A generic coordinate descent solver for non-smooth convex optimisation

Olivier Fercoq

ABSTRACT

We present a generic coordinate descent solver for the minimisation of a non-smooth convex objective with structure. The method can deal in particular with problems with linear constraints. The implementation makes use of efficient residual updates and automatically determines which dual variables should be duplicated. A list of basic functional atoms is pre-compiled for efficiency and a modelling language in Python allows the user to combine them at run time. So, the algorithm can be used to solve a large variety of problems including Lasso, sparse multinomial logistic regression, linear and quadratic programmes.

1. Introduction

Coordinate descent methods decompose a large optimisation problem into a sequence of one-dimensional optimisation problems. The algorithm was first described for the minimisation of quadratic functions by Gauss and Seidel in [49]. Coordinate descent methods have become unavoidable in machine learning because they are very efficient for key problems, namely Lasso [20], logistic regression [67] and support vector machines [43,52]. Moreover, the decomposition into small subproblems means that only a small part of the data is processed at each iteration and this makes coordinate descent easily scalable to high dimensions.

One of the main ingredients of an efficient coordinate descent solver is its ability to compute efficiently partial derivatives of the objective function [39]. In the case of least squares for instance, this involves the definition of a vector of residuals that will be updated during the run of the algorithm. As this operation needs to be performed at each iteration, and millions of iterations are usually needed, the residual update and directional derivative computation must be coded in a compiled programming language.

Many coordinate descent solvers have been written in order to solve a large variety of problems. However, most of the existing solvers can only solve problems of the type

$$\min_{x \in \mathbb{R}^N} \sum_{j=1}^{J} f(A_j x - b_j) + \sum_{i=1}^{I} g(x^{(i)})$$
where \( x^{(i)} \in \mathbb{R}^{N_i} \) is the \( i \)th block of \( x \), \( \sum_{i=1}^{I} N_i = N \), \( A_j \in \mathbb{R}^{M_j \times N} \) is a matrix and \( b_j \in \mathbb{R}^{M_j} \) is a vector, and where \( f \) is a convex differentiable function and \( g \) is a convex lower-semicontinuous function whose proximal operator is easy to compute (a.k.a. a proximal-friendly convex function). Each piece of code usually covers only one type of function \([14,42]\). Moreover, even when the user has a choice of objective function, the same function is used for every block \([6]\).

In this work, we propose a generic coordinate descent method for the resolution of the convex optimisation problem

\[
\min_{x \in \mathbb{R}^N} \frac{1}{2} x^\top Q x + \sum_{j=1}^{J} c_{j}^f f_j(A_j^f x - b_j^f) + \sum_{i=1}^{I} c_{i}^g g_i(D_i^g x^{(i)} - b_i^g) + \sum_{l=1}^{L} c_{l}^h h_l(A_l^h x - b_l^h). \tag{1}
\]

We shall call \( f_j, g_i \), and \( h_l \) atom functions. Each of them may be different. We will assume that \( f_j \)'s are differentiable and convex, \( g_i \)'s and \( h_l \)'s are proximal-friendly convex functions. As before \( A_j^f \in \mathbb{R}^{M_j \times N} \) and \( A_l^h \in \mathbb{R}^{M_l \times N} \) are matrices, \( D_i^g \) is a multiple of the identity matrix of size \( N_i \), \( b_j^f \in \mathbb{R}^{M_j} \), \( b_i^g \in \mathbb{R}^{N_i} \) and \( b_l^h \in \mathbb{R}^{M_l} \) are vectors, \( c_{j}^f, c_{i}^g \), and \( c_{l}^h \) are positive real numbers, \( Q \) is a \( N \times N \) positive semi-definite matrix.

The algorithm we implemented is described in \([15]\) and can be downloaded on https://bitbucket.org/ofercoq/cd_solver. The present paper focuses on important implementation details about residual updates and dual variable duplication. The novelty of our code is that it allows a generic treatment of these algorithmic steps and includes a modelling interface in Python for the definition of the optimisation problem. Note that unlike most coordinate descent implementations, it can deal with non-separable non-smooth objectives and linear constraints.

### 2. Literature review on coordinate descent methods

A thorough review on coordinate descent is beyond the scope of this paper. We shall refer the interested reader to the review papers \([54,63]\). Instead, for selected papers dealing with smooth functions, separable non-smooth functions or non-separable non-smooth function, we list their main features. We also quickly review what has been done for non-convex functions. We sort papers in order of publication except when there are an explicit dependency between a paper and a follow-up.

#### 2.1. Smooth functions

Smooth objectives are a natural starting point for algorithmic innovations. The optimisation problem at stake writes

\[
\min_{x \in \mathbb{R}^N} f(x),
\]

where \( f \) is a convex differentiable function with Lipschitz-continuous partial derivatives.

In Table 1, we compare papers that introduced important improvements to coordinate descent methods. We shall in particular stress the seminal paper by Tseng and Yun \([60]\). It features coordinate gradient steps instead of exact minimisation. Indeed a coordinate
Table 1. Selected papers for the minimisation of smooth functions.

| Paper | Rate | Rand | Grad | Bck | Par | Acc | Notable feature |
|-------|------|------|------|-----|-----|-----|-----------------|
| Seidel 1974 [49] | × | N | × | × | × | × | Quadratic |
| Warga 1963 [62] | × | N | × | × | × | × | Strictly convex |
| Luo and Tseng 1992 [34] | asymp | N | × | × | × | × | Rate for weakly convex |
| Leventhal and Lewis 2008 [28] | ✓ | Y | × | × | × | × | Quadratic |
| Tseng and Yun 2009 [60] | asymp | N | ✓ | ✓ | × | × | Line search, proximal operator |
| Nesterov 2012 [39] | ✓ | Y | × | × | × | Not eff | First acc and first non-uniform |
| Beck and Tetruashvili 2013 [4] | ✓ | N | ✓ | ✓ | × | Not eff | Finite-time analysis cyclic CD |
| Lin and Xiao 2013 [33] | ✓ | Y | ✓ | ✓ | × | Not eff | Improvements on [39,45] |
| Lee and Sidford 2013 [27] | ✓ | Y | ✓ | × | × | ✓ | First efficient accelerated |
| Liu et al. 2013 [31] | ✓ | Y | ✓ | × | × | × | First asynchronous |
| Glasmachers and Dogan 2013 [22] | × | Y | ✓ | × | × | × | Heuristic sampling |
| Richtárik and Takáč 2016 [48] | ✓ | Y | ✓ | × | ✓ | × | First arbitrary sampling |
| Allen-Zhu et al. 2016 [2] | ✓ | Y | ✓ | ✓ | × | ✓ | Non-uniform sampling |
| Sun et al. 2017 [55] | ✓ | Y | ✓ | ✓ | × | ✓ | Better asynchrony than [31] |

Notes: Rate: we check whether the paper proves convergence (×), an asymptotic rate (asymp) or a finite-time iteration complexity (√). Rand: deterministic (N) or randomised (Y) selection of coordinates. Grad: exact minimisation (×) or use of partial derivatives (√). Bck: the paper considers 1D coordinates (×) or blocks of coordinates (√). Par: Algorithm designed for parallel computing (√). Acc: no momentum (×), accelerated but not efficient in practice (not eff), accelerated algorithm (√).

Gradient steps gives similar iteration complexity both in theory and in practice for a much cheaper iteration cost. Moreover, this opened the door for many innovations: blocks of coordinates and the use of proximal operators were developed in the same paper. Another crucial step was made in [39]: Nesterov showed how randomisation can help finding finite-time complexity bounds and proposed an accelerated version of coordinate descent. He also proposed to use a non-uniform sampling of coordinates depending on the coordinate-wise Lipschitz constants.

2.2. Separable non-smooth functions

A large literature has been devoted to composite optimisation problems with separable non-smooth functions:

$$\min_{x \in \mathbb{R}^N} f(x) + \sum_{i=1}^{n} g_i(x^{(i)}),$$

where $f$ is a convex differentiable function with Lipschitz-continuous partial derivatives and for all $i$, $g_i$ is a convex function whose proximal operator is easy to compute. Indeed, regularised expected risk minimisation problems often fit into this framework and this made the success of coordinate descent methods for machine learning applications.

