Piece-wise quadratic approximations of arbitrary error functions for fast and robust machine learning

A.N. Gorban, E.M. Mirkes, A. Zinovyev

Department of Mathematics, University of Leicester, Leicester, LE1 7RH, UK
Institut Curie, PSL Research University, Mines Paris Tech, Inserm, U900, F-75005, Paris, France

Abstract
Most of machine learning approaches have stemmed from the application of minimizing the mean squared distance principle, based on the computationally efficient quadratic optimization methods. However, when faced with high-dimensional and noisy data, the quadratic error functionals demonstrated many weaknesses including high sensitivity to contaminating factors and dimensionality curse. Therefore, a lot of recent applications in machine learning exploited properties of non-quadratic error functionals based on \( L_1 \) norm or even sub-linear potentials corresponding to quasinorms \( L_p \) (\( 0 < p < 1 \)). The back side of these approaches is increase in computational cost for optimization. Till so far, no approaches have been suggested to deal with arbitrary error functionals, in a flexible and computationally efficient framework. In this paper, we develop a theory and basic universal data approximation algorithms (\( k \)-means, principal components, principal manifolds and graphs, regularized and sparse regression), based on piece-wise quadratic error potentials of subquadratic growth (PQSQ potentials). We develop a new and universal framework to minimize arbitrary sub-quadratic error potentials using an algorithm with guaranteed fast convergence to the local or global error minimum. The theory of PQSQ potentials is based on the notion of the cone of minorant functions, and represents a natural approximation formalism based on the application of min-plus algebra. The approach can be applied in most of existing machine learning methods, including methods of data approximation and regularized and sparse regression, leading to the improvement in the computational cost/accuracy trade-off. We demonstrate that on synthetic and real-life datasets PQSQ-based machine learning methods achieve orders of magnitude faster computational performance than the corresponding state-of-the-art methods, having similar or better approximation accuracy.

Keywords: data approximation, nonquadratic potential, principal components, clustering, regularized regression, sparse regression

1. Introduction
Modern machine learning and artificial intelligence methods are revolutionizing many fields of science today, such as medicine, biology, engineering, high-energy physics and sociology, where large amounts of data have been collected due to the emergence of new high-throughput computerized technologies. Historically and methodologically speaking, many machine learning algorithms have been based on minimizing the mean squared error potential, which can be explained by tractable properties of normal distribution and existence of computationally efficient methods for quadratic optimization. However, most of the real-life datasets are characterized by strong noise, long-tailed distributions, presence of contaminating factors, large dimensions. Using quadratic potentials can be drastically compromised by all these circumstances: therefore, a lot of practical and theoretical efforts have been made in order to exploit the properties of non-quadratic error potentials which can be more appropriate in certain contexts. For example, methods of regularized and sparse regression such as lasso and elastic net based on the proper-
ties of $L_1$ metrics \cite{1,2} found numerous applications in bioinformatics \cite{3}, and $L_1$ norm-based methods of dimension reduction are of great use in automated image analysis \cite{4}. Not surprisingly, these approaches come with drastically increased computational cost, for example, connected with applying linear programming optimization techniques which are substantially more expensive compared to mean squared error-based methods.

In practical applications of machine learning, it would be very attractive to be able to deal with arbitrary error potentials, including those based on $L_1$ or fractional quasinorms $L_p$ ($0 < p < 1$), in a computationally efficient and scalable way. There is a need in developing methods allowing to tune the computational cost/accuracy of optimization trade-off accordingly to various contexts.

In this paper, we suggest such a universal framework able to deal with a large family of error potentials. We exploit the fact that finding a minimum of a piece-wise quadratic function, or, in other words, a function which is the minorant of a set of quadratic functionals, can be almost as computationally efficient as optimizing the standard quadratic potential. Therefore, if a given arbitrary potential (such as $L_1$-based or fractional quasinorm-based) can be approximated by a piece-wise quadratic function, this should lead to relatively efficient and simple optimization algorithms. It appears that only potentials of quadratic or sub-quadratic growth are possible in this approach: however, these are the most usefull ones in data analysis. We introduce a rich family of piece-wise quadratic potentials of sub-quadratic growth (PQSQ-potentials), suggest general approach for their optimization and prove convergence of a simple iterative algorithm in the most general case. We focus on the most used methods of data dimension reduction and regularized regression: however, potential applications of the approach can be much wider.

Data dimension reduction by constructing explicit low-dimensional approximators of a finite set of vectors is one of the most fundamental approach in data analysis. Starting from the classical data approximators such as k-means \cite{5} and linear principal components (PCA) \cite{6}, multiple generalizations have been suggested in the last decades (self-organizing maps, principal curves, principal manifolds, principal graphs, principal trees, etc.) \cite{7,8} in order to make the data approximators more flexible and suitable for complex data structures.

We solve the problem of approximating a finite set of vectors $\vec{x}_i \in \mathbb{R}^m$, $i = 1, \ldots, N$ (data set) by a simpler object $L$ embedded into the data space, such that for each point $\vec{x}_i$ an approximation error $err(\vec{x}_i, L)$ function can be defined. We assume this function in the form

$$\text{err}(\vec{x}_i, L) = \min_{y \in L} \sum_{k} u(x_i^k - y^k), \quad (1)$$

where the upper $k = 1, \ldots, m$ stands for the coordinate index, and $u(x)$ is a monotonously growing symmetrical function, which we will be calling the error potential. By data approximation we mean that the embedment of $L$ in the data space minimizes the error

$$\sum_i \text{err}(\vec{x}_i, L) \to \min.$$ 

Note that our definition of error function is coordinate-wise (it is a sum of error potential over all coordinates).

The simplest form of the error potential is quadratic $u(x) = x^2$, which leads to the most known data approximators: mean point ($L$ is a point), principal points ($L$ is a set of points) \cite{9}, principal components ($L$ is a line or a hyperplane) \cite{6}. In more advanced cases, $L$ can posses some regular properties leading to principal curves ($L$ is a smooth line or spline) \cite{10}, principal manifolds ($L$ is a smooth low-dimensional surface) and principal graphs (eg., $L$ is a pluri-harmonic graph embedment) \cite{11,12,3}.

