subset of $\mathbb{R}^n$ and is a smooth manifold with boundary $\partial\Omega$. The interior Ω is an open subset of $\mathbb{R}^n$ and is a smooth manifold.

Move the above bit to 1.3

Definition 3. A point $\boldsymbol{x}^* \in \Omega$ is a **critical point** of f if the differentiable map $df_p : T_p\Omega \to \mathbb{R}$ is zero. (Here $T_p\Omega$ is a tangent space of the Manifold M at p.) The set of critical points of f is denoted by $\text{crit}(f)$.

Definition 4. A point $\boldsymbol{x}^* \in \Omega$ is a **non-degenerate critical point** of f if the Hessian $H_p f$ is non-singular.

Definition 5. The **index** of a *non-degenerate critical point* $\boldsymbol{x}^*$ is defined to be the dimension of the negative eigenspace of the Hessian $H_p f$.

- local minima at $\boldsymbol{x}^*$ have index 0.
- local maxima at $\boldsymbol{x}^*$ have index n.
- saddle points at $\boldsymbol{x}^*$ have index k where $0 < k < n$.

We reserve the integers $c_0, c_1, \ldots, c_i, \ldots, c_n$ to denote the number of critical points of index i.

Definition 6. A **Morse function** is a smooth function $f : \Omega \to \mathbb{R}$ such that all critical points of f are non-degenerate.

2.1 Morse Theory in a Metric Space

theorem. Let f be a Morse function on Ω, then the Euler characteristic of Ω is given by

$$\chi(\Omega) = \sum_{i=0}^{n} (-1)^i c_i$$

where c_i is the number of critical points of index i.

Remark. The Euler characteristic $\chi(\Omega)$ is a topological invariant of the manifold Ω and is independent of the choice of Morse function f. The Euler characteristic is a measure of the "shape" of the manifold and can be used to distinguish between different topological spaces. The Euler characteristic may be defined by the alternating sum of the Betti numbers b_i of the manifold Ω

$$\chi(\Omega) = \sum_{i=0}^{n} (-1)^i b_i$$

where b_i is the i-th Betti number of the manifold Ω.

theorem. (Sard's theorem) Let f be a Morse function on Ω, then the image $f(\text{crit}(f))$ has Lebesque measure zero in $\mathbb{R}$.

Remark. We state a particular instance of Sard's theorem for continuous scalar-valued functions f, which was first proved by Anothony P. Morse in 1939. The theorem asserts that the image of the critical points of a Morse function is a set of measure zero in $\mathbb{R}$. This means that the critical points of a Morse function are "rare" in the sense that they do not form a dense subset of the manifold Ω. Consequently, selecting $\boldsymbol{x} \in \Omega$ at random will almost never yeild a critical point of f.

Remark. The property that $\boldsymbol{x}^* \in \Omega$ being a *critical point* of a Morse function f is not dependent of the metric of $\Omega \subset \mathbb{R}^n$ (and consequently, the norm induced by the metric)

theorem. For a Morse function f on Ω, the gradient of f is either zero or orthogonal to the tangent space of the level set L_c at $\boldsymbol{x} \in L_c$.

The above theorem implies that at a stationary point $\boldsymbol{x^*}$, a level set $L_{\boldsymbol{x}^*}$ is reduced to a single point when $\boldsymbol{x^*}$ is a local minimum or maximum. Otherwise, the level set may have a singularity such as a self-intersection or a cusp.

Signed Distance Function

TODO: Add signed distance function theory notes and cite ref. The only reason to do this is to talk about geodesics, height parameterized flows, mean curvature flow, minimal surfaces and links to dimensionality reduction. Ref: Chapter 6? Stanley Osher and Ronald Fedkiw, Level Set Methods and Dynamic Implicit Surfaces, Springer, 2003.

Time Marching Level Set Methods TODO: Explain how level set methods would allow us to classify the critical point structure of f. And their role in low-dimensional topology. Explain curse of dimensionality and how level set methods can be used to classify the critical point structure of f. Chapter 7? Osher and Fedkiw, Level Set Methods and Dynamic Implicit Surfaces, Springer, 2003.
Discontinuous Galerkin Methods / Transportation Methods High dimensional sparse grid methods ... dependant on the sturcture of f. Reference recent work in optimal transport theory.