Some papers study $\min_{x \in \mathbb{R}^p} f(x) + \sum_{i=1}^{n} g_i((Ax)^{(i)})$, where $f$ is strongly convex and apply coordinate descent to a dual problem written as

$$\min_{y \in \mathbb{R}^N} f^*(-A^T y) + \sum_{i=1}^{n} g_i^*(y^{(i)}).$$

One of the challenges of these works is to show that even though we are solving the dual problem, one can still recover rates for a sequence minimising the primal objective.

We present our selection of papers devoted to this type of problems in Table 2.
Table 2. An overview of selected papers proposing and analysing the iteration complexity of coordinate descent methods for separable non-smooth objectives.

| Paper | Prx | Par | Acc | Dual | Notable feature |
|-------|-----|-----|-----|------|-----------------|
| Tseng and Yun 2009 [60] | ✓ | × | × | × | First prox, line search, deterministic |
| S-Shwartz and Tewari 2009 [50] | ✓ | × | ✓ | × | First ℓ₁-regularised with finite-time bound |
| Bradley et al. 2011 [7] | × | ✓ | × | × | First proximal with finite-time bound |
| S-Shwartz and Zhang 2013 [51] | ✓ | × | × | ✓ | First dual |
| Richtárik and Takáč 2014 [45] | ✓ | × | ✓ | × | First ℓ₁-regularised parallel |
| S-Shwartz and Zhang 2014 [53] | ✓ | × | ✓ | ✓ | Acceleration in the primal |
| Yun 2014 [68] | ✓ | × | × | × | Analysis of cyclic CD |
| S-Shwartz and Zhang 2013 [51] | ✓ | × | ✓ | ✓ | First dual and parallel |
| Richtárik and Takáč 2015 [46] | ✓ | × | × | ✓ | First general parallel |
| Bradley et al. 2011 [7] | × | ✓ | × | × | First ℓ₁-regularised parallel |
| S-Shwartz and Zhang 2013 [51] | ✓ | × | × | ✓ | First dual |
| Richtárik and Takáč 2015 [46] | ✓ | × | × | ✓ | First general parallel |
| S-Shwartz and Zhang 2014 [53] | ✓ | × | ✓ | ✓ | Acceleration in the primal |
| Yun 2014 [68] | ✓ | × | × | × | Analysis of cyclic CD |
| S-Shwartz and Zhang 2013 [51] | ✓ | × | ✓ | ✓ | First dual and parallel |
| Richtárik and Takáč 2015 [46] | ✓ | × | × | ✓ | First general parallel |
| S-Shwartz and Zhang 2014 [53] | ✓ | × | ✓ | ✓ | Acceleration in the primal |
| Yun 2014 [68] | ✓ | × | × | × | Analysis of cyclic CD |
| S-Shwartz and Zhang 2013 [51] | ✓ | × | ✓ | ✓ | First dual and parallel |
| Richtárik and Takáč 2015 [46] | ✓ | × | × | ✓ | First general parallel |
| S-Shwartz and Zhang 2014 [53] | ✓ | × | ✓ | ✓ | Acceleration in the primal |

Notes: Prx: uses a proximal operator to deal with the non-smooth part of the objective. Par: updates several blocks of coordinates in parallel. Acc: uses momentum to obtain an improved rate of convergence. Dual: solves a dual problem but still proves rates in the primal (only relevant for weakly convex duals).

2.3. Non-separable non-smooth functions

Non-separable non-smooth objective functions are much more challenging to coordinate descent methods. One wishes to solve

$$\min_{x \in \mathbb{R}^N} f(x) + g(x) + h(Ax),$$

where $f$ is a convex differentiable function with Lipschitz-continuous partial derivatives, $g$ and $h$ are convex functions whose proximal operator are easy to compute and $A$ is a linear operator. Indeed, the linear operator introduces a coupling between the coordinates and a naive approach leads to a method that does not converge to a minimiser [3]. When $h = \iota_{\{b\}}$, the convex indicator function of the set $\{b\}$, we have equality constraints.

Table 3. Selected papers for the minimisation of non-smooth non-separable functions.

| Paper | Rate | Const | PD-CD | Notable feature |
|-------|------|-------|-------|----------------|
| Platt 1999 [43] | × | ✓ | P | For SVM |
| Tseng and Yun 2009 [61] | ✓ | × | P | Adapts Gauss–Southwell rule |
| Tao et al. 2012 [57] | ✓ | × | P | Uses averages of subgradients |
| Necoara and Clipici 2013 [37] | ✓ | × | P | Two-coordinate descent |
| Nesterov 2012 [40] | ✓ | × | P | Uses subgradients |
| Combettes and Pesquet 2014 [11] | ✓ | ✓ | P | Coupled constraints |
| Bianchi et al. 2014 [5] | ✓ | × | P | Distributed optimisation |
| Hong et al. 2014 [24] | × | ✓ | P | Updates all dual variables |
| Alacaoglu et al. 2017 [19] | ✓ | × | P | Uses smoothing |
| Chambolle et al. 2017 [1] | ✓ | × | P | Updates all primal variables |
| Xu and Zhang 2018 [66] | ✓ | × | P | Better rate than [21] |
| Fercioq and Bianchi 2019 [15] | ✓ | ✓ | P | First PD-CD w rate for constraints |
| Gao et al. 2019 [21] | ✓ | ✓ | P | First primal–dual w rate for constraints |
| Latafat et al. 2019 [26] | ✓ | ✓ | P | Linear conv w growth condition |

Notes: Rate: does the paper prove rates? Const: can the method solve problems with linear equality constraints? PD-CD: the method is purely primal (P), the method updates some primal variables but all the Lagrange multipliers or some dual variables but all the primal variables (×), the method updates some primal and some dual variables at each iteration (√).
Table 4. Selected papers for the local minimisation of non-convex problems.

| Paper                  | Conv | Smth | Nsmth | Notable feature                          |
|------------------------|------|------|-------|------------------------------------------|
| Grippo and Sciandrone 2000 [23] | ×    | ✓    | ×     | Two-block or coordinate-wise quasiconvex |
| Tseng and Yun 2009 [60]  | ✓    | ×    | ✓     | Convergence under error bound            |
| Hsieh and Dhillon 2011 [25] | ×    | ✓    | ×     | Non-negative matrix factorisation        |
| Breheny and Huang 2011 [8] | ×    | ✓    | ✓     | Regularised least squares                |
| Mazumder et al. 2011 [35]  | ×    | ✓    | ✓     | Regularised least squares                |
| Razaviyayn et al. 2013 [44] | ×    | ✓    | ✓     | Requires uniqueness of the prox          |
| Xu and Yin 2013 [64]  | ✓    | ✓    | ×     | Multiconvex                              |
| Lu and Xiao 2013 [32]  | ×    | ✓    | ✓     | Random sampling                          |
| Patrascu and Necoara 2015 [41] | ✓    | ✓    | ×     | Randomised + 1 linear constraint         |
| Xu and Yin 2017 [65]  | ✓    | ✓    | ✓     | Convergence under KL                     |

Notes: Conv: study only limit points (×) or convergence of the sequence proved (✓). Smth: can deal with non-convex smooth functions. Nsmth: can deal with non-convex and non-smooth functions.

We present our selection of papers devoted to this type of problems in Table 3.

2.4. Non-convex functions

The goal here is to find a local minimum to the problem

$$\text{loc} - \min_{x \in \mathbb{R}^N} f(x) + \sum_{i=1}^n g_i(x^{(i)})$$

without any assumption on the convexity of $f$ nor $g_i$. The function $f$ should be continuously differentiable and the proximal operator of each function $g_i$ should be easily computable. Note that in the non-convex setting, the proximal operator may be set-valued (Table 4).

3. Description of the algorithm

3.1. General scheme

The algorithm we implemented is a coordinate descent primal–dual method developed in [15]. Let introduce the notation $F(x) = \frac{1}{2}x^\top Qx + \sum_{j=1}^J c_j f_j(A_j^f x - b_j^f)$, $G(x) = \sum_{i=1}^I c_i^g g_i(D_i^f x^{(i)} - b_i^f)$, $H(z) = \sum_{l=1}^L c_l^h h_l(z^{(l)} - b_l^h)$, $\mathcal{J}(i) = \{j : A_{j,i}^h \neq 0\}$, $\mathcal{I}(j) = \{i : A_{i,j}^f \neq 0\}$, $m_j = |\mathcal{I}(j)|$ and $\rho(A)$ the spectral radius of matrix $A$. We shall also denote $\mathcal{J}^f(i) = \{j : A_{j,i}^f \neq 0\}$, $\mathcal{J}^Q(i) = \{j : Q_{ij} \neq 0\}$, $A_i^f \in \mathbb{R}^{\sum_j M_j^f \times N}$ the matrix which stacks the matrices $(A_{i,j}^f)_{1 \leq j \leq J}$ and $A_i^h$ the matrix which stacks the matrices $(A_{i,j}^h)_{1 \leq j \leq L}$. The algorithm writes then as Algorithm 1.

