Homework 1 - Optimization

Francesco Sartori Pietro Sittoni Irene Caria Gianmarco Lattaruolo

1 Introduction

In this homework we worked on a semi-supervised learning problem of classification. Given n points in the $\mathbb{R}^2$ with a few of them labeled with $+1$ or -1, we want to minimize the function

$$f(y) = \sum_{i=1}^{l}\sum_{j=1}^{u} \overline{w}_{ij}(y^j - \overline{y}^i)^2 + \frac{1}{2}\sum_{i=1}^{u}\sum_{j=1}^{u} w_{ij}(y^i - y^j)^2, \tag{1}$$

where:

- $y \in \mathbb{R}^u$ with u the number of unlabeled examples and our prediction for the label of i-th sample will be $sign(y_i)$.
- l is the number of labeled samples and $\overline{y}^i = \pm 1$ for $i = 1, \dots, l$ depending on the label of i-th sample.
- $\overline{w}_{ij}$ is the weight related to the i-th unlabeled sample and the j-th labeled one.
- w_{ij} is the weight related to the i-th unlabeled sample and the j-th unlabeled one.

The weights are computed according to certain similarity measure as we will see later.

1.1 Toy Model

We tried to solve the problem with two datasets of points. The first was a simple model we called "Toy Model". We randomly generated two clusters in $\mathbb{R}^2$, one centered in $(-2, -2)$ and the other one in $(4, 4)$; each cluster is a set of 5000 points generated from a normal distribution around the centers. We labeled the points with $\{-1, 1\}$ according to the cluster which they belongs to (see Figure 1).

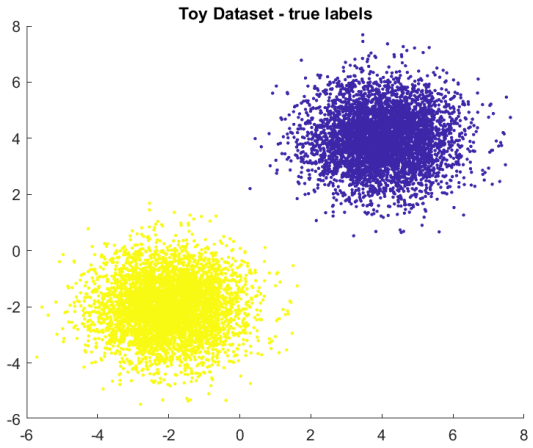

Figure 1: Random points with two classes

Then we selected a sample X_{samp} of 2% of the points we generated and saved their labels in y_{samp} (see Figure 2). Actually we consider these points as the only ones for which we know the labels (a sort of a very small training set). We called $X_{unlabeled} := X \backslash X_{samp}$ and y_{exact} their labels.

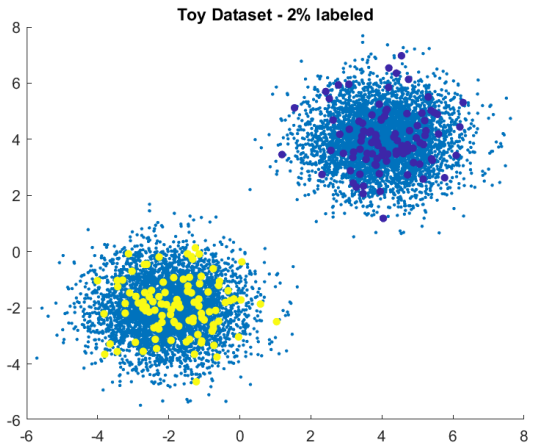

Figure 2: Random 2% labeled points

1.2 Real dataset

We have considered the *Occupancy Detection Dataset* from the *UCI Machine Learning Repository*. This dataset is an experimental one used for binary

classification (room occupancy) from Temperature, Humidity, Light and CO2. Ground-truth occupancy was obtained from time stamped pictures that were taken every minute. We combined the training and test sets to obtain a set of 10803 observations and we considered only the Temperature and Light features. There are 8107 elements of class 0, which corresponds to not occupied status, and 2696 elements of class 1, which corresponds to occupied status. For convenience, we have called -1 the class 0. We then removed the outliers by selecting the points whose Light was under 900 and we normalized the features. Trying the methods on this dataset we saw that all points were classified with the most frequent class (i.e.:-1). We noticed that all the points were very close to each other, consequently the weight were close one to an other even for distant points; so we decided to expand our features space by multiplying both features by 100. We reported our data in Figure 3.

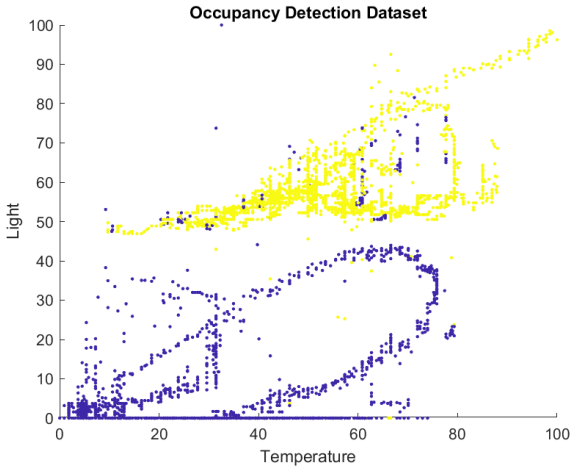

Figure 3: Occupancy Detection Dataset

We then randomly selected a sample of 2% of the points and saved their labels $\{-1, 1\}$ (see Figure 4).