There exist multiple advantages of using quadratic potential $u(x)$, because it leads to the most computationally efficient algorithms usually based on the splitting schema, a variant of expectation-minimization approach \cite{7}. For example, $k$-means algorithm solves the problem of finding the set of principal points and the standard iterative Singular Value Decomposition finds principal components. However, quadratic potential is known to be sensitive to outliers in the data set. Also, purely quadratic potentials can suffer from the curse of dimensionality, not being able to robustly discriminate ‘close’ and ‘distant’ point neighbours in a high-dimensional space \cite{12}.

There exist several widely used ideas for increasing approximator’s robustness in presence of strong noise in data such as: (1) using medians instead of mean values, (2) substituting quadratic norm by $L_1$ norm (e.g. \cite{13,14}), (3) outliers exclusion or
fixed weighting or iterative reweighting during optimizing the data approximators (e.g. [15, 16, 17]), and (4) regularizing the PCA vectors by $L_1$ norm [18, 19, 20]. In some works, it was suggested to utilize ‘trimming’ averages, e.g. in the context of the $k$-means clustering or some generalizations of PCA [21, 14]. In the context of regression, iterative reweighting is exploited to mimic the properties of Lasso and Elastic Net. Thus, for each coordinate $k$, we have

$$a_k = \frac{f(r_k) - f(r_{k+1})}{r_k - r_{k+1}},$$

$$b_k = \frac{f(r_{k+1})r_k^2 - f(r_k)r_{k+1}^2}{r_k^2 - r_{k+1}^2},$$

where $f(x)$ is a majorizing function, which is to be approximated (imitated) by $u(x)$. For example, in the simplest case $f(x)$ can be a linear function: $f(x) = x$, in this case, $\sum_k u(x^k)$ will approximate the $L_1$-based error function.

Note that accordingly to (3, 4), $b_0 = 0, a_p = 0, b_p = f(r_p)$. Therefore, the choice of $r_p$ can naturally create a ‘trimmed’ version of error potential $u(x)$ such that some data points (outliers) do not have any contribution to the gradient of $u(x)$, hence, will not affect the optimization procedure. However, this set of points can change during minimization of the potential.

The condition of subquadratic growth consists in the requirement $a_{k+1} \leq a_k$ and $b_{k+1} \geq b_k$. To guarantee this, the following simple condition on $f(x)$ should be satisfied:

$$f' > 0, \quad f''x \leq f'.$$

Therefore, $f(x)$ is a monotonic concave function of $q = x^2$:

$$\frac{d^2 f(\sqrt{q})}{dq^2} = \frac{1}{4x^2}f''(x) - \frac{1}{4x^3}f'(x) \leq 0.$$

In particular, $f(x)$ should grow not faster than any parabola $ax^2 + c, c > 0$, which is tangent to $f(x)$.

2. Piecewise quadratic potential of subquadratic growth (PQSQ)

2.1. Definition of the PQSQ potential

Let us split all non-negative numbers $x \in R_{\geq 0}$ into $p + 1$ non-intersecting intervals $R_0 = [0; r_1), R_1 = [r_1; r_2), \ldots, R_k = [r_k; r_{k+1}), \ldots, R_p = [r_p; \infty)$, for a set of thresholds $r_1 < r_2 < \ldots < r_p$. For convenience, let us denote $r_0 = 0, r_{p+1} = \infty$. Piecewise quadratic potential is a continuous monotonously growing function $u(x)$ constructed from pieces of centered at zero parabolas $y = b_k + a_kx^2$, defined on intervals $x \in [r_k; r_{k+1})$, satisfying $y(r_k) = f(r_k)$ (see Figure 1):

$$u(x) = b_k + a_kx^2, \text{ if } r_k \leq |x| < r_{k+1}, \quad k = 0, \ldots, p.$$

2.2. Basic approach for optimization

In order to use the PQSQ potential in an algorithm, a set of $p$ interval thresholds $r_k, s = 1, \ldots, p$ for each coordinate $k = 1, \ldots, m$ should be provided. Matrices of $a$ and $b$ coefficients defined by (3, 4) based on interval definitions: $a_k, b_k, s = 0, \ldots, p, k = 1, \ldots, m$ are computed separately for each coordinate $k$.

Minimization of PQSQ-based functional consists in several basic steps which can be combined in an algorithm:

1) For each coordinate $k$, split all data point indices into non-overlapping sets $R^k_s$:

$$R^k_s = \{i : r^k_s \leq |x^i - \beta^k_s| < r^k_{s+1}\}, \quad s = 0, \ldots, p.$$
For any convenience, in this section, we will define the potential instead of the operational definition (2) in a different way when convenient to construct $u(x)$. The last parabola is flat ($a_p = 0$) which corresponds to trimmed potential.

where $\beta$ is a matrix which depends on the nature of the algorithm.

2) Minimize PQSQ-based functional where each set of points $\{x_i \in \mathbb{R}^d\}$ contributes to the functional quadratically with coefficient $a_i^k$. This is a quadratic optimization task.

3) Repeat (1)-(2) till convergence.

3. General theory of the piece-wise convex potentials as the cone of minorant functions

In order to deal in most general terms with the data approximation algorithms based on PQSQ potentials, let us consider a general case where a potential can be constructed from a set of functions $\{q_i(x)\}$ with only two requirements: 1) that each $q_i(x)$ has a (local) minimum; 2) that the whole set of all possible $q_i(x)$ forms a cone. In this case, instead of the operational definition (2) it is convenient to define the potential $u(x)$ as the minorant function for a set of functions as follows. For convenience, in this section, $x$ will notify a vector $\vec{x} \in \mathbb{R}^m$.

Let us consider a generating cone of functions $Q$. We remind that the definition of a cone implies that for any $q(x) \in Q, p(x) \in Q$, we have $\alpha q(x) + \beta p(x) \in Q$, where $\alpha \geq 0, \beta \geq 0$.