We will denote $U_1, \ldots, U_I$ the columns of the identity matrix corresponding to the blocks of $x = (x^{(1)}, \ldots, x^{(I)})$, so that $U_i x^{(i)} \in \mathbb{R}^N$ and $V_1, \ldots, V_I$ the columns of the identity matrix corresponding to the blocks of $A_i^f x - b_i^f = (A_{1,i}^f x - b_1^f, \ldots, A_{I,i}^f x - b_I^f)$.

3.2. Computation of partial derivatives

For simplicity of implementation, we are assuming that $G$ is separable and the blocks of variable will follow the block structure of $G$. This implies in particular that at each
Algorithm 1 Coordinate descent primal–dual algorithm with duplicated variables (PD-CD)

**Input**: Differentiable function $F : \mathbb{R}^N \to \mathbb{R}$, matrix $A^h \in \mathbb{R}^{M \times N}$, functions $G$ and $H$ whose proximal operators are available.

**Initialisation**: Choose $x_0 \in \mathbb{R}^N, y_0 \in \mathbb{R}^{\text{nnz}(A^h)}$. Denote $\mathcal{J}(i) = \{ j : A^h_{j,i} \neq 0 \}, \mathcal{I}(j) = \{ i : A^h_{j,i} \neq 0 \}, m_j = |\mathcal{I}(j)|$ and $\rho(A)$ the spectral radius of matrix $A$. Choose step sizes $\tau \in \mathbb{R}_+$ and $\sigma \in \mathbb{R}_+$ such that $\forall i \in \{1, \ldots, I\}$,

$$
\tau_i < \frac{1}{\beta_i + \rho \left( \sum_{j \in \mathcal{J}(i)} m_j \sigma_j \langle A^h_j, A^h_{j,i} \rangle \right)}.
$$

(1)

For all $i \in \{1, \ldots, I\}$, set $w^{(i)}_0 = \sum_{j \in \mathcal{J}(i)} (A^h_{j,i})^\top y^{(j)}_0(i)$.

For all $j \in \{1, \ldots, J\}$, set $z^{(j)}_0 = \frac{1}{m_j} \sum_{i \in \mathcal{I}(j)} y^{(j)}_0(i)$.

**Iteration $k$**: Define:

$$
\bar{y}^{(i)}_{k+1} = \text{prox}_{\sigma, H^*}(z^{(i)}_k + D(\sigma)A^h x_k)
$$

$$
\bar{x}^{(i)}_{k+1} = \text{prox}_{\tau, G}(x_k - D(\tau) \left( \nabla F(x_k) + 2(A^h)^\top \bar{y}^{(i)}_{k+1} - w^k_k \right)).
$$

For $i = i_{k+1} \sim U(\{1, \ldots, I\})$ and for each $j \in \mathcal{J}(i_{k+1})$, update:

$$
x^{(i)}_{k+1} = \bar{x}^{(i)}_{k+1}
$$

$$
y^{(j)}_{k+1}(i) = \bar{y}^{(j)}_{k+1}
$$

$$
w^{(i)}_{k+1} = w^k_k + \sum_{j \in \mathcal{J}(i)} (A^h_{j,i})^\top (y^{(j)}_{k+1}(i) - y^{(j)}_k(i))
$$

$$
z^{(j)}_{k+1} = z^k_k + \frac{1}{m_j} (y^{(j)}_{k+1}(i) - y^{(j)}_k(i)).
$$

Otherwise, set $x^{(i')}_{k+1} = x^{(i')}_k$, $w^{(i')}_{k+1} = w^{(i')}_k$, $z^{(i')}_{k+1} = z^{(i')}_k$ and $y^{(i')}_{k+1}(i') = y^{(i')}_{k}(i')$.

This algorithm allows to efficiently compute only the partial derivative $\nabla_i F(x_k)$, which needs to be computed. It is thus inefficient because it needs to be performed at each iteration of the algorithm. We now describe the efficient residual update method, which is classically used in coordinate descent implementations [39].

Denote $r^{f_x}_k = A^f x_k - b^f$ and $r^{Q_x}_k = Q x_k$. By the chain rule, we have

$$
\nabla_i F(x_k) = \sum_{j=1}^J c^f_j (A^f_{j,i})^\top \nabla f_j(A^f_j x_k - b^f_j) + Q x_k
$$

$$
= \sum_{j \in \mathcal{J}(i)} c^f_j (A^f_{j,i})^\top \nabla f_j((r^{f_x}_k)_j) + \sum_{j \in \mathcal{I}(i)} U_i r^{Q_x}_k
$$
If $r_k^{f,x}$ and $r_k^{Q,x}$ are pre-computed, only $O(|\mathcal{J}(i)| + |\mathcal{J}(i)|)$ operations are needed.

For an efficient implementation, we will update the residuals $r_k^{f,x}$ as follows, using the fact that only the coordinate block $i_{k+1}$ is updated:

$$
\begin{align*}
  r_{k+1}^{f,x} &= A^f x_{k+1} - b^f = A^f (x_k + U_{ik+1}(x_{i_{k+1}}^{(i_{k+1})}) - x_k^{(i_{k+1})}) - b^f \\
  &= r_k^{f,x} + A^f U_{ik+1}(x_{i_{k+1}}^{(i_{k+1})}) - x_k^{(i_{k+1})} \\
  &= r_k^{f,x} + \sum_{j \in \mathcal{J}(i_{k+1})} V_j A_{j,i_{k+1}}^{f}(x_{i_{k+1}}^{(i_{k+1})}) - x_k^{(i_{k+1})}
\end{align*}
$$

Hence, updating $r_{k+1}^{f,x}$ also requires only $O(|\mathcal{J}(i_{k+1})|)$ iterations.

Similarly, updating the residuals $r_k^{Q,x}$, $r_k^{h,x} = A^h x_k - b^h$, $w_k$ and $z_k$ can be done in $O(|\mathcal{J}(i_{k+1})|) + O(|\mathcal{J}(i_{k+1})|)$ operations.

Although this technique is well known, it is not trivial how to write it in a generic fashion, since residual updates are needed at each iteration and should be written in a compiled language. We coded the residual update using abstract atom functions in order to achieve this goal.

### 3.3. Computation of proximal operators using atom functions

Another major step in the method is the computation of the $i$th coordinate of $\text{prox}_{\tau,G}(x)$ for a given $x \in \mathbb{R}^N$.

As $D^g$ is assumed to be diagonal, $G$ is separable. Hence, by the change of variable $\bar{z} = D^g_i \bar{x} - b^g_i$,

$$(\text{prox}_{\tau,G}(x))_i = \arg \min_{\bar{x} \in \mathbb{R}^N} c_i^g(D^g_i \bar{x} - b^g_i) + \frac{1}{2\tau_i} \| \bar{x} - x^{(i)} \|^2$$

$$= (D^g_i)^{-1} \left( b^g_i + \arg \min_{\bar{z} \in \mathbb{R}^Ni} c_i^g(\bar{z}) + \frac{1}{2\tau_i} \| (D^g_i)^{-1}(b^g_i + \bar{z}) - x^{(i)} \|^2 \right)$$

$$= (D^g_i)^{-1} \left( b^g_i + \arg \min_{\bar{z} \in \mathbb{R}^Ni} g_i(\bar{z}) + \frac{1}{2\tau_i c_i^g(D^g_i)^2} \| \bar{z} - (D^g_i x^{(i)} - b^g_i) \|^2 \right)$$

$$= (D^g_i)^{-1} \left( b^g_i + \text{prox}_{\frac{1}{\tau_i c_i^g(D^g_i)^2}}(D^g_i x^{(i)} - b^g_i) \right),$$

where we used the abuse of notation that $D^g_i$ is either the scaled identity matrix or any of its diagonal elements. This derivation shows that to compute $(\text{prox}_{\tau,G}(x))_i$ we only need linear algebra and the proximal operator of the atom function $g_i$.