2.2 Machine Learning

Universal nonlinear approximation theorems and PINNS/Deep Learning methods. Have the luxury of economies of scale, however, they still suffer from the curse of dimensionality. Ref recent work on learning approximate level set flows in up to a dimension of 100. **TODO:** Interesting to think about learning the critical point structure of f via a neural network in small dimensions.. Ecspecially trace data, to help us better understand why some Quasi-Newton methods work better than others.

3 Problem Selection Criterion and Software Stack

4 Numerical Experiments

An uniformed sampling of the problem space Ω is performed in Code Listings 1.

5 Conclusion

reference introduction section.

6 Apendix

References

[KS09] Schalk Kok and Carl Sandrock. Locating and characterizing the stationary points of the extended rosenbrock function. *Evolutionary Computation*, 17(3):437–453, 2009. © 2009 by the Massachusetts Institute of Technology.

[MOS] Tangi Migot, Dominique Orban, and Abel Soares Siqueira. Optimizationproblems.jl: A collection of optimization problems in julia. If you use this software, please cite it using the metadata from this file.

[NW06] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, second edition, 2006.

Notation

We assume the following notation throughout

- $\|\cdot\|$ denotes the usual ℓ_2 norm for vectors $\boldsymbol{x}$ in $\mathbb{R}^n$ and $p = 2$ norm for matrices in $\mathbb{R}^{n\times m}$. i.e.,

$$\|x\| := \left(\sum_i x_i^2\right)^{1/2}$$
$$\|A\| := (\lambda_{\max}(A^\top A))^{1/2} = \max(\sigma(A))$$

- $\sigma(A) := \{\text{singular values of } A\}$.
- $A \in \mathbb{R}^{n\times n} \implies \sigma(A) = \{\text{eigenvalues of } A \text{ (i.e. spectrum)}\}$
- $\sigma_{\max}(A) := \max(\sigma(A))$ and $\sigma_{\min}(A) := \min(\sigma(A))$.
- $A \in \mathbb{R}^{n\times n} \implies \lambda_{\max}(A) := \sigma_{\max}(A)$ and $\lambda_{\min}(A) = \sigma_{\min}(A)$
- $\langle\cdot,\cdot\rangle$ denotes the usual inner product on $\mathbb{R}^n$, i.e.,

$$\langle \boldsymbol{x}, \boldsymbol{y}\rangle := \boldsymbol{x}^\top \boldsymbol{y} = \sum_{i=1}^n \boldsymbol{x}_i \boldsymbol{y}_i = \|\boldsymbol{x}\|\|\boldsymbol{y}\| \cos(\theta)$$

  where θ is the angle between $\boldsymbol{x}$ and $\boldsymbol{y}$.

- $\mathcal{B}_r(\boldsymbol{x}) := \{\boldsymbol{y} \in \mathbb{R}^n : \|\boldsymbol{y} - \boldsymbol{x}\| < r\}$ is the open ball of radius r centered at $\boldsymbol{x} \in \mathbb{R}^n$.
- $\text{crit}(f) = \{\boldsymbol{x}^* \in \mathbb{R}^n : \nabla f(\boldsymbol{x}^*) = 0\}$ is the set of critical points of f.

6.1 Definitions

TODO: integrated into article as needed. At a minimum labled and references

Definition 7. f is L**-Lipschitz** if $\forall\ \boldsymbol{x_1}, \boldsymbol{x_2}$

$$\exists\ L \geq 0\ :\ \|f(\boldsymbol{x_1}) - f(\boldsymbol{x_2})\| \leq L\|\boldsymbol{x_1} - \boldsymbol{x_2}\|$$

Definition 8. f has ℓ**-Lipschitz gradient**, or, f is ℓ**-smooth** if $\forall\ \boldsymbol{x_1}, \boldsymbol{x_2}$

$$\exists\ \ell \geq 0\ :\ \|\nabla f(\boldsymbol{x_1}) - \nabla f(\boldsymbol{x_2})\| \leq \ell\|\boldsymbol{x_1} - \boldsymbol{x_2}\|$$