For any finite set of functions $q_1(x) \in Q, q_2(x) \in Q, \ldots, q_s(x) \in Q,$ we define the minorant function (Figure 2):

$$u_{q_1,q_2,\ldots,q_s}(x) = \min(q_1(x), q_2(x), \ldots, q_s(x)). \quad (7)$$

It is convenient to introduce a multiindex

$$I_{q_1,q_2,\ldots,q_s}(x)$$

indicating which particular function(s) $q_i$ corresponds to the value of $u(x)$, i.e.

$$I_{q_1,q_2,\ldots,q_s}(x) = \{i | u_{q_1,q_2,\ldots,q_s}(x) = q_i(x)\}. \quad (8)$$

For a cone $Q$ let us define a set of all possible minorant functions $M(Q)$

$$M(Q) = \{u_{q_1,q_2,\ldots,q_n} | q_1 \in Q, q_2 \in Q, \ldots, q_n \in Q, n = 1, 2, 3, \ldots \}. \quad (9)$$

**Proposition 1.** $M(Q)$ is a cone.

**Proof** For any two minorant functions $u_{q_1,q_2,\ldots,q_n}, u_{q_1,q_2,\ldots,q_n} \in M(Q)$ we have

$$\alpha u_{q_1,q_2,\ldots,q_n} + \beta u_{q_1,q_2,\ldots,q_n} = u_{\alpha q_1 + \beta q_1} \in M(Q), \quad (10)$$

where $\{\alpha q_1 + \beta q_1\}$ is a set of all possible linear combinations of functions from $\{q_1, q_2, \ldots, q_n\}$ and $\{q_1, q_2, \ldots, q_n\}$.

**Proposition 2.** Any restriction of $M(Q)$ onto a linear manifold $L$ is a cone.

**Proof** Let us denote $q(x)|_L$ a restriction of $q(x)$ function onto $L$, i.e. $q(x)|_L = \{q(x) | x \in L\}, q(x)|_L$ is a part of $Q$. Set of all $q(x)|_L$ forms a restriction $M(Q)$ of $Q$ onto $L$. $Q|_L$ is a cone, hence, $M(Q)|_L = M(Q)|_L$ is a cone (Proposition 1).

**Definition** Splitting algorithm minimizing

$$u_{q_1,q_2,\ldots,q_n}(x)$$

is defined as Algorithm 1.

**Theorem 3.1.** Splitting algorithm (Algorithm 1) for minimizing $u_{q_1,q_2,\ldots,q_n}(x)$ converges in a finite number of steps.

Figure 1: Trimmed piecewise quadratic potential of sub-quadratic growth $u(x)$ (solid blue line) defined for the majorizing function $f(x)$ (red dashed line) and several thresholds $r_k$. Dotted lines show the parabolas which fragments are used to construct $u(x)$. The last parabola is flat ($a_p = 0$) which corresponds to trimmed potential.
Correspondingly to the definition (7), (2) Or $u_{q_1,q_2,...,q_n}(x') < u_{q_1,q_2,...,q_n}(x)$ since, accordingly to the definition (11), $q_i(x') < q_i(x)$, for any $i' \in I_{q_1,q_2,...,q_n}(x')$, $i \in I_{q_1,q_2,...,q_n}(x)$ (see Figure 2).

Note that in Algorithm 1 we do not specify exactly the way to find the local minimum of $q_i(x)$. To be practical, the cone $Q$ should contain only functions for which finding a local minimum is fast and explicit. Evident candidates for this role are positively defined quadratic functionals $q(x) = q_0 + (q_1, x) + (x, Q_2 x)$, where $Q_2$ is a positively defined symmetric matrix. Any minorant function constructed from positively defined quadratic functions will automatically provide subquadratic growth, since the minorant can not grow faster than any of the quadratic forms by which it is defined.

Operational definition of PQSQ given above (2), corresponds to a particular form of the quadratic functional, with $Q_2$ being diagonal matrix. This choice corresponds to the coordinate-wise definition of data approximation error function (11) which is particularly simple to minimize. This circumstance is used in Algorithms 2, 3.

**Algorithm 1** Finding local minimum of a minorant function $u_{q_1,q_2,...,q_n}(x)$

1: **procedure** MINIMIZING MINORANT FUNCTION
2: initialize $x \leftarrow x_0$
3: repeat until stopping criterion has been met:
4: compute multiindex $I_{q_1,q_2,...,q_n}(x)$
5: for all $i \in I_{q_1,q_2,...,q_n}(x)$
6: \[ x_i = \arg \min q_i(x) \]
7: end for
8: select optimal $x_i$:
9: $x_{opt} \leftarrow \arg \min_{x_i} u(x_i)$
10: $x \leftarrow x_{opt}$
11: **stopping criterion**: check if the multiindex $I_{q_1,q_2,...,q_n}(x)$ does not change compared to the previous iteration.

**Proof** Since the set of functions $\{q_1,q_2,\ldots,q_n\}$ is finite then we only have to show that at each step the value of the function $u_{q_1,q_2,...,q_n}(x)$ can not increase. For any $x$ and the value $x' = \arg \min q_i(x)$ for $i \in I_{q_1,q_2,...,q_n}(x)$ we can have only two cases:

1. Either $I_{q_1,q_2,...,q_n}(x) = I_{q_1,q_2,...,q_n}(x')$ (convergence, and in this case $q_i(x') \leq q_i(x)$ for any $i' \in I_{q_1,q_2,...,q_n}(x')$);
2. Or $u_{q_1,q_2,...,q_n}(x') < u_{q_1,q_2,...,q_n}(x)$ since, accordingly to the definition (11), $q_i(x') < q_i(x)$, for any $i' \in I_{q_1,q_2,...,q_n}(x')$, $i \in I_{q_1,q_2,...,q_n}(x)$ (see Figure 2).

4. Commonly used data approximators with PQSQ potential

4.1. Mean value and k-means clustering in PQSQ approximation measure

Mean vector $\bar{X}_k$ for a set of vectors $X = \{x_i\}$, $i = 1, \ldots, N$, $k = 1, \ldots, m$ and an approximation error defined by potential $f(x)$ can be defined as a point minimizing the mean error potential for all points in $X$:

\[
\sum_i \sum_k f(x^k_i - \bar{X}^k) \rightarrow \min.
\]

For Euclidean metrics $L_2$ ($f(x) = x^2$) it is the usual arithmetic mean.

For $L_1$ metrics ($f(x) = |x|$), (11) leads to the implicit equation $\#(x^k_i > \bar{X}^k) = \#(x^k_i < \bar{X}^k)$, where $\#$ stands for the number of points, which corresponds to the definition of median. This equation can have a non-unique solution in case of even number of points or when some data point coordinates coincide: therefore, definition of median is usually accompanied by heuristics used for breaking ties, i.e. to deal with non-uniquely defined rankings. This situation reflects the general situation of existence of multiple local minimums and possible non-uniqueness of global minimum of (11) (Figure 3).