We can similarly compute $\text{prox}_H$. To compute $\text{prox}_{\sigma,H^*}$, we use Moreau’s formula:

$$(\text{prox}_{\sigma,H^*}(z)) = z - D(\sigma) \text{prox}_{\sigma^{-1},H}(D(\sigma)^{-1} z)$$

### 3.4. Duplication of dual variables

Algorithm 1 maintains duplicated dual variables $y_k \in \mathbb{R}^{nnz(A^h)}$ as well as averaged dual variables $z_k \in \mathbb{R}^{M^h}$ where $M^h = \sum_{i=1}^{L} M^h_i$ and $A^h_{i,j}$ is of size $M^h_i \times N_i$. The sets $\mathcal{J}(i)$
for all $i$ are given by the sparse column format representation of $A^h$. Yet, for all $i$, we need to construct the set of indices of $y_{k+1}$ that need to be updated. This is the table dual_vars_to_update in the code. Moreover, as $H$ is not separable in general, in order to compute $\tilde{y}_{k+1}^j$, for $j \in \mathcal{J}(ik+1)$, we need to determine the set of dual indices $j'$ that belong to the same block as $j$ with respect to the block decomposition of $H$. This is the purpose of the tables inv_blocks_h and blocks_h.

The procedure allows us to only compute the entries of $\tilde{y}_{k+1}$ that are required for the update of $y_k$.

4. Code structure

The code is organised in nine files. The main file is cd_solver.pyx. It contains the Python callable and the data structure for the problem definition. The other files are atoms.pyx/pxd, algorithm.pyx/pxd, helpers.pyx/pxd and

![Figure 1. Code structure when using no screening and no inertial acceleration.](image-url)
screening.pyx/pxd. They contain the definition of the atom functions, the algorithms and the functions for computing the objective value. In Figure 1, we show for each subfunction, in which function it is used. The user needs to call the Python class Problem and the Python function coordinate_descent. Atom functions can be added by the user without modifying the main algorithm.

All tables are defined using NumPy’s array constructor in the coordinate_descent function. The main loop of coordinate descent and the atom functions are pre-compiled for efficiency.

5. Atom functions

The code allows us to define atom functions independently of the coordinate descent algorithm. As an example, we provide in Figure 2 the code for the square function atom.

As inputs, it gets \( x \) (an array of numbers which is the point where the operation takes place), \( \text{buff} \) (the buffer for vectorial outputs), \( \text{nbCoord} \) (is the size of \( x \)), \( \text{mode} \), \( \text{proxParam} \) and \( \text{proxParam2} \) (numbers which are needed when computing the proximal operator). The input \( \text{mode} \) can be:

- **GRAD** in order to compute the gradient.
- **PROX** to compute the proximal operator.
- **PROX_CONJ** uses Moreau’s formula to compute the proximal operator of the conjugate function.
- **LIPSCHITZ** to return the Lipschitz constant of the gradient.

cdef DOUBLE square(DOUBLE[:,: ] x, DOUBLE[:,: ] buff, int nb_coord, MODE mode, 
                    DOUBLE prox_param, DOUBLE prox_param2) nogil:
    # Function x -> x**2
    cdef int i
    cdef DOUBLE val = 0.
    if mode == GRAD:
        for i in range(nb_coord):
            buff[i] = 2. * x[i]
        return buff[0]
    elif mode == PROX:
        for i in range(nb_coord):
            buff[i] = x[i] / (1. + 2. * prox_param)
        return buff[0]
    elif mode == PROX_CONJ:
        return prox_conj(square, x, buff, nb_coord, prox_param, prox_param2)
    elif mode == LIPSCHITZ:
        buff[0] = 2.
        return buff[0]
    elif mode == VAL_CONJ:
        return val_conj_not_implemented(square, x, buff, nb_coord)
    else:    # mode == VAL
        for i in range(nb_coord):
            val += x[i] * x[i]
        return val

**Figure 2.** Code for the square function atom.
- VAL_CONJ to return the value of the conjugate function. As this mode is used only by compute_smoothed_gap for printing purposes, its implementation is optional and can be approximated using the helper function val_conj_not_implemented. Indeed, for a small $\epsilon > 0$, $h^*(y) = \sup_z \langle z, y \rangle - h(z) \approx \sup_z \langle z, y \rangle - h(z) - \frac{\epsilon}{2} \|z\|^2 = \langle p, y \rangle - h(p) - \frac{\epsilon}{2} \|p\|^2$, where $p = \text{prox}_{h/\epsilon}(y/\epsilon)$.
- VAL to return the value of the function.

Some functions naturally require multi-dimensional inputs, like $\|\cdot\|_2$ or the log-sum-exp function. For consistency, we define all the atoms with multi-dimensional inputs: for an atom function $f_0 : \mathbb{R} \rightarrow \mathbb{R}$, we extend it to an atom function $f : \mathbb{R}^{N_i} \rightarrow \mathbb{R}$ by $f(x) = \sum_{i=1}^{N_i} f_0(x_i)$.

For efficiency purposes, we are bypassing the square atom function when computing a gradient and implemented it directly in the algorithm.

### 6. Modelling language

In order to use the code in all its generality, we defined a modelling language that can be used to define the optimisation problem we want to solve (1).

The user defines a problem using the class `Problem`. Its arguments can be:

- $N$ the number of variables, `blocks` the blocks of coordinates coded in the same fashion as the `indptr` index of sparse matrices (default [0, 1, ..., $N$]), `x_init` the initial primal point (default 0) and `y_init` the initial duplicated dual variable (default 0)
- Lists of strings `f`, `g` and `h` that code for the atom functions used. The function `string_to_func` is responsible for linking the atom function that corresponds to the string. Our convention is that the string code is exactly the name of the function in `atoms.pyx`. The size of the input of each atom function is defined in `blocks_f`, `blocks` and `blocks_h`. The function strings `f`, `g` or `h` may be absent, which means that the function does not appear in the problem to solve.
- Arrays and matrices `cf`, `Af`, `bf`, `cg`, `Dg`, `bg`, `ch`, `Ah`, `bh`, `Q`. The class initiator transforms matrices into the sparse column format and checks whether `Dg` is diagonal.

For instance, in order to solve the Lasso problem, $\min_x \frac{1}{2} \|Ax - b\|_2^2 + \lambda \|x\|_1$, one can type

```python
pb_lasso = cd_solver.Problem(N=A.shape[1],
    f=["square"]*A.shape[0], Af=A,
    bf=b, cf=[0.5]*A.shape[0], g=["abs"]*A.shape[1],
    cg=[lambda]*A.shape[1])
    cd_solver.coordinate_descent(pb_lasso)
```

### 7. Extensions

#### 7.1. Non-uniform probabilities

We added the following feature for an improved efficiency. Under the argument `sampling = 'kink_half'`, the algorithms periodically detects the set of blocks $I_{kink}$
such that \( i \in I_{\text{kink}} \) if \( x^{(i)} \) is at a kink of \( g_i \). Then, block \( i \) is selected with probability law

\[
P(i_{k+1} = i) = \begin{cases} 
\frac{1}{n} & \text{if } |I_{\text{kink}}| = n \\
\frac{1}{2n} & \text{if } |I_{\text{kink}}| < n \quad \text{and} \quad i \in I_{\text{kink}} \\
\frac{1}{2n} + \frac{1}{2(n-|I_{\text{kink}}|)} & \text{if } |I_{\text{kink}}| < n \quad \text{and} \quad i \not\in I_{\text{kink}}
\end{cases}
\]

The rationale for this probability law is that blocks at kinks are likely to incur no move when we try to update them. We thus put more computational power for non-kinks. On the other hand, we still keep an update probability weight of at least \( \frac{1}{2n} \) for each block, so even in unfavourable cases, we should not observe too much degradation in the performance as compared to the uniform law.

### 7.2. Acceleration

We also coded accelerated coordinate descent [18], as well as its restarted [16] and primal–dual [1] variants. The algorithm is given in Algorithm 2. As before, \( \tilde{y}_{k+1} \) and \( \tilde{x}_{k+1} \) should not be computed: only the relevant coordinates should be computed.