Definition 9. f has ρ**-Lipschitz Hessian** if $\forall\ \boldsymbol{x_1}, \boldsymbol{x_2}$

$$\exists\ \rho \geq 0\ :\ \|\nabla^2 f(\boldsymbol{x_1}) - \nabla^2 f(\boldsymbol{x_2})\| \leq \rho\|\boldsymbol{x_1} - \boldsymbol{x_2}\|$$

Definition 10. f is **convex** if $\forall\ \boldsymbol{x_1}, \boldsymbol{x_2}$

$$\begin{aligned} f(\boldsymbol{x_2}) &\geq f(\boldsymbol{x_1}) + \langle \boldsymbol{x_2} - \boldsymbol{x_1}, \nabla f(\boldsymbol{x_1})\rangle \\ &= f(\boldsymbol{x_1}) + \nabla f(\boldsymbol{x_1})^T(\boldsymbol{x_2} - \boldsymbol{x_1}) \end{aligned}$$

Definition 11. f is **strictly convex** if

$$\begin{aligned} \exists\ \mu > 0\ &: \nabla^2 f \succeq \mu I \\ &\iff \lambda_{\min}(\nabla^2 f) \geq \mu > 0 \end{aligned}$$

Definition 12. f is α**-strongly convex** if $\forall\ \boldsymbol{x_1}, \boldsymbol{x_2} \exists\ \alpha > 0$ s.t.

$$\begin{aligned} &f(\boldsymbol{x_2}) \geq f(\boldsymbol{x_1}) + \langle \nabla f(\boldsymbol{x_1}), \boldsymbol{x_2} - \boldsymbol{x_1}\rangle + \frac{\alpha}{2}\|\boldsymbol{x_2} - \boldsymbol{x_1}\|^2 \\ &\iff \lambda_{\min}(\nabla^2 f(\boldsymbol{x})) \geq -\alpha. \end{aligned}$$

Definition 13. $\boldsymbol{x^*}$ is a **first-order stationary point** if $\|\nabla f(x^*)\| = 0$.

Definition 14. $\boldsymbol{x^*}$ is an ϵ**-first-order stationary point** if $\|\nabla f(x^*)\| \leq \epsilon$.

Definition 15. $\boldsymbol{x}^* \in \mathbb{R}^n$ is a **second-order stationary point** if $\|\nabla f(x^*)\| = 0$ and $\nabla^2 f(x^*) \succeq 0$.

Definition 16. if f has ρ-Lipschitz Hessian, $\boldsymbol{x}^* \in \mathbb{R}^n$ is a ϵ**-second-order stationary point** if

$$\|\nabla f(x^*)\| \leq \epsilon \text{ and } \nabla^2 f(x^*) \succeq -\sqrt{\rho\epsilon}$$

Remark. Note that the Hessian is not required to be positive definite, but it is required to have a small eigenvalue.

Definition 17. A point $\boldsymbol{x^*} \in \Omega$ is a **critical point** of f if the differentiable map $df_p : T_p\Omega \to \mathbb{R}$ is zero. (Here $T_p\Omega$ is a tangent space of the Manifold M at p.) The set of critical points of f is denoted by $\mathrm{crit}(f)$.

Definition 18. A point $\boldsymbol{x^*} \in \Omega$ is a **non-degenerate critical point** of f if the Hessian $H_p f$ is non-singular.

Definition 19. The **index** of a *non-degenerate critical point* $\boldsymbol{x^*}$ is defined to be the dimension of the negative eigenspace of the Hessian $H_p f$.

- local minima at $\boldsymbol{x}^*$ have index 0.
- local maxima at $\boldsymbol{x}^*$ have index n.
- saddle points at $\boldsymbol{x}^*$ have index k where $0 < k < n$.

We reserve the integers $c_0, c_1, \ldots, c_i, \ldots, c_n$ to denote the number of critical points of index i.

Remark. For each objective function f we are interested in determining the critical points of f

Remark. The **Morse function** is a smooth function $f : \Omega \to \mathbb{R}$ such that all critical points of f are non-degenerate.