For PQSQ approximation measure (2) it is difficult to write down an explicit formula for computing the mean value corresponding to the global minimum of (11). In order to find a point $\bar{X}_{PQSQ}$ minimizing mean PQSQ potential, a simple iterative algorithm can be used (Algorithm 2). The sug-
gested algorithm converges to the local minimum which depends on the initial point approximation.

**Algorithm 2** Computing PQSQ mean value

1. procedure PQSQ MEAN VALUE
2. define intervals \( r^k_s, s = 0, \ldots, p, k = 1, \ldots, m \)
3. compute coefficients \( a^k_s \)
4. initialize \( \bar{X}_{PQSQ} \)
eg., by arithmetic mean
5. repeat till convergence of \( \bar{X}_{PQSQ} \):
6. for each coordinate \( k \)
7. define sets of indices \( \mathcal{R}^k_s = \{ i : r^k_s \leq |x^k_s - \bar{X}^k_{PQSQ}| < r^k_{s+1} \}, \)
8. compute new approximation for \( \bar{X}_{PQSQ} \):
9. \( \bar{X}^k_{PQSQ} \leftarrow \frac{\sum_{i=1}^s a^k_s \sum_{i=1}^p a^k_i x^k_i}{\sum_{i=1}^s a^k_i} \)
10. end for
11. goto repeat till convergence

Based on the PQSQ approximation measure and the algorithm for computing the PQSQ mean value (Algorithm 2), one can construct the PQSQ-based \( k \)-means clustering procedure in the usual way, splitting estimation of cluster centroids given partitioning of the data points into \( k \) disjoint groups, and then re-calculating the partitioning using the PQSQ-based proximity measure.

4.2. Principal Component Analysis (PCA) in PQSQ metrics

Accordingly to the classical definition of the first principal component, it is a line best fit to the data set \( X \) \[6\]. Let us define a line in the parametric form \( \vec{y} = \vec{V} \nu + \vec{\delta} \), where \( \nu \in \mathbb{R}^l \) is the parameter. Then the first principal component will be defined by vectors \( \vec{V}, \vec{\delta} \) satisfying

\[
\sum_i \sum_k u(x_i^k - V^k \nu - \delta^k) \rightarrow \min, \quad (12)
\]

where

\[
\nu = \arg \min_s \sum_k u(x_i^k - V^k s - \delta^k). \quad (13)
\]

The standard first principal component (PC1) corresponds to \( u(x) = x^2 \) when the vectors \( \vec{V}, \vec{\delta} \) can be found by a simple iterative splitting algorithm for Singular Value Decomposition (SVD). If \( X \) does not contain missing values then \( \vec{\delta} \) is the vector of arithmetic mean values. By contrast, computing \( L_1 \)-based principal components \( (u(x) = |x|) \) represents a much more challenging optimization problem \[24\]. Several approximative algorithms for computing \( L_1 \)-norm PCA have been recently suggested and benchmarked \[23, 24, 25, 26, 27\]. To our knowledge, there have not been a general efficient algorithm suggested for computing PCA in case of arbitrary approximation measure for some monotonous function \( u(x) \).

Computing PCA based on PQSQ approximation error is only slightly more complicated than computing the standard \( L_2 \) PCA by SVD. Here we provide a pseudo-code (Algorithm 3) of a simple iterative algorithm (similar to Algorithm 2) with guaranteed convergence (see Section 3).

Computation of second and further principal components follows the standard deflation approach: projections of data points onto the previously computed component are subtracted from the data set, and the algorithm is applied to the residues. However, as it is the case in any non-quadratic metrics, the resulting components can be non-orthogonal or even not invariant with respect to the dataset rotation. Moreover, unlike \( L_2 \)-based principal components, the Algorithm 3 does not always converge to a unique global minimum; the computed components can depend on the initial estimate of \( \vec{V} \). The situation is somewhat similar to the standard \( k \)-means algorithm. Therefore, in order to achieve the least possible approximation error to the linear subspace, \( \vec{V} \) can be initialized randomly or by data vectors \( \vec{x}_i \) many times and the deepest in PQSQ approximation error \( 1 \) minimum should be selected.

How does the Algorithm 1 serve a more abstract version of the Algorithms 2,3? For example, the ‘variance’ function \( m(\vec{x}) = \frac{1}{l} \sum_i u(\vec{x}_j - \vec{x}) \) to be minimized in Algorithm 2 uses the generating functions in the form \( Q = \{ h_{j_i} + \sum_k a^k_j (x^k_j - \bar{x}^k_j)^2 \} \), where \( i \) is the index of the interval in \[2\]. Hence, \( m(\vec{x}) \) is a minorant function, belonging to the cone \( \mathcal{M}(Q) \), and must converge (to a local minimum) in a finite number of steps accordingly to Theorem 3.1

4.3. Nonlinear methods: PQSQ-based Principal Graphs and Manifolds

In a series of works, the authors of this article introduced a family of methods for constructing principal objects based on graph approximations
Let $G$ be a simple undirected graph with set of vertices $Y$ and set of edges $E$. For $k \geq 2$ a $k$-star in $G$ is a subgraph with $k + 1$ vertices $y_0, y_1, ..., y_k \in Y$ and $k$ edges \{$(y_0, y_i)$ \mid $i = 1, ..., k$\} $\subset E$. Suppose for each $k \geq 2$, a family $S_k$ of $k$-stars in $G$ has been selected. We call a graph $G$ with selected families of $k$-stars $S_k$ an elastic graph if, for all $E^{(i)} \in E$ and $S_k^{(j)} \in S_k$, the correspondent elasticity moduli $\lambda_i > 0$ and $\mu_{kj} > 0$ are defined. Let $E^{(i)}(0), E^{(i)}(1)$ be vertices of an edge $E^{(i)}$ and $S_k^{(j)}(0), ..., S_k^{(j)}(k)$ be vertices of a $k$-star $S_k^{(j)}$ (among them, $S_k^{(j)}(0)$ is the central vertex).