The accelerated algorithms improve the worst case guarantee as explained in Table 5:

However, accelerated algorithms do not take profit of regularity properties of the objective like strong convexity. Hence, they are not guaranteed to be faster, even though restart may help.

### 7.3. Variable screening

The code includes the Gap Safe screening method presented in [36]. Note that the method has been studied only for the case where \( h = 0 \). Given a non-differentiability point \( \hat{x}^{(i)} \) of the function \( g_i \) where the subdifferential \( \partial g_i(\hat{x}^{(i)}) \) has a non-empty interior, a test is derived to check whether \( \hat{x}^{(i)} \) is the \( i \)th variable of an optimal solution. If this is the case, one can set \( x^{(i)} = \hat{x}^{(i)} \) and stop updating this variable. This may lead to a huge speed up in some cases. As the test relies on the computation of the duality gap, which has a nonnegligible cost, it is only performed from time to time.

In order to state Gap Safe screening in a general setting, we need the concept of polar of a support function. Let \( C \) be a convex set. The support function of \( C \) is \( \sigma_C \) defined by

\[
\sigma_C(x) = \sup_{y \in C} \langle y, x \rangle.
\]

The polar to \( \sigma_C \) is

\[
\sigma_C^\circ(\tilde{x}) = \sup_{x : \sigma_C(x) \leq 1} \langle \tilde{x}, x \rangle.
\]

### Table 5. Convergence speed of the algorithms implemented

| Algorithm                          | \( h = 0 \)         | \( h \neq 0 \)       |
|-----------------------------------|---------------------|----------------------|
| PD-CD Algorithm 1                 | \( O(1/k) \)        | \( O(1/\sqrt{k}) \)  |
| APPROX / SMART-CD Algorithm 2     | \( O(1/k^2) \)      | \( O(1/k) \)         |
\begin{algorithm}
\textbf{Algorithm 2} Smooth, accelerate, randomise the Coordinate Descent (APPROX/SMART-CD)
\end{algorithm}

\textbf{Input:} Differentiable function $F : \mathbb{R}^N \to \mathbb{R}$, matrix $A^h \in \mathbb{R}^{M^h \times N}$, functions $G$ and $H$ whose proximal operators are available.

\textbf{Initialisation:} Choose $x_0 \in \mathbb{R}^N$, $y_0 \in \mathbb{R}^{M^h}$. Choose $\gamma_1 > 0$ and denote $B_0^i = \beta_i + \frac{\rho((A^h_i)^T A^h_i)}{\gamma_1}$.

Set $s = 0$, $\theta_0 = \frac{1}{\theta_0}$, $c_0 = 1$, $\hat{x}_0 = 0 \in \mathbb{R}^N$ and $\bar{x}_0 = x_0$.

\textbf{Iteration $k$}: Define:

\begin{align*}
\bar{y}_{k+1} &= \text{prox}_{\gamma_{k+1}H^*}(\bar{y}_s + D(\gamma_{k+1})^{-1}(c_k A^h \bar{x}_k + A^h \bar{x}_{k})) \\
\bar{x}_{k+1} &= \text{prox}_{\theta_k B_k^{-1}} G\left(\bar{x}_k - \frac{\theta_0}{\theta_k} D(B_k)^{-1}\left(\nabla F(c_k \hat{x}_k + \bar{x}_k) + (A^h)^T \bar{y}_{k+1}\right)\right).
\end{align*}

For $i = i_{k+1} \sim U(\{1, \ldots, I\})$, update:

\begin{align*}
\hat{x}_{k+1}^{(i)} &= \bar{x}_{k+1}^{(i)}, \\
\check{x}_{k+1}^{(i)} &= \hat{x}_k^{(i)} - \frac{1 - \theta_k/\theta_0}{c_k}(\bar{x}_{k+1}^{(i)} - \hat{x}_k^{(i)}).
\end{align*}

Otherwise, set $x_{k+1}^{(i)} = \hat{x}_k^{(i)}$.

Compute $\theta_{k+1} \in (0, 1)$ as the unique positive root of

\begin{align*}
\begin{cases}
\theta^3 + \theta^2 + \theta^2 \theta - \theta^2 = 0 & \text{if } h \neq 0 \\
\theta^2 + \theta^2 \theta - \theta^2 = 0 & \text{if } h = 0
\end{cases}
\end{align*}

Update $\gamma_{k+2} = \frac{\gamma_{k+1}}{1 + \theta_{k+1}}$, $c_{k+1} = (1 - \theta_{k+1})c_k$ and $B_{k+1}^i = \beta_i + \frac{\rho((A^h_i)^T A^h_i)}{\gamma_{k+2}}$ for all $i$.

If Restart($k$) is true:

Set $x_{k+1} = \hat{x}_{k+1} + c_k \hat{x}_{k+1}$.

Set $\bar{y}_{k+1} = \bar{y}_{k+1}$ and $s \leftarrow s + 1$.

Reset $\hat{x}_{k+1} \leftarrow 0$, $\check{x}_{k+1} \leftarrow x_{k+1}$, $c_{k+1} \leftarrow 1$, $\theta_{k+1} \leftarrow 0$, $\beta_{k+1} \leftarrow 1$.

In particular, if $\bar{x} \in C$, then $\sigma_C^2(\bar{x}) \leq 1$. Denote $f(z) = \sum_{j=1}^J c_j f_j(z^{(i)} - b_j^i)$, so that $(\nabla f(A^f x))_j = c_j f_j(A^f x - b_j^i)$, $G_i(x) = c_i g_i(D_i^f x^{(i)} - b_j^i)$, $x_*$ a solution to the optimisation problem (1) and suppose we have a set $\mathcal{R}$ such that $(\nabla f(A^f x_* - b^f), Q x_*) \in \mathcal{R}$. Gap Safe screening states that

\[
\max_{(\zeta, \omega) \in \mathcal{R}} \sigma_{\partial G_i(\zeta^{(i)})}^2((A^f_i)^T \zeta + \omega^{(i)}) < 1 \implies x_*^{(i)} = \hat{x}^{(i)}.
\]

Denote $z = \nabla f(A^f x)$ and $w = Q x^1$. We choose $\mathcal{R}$ as a sphere centred at

\[
(\bar{\zeta}, \bar{\omega}) = \frac{(z, w)}{\max(1, \max_{1 \leq i \leq I} (\sigma_{\partial G_i}^2((A^f_i)^T z + w^{(i)})))}.
\]
and with radius
\[
r = \sqrt{\frac{2\text{Gap}(x, \tilde{\zeta}, \tilde{\omega})}{L_{f,Q}}},
\]
where
\[
L_{f,Q} = \max(\max_{1 \leq j \leq J} L(\nabla f_j), \max_{1 \leq i \leq I} \rho(Q_i,i))
\]
and
\[
\text{Gap}(x, \tilde{\zeta}, \tilde{\omega}) = \frac{1}{2}x^\top Qx + f(A^fx) + G(x) + G^*(-(A^f)^\top \tilde{\zeta} - \tilde{\omega}) + \frac{1}{2}\tilde{\omega}^\top Q^\dagger \tilde{\omega} + f^*(\tilde{\zeta}).
\]

Note that as \(\tilde{\omega}\) is a rescaled version of \(w = Qx\), we do not need to know \(Q^\dagger\) in order to compute \(\tilde{\omega}^\top Q^\dagger \tilde{\omega}\). It is proved in [36] that this set \(\mathcal{R}\) contains \((\nabla f(A^fx^*) - b^f), Qx^*)\) for any optimal solutions \(x^*\) to the primal problem. In the case where \(G_i\) is a norm and \(x^{(i)} = 0\), these expressions simplify since the \(\sigma^\circ_{\partial G_i(0)} = \sigma^\circ_{\text{dom}G_i^*}\) is nothing else than the dual norm associated to \(G_i\).