6.2 Test Problems

1. **Extended Trigonometric function:**

$$f(x) = \sum_{i=1}^{n} \left((n - \sum_{j=1}^{n} \cos(x_j)) + i\,(1 - \cos(x_i)) - \sin(x_i) \right)^2,$$

$$x_0 = [0.2, 0.2, \ldots, 0.2].$$

2. **Generalized Rosenbrock function:**

$$f(x) = \sum_{i=1}^{n-1} \left(c(x_{i+1} - x_i^2)^2 + (1 - x_i)^2 \right),$$

$$x_0 = [-1.2, 1, -1.2, 1, \ldots, -1.2, 1], \quad c = 100.$$

3. **Extended Penalty function:**

$$f(x) = \sum_{i=1}^{n-1} (x_i - 1)^2 + \left(\sum_{j=1}^{n} x_j^2 - 0.25 \right)^2,$$

$$x_0 = [1, 2, \ldots, n].$$

4. **Perturbed Quadratic function:**

$$f(x) = \sum_{i=1}^{n} i x_i^2 + \frac{1}{100} \left(\sum_{i=1}^{n} x_i \right)^2$$

$$x_0 = [0.5, 0.5, \ldots, 0.5].$$

5. **Generalized Tridiagonal 1 function:**

$$f(x) = \sum_{i=1}^{n-1} (x_i + x_{i+1} - 3)^2 + (x_i - x_{i+1} + 1)^4,$$

$$x_0 = [2, 2, \ldots, 2].$$

6. **Generalized White & Holst function:**

$$f(x) = \sum_{i=1}^{n-1} c(x_{i+1} - x_i^3)^2 + (1 - x_i)^2, \quad c = 100,$$

$$x_0 = [-1.2, 1, -1.2, 1, \ldots, -1.2, 1].$$

7. **Generalized PSC1 function:**

$$f(x) = \sum_{i=1}^{n-1} \left(x_i^2 + x_{i+1}^2 + x_i x_{i+1} \right)^2 + \sin^2(x_i) + \cos^2(x_i),$$

$$x_0 = [3, 0.1, \ldots, 3, 0.1].$$

8. **Full Hessian FH1 function:**

$$f(x) = (x_1 - 3)^2 + \sum_{i=2}^{n} \left(x_1 - 3 - 2 \left(\sum_{j=1}^{i} x_j \right)^2 \right)^2,$$

$$x_0 = [0.01, 0.01, \ldots, 0.01].$$

9. **Full Hessian FH2 function:**

$$f(x) = (x_1 - 5)^2 + \sum_{i=2}^{n} \left(\sum_{j=1}^{i} x_j - 1 \right)^2,$$

$$x_0 = [0.01, 0.01, \ldots, 0.01].$$

10. **Perturbed Quadratic Diagonal function:**

$$f(x) = \left(\sum_{i=1}^{n} x_i \right)^2 + \sum_{i=1}^{n} \frac{i}{100} x_i^2,$$

$$x_0 = [0.5, 0.5, \ldots, 0.5].$$

11. **Quadratic QF1 function:**

$$f(x) = \frac{1}{2} \sum_{i=1}^{n} i x_i^2 - x_n,$$

$$x_0 = [1, 1, 1, \ldots, 1]$$

12. **Extended quadratic penalty QP1 function:**

$$f(x) = \sum_{i=1}^{n-1} \left(x_i^2 - 2 \right)^2 + \sum_{i=1}^{n} \left(x_i^2 - 0.5 \right)^2$$

$$x_0 = [1, 1, 1, \ldots, 1]$$

13. **Extended quadratic penalty QP2 function:**

$$f(x) = \left(\sum_{i=1}^{n} x_i^2 - 100 \right)^2 + \sum_{i=1}^{n-1} \left(x_i^2 - \sin x_i \right)^2$$

$$x_0 = [1, 1, 1, \ldots, 1]$$

14. **Quadratic QF2 function:**

$$f(x) = \frac{1}{2}\sum_{i=1}^{n} i\left(x_i^2 - 1\right)^2 - x_n$$

$$x_0 = [0.5, 0.5, 0.5, \ldots, 0.5]$$

15. **Extended Tridiagonal 2 function:**

$$f(x) = \sum_{i=1}^{n-1}(x_i x_{i+1} - 1)^2 + c(x_i + 1)(x_{i+1} + 1)$$

$$x_0 = [1, 1, 1, \ldots, 1], \quad c = 0.1$$

16. **FLETCBV3 function (CUTE):**

$$f(x) = \frac{1}{2}p(x_1^2 + x_n^2) + \frac{p}{2}\sum_{i=1}^{n-1}(x_i - x_{i+1})^2 - \sum_{i=1}^{n}\left(\frac{p(h^2+2)}{h^2}x_i + \frac{cp}{h^2}\cos(x_i)\right),$$