For any map $\phi : Y \rightarrow R^m$ the energy of the graph is defined as

$$U^{\phi}(G) := \sum_{E^{(i)}} \lambda_i \left\| \phi(E^{(i)}(0)) - \phi(E^{(i)}(1)) \right\|^2 + \sum_{S_k^{(j)}} \mu_{kj} \left\| \sum_{i=1}^k \phi(S_k^{(j)}(i)) - k \phi(S_k^{(j)}(0)) \right\|^2.$$ 

For a given map $\phi : Y \rightarrow R^m$ we divide the dataset $D$ into node neighborhoods $K^y, y \in Y$. The set $K^y$ contains the data points for which the node $\phi(y)$ is the closest one in $\phi(y)$. The energy of approximation is:

$$U^{\phi}_A(G, D) = \sum_{y \in Y} \sum_{x \in K^y} w(x) \| x - \phi(y) \|^2, \quad (14)$$

where $w(x) \geq 0$ are the point weights. Simple and fast algorithm for minimization of the energy

$$U^{\phi} = U^{\phi}_A(G, D) + U^{\phi}(G) \quad (15)$$

is the splitting algorithm, in the spirit of the classical $k$-means clustering; for a given system of sets \{ $K^y \mid y \in Y$ \} we minimize $U^{\phi}$ (optimization step, it is the minimization of a positive quadratic functional), then for a given $\phi$ we find new \{ $K^y$ \} (repartitioning), and so on; stop when no change.
Algorithm 3 Computing PQSQ PCA

1: procedure PQSQ FIRST PRINCIPAL COMPONENT
2: define intervals \(r_k^i, s = 0, \ldots, p, \kappa = 1, \ldots, m\)
3: compute coefficients \(a_k^i\)
4: \(\bar{\delta} \leftarrow X_{PQSQ}\)
5: initialize \(\tilde{V} : \text{eg., by } L_2\)-based PC1
6: initialize \(\{\nu_i\} : \text{eg., by }\)
7: \(\nu_i = \frac{\sum_k V_i^k (x_i^k - \delta^k)}{\sum_k (V_i^k)^2}\)
8: repeat until convergence of \(\tilde{V}\):
9: for each coordinate \(k\)
10: define sets of indices \(\mathcal{R}_k = \{i : r_k^i \leq |x_i^k - V_i^k \nu_i - \delta^k| < r_{k+1}^i\}\), \(s = 0, \ldots, p\)
11: end for
12: for each data point \(i\) and coordinate \(k\)
13: find all \(s_{i,k}\) such that \(i \in \mathcal{R}_{s_{i,k}}^k\)
14: if all \(a_{s_{i,k}}^k = 0\) then \(\nu_i^k \leftarrow 0\) else
15: \(\nu_i^k \leftarrow \frac{\sum_k a_{s_{i,k}}^k V_i^k (x_i^k - \delta^k)}{\sum_k a_{s_{i,k}}^k (V_i^k)^2}\)
16: end for
17: for each coordinate \(k\)
18: \(V_i^k \leftarrow \frac{\sum a_{s_{i,k}}^k \sum_{i \in \mathcal{R}_k} (x_i^k - \delta^k) \nu_i}{\sum a_{s_{i,k}}^k \sum_{i \in \mathcal{R}_k} (\nu_i)^2}\)
19: end for
20: \(\nu_i \leftarrow \nu_i^k\)
21: end for
22: goto repeat until convergence

Application of PQSQ-based potential is straightforward in this approach. It consists in replacing (14) with

\[
U^\phi_{\lambda}(G, D) = \sum_{y \in Y} \sum_{x \in K^y} w(x) \sum_k u(x^k - \phi(y^k)),
\]

where \(u\) is a chosen PQSQ-based error potential. Partitioning of the dataset into \(\{K^y\}\) can be also based on calculating the minimum PQSQ-based error to \(y\), or can continue enjoying nice properties of \(L_2\)-based distance calculation.

5. PQSQ-based regularized regression

5.1 Regularizing linear regression with PQSQ potential

One of the major application of non-Euclidean norm properties in machine learning is using non-quadratic terms for penalizing large absolute values of regression coefficients [1, 2]. Depending on the chosen penalization term, it is possible to achieve various effects such as sparsity or grouping coefficients for redundant variables. In a general form, regularized regression solves the following optimization problem

\[
\frac{1}{N} \sum_{i=1}^{N} \left( y_i - \sum_{k=1}^{m} \beta_k^i x_i^k \right)^2 + \lambda f(\beta) \rightarrow \min, \quad (16)
\]

where \(N\) is the number of observations, \(m\) is the number of independent variables in the matrix \(\{x_i^k\}\), \(\{y_i\}\) are dependent variables (to be predicted), \(\lambda\) is an internal parameter controlling the strength of regularization (penalty on the amplitude of regression coefficients \(\beta\)), and \(f(z)\) is the regularizer function, which is \(f(z) = \|z\|_{L_2}^2\) for ridge regression, \(f(z) = \|z\|_{L_2}^2 + \alpha\|z\|_{L_1}\) for lasso and \(f(z) = \frac{1}{\alpha} \|z\|_{L_2}^2 + \alpha\|z\|_{L_1}\) for elastic net methods correspondingly.

Here we suggest to imitate \(f(x)\) with a PQSQ potential function, i.e. instead of (16) solving the problem

\[
\frac{1}{N} \sum_{i=1}^{N} \left( y_i - \sum_{k=1}^{m} \beta_k^i x_i^k \right)^2 + \lambda \sum_{k=1}^{m} u(\beta^k) \rightarrow \min,
\]

where \(u(\beta)\) is a PQSQ potential imitating arbitrary subquadratic regression regularization penalty.
Solving (17) is equivalent to iteratively solving a system of linear equations

\[
\frac{1}{N} \sum_{k=1}^{m} \beta^k \sum_{i=1}^{N} x_i^k x_i^j + \lambda a_{I(\beta^k)} = \sum_{i=1}^{N} y_i x_i^j, \quad j = 1, \ldots, m, \tag{18}
\]

where \( a_{I(\beta^k)} \) is a constant (where \( I \) index is defined from \( r_I \leq \beta^k < r_{I+1} \)) computed accordingly to the definition of \( u(x) \) function (see (3)), given the estimation of \( \beta^k \) regression coefficients at the current iteration. In practice, iterating (18) converges in a few iterations, therefore, the algorithm can work very fast and outperform the widely used least angle regression algorithm for solving (16) in case of \( L_1 \) penalties.