For the estimation of \(\max(\zeta, \omega) \in \mathcal{R} \sigma^\circ_{\partial G_i(\hat{x}^{(i)})}((A^f)^\top \zeta + \omega^{(i)})\), we use the fact that the polar of a support function is sublinear and positively homogeneous. Indeed, we have
\[
\sigma^\circ_{\partial G_i(\hat{x}^{(i)})}((A^f)^\top \zeta + \omega^{(i)}) = \sup_{x : \sigma^\circ_{\partial G_i(\hat{x}^{(i)})}(x) \leq 1} \langle (A^f)^\top \zeta + \omega^{(i)}, x \rangle
\]
\[
= \sup_{x : \sigma^\circ_{\partial G_i(\hat{x}^{(i)})}(x) \leq 1} \langle (A^f)^\top \zeta + \omega^{(i)}, x \rangle
\]
\[
= \frac{1}{c_i^D_i} \sigma^\circ_{\partial G_i(D_i^\hat{x}^{(i)} - b_i^f)}((A^f)^\top \zeta + \omega^{(i)})
\]
\[
= \frac{1}{c_i^D_i} \sigma^\circ_{\partial G_i(\hat{x}_i)}((A^f)^\top \zeta + \omega^{(i)})
\]
\[
\leq \frac{1}{c_i^D_i} \sigma^\circ_{\partial G_i(\hat{x}_i)}((A^f)^\top \zeta + \omega^{(i)}) + r \sup_{(u,v)||u,v||=1} \frac{1}{c_i^D_i} \sigma^\circ_{\partial G_i(\hat{x}_i)}((A^f)^\top u + v^{(i)}).
\]

Here \(\hat{x}_i = D_i^\hat{x}^{(i)} - b_i^f\) is a point where \(\partial G_i(\hat{x}_i)\) has a non-empty interior. Some care should be taken when \(\partial G_i(\hat{x}^{(i)})\) is unbounded, so that we first check whether \((A^f)^\top \zeta + \omega^{(i)} \in \text{dom} \sigma^\circ_{\partial G_i(\hat{x}_i)}\) for all \((\zeta, \omega) \in \mathcal{R}\).

Here also, the novelty lies in the generality of the implementation.

8. Numerical validation

8.1. Performance

In order to evaluate the performance of the implementation, we compare our implementation with a pure Python coordinate descent solver and code written for specific problems:
Table 6. Comparison of our code with a pure Python code and reference implementations for performing $100N$ coordinate descent iterations for the Lasso problem on the Leukemia dataset with regularisation parameter $\lambda = 0.1 \| (A_f^T b_f^T) \|_\infty$, and for $10N$ coordinate descent iterations for the dual SVM problem on the RCV1 dataset with penalty parameter $C = 10$.

|         | Lasso | SVM   |
|---------|-------|-------|
| Pure Python | 308.76 s | Pure Python | 126.24 s |
| cd_solver | 0.43 s | cd_solver | 0.31 s |
| Scikit-learn Lasso | 0.11 s | Liblinear SVM | 0.13 s |

Scikit-learn’s Lasso solver and Liblinear’s SVM solver. We run the code on an Intel Xeon CPU at 3.07 GHz.

We can see on Table 6 that our code is hundreds of times faster than the pure Python code. This is due to the compiled nature of our code, that does not suffer from the huge number of iterations required by coordinate descent. On the other hand, our code is about four times slower than state-of-the-art coordinate descent implementations designed for a specific problem. We can see it in both examples we chose. This overhead is the price of genericity.

We believe that, except for critical applications like Lasso or SVM, a four times speed-up does not justify writing a new code from scratch, since a separate piece of code for each problem makes it difficult to maintain and to improve with future algorithmic advances.

8.2. Genericity

We tested our algorithm on the following problems:

- **Lasso problem**

  $$\min_{x \in \mathbb{R}^n} \frac{1}{2} \| Ax - b \|^2 + \lambda \| x \|_1$$

- **Binomial logistic regression**

  $$\min_{x \in \mathbb{R}^n} \sum_{i=1}^m \log(1 + \exp(b_i (Ax)_i)) + \frac{\lambda}{2} \| x \|^2_2,$$

  where $b_i \in \{-1, 1\}$ for all $i$.

- **Sparse binomial logistic regression**

  $$\min_{x \in \mathbb{R}^n} \sum_{i=1}^m \log(1 + \exp(b_i (Ax)_i)) + \lambda \| x \|_1$$

- **Dual SVM without intercept**

  $$\min_{x \in \mathbb{R}^n} \frac{1}{2\alpha} \| A^T D(b)x \|^2_2 - e^T x + \iota_{[0,1]^n}(x),$$

  where $\iota_{[0,1]^n}$ is the convex indicator function of the set $[0, 1]^n$ and encodes the constraint $x \in [0, 1]^n$. 

• Dual SVM with intercept

\[
\min_{x \in \mathbb{R}^n} \frac{1}{2\alpha} \|A^T D(b) x\|_2^2 - e^T x + \tau_{[0,1]}(x) + \tau_{\{0\}}(b^T x)
\]

• Linearly constrained quadratic programme

\[
\min_{x \in \mathbb{R}^n} \frac{1}{2} \|(A^f)^T x - b^f\|_2^2 + \tau_{\{0\}}(A^h x - b^h)
\]

• Linear programme

\[
\min_{x \in \mathbb{R}^n} c^T x + \tau_{\mathbb{R}^n_+}(x) + \tau_{\mathbb{R}^m}(Ax - b)
\]

• TV+$\ell_1$-regularised regression

\[
\min_{x \in \mathbb{R}^{n_1n_2n_3}} \frac{1}{2} \|Ax - b\|_2^2 + \alpha_1 \|Dx\|_{2,1} + \alpha_2 \|x\|_1,
\]

where $D$ is the discrete gradient operator and $\|y\|_{2,1} = \sum_{i,j,k} \sqrt{\sum_{l=1}^3 y_{i,j,k,l}^2}$.

• Sparse multinomial logistic regression

\[
\min_{x \in \mathbb{R}^{n \times q}} \sum_{i=1}^m \log \left( \sum_{j=1}^q \exp \left( \sum_{l=1}^n A_{i,j,l} x_{l,j} \right) \right) + \sum_{i=1}^n \sum_{j=1}^q x_{i,j} b_{i,j} + \sum_{l=1}^n \sqrt{\sum_{j=1}^q x_{i,j}^2},
\]

where $b_{i,j} \in \{0, 1\}$ for all $i,j$.

This list demonstrates that the method is able to deal with differentiable functions, separable or non-separable non-differentiable functions, as well as use several types of atom function in a single problem.

8.3. Benchmarking

In Table 7, we compare the performance of Algorithm 1 with and without screening, Algorithm 2 with and without screening as well as 2 alternative solvers for 3 problems exhibiting various situations:

• Lasso: the non-smooth function in the Lasso problem is separable;
• the TV-regularised regression problem has a non-smooth, non-separable regulariser whose matrix $A^h$ is sparse;
• the dual SVM with intercept has a single linear non-separable constraint.

For Algorithm 2, we set the restart with a variable sequence as in [16]. We did not tune the algorithmic parameters for each instance. We evaluate the precision of a primal–dual
Table 7. Time to reach precision $\epsilon$. \(scr\) with gap safe screening. For SVM, we scale the data so that in each column of the data matrix \(A, A_{ij} \in [-1, 1]\).