where:

$$p = \frac{1}{10^8}, \quad h = \frac{1}{n+1}, \quad c = 1,$$

$$x_0 = [h, 2h, \ldots, nh].$$

17. **FLETCHCR function (CUTE):**

$$f(x) = \sum_{i=1}^{n-1} c(x_{i+1} - x_i + 1 - x_i^2)^2$$

$$x_0 = [0, 0, 0, \ldots, 0], \quad c = 100$$

18. **BDQRTIC function (CUTE):**

$$f(x) = \sum_{i=1}^{n-4}(-4x_i + 3)^2 + \left(x_i^2 + 2x_{i+1}^2 + 3x_{i+2}^2 + 4x_{i+3}^2 + 5x_n^2\right)^2,$$

$$x_0 = [1, 1, \ldots, 1].$$

19. **TRIDIA function (CUTE):**

$$f(x) = \gamma(\delta x_1 - 1)^2 + \sum_{i=2}^{n} i\left(\alpha x_i - \beta x_{i-1}\right)^2$$

where:

$$\alpha = 2, \quad \beta = 1, \quad \gamma = 1, \quad \delta = 1$$

$$x_0 = [1, 1, 1, \ldots, 1]$$

20. **ARWHEAD function (CUTE):**

$$f(x) = \sum_{i=1}^{n-1} (-4x_i + 3) + \sum_{i=1}^{n-1} \left(x_i^2 + x_n^2\right)^2$$

$$x_0 = [1, 1, 1, \ldots, 1]$$

21. **NONDIA function (CUTE):**

$$f(x) = (x_1 - 1)^2 + \sum_{i=2}^{n} 100 \left(x_1 - x_{i-1}^2\right)^2,$$

$$x_0 = [-1, -1, -1, \ldots, -1]$$

22. **NONDQUAR function (CUTE):**

$$f(x) = (x_1 - x_2)^2 + \sum_{i=1}^{n-2} (x_i + x_{i+1} + x_n)^4 + (x_{n+1} + x_n)^2,$$

$$x_0 = [1, -1, , \ldots, 1, -1]$$

23. **EG2 function (CUTE):**

$$f(x) = \sum_{i=1}^{n-1} \sin(x_1 + x_i^2 - 1) + \frac{1}{2} \sin(x_n^2)$$

$$x_0 = [1, 1, 1, \ldots, 1]$$

24. **CURLY20 function (CUTE):**

$$f(x) = \sum_{i=1}^{n} \left(q_i^4 - 20q_i - 0.1q_i\right)$$

where:

$$q_i = \begin{cases} x_i + x_{i+1} + x_{i+2} + \cdots + x_{i+k}, & \text{if } i \leq n - k \\ x_i + x_{i+1} + x_{i+2} + \cdots + x_n, & \text{if } i > n - k \end{cases}$$

$$k = 20, \quad x_0 = \left[\frac{0.001}{n+1}, \ldots, \frac{0.001}{n+1}\right].$$

25. **Partially Perturbed Quadratic function:**

$$f(x) = x_1^2 + \sum_{i=1}^{n} \left[i x_i^2 + \frac{1}{100} \left(\sum_{j=1}^{i} x_j \right)^2 \right],$$

$$x_0 = [0.5, 0.5, \ldots, 0.5].$$

26. **Broyden Tridiagonal function:**

$$f(x) = \left(3x_1 - 2x_1^2\right)^2 + \sum_{i=2}^{n-1} \left(3x_i - 2x_i^2 - x_{i-1} - 2x_{i+1} + 1\right)^2 + \left(3x_n - 2x_n^2 - x_{n-1} + 1\right)^2,$$