5.2. Introducing sparsity by ‘black hole’ trick

Any PQSQ potential \( u(x) \) is characterized by zero derivative for \( x = 0 \) by construction: \( u'(x)|_{x=0} = 0 \), which means that the solution of (17) does not have to be sparse for any \( \lambda \). Unlike pure \( L_1 \)-based penalty, the coefficients of regression diminish with increase of \( \lambda \) but there is nothing to shrink them to exact zero values, similar to the ridge regression. However, it is relatively straightforward to modify the algorithm, to achieve sparsity of the regression solution. The ‘black hole’ trick consists in eliminating from regression training after each iteration (15) all regression coefficients \( \beta^k \) smaller by absolute value than a given parameter \( \epsilon \) (‘black hole radius’). Those regression coefficients which have passed the ‘black hole radius’ are put to zero and do not have any chance to change their values in the next iterations.

The optimal choice of \( \epsilon \) value requires a separate study. From general considerations, it is preferable that the derivative \( u'(x)|_{x=\epsilon} \) would not be very close to zero. As a pragmatic choice for the numerical examples in this article, we define \( \epsilon \) as the midst of the smallest interval in the definition of PQSQ potential (see Figure 4), i.e., \( \epsilon = r_1/2 \), which guarantees far from zero \( u'(x)|_{x=\epsilon} \) value. It might happen that this value of \( \epsilon \) would collapse all \( \beta^k \) to zero even without regularization (i.e., with \( \lambda = 0 \)). In this case, the ‘black hole radius’ is divided by half \( \epsilon \rightarrow \epsilon/2 \) and it is checked that for \( \lambda = 0 \) the iterations would leave at list half of the regression coefficients. If it is not the case then the process of diminishing the ‘black hole radius’ repeated recursively till meeting the criterion of preserving the majority of regression coefficients. In practice, it requires only few (very fast) additional iterations of the algorithm.

As in the lasso methodology, the problem (17) is solved for a range of \( \lambda \) values, calibrated such that the minimal \( \lambda \) would select the maximum number of regression variables, while the maximum \( \lambda \) value would lead to the most sparse regression (selecting only one single non-zero regression coefficient).

6. Numerical examples

6.1. Practical choices of parameters

The main parameters of PQSQ are (a) majorating function \( f(x) \) and (b) decomposition of each coordinate range into \( p + 1 \) non-overlapping intervals. Depending on these parameters, various approximation error properties can be exploited, including those providing robustness to outlier data points.

When defining the intervals \( r_j, j = 1, \ldots, p \), it is desirable to achieve a small difference between \( f(\Delta x) \) and \( u(\Delta x) \) for expected argument values \( \Delta x \) (differences between an estimator and the data point), and choose the suitable value of the potential trimming threshold \( r_p \), in order to achieve the desired robustness properties. If no trimming is needed, then \( r_p \) should be made larger than the maximum expected difference between coordinate values (maximum \( \Delta x \)).
In our numerical experiments we used the following definition of intervals. For any data coordinate \( k \), we define a characteristic difference \( D^k \), for example

\[
D^k = \alpha_{\text{scale}}(\max_i (x^k_i) - \min_j (x^k_j)),
\]

where \( \alpha_{\text{scale}} \) is a scaling parameter, which can be put at 1 (in this case, the approximating potential will not be trimmed). In case of existence of outliers, for defining \( D^k \), instead of amplitude one can use other measures such as the median absolute deviation (MAD):

\[
D^k = \alpha_{\text{scale}} \text{median}_i(|x^k_i - \text{median}_j(x^k_j)|);
\]

in this case, the scaling parameter should be made larger, i.e. \( \alpha_{\text{scale}} = 10 \), if no trimming is needed.

After defining \( D^k \) we use the following definition of intervals:

\[
r^k_j = D^k \frac{j^2}{p^2}, \quad j = 0, \ldots, p.
\]

More sophisticated approaches are also possible to apply such as, given the number of intervals \( p \) and the majorating function \( f(x) \), choose \( r_j, j = 1, \ldots, p \) in order to minimize the maximal difference

\[
d = \max_j |f(x) - u(x)| \to \min.
\]

The calculation of intervals is straightforward for a given value of \( d \) and many smooth concave functions \( f(x) \) like \( f(x) = x^p (0 < p \leq 1) \) or \( f(x) = \ln(1 + x) \).

### 6.2. Implementation

We provide Matlab implementation of PQSQ approximators (in particular, PCA) together with the Matlab and R code used to generate the example figures in this article at ‘PQSQ-DataApproximators’ GitHub repository\(^1\) and Matlab implementation of PQSQ-based regularized regression with build-in imitations of \( L_1 \) (lasso-like) and \( L_1 \& L_2 \) mixture (elastic net-like) penalties at ‘PQSQ-regularized-regression’ GitHub repository\(^2\).

The Java code implementing elastic graph-based non-linear approximator implementations is available from the authors on request.

---

\(^1\)https://github.com/auranic/PQSQ-DataApproximators

\(^2\)https://github.com/Mirkes/PQSQ-regularized-regression

### 6.3. Motivating example: dense two-cluster distribution contaminated by sparse noise

We demonstrate the value of PQSQ-based computation of \( L_1 \)-based PCA by constructing a simple example of data distribution consisting of a dense two-cluster component superimposed with a sparse contaminating component with relatively large variance whose co-variance does not coincide with the dense signal (Figure 5). We study the ability of PCA to withstand certain level of sparse contamination and compare it with the standard \( L_2 \)-based PCA. In this example, without noise the first principal component coincides with the vector connecting the two cluster centers: hence, it perfectly separates them in the projected distribution. Noise interferes with the ability of the first principal component to separate the clusters to the degree when the first principal component starts to match the principal variance direction of the contaminating distribution (Figure 5A,B). In higher dimensions, not only the first but also the first two principal components are not able to distinguish two clusters, which can hide an important data structure when applying the standard data visualization tools.