| Problem | Dataset | Alternative solver 1 | Alternative solver 2 | Algorithm 1 | Algorithm 2 |
|---------|---------|----------------------|----------------------|-------------|-------------|
| Lasso ($\lambda = 0.1 \|X^Ty\|_\infty$) | triazines ($J = 186, I = 60$) | Scikit-learn: 0.005 s | OSQP: 0.033 s | 0.107 s; scr: 0.101 s | 0.066 s; scr: 0.079 s |
| $\epsilon = 10^{-6}$ | scm1d ($J = 9803, I = 280$) | Scikit-learn: 24.40 s | OSQP: 33.21 s | 91.97 s; scr: 8.73 s | 24.13; scr: 3.21 s |
| | news20.binary ($J = 19996, I = 1355191$) | Scikit-learn: 64.6 s | OSQP: > 2000 s | 267.3 s; scr: 114.4 s | 169.8 s; scr: 130.7 s |
| TV-regularised regression | fMRI [13,58] ($\alpha_1 = \alpha_2 = 5 \times 10^{-3}$) | inexact FISTA: 24,341 s | LBFGS+homotopy: 6,893 s | > 25,000 s | 2,734 s |
| $\epsilon = 1$ | | | | | |
| Dual SVM with intercept | ionosphere ($J = 14, I = 351$) | libsvm: 0.04 s | OSQP: 0.12 s | 3.23 s | 0.42 s |
| $\epsilon = 10^{-3}$ | leukaemia ($J = 7,129, I = 72$) | libsvm: 0.1 s | OSQP: 3.2 s | 25.5 s | 0.8 s |
| | madelon ($J = 500, I = 2,600$) | libsvm: 50 s | OSQP: 37 s | 3842 s | 170 s |
| | gisette ($J = 5,000, I = 6,000$) | libsvm: 70 s | OSQP: 936 s | 170 s | 901 s |
| | rcv1 ($J = 47,236, I = 20,242$) | libsvm: 195 s | OSQP: Memory error | > 5000 s | 63 s |
pair \((x, y)\) as follows. We define the smoothed gap \([59]\) as
\[
G_{\beta, \gamma}(x, y, \zeta, \omega) = \frac{1}{2} x^\top Q x + f(A^T x) + G(x) + \max_y \langle A^h x, y \rangle - H^*(y) - \frac{\beta}{2} \|y - y'\|^2 \\
+ H^*(y) + \frac{1}{2} \omega^\top Q^\top \omega + f^*(\zeta) + \max_{x'} \langle - (A^h)^\top y - (A^f)^\top \zeta - \omega, x' \rangle - G(x') \\
- \frac{\gamma}{2} \|x - x'\|^2
\]
and we choose the positive parameters \(\beta\) and \(\gamma\) as
\[
\beta = \text{dist}(A^h x, \text{dom}H) \quad \gamma = \text{dist}(-(A^h)^\top y - (A^f)^\top \zeta - \omega, \text{dom}G^*) .
\]
It is shown in \([59]\) that when \(G_{\beta, \gamma}(x, y, \zeta, \omega)\), \(\beta\) and \(\gamma\) are small then the objective value and feasibility gaps are small.

For the Lasso problems, we compare our implementations of the coordinate descent method with scikit-learn’s coordinate descent \([42]\) and CVXPY’s augmented Lagrangian method \([12]\) called OSQP. As in Table 6, we have a factor 4 between Algorithm 1 without screening and scikit-learn. Acceleration and screening allows us to reduce this gap without sacrificing generality. OSQP is efficient on small problems but is not competitive on larger instances.

For the TV-regularised regression problems, we compare ourselves with FISTA where the proximal operator of the total variation is computed inexactly and with LBFGS where the total variation is smoothed with a decreasing smoothing parameter. Those two methods have been implemented for \([13]\). They manage to solve the problem to an acceptable accuracy in a few hours. As the problem is rather large, we did not run OSQP on it. For our methods, as \(h\) is nonzero, we cannot use variable screening with the current theory. Algorithm 1 quickly reduces the objective value but fails to get a high precision in a reasonable amount of time. On the other hand, Algorithm 2 is the quickest among the four solvers tested here (Table 7).

The third problem we tried is dual support vector machine with intercept. A very famous solver is libsvm \([10]\), which implements SMO \([43]\), a 2-coordinate descent method that ensures the feasibility of the constraints at each iteration. The conclusions are similar to what we have seen above. The specialised solver remains a natural choice. OSQP can only solve small instances. Algorithm 1 has trouble finding high accuracy solutions. Algorithm 2 is competitive with respect to the specialised solver.

9. Conclusion

This paper introduces a generic coordinate descent solver. The technical challenge behind the implementation is the fundamental need for a compiled language in the low-level operations that are partial derivative and proximal operator computations. We solved it using pre-defined atom functions that are combined at run time using a python interface.

We show how genericity allows us to decouple algorithm development from a particular application problem. As an example, our software can solve at least 12 types of optimisation problems on large instances using primal–dual coordinate descent, momentum acceleration, restart and variable screening.
As future works, apart from keeping the algorithm up to date with the state of the art, we plan to bind our solver with CVXPY in order to simplify further the user experience.

Note
1. We reuse the notation $w$ and $z$ here for the purpose of this section.

Disclosure statement
No potential conflict of interest was reported by the author.

ORCID
Olivier Fercoq http://orcid.org/0000-0002-3393-9757

References
[1] A. Alacaoglu, Q.T. Dinh, O. Fercoq, and V. Cevher, Smooth primal-dual coordinate descent algorithms for nonsmooth convex optimization, in Advances in Neural Information Processing Systems, pp. 5852–5861, 2017.
[2] Z. Allen-Zhu, Z. Qu, P. Richtárik, and Y. Yuan, Even faster accelerated coordinate descent using non-uniform sampling, in International Conference on Machine Learning, pp. 1110–1119, 2016.
[3] A. Auslender, Optimisation: Méthodes Numériques, Masson, Paris, 1976.
[4] A. Beck and L. Teteuashvili, On the convergence of block coordinate descent type methods, SIAM. J. Optim. 23(4) (2013), pp. 2037–2060.
[5] P. Bianchi, W. Hachem, and I. Franck, A stochastic coordinate descent primal-dual algorithm and applications, in 2014 IEEE International Workshop on Machine Learning for Signal Processing (MLSP), IEEE, pp. 1–6, 2014.
[6] M. Blondel and F. Pedregosa, Lightning: Large-scale linear classification, regression and ranking in Python, 2016.
[7] J.K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin, Parallel coordinate descent for L1-Regularized loss minimization, in 28th International Conference on Machine Learning, 2011.
[8] P. Breheny and J. Huang, Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection, Ann. Appl. Stat. 5(1) (2011), pp. 232.
[9] A. Chambolle, M.J. Ehrhardt, P. Richtárik, and C.-B. Schonlieb, Stochastic primal-dual hybrid gradient algorithm with arbitrary sampling and imaging applications, SIAM. J. Optim. 28(4) (2018), pp. 2783–2808.
[10] C.-C. Chang and C.-J. Lin, LIBSVM: A library for support vector machines, ACM Trans. Intell. Syst. Technol. (TIST) 2(3) (2011), pp. 27.
[11] P.L. Combettes and J.-C. Pesquet, Stochastic Quasi-Fejér block-coordinate fixed point iterations with random sweeping, SIAM. J. Optim. 25(2) (2015), pp. 1221–1248.
[12] S. Diamond and S. Boyd, CVXPY: A Python-embedded modeling language for convex optimization, J. Mach. Learn. Res. 17(83) (2016), pp. 1–5.
[13] E. Dohmatob, A. Gramfort, B. Thirion, and G. Varoquaux, Benchmarking solvers for TV-L1 least-squares and logistic regression in brain imaging, in Pattern Recognition in Neuroimaging (PRNI). IEEE, 2014.
[14] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, Liblinear: A library for large linear classification, J. Mach. Learn. Res. 9(Aug) (2008), pp. 1871–1874.
[15] O. Fercoq and P. Bianchi, A coordinate-descent primal-dual algorithm with large step size and possibly nonseparable functions, SIAM. J. Optim. 29(1) (2019), pp. 100–134.
[16] O. Fercoq and Z. Qu, Restarting the accelerated coordinate descent method with a rough strong convexity estimate. arXiv preprint arXiv:1803.05771, 2018.
[17] O. Fercoq, Z. Qu, P. Richtárik, and M. Takáč, Fast distributed coordinate descent for minimizing non-strongly convex losses, in IEEE International Workshop on Machine Learning for Signal Processing, 2014.

[18] O. Fercoq and P. Richtárik, Accelerated, parallel and proximal coordinate descent, SIAM. J. Optim. 25(4) (2015), pp. 1997–2023.

[19] O. Fercoq and P. Richtárik, Smooth minimization of nonsmooth functions with parallel coordinate descent methods, in Modeling and Optimization: Theory and Applications, Springer, pp. 57–96, 2017.

[20] J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani, Pathwise coordinate optimization, Ann. Appl. Stat. 1(2) (2007), pp. 302–332.

[21] X. Gao, Y.-Y. Xu, and S.-Z. Zhang, Randomized primal–dual proximal block coordinate updates, J. Oper. Res. Soc. China 7(2) (2019), pp. 205–250.

[22] T. Glasmachers and U. Dogan, Accelerated coordinate descent with adaptive coordinate frequencies, in Asian Conference on Machine Learning, pp. 72–86, 2013.

[23] L. Grippo and M. Sciandrone, On the convergence of the block nonlinear Gauss–Seidel method under convex constraints, Oper. Res. Lett. 26(3) (2000), pp. 127–136.