$$x_0 = [-1, -1, \ldots, -1].$$

27. **Perturbed Tridiagonal Quadratic function:**

$$f(x) = x_1^2 + \sum_{i=2}^{n-1} i x_i^2 + (x_{i-1} + x_i + x_{i+1})^2,$$

$$x_0 = [0.5, 0.5, \ldots, 0.5].$$

28. **Staircase 1 function:**

$$f(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2,$$

$$x_0 = [1, 1, \ldots, 1].$$

29. **Staircase 2 function:**

$$f(x) = \sum_{i=1}^{n} \left[\left(\sum_{j=1}^{i} x_j \right) - i \right]^2,$$

$$x_0 = [0, 0, \ldots, 0].$$

30. **ENGVAL1 function (CUTE):**

$$f(x) = \sum_{i=1}^{n-1} \left(x_i^2 + x_{i+1}^2\right)^2 + \sum_{i=1}^{n-1} \left(-4x_i + 3\right),$$

$$x_0 = [2, 2, \ldots, 2].$$

31. **EDENSCH function (CUTE):**

$$f(x) = 16 + \sum_{i=1}^{n-1} \left[(x_i - 2)^4 + (x_i x_{i+1} - 2x_{i+1})^2 + (x_{i+1} + 1)^2\right]$$

$$x_0 = [0, 0, \ldots, 0].$$

32. **INDEF function (CUTE):**

$$f(x) = \sum_{i=2}^{n-1} \cos(2x_i - x_n - x_1) + \sum_{i=1}^{n} x_i,$$

$$x_0 = \left[\frac{1}{n+1}, \frac{2}{n+1}, \ldots, \frac{n}{n+1}\right].$$

33. **CUBE function (CUTE):**

$$f(x) = (x_1 - 1)^2 + \sum_{i=2}^{n} 100 \left(x_i - x_{i-1}^3\right)^2,$$

$$x_0 = [-1.2, 1, -1.2, 1, \dots].$$

34. **EXPLIN1 function (CUTE):**

$$f(x) = \sum_{i=1}^{n} \exp(0.1 x_i x_{i+1}) - 10 \sum_{i=1}^{n} i x_i,$$

$$x_0 = [0, 0, \dots, 0].$$

35. **BDEXP function (CUTE):**

$$f(x) = \sum_{i=1}^{n-2} (x_i + x_{i+1}) \exp(-x_{i+2}(x_i + x_{i+1}))$$

$$x_0 = [1, 1, \dots, 1].$$

36. **HARKERP2 function (CUTE):**

$$f(x) = \left(\sum_{i=1}^{n} x_i\right)^2 - \sum_{i=1}^{n} (x_i + \frac{1}{2} x_i^2) + 2 \sum_{i=2}^{n} \left(\sum_{i=j}^{n} x_i\right)^2,$$

$$x_0 = [1, 2, ..., n].$$

37. **GENHUMPS function (CUTE):**

$$f(x) = \sum_{i=1}^{n-1} \left(\sin(2x_i)^2 \sin(2x_{i+1})^2 + 0.05(x_i^2 + x_{i+1}^2)\right),$$

$$x_0 = [-506, 506.2, \dots, 506.2].$$

38. **MCCORMCK function (CUTE):**

$$f(x) = \sum_{i=1}^{n-1} \left(-1.5x_i + 2.5x_{i+1} + 1 + (x_i - x_{i+1})^2 + \sin(x_i + x_{i+1})\right)$$

$$x_0 = [1, 1, \dots].$$

39. **NONSCOMP function (CUTE):**

$$f(x) = (x_1 - 1)^2 + \sum_{i=2}^{n} 4(x_i - x_{i-1}^2)^2$$

$$x_0 = [3, 3, \dots].$$

40. **SINQUAD function (CUTE):**

$$f(x) = (x_1 - 1)^4 + \sum_{i=2}^{n-1} \left(\sin(x_i - x_n) - x_1^2 + x_i^2\right)^2 + (x_n^2 - x_1^2)^2,$$

$$x_0 = [0.1, 0.1, \dots].$$

41. **LIARWHD function (CUTE):**

$$f(x) = \sum_{i=1}^{n} 4\left(x_i^2 - x_1\right)^2 + \sum_{i=1}^{n} (x_i - 1)^2$$