In the first test we study a switch of the first principal component from following the variance of the dense informative distribution (abscissa) to the sparse noise distribution (ordinate) as a function of the number of contaminating points, in \( R^2 \) (Figure 5A-C). We modeled two clusters as 100-point samples from normal distribution centered in points \([-1; 0]\) and \([1; 0]\) with isotropic variance with the standard deviation 0.1. The sparse noise distribution was modeled as a \( k \)-point sample from the product of two Laplace distributions of zero means with the standard deviations 2 along abscissa and 4 along ordinate. The intervals for computing the PQSQ functional were defined by thresholds \( R = \{0; 0.01; 0.1; 0.5; 1\} \) for each coordinate. Increasing the number of points in the contaminating distribution diminishes the average value of the abscissa coordinate of PC1, because the PC1 starts to be attracted by the contaminating distribution (Figure 5C). However, it is clear that on average PQSQ \( L_1 \)-based PCA is able to withstand much larger amplitude of the contaminating signal (very robust up to 20-30 points, i.e. 10-20% of strong noise contamination) compared to the standard \( L_2 \)-based PCA (which is robust to 2-3% of contamination).

In the second test we study the ability of the first two principal components to separate two clus-
Performance/stability trade-off benchmarking

In order to compare the computation time and the robustness of PQSQ-based PCA algorithm for the case \( u(x) = |x| \) with existing R-based implementations of \( L_1 \)-based PCA methods (pcaL1 package), we followed the benchmark described in [26]. We compared performance of PQSQ-based PCA based on Algorithm 3 with several \( L_1 \)-based PCA algorithms: L1-PCA* [25], L1-PCA [23], PCA-PP [34], PCA-L1 [21]. As a reference point, we used the standard PCA algorithm based on quadratic norm and computed using the standard SVD iterations.

The idea of benchmarking is to generate a series of datasets of the same size (\( N = 1000 \) objects in \( m = 10 \) dimensions) such that the first 5 dimensions would be sampled from a uniform distribution \( U(-10, 10) \). Therefore, the first 5 dimensions represent ‘true manifold’ sampled by points. The values in the last 5 dimensions represent ‘noise + outlier’ signal. The background noise is represented by Laplacian distribution of zero mean and 0.1 variance. The outlier signal is characterized by mean value \( \mu \), dimension \( p \) and frequency \( \phi \). Then, for each data point with a probability \( \phi \), in the first \( p \) outlier dimensions a value is drawn from Laplace(\( \mu, 0.1 \)). The rest of the values is drawn from background noise distribution.

As in [20], we’ve generated 1300 test sets corresponding to \( \phi = 0.1 \), with 100 examples for each combination of \( \mu = 1, 5, 10, 25 \) and \( p = 1, 2, 3 \).

For each test set 5 first principal components \( \vec{V}_1, \ldots, \vec{V}_5 \) of unit length were computed, with corresponding point projection distributions \( U^1, \ldots, U^5 \) and the mean vector \( \vec{C} \). Therefore, for each initial data point \( \vec{x}_i \), we have the ‘restored’ data point

\[
P(\vec{x}_i) = \sum_{k=1,\ldots,5} U^k \vec{V}_k + \vec{C}.
\]

For computing the PQSQ-based PCA we used 5 intervals without trimming. Changing the number of intervals did not significantly changed the benchmarking results.

Two characteristics were measured: (1) computation time measured as a ratio to the computation of 5 first principal components using the standard \( L_2 \)-based PCA and (2) the sum of absolute values of the restored point coordinates in the ‘outlier’ dimensions normalized on the number of points:

\[
\sigma = \frac{1}{N} \sum_{i=1,\ldots,N} \sum_{k=6,\ldots,10} |P^k(\vec{x}_i)|. \tag{22}
\]
Formally speaking, $\sigma$ is $L_1$-based distance from the point projection onto the first five principal components to the ‘true’ subspace. In simplistic terms, larger values of $\sigma$ correspond to the situation when the first five principal components do not represent well the first ‘true’ dimensions having significant loadings into the ‘outlier dimensions’. $\sigma = 0$ if and only if the first five components do not have any non-zero loadings in the dimensions 6, …, 10.

The results of the comparison averaged over all 1300 test sets are shown in Figure 6. The PQSQ-based computation of PCA outperforms by accuracy the existing heuristics such as PCA-L1 but remains computationally efficient. It is 100 times faster than L1-PCA giving almost the same accuracy. It is almost 500 times faster than the L1-PCA* algorithm, which is, however, better in accuracy (due to being robust with respect to strong outliers). One can see from Figure 6 that PQSQ-based approach is the best in accuracy among fast iterative methods. The detailed tables of comparison for all combinations of parameters are available on GitHub\(^3\). The scripts used to generate the datasets and compare the results can also be found there\(^4\).

6.5. Comparing performances of PQSQ-based regularized regression and lasso algorithms

We compared performance of PQSQ-based regularized regression method imitating $L_1$ penalty with lasso implementation in Matlab, using 8 datasets from UCI Machine Learning Repository\(^3\), Regression Task section. In the selection of datasets

\(^3\)http://goo.gl/sXBvqh
\(^4\)https://github.com/auranic/PQSQ-DataApproximators
Table 1: Comparing time performance (in seconds, on ordinary laptop) of lasso vs PQSQ-based regularized regression imitating $L_1$ penalty. Average acceleration of PQSQ-based method vs lasso in these 8 examples is 120 fold with comparable accuracy.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Objects</th>
<th>Variables</th>
<th>lasso</th>
<th>PQSQ</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast cancer</td>
<td>47</td>
<td>31</td>
<td>10.50</td>
<td>0.05</td>
<td>233.33</td>
</tr>
<tr>
<td>Prostate cancer</td>
<td>97</td>
<td>8</td>
<td>0.07</td>
<td>0.02</td>
<td>4.19</td>
</tr>
<tr>
<td>ENB2012</td>
<td>768</td>
<td>8</td>
<td>0.53</td>
<td>0.03</td>
<td>19.63</td>
</tr>
<tr>
<td>Parkinson</td>
<td>5875</td>
<td>26</td>
<td>20.30</td>
<td>0.04</td>
<td>548.65</td>
</tr>
<tr>
<td>Crime</td>
<td>1994</td>
<td>100</td>
<td>10.50</td>
<td>0.19</td>
<td>56.24</td>
</tr>
<tr>
<td>Crime reduced</td>
<td>200</td>
<td>100</td>
<td>17.50</td>
<td>0.17</td>
<td>102.94</td>
</tr>
<tr>
<td>Forest fires</td>
<td>517</td>
<td>8</td>
<td>0.05</td>
<td>0.02</td>
<td>3.06</td>
</tr>
<tr>
<td>Random regression (1000×250)</td>
<td>1000</td>
<td>250</td>
<td>2.82</td>
<td>0.58</td>
<td>4.86</td>
</tr>
</tbody>
</table>

we chose very different numbers of objects and variables for regression construction (Table 1). All table rows containing missing values were eliminated for the tests.