[24] M. Hong, T.-H. Chang, X. Wang, M. Razaviyayn, S. Ma, and Z.-Q. Luo, A block successive upper bound minimization method of multipliers for linearly constrained convex optimization. arXiv preprint arXiv:1401.7079, 2014.

[25] C.-J. Hsieh and I.S. Dhillon, Fast coordinate descent methods with variable selection for non-negative matrix factorization, in Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, pp. 1064–1072, 2011.

[26] P. Latafat, N.M. Freris, and P. Patrinos, A new randomized block-coordinate primal-dual proximal algorithm for distributed optimization, IEEE. Trans. Autom. Control (2019).

[27] Y.T. Lee and A. Sidford, Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. pp. 147–156, 2013.

[28] D. Leventhal and A. Lewis, Randomized methods for linear constraints: Convergence rates and conditioning, Math. Oper. Res. 35(3) (2010), pp. 641–654.

[29] H. Li and Z. Lin, On the complexity analysis of the primal solutions for the accelerated randomized dual coordinate ascent. arXiv preprint arXiv:1807.00261, 2018.

[30] Q. Lin, Z. Lu, and L. Xiao, An accelerated proximal coordinate gradient method. in Advances in Neural Information Processing Systems, pp. 3059–3067, 2014.

[31] J. Liu, S.J. Wright, C. Ré, V. Bittorf, and S. Sridhar, An asynchronous parallel stochastic coordinate descent algorithm, J. Mach. Learning Res. 16(1) (2015), pp. 285–322.

[32] Z. Lu and L. Xiao, Randomized block coordinate non-monotone gradient method for a class of nonlinear programming. arXiv preprint arXiv:1306.5918, 1273, 2013.

[33] Z. Lu and L. Xiao, On the complexity analysis of randomized block-coordinate descent methods, Math. Program. 152(1) (2015), pp. 615–642.

[34] Z.-Q. Luo and P. Tseng, On the convergence of the coordinate descent method for convex differentiable minimization, J. Optim. Theory. Appl. 72(1) (1992), pp. 7–35.

[35] R. Mazumder, J.H. Friedman, and T. Hastie, Spar森: Coordinate descent with nonconvex penalties, J. Am. Stat. Assoc. 106(495) (2011), pp. 1125–1138.

[36] E. Ndiaye, Safe optimization algorithms for variable selection and hyperparameter tuning, PhD thesis, Université Paris-Saclay, 2018.

[37] I. Necoara and D. Clipici, Efficient parallel coordinate descent algorithm for convex optimization problems with separable constraints: Application to distributed MPC, J. Process Control 23 (2013), pp. 243–253.

[38] I. Necoara, Y. Nesterov, and F. Glineur, Efficiency of randomized coordinate descent methods on optimization problems with linearly coupled constraints, Technical report, Politehnica University of Bucharest, 2012.

[39] Y. Nesterov, Efficiency of coordinate descent methods on huge-scale optimization problems, SIAM. J. Optim. 22(2) (2012), pp. 341–362.

[40] Y. Nesterov, Subgradient methods for huge-scale optimization problems. Core Discussion Paper 2012/2, 2012.
[41] A. Patrascu and Ion Necoara, *Efficient random coordinate descent algorithms for large-scale structured nonconvex optimization*, J. Global Optim. 61(1) (2015), pp. 19–46.

[42] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and J. Vanderplas, *Scikit-learn: Machine learning in python*, J. Mach. Learn. Res. 12(Oct) (2011), pp. 2825–2830.

[43] J.C. Platt, *Fast training of support vector machines using sequential minimal optimization*. in Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola, editors, *Advances in Kernel Methods –Support Vector Learning*. MIT Press, 1999. http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets.

[44] M. Razaviyayn, M. Hong, and Z.-Q. Luo, *A unified convergence analysis of block successive minimization methods for nonsmooth optimization*, SIAM. J. Optim. 23(2) (2013), pp. 1126–1153.

[45] P. Richtárik and M. Takáč, *Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function*, Math. Program. Ser. A 144(1–2) (2014), pp. 1–38.

[46] P. Richtárik and M. Takáč, *Parallel coordinate descent methods for big data optimization*, in Mathematical Programming, pp. 1–52, 2015.

[47] P. Richtárik and M. Takáč, *Distributed coordinate descent method for learning with big data*, J. Mach. Learning Res. 17(1) (2016), pp. 2657–2681.

[48] P. Richtárik and M. Takáč, *On optimal probabilities in stochastic coordinate descent methods*, Optim. Lett. 10(6) (2016), pp. 1233–1243.

[49] L. Seidel, *Ueber ein Verfahren, die Gleichungen, auf welche die Methode der kleinsten Quadrate führt, sowie lineäre Gleichungen überhaupt, durch successive Annäherung aufzulösen:* (Aus den Abhandl. der bayer. Akademie II. Cl. XI. Bd. III. Abth. Verlag der k. Akademie, 1874.

[50] S. Shalev-Shwartz and A. Tewari, *Stochastic methods for l1-regularized loss minimization*, J. Mach. Learn. Res. 12 (2011), pp. 1865–1892.

[51] S. Shalev-Shwartz and T. Zhang, *Proximal stochastic dual coordinate ascent*. arXiv:1211.2717, 2012.

[52] S. Shalev-Shwartz and T. Zhang, *Stochastic dual coordinate ascent methods for regularized loss minimization*, J. Mach. Learn. Res. 14(1) (2013), pp. 567–599.

[53] S. Shalev-Shwartz and T. Zhang, *Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization*. in International Conference on Machine Learning, pp. 64–72, 2014.

[54] H.-J.M. Shi, S. Tu, Y. Xu, and W. Yin, *A primer on coordinate descent algorithms*. arXiv preprint arXiv:1610.00040, 2016.

[55] T. Sun, R. Hannah, and W. Yin, *Asynchronous coordinate descent under more realistic assumptions*. in Advances in Neural Information Processing Systems, pp. 6182–6190, 2017.

[56] M. Takáč, A. Bijral, P. Richtárik, and N. Srebro, *Mini-batch primal and dual methods for SVMs*. in 30th International Conference on Machine Learning, 2013.

[57] Q. Tao, K. Kong, D. Chu, and G. Wu, *Stochastic coordinate descent methods for regularized smooth and nonsmooth losses*, in Machine Learning and Knowledge Discovery in Databases, pp. 537–552, 2012.

[58] S.M. Tom, C.R. Fox, C. Trepel, and R.A. Poldrack, *The neural basis of loss aversion in decision-making under risk*, Science 315(5811) (2007), pp. 515–518.

[59] Q. Tran-Dinh, O. Fercoq, and V. Cevher, *A smooth primal-dual optimization framework for nonsmooth composite convex minimization*, SIAM. J. Optim. 28(1) (2018), pp. 96–134.

[60] P. Tseng and S. Yun, *A coordinate gradient descent method for nonsmooth separable minimization*, Math. Program. 117 (2009), pp. 387–423.

[61] P. Tseng and S. Yun, *Block-coordinate gradient descent method for linearly constrained non separable optimization*, J. Optim. Theory. Appl. 140 (2009), pp. 513–535.

[62] J. Warga, *Minimizing certain convex functions*, J. Soc. Ind. Appl. Math. 11(3) (1963), pp. 588–593.

[63] S.J. Wright, *Coordinate descent algorithms*, Math. Program. 151(1) (2015), pp. 3–34.
[64] Y. Xu and W. Yin, *A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion*, SIAM. J. Imaging. Sci. 6(3) (2013), pp. 1758–1789.

[65] Y. Xu and W. Yin, *A globally convergent algorithm for nonconvex optimization based on block coordinate update*, J. Sci. Comput. 72(2) (2017), pp. 700–734.

[66] Y. Xu and S. Zhang, *Accelerated primal–dual proximal block coordinate updating methods for constrained convex optimization*, Comput. Optim. Appl. 70(1) (2018), pp. 91–128.

[67] H.-F. Yu, F.-L. Huang, and C.-J. Lin, *Dual coordinate descent methods for logistic regression and maximum entropy models*, Mach. Learn. 85(1–2) (2011), pp. 41–75.

[68] S. Yun, *On the iteration complexity of cyclic coordinate gradient descent methods*, SIAM. J. Optim. 24(3) (2014), pp. 1567–1580.