$$x_0 = [4, 4, 4, \dots, 4]$$

42. **DIXON3DQ function (CUTE):**

$$f(x) = (x_1 - 1)^2 + \sum_{i=1}^{n-1} (x_i - x_{i+1})^2 + (x_n - 1)^2,$$

$$x_0 = [-1, -1, \dots, -1].$$

43. **COSINE function (CUTE):**

$$f(x) = \sum_{i=1}^{n-1} \cos(-0.5x_{i+1} + x_i^2),$$

$$x_0 = [1, 1, \dots, 1].$$

44. **SINE function:**

$$f(x) = \sum_{i=1}^{n-1} \sin(-0.5x_{i+1} + x_i^2),$$

$$x_0 = [1, 1, \dots, 1].$$

45. **BIGGSB1 function (CUTE):**

$$f(x) = (x_1 - 1)^2 + \sum_{i=1}^{n-1} (x_{i+1} - x_i)^2 + (1 - x_n)^2,$$

$$x_0 = [0, 0, \dots].$$

46. **Generalized Quartic function:**

$$f(x) = \sum_{i=1}^{n-1} x_i^2 + (x_{i+1} + x_i^2)^2,$$

$$x_0 = [1, 1, \dots].$$

47. **Full Hessian FH3 function:**

$$f(x) = \left(\sum_{i=1}^{n} x_i\right)^2 + \sum_{i=1}^{n}(x_i \exp(x_i) - 2x_i - x_i^2),$$

$$x_0 = [1, 1, \ldots, 1].$$

6.3 Code Listings

The following code is available in the source code repository https://github.com/danphenderson/numerical-optimization/blob/main/julia/saddles.jl