We observed up to two orders of magnitude acceleration of PQSQ-based method compared to the lasso method implemented in Matlab (Table 1), with similar sparsity properties and approximation power as lasso (Figure 7).

While comparing time performances of two methods, we’ve noticed that lasso (as it is implemented in Matlab) showed worse results when the number of objects in the dataset approaches the number of predictive variables (see Table 1). In order to test this observation explicitly, we took a particular dataset (‘Crime’) containing 1994 observations and 100 variables and compared the performance of lasso in the case of complete table and a reduced table (‘Crime reduced’) containing only each 10th observation. Paradoxically, lasso converges almost two times slower in the case of the smaller dataset, while the PQSQ-based algorithm worked slightly faster in this case.

It is necessary to stress that here we compare the basic algorithms without many latest technical improvements which can be applied both to $L_1$ penalty and its PQSQ approximation (such as fitting the whole lasso path). Detailed comparison of all the existent modifications if far beyond the scope of this work.

For comparing approximation power of the PQSQ-based regularized regression and lasso, we used two versions of PQSQ potential for regression coefficients: with and without trimming. In order to represent the results, we used the ‘Number of non-zero parameters vs Fraction of Variance Unexplained (FVU)’ plots (see two representative examples at Figure 7). We suggest that this type of plot is more informative in practical applications than the ‘lasso plot’ used to calibrate the strength of regularization, since it is a more explicit representation for optimizing the accuracy vs complexity ratio of the resulting regression.

From our testing, we can conclude that PQSQ-based regularized regression has similar properties of sparsity and approximation accuracy compared to lasso. It tends to slightly outperform lasso (to give smaller FVU) in case of $N \approx P$. Introducing trimming in most cases does not change the best FVU for a given number of selected variables, but tends to decrease its variance (has a stabilization effect). In some cases, introducing trimming is the most advantageous method (Figure 7B).

The GitHub ‘PQSQ-regularized-regression’ repository contains exact dataset references and more complete report on comparing approximation ability of PQSQ-based regularized regression with lasso.

7. Conclusion

In this paper we develop a new machine learning framework (theory and application) allowing one to deal with arbitrary error potentials of not-faster than quadratic growth, imitated by piecewise quadratic function of subquadratic growth (PQSQ error potential).

We develop methods for constructing the standard data approximators (mean value, $k$-means clustering, principal components, principal graphs) for arbitrary non-quadratic approximation error with subquadratic growth and regularized linear regression with arbitrary subquadratic penalty by using a piecewise-quadratic error functional (PQSQ...

---

https://github.com/Mirkes/PQSQ-regularized-regression/wiki
potential). These problems can be solved by applying quasi-quadratic optimization procedures, which are organized as solutions of sequences of linear problems by standard and computationally efficient algorithms.

The suggested methodology have several advantages over existing ones:

(a) **Scalability**: the algorithms are computationally efficient and can be applied to large data sets containing millions of numerical values.

(b) **Flexibility**: the algorithms can be adapted to any type of data metrics with subquadratic growth, even if the metrics can not be expressed in explicit form. For example, the error potential can be chosen as adaptive metrics [36, 37].

(c) **Built-in (trimmed) robustness**: choice of intervals in PQSQ can be done in the way to achieve a trimmed version of the standard data approximators, when points distant from the approximator do not affect the error minimization during the current optimization step.

(d) **Guaranteed convergence**: the suggested algorithms converge to local or global minimum just as the corresponding predecessor algorithms based on quadratic optimization and expectation/minimization-based splitting approach.

In theoretical perspective, using PQSQ-potentials in data mining is similar to existing applications of min-plus (or, max-plus) algebras in non-linear optimization theory, where complex non-linear functions are approximated by infimum (or supremum) of finitely many ‘dictionary functions’ [38, 39]. We can claim that just as using polynomials is a natural framework for approximating in rings of functions, using min-plus algebra naturally leads to introduction of PQSQ-based functions and the cones of minorants of quadratic dictionary functions.

One of the application of the suggested methodology is approximating the popular in data mining $L_1$ metrics. We show by numerical simulations that PQSQ-based approximators perform as fast as the fast heuristical algorithms for computing $L_1$-based PCA but achieve better accuracy in a previously suggested benchmark test. PQSQ-based approximators can be less accurate than the exact algorithms for optimizing $L_1$-based functions utilizing linear programming: however, they are several orders of magnitude faster. PQSQ potential can be applied in the task of regression, replacing the classical Least Squares or $L_1$-based Least Absolute Deviation methods. At the same time, PQSQ-based approximators can imitate a variety of subquadratic error potentials (not limited to $L_1$ or variations), including fractional quasinorms $L_p$ ($0 < p < 1$). We demonstrate that the PQSQ potential can be easily adapted to the problems of sparse regularized regression with non-quadratic penalty on regression coefficients (including imitations of lasso and elastic net). On several real-life dataset examples we show that PQSQ-based regularized regression can
### Table 2: List of methods which can use PQSQ-based error potentials

<table>
<thead>
<tr>
<th>Data approximation/Clustering/Manifold learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Principal Component Analysis</td>
</tr>
<tr>
<td>Principal curves and manifolds</td>
</tr>
<tr>
<td>Self-Organizing maps</td>
</tr>
<tr>
<td>Principal graphs/trees</td>
</tr>
<tr>
<td>K-means</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>High-dimensional data mining</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use of fractional quasinorms $L_p$ ($0 &lt; p &lt; 1$)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Regularized and sparse regression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lasso</td>
</tr>
<tr>
<td>Elastic net</td>
</tr>
</tbody>
</table>

perform two orders of magnitude faster than the lasso algorithm implemented in the same programming environment.

To conclude, in Table 2 we list possible applications of the PQSQ-based framework in machine learning.

**Acknowledgement**

This study was supported in part by Big Data Paris Science et Lettre Research University project ‘PSL Institute for Data Science’.

**References**