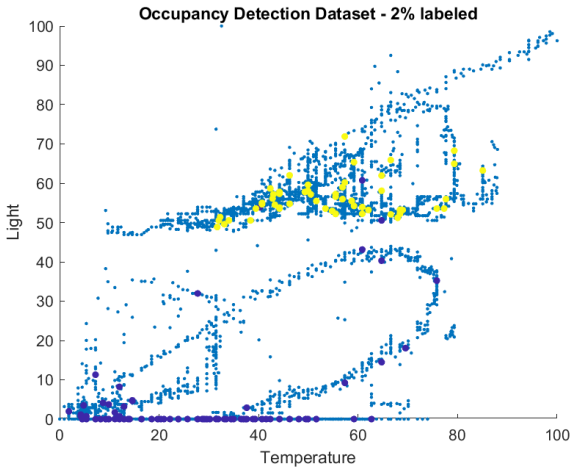

Figure 4: Occupancy Detection Dataset with 2% data labeled

1.3 Similarity measures

The distances between the labeled points X_{samp} and the unlabeled points $X_{unlabeled}$ have been saved in a matrix $\overline{D} \in \mathbb{R}^{l \times u}$, where $\overline{D}_{i,j} := d(X_{samp_i}, X_{unlabeled_j})$ and $d(\cdot, \cdot)$ is the euclidean distance. The distances between the unlabeled points $X_{unlabeled}$ and themselves have been saved in a matrix $D \in \mathbb{R}^{u \times u}$ as well, where $D_{i,j} := d(X_{unlabeled_i}, X_{unlabeled_j})$ and clearly $D = D^t$. We calculated the weights in the following way:[1]

$$\overline{W}_{i,j} := \exp(-\overline{D}_{ij}), \quad i = 1, .., l, \quad j = 1, ..., u$$
$$W_{i,j} := exp(-D_{i,j}), \quad i, j = 1, .., u.$$

We decided to use this similarity function because we assumed that two examples share the same class if their distance is small. Thus we wanted a decreasing function and we tested several ones: $1/d, e^{-\frac{d^2}{2}}$ and e^{-d}, and finally we selected e^{-d}. After this, to have a more sparse matrix, if a weight related to two examples was less than 10^{-2}, we directly put it equal to 0. We gained a computational advantage and this choice did not affect the accuracy score and the decreasing of the gradient's norm with respect to the full matrix of weights.

2 Algorithms

To optimize the function f in (1) we used some classical gradient-based methods:

- the Gradient Descent method,

[1] In our codes we call `D_samp` and `W_samp` the matrices $\overline{D}$ and $\overline{W}$ above.

- the Block Coordinate Gradient Descent method, with cyclic rule for the update, and unitary block sizes i.e. one coordinate for each block,
- the Block Coordinate Gradient Descent method with randomized rule for choosing with uniform probability and updating one block, always of unitary size.

For each of these algorithms we tested three step size rules: the fixed step size, the Armijo rule and the exact line search. In the functions for the gradient methods (`GD.m`, `BCGD_cyclic.m` and `BCGD_random.m`) thus we have three cases and we use in all of them other auxiliaries functions. The main difference we took in consideration, as explained below, is the case in which we have to compute the full gradient (see the function `f_deriv.m`) and the case in which is enough to compute one coordinate derivative (see the function `f_deriv_block.m`). In most of the cases we have done some mathematics to have a simplest formula for the results we need, in particular:

Fixed Step Size Due to the convexity of the function f, which is a quadratic function, and its continuous gradient we set the fixed step size α using the theorem we saw during the lectures $\alpha = 1/L$ where L is the Lipschitz constant of ∇f. Solving the inequality $||\nabla f(y) - \nabla f(y')||_2 \leq L||y - y'||_2$ we estimate that $L \leq 2(C_{max}^2 + \lambda_{max}^2 - 2C_{min}\lambda_{min})^{1/2}$, where $C_{max} = \max_{j=1,\dots,u} \{\sum_i^u \overline{w}_{ij} + \sum_i^u w_{ij}\}$ and λ_{max} is the largest eigenvalue of W. Analogous definition for C_{min} and λ_{min}[2]. Clearly there is not difference computing the fixed step size $\alpha = 1/L$ in the Gradient Descent method and in the other two methods.

Armijo Rule For the Armijo rule we wrote a code to find the biggest value of α such that $f(y + \alpha d) \leq f(y) + \gamma\alpha\nabla f(y)^t d$ where d is the opposite direction given by the gradient of f in y and γ is a parameter. The research for α is made following the rule and putting $\alpha = \delta^m\Delta$ for the full gradient[3]. In the case of Block Coordinate Gradient Descent methods we have a single coordinate, say the m-th, and we solved the inequality $f(y + \alpha d_m \mathbf{e}_m) \leq f(y) + \gamma\alpha d_m^2$ where $\mathbf{e}_i, i = 1, \dots, m$ is the canonical base in $\mathbb{R}^u$ and $d_m = -\partial f/\partial y_m$. Calling $C_m = \sum_i^u \overline{w}_{im} + \sum_i^u w_{im}$ the solution leads to $\alpha \leq (1 - \gamma)/(C_m - w_{mm})$.

Exact Line Search Computing the derivative of $\phi(\alpha) := f(y + \alpha d)$ we know that, since it is a convex function, the value for which $\phi'(\alpha) = 0$ is the unique minimum in the direction d starting from y, therefore is the step size we need. Doing the math we obtained the following formula:

$$\alpha = \frac{\sum_{j=1}^u d_j^2}{2\sum_{i=1}^l \sum_{j=1}^u \overline{w}_{ij} d_j^2 + 2\sum_{i=1}^u \sum_{j=1}^u w_{ij}(d_i - d_j)d_i} \tag{2}$$

[2]Actually in the codes we do not consider the term $-2C_{min}\lambda_{min}$ since its value was negligible.

[3]In our code we test some values for the parameters and finally we set $\delta = 0.5, \Delta = 1$ and $\gamma = 0.25$.

In the case of BCGD algorithms we can highlight a great simplification of the previous equation which becomes $\alpha = \frac{1}{2(C_m - w_{mm})}$ (with the same notation as before). Notice that it is the same result of Armijo rule with $\gamma = 1/2$ (actually the largest admissible value following the rule).

The language we have used to code is **MATLAB**; to gain some computational advantages we spent a lot of time in try to avoiding `for` loops, using instead matrix based formulas for computing the summations. For example, with these conventions to have formal expressions:

- if $v \in \mathbb{R}^n$ then $v.^2 \in \mathbb{R}^n$ is the vector such that $(v.^2)_i = v_i^2, i = 1, \dots, n$,
- $\mathbb{1}_n := \underbrace{(1 \cdots 1)^t}_{n\ times}$ is the column vector with every entry equal to 1,
- $v. * w = z$ is the vector such that $z_i = v_i \cdot w_i, i = 1, \dots, n$

the equation (2) becomes

$$\alpha = \frac{d^t d}{\mathbb{1}_l^t \overline{W}(d.^2) + \mathbb{1}_u W(d.^2) - d^t W d} \tag{3}$$

In the same way we wrote the code for the objective function (1) as follows: $f(y) = \mathbb{1}_l^t \overline{W}(y.^2) - 2\overline{y}^t \overline{W} y + (\overline{y}.^2)^t \overline{W} \mathbb{1}_u + \mathbb{1}_u^t W(y.^2) - y^t W y$. Furthermore, with this notation we can write in a closed formula also the gradient vector of f:

$$\nabla f(y) = 2(\overline{W}^t \mathbb{1}_l + W^t \mathbb{1}_u). * y - 2(\overline{W}^t \overline{y} + W^t y) \tag{4}$$

For the proofs of all the statement above see the appendix at the end of the report.

3 Results

To compute the class of the unlabeled data, as said before, we considered the sign of the entries of $y^{(k)}$, the vector obtained at iteration k-th of our algorithms. Thus if $y^{(k)}$ converge to the exact solution then it converges also in sign but vice versa in general is not true, so we can expect a faster convergence with the sign. In order to had a control on the execution time, we set an upper bound on the iteration. For the classic GD and the cyclic BCGD methods we set 1000 iterations as a threshold, meanwhile for the randomized we chose 10^5. This is a point that must be stressed: for the BCGD randomized method we have a threshold for the number of iteration that is 100 times with respect to the other two methods. This choice was lead by a simple consideration: in the Gradient Descent and in BCGD with cyclic rule, at each iteration there is an update of all the variables (9800 for the toy model), instead in the randomized BCGD only one variable is changed at each step. For this reason one iteration (i.e: one oracle call) has a computational cost considerably lower than the others two

methods, and globally the randomized BCGD method takes less time to execute, but it needs as well more iteration to reach the convergence (or our stopping condition). To make more comparable the results of randomized BCGD with the others, and also to avoiding some computation, for this method we collected the accuracy, the gradient norms and the CPU-time ones every 100 iterations; looking at the first two plots we should take in account this fact.

Another stopping criterion we used is the norm of the gradient, that since the aim is to find a minimum of a convex function, it embeds the error of our research. For the classic GD and for cyclic BCGD we set a lower bound of 10^{-4} for the norm of the gradient. Notice that to compute the full gradient in the cyclic BCGD, instead of evaluating it from the function in the new point y at the end of each iteration, we stored the entries of the gradient already computed for updating each block (which actually is one coordinate).

For the randomized BCGD we consider others stopping conditions. As said before we calculated the full gradient only once every 100 iteration, and we stop the algorithm if $(u|\nabla_