Code 1: Algorithm 16.5

```
using CSV, DataFrames, OptimizationProblems, ADNLPModels, NLPModels
using Random, Arpack, Optim, LineSearches
using LinearAlgebra, Statistics, Distributions

Random.seed!(1234)


function get_test_set(n::Int=40)
    """
    List of unconstrained scalable ADNLPModels within OptimizationProblems.jl
    """
    # meta = OptimizationProblems.meta
    # names_pb_vars = meta[
    # (meta.variable_nvar .== true) .& (meta.ncon .== 0) .& (5 .<= meta.nvar .<= 100),
    #     [:nvar, :name]
    # ]
    # test_set_generator = (
    #     eval(Meta.parse("ADNLPProblems.$(pb[:name])(n=$n)")) for pb in eachrow(names_pb_vars)
    # )
    # return [p for p in test_set_generator]
    return [
        "genrose",
        "arglina",
        "freuroth",
        "eg2",
        "cosine",
        "arglinb",
        "arglinc",
        "argtrig",
        "arwhead",
        "bdqrtic",
        "brownal",
        "broyden3d",
        "chnrosnb_mod",
        "cragglvy",
        "cragglvy2",
        "curly10",
        "curly10",
        "curly20",
        "curly30",
        "dixon3dq",
        "dqdrtic",
        "dqrtic",
        "edensch",
        "engval1",
        "errinros_mod",
        "extrosnb",
        "fletcbv2",
```

```
        "fletcbv3_mod",
        "fletchcr",
        "genhumps",
        "genrose_nash",
        "indef_mod",
        "integreq",
        "liarwhd",
        "morebv",
        "noncvxu2",
        "noncvxun",
        "nondia",
        "nondquar",
        "penalty1",
        "penalty2",
        "penalty3",
        "power",
        "quartc",
        "sbrybnd",
        "schmvett",
        "scosine",
        "sinquad",
        "tointgss",
        "tquartic",
        "tridia",
        "vardim"];
end

function get_problem(name::String, n::Int=40)
    """
    Get a problem by name
    """
    return eval(Meta.parse("ADNLPProblems.$(name)(n=$n)"))
end

function get_optim_options()
    """
    Using really stict conditions in low dimensions.

    Tacking earlier in routine may be obpuscated by extra
    iterations to obtain terminal convergence.
    """
    return Optim.Options(
        iterations = 10000000,
        g_abstol = eps(),
        store_trace = false, # Trace has a lot of useful stuff...
        show_trace = false,
        extended_trace = false,
    )
end

function build_sample_box(problem::ADNLPModel)
    """
    Box centered around the minimizer
    """
```

```
    x0 = problem.meta.x0
    scale_vector = 2 .* abs.(x0)
    scale_vector[scale_vector .<= 1.0] .= 1.0
    return scale_vector
end

function pull_sample(problem, box::Vector)
    """
    Pulls a sample uniformly from the box box surrounding x0
    """
    return rand.(Uniform.(-box, box))
end

function bfgs_linesearch()
    """
    Define the algorithm for the optimization
    """
    return BFGS(;
        alphaguess = LineSearches.InitialStatic(),
        linesearch = LineSearches.HagerZhang(),
        initial_invH = x -> Matrix{eltype(x)}(I, length(x), length(x)),
        manifold = Flat(),
    )
end

function gradiant_descent_linesearch()
    """
    Defines the Quasi-Newton algorithm for the optimization.

    P is our H. Currently P = \nabla^2 f(x) = I and we fallback
    to gradient descent.

    TODO: Accept P as an argument.
    """
    GradientDescent(;
        alphaguess = LineSearches.InitialHagerZhang(),
        linesearch = LineSearches.HagerZhang(),
        P = nothing,
        precondprep = (P, x) -> nothing
    )
end

function newton_trust_region()
    """
    Defines the Newton Trust Region algorithm for the optimization.
    """
    return NewtonTrustRegion(; initial_delta = 1.0,
        delta_hat = 100.0,
        eta = 0.1,
        rho_lower = 0.25,
        rho_upper = 0.75)
end

function eigs_hess(problem::ADNLPModel, x_cp::Vector)
```

```
    H = hess(problem, x_cp)
    λ, _ = eigs(H, nev=problem.meta.nvar - 1, maxiter=10000, which=:LM)
    λmin, _ = eigs(H, nev=1, maxiter=10000, which=:SR)
    push!(λ, pop!(λmin))
    return λ
end

function run_sample(problem::ADNLPModel, sample::Vector)
    """
    Run the optimization algorithm on the problem
    """
    # Define objective and in-place gradient aligning with optim's interface.
    x0 = sample
    f(x) = obj(problem, x)
    g!(G, x) = grad!(problem, x, G)
    h!(H, x) = hess!(problem, x, H)

    # Run the optimization
    res = Optim.optimize(
        f,
        g!,
        x0,
        bfgs_linesearch(),
        get_optim_options()
    )

    # Reset problem counters.
    NLPModels.reset!(problem)
    return res
end

function run(problem)
    box = build_sample_box(problem)
    df = DataFrame(
        "initial_objective" => Vector{Float64}(),
        "final_objective" => Vector{Float64}(),
        "g_residual" => Vector{Float64}(),
        "is_saddle" => Vector{Bool}(),
        "pos_curvature_directions" => Vector{Int}(),
        "neg_curvature_directions" => Vector{Int}(),
        "zero_curvature_directions" => Vector{Int}(),
        "max_lambda" => Vector{Float64}(),
        "min_lambda" => Vector{Float64}(),
        "median_lambda" => Vector{Float64}(),
        "iterations" => Vector{Int}(),
        "critical_point" => Vector{Vector{Float64}}(),
    )
    for _ in 1:1000
        sample = pull_sample(problem, box)
        fx0 = obj(problem, sample)
        try
            res = run_sample(problem, sample)
            if !res.f_converged
                continue
```

```
            end
            λ = eigs_hess(problem, res.minimizer)
            push!(df, (
                initial_objective = fx0,
                final_objective = res.minimum,
                g_residual = res.g_residual,
                is_saddle = λ[end] < 0,
                pos_curvature_directions = sum(λ .> 0),
                neg_curvature_directions = sum(λ .< 0),
                zero_curvature_directions = sum(abs.(λ) .< 1e-12),
                max_lambda = maximum(λ),
                min_lambda = minimum(λ),
                median_lambda = median(λ),
                iterations = res.iterations,
                critical_point = res.minimizer,
            ))
        catch
            continue
        end
    end
    return df
end

function unique_critical_points(df::DataFrame)
    """
    Compares the critical points locations to determine
    the number of unique critical points.
    """
    critical_points = df.critical_point
    critical_points = [round.(x, digits=4, base=2) for x in critical_points] # HACK d
    return length(unique(critical_points))
end

function run_all()
    test_set = get_test_set()
    mkpath("public/saddles/dim-40")
    for pb in test_set
        problem = get_problem(pb, 40)
        df = run(problem)
        CSV.write("public/saddles/dim-40/$(pb).csv", df)
        println("$(pb) has $(unique_critical_points(df)) unique critical points.")
        println("   $(pb) total saddles $(sum(df.is_saddle))")
    end
end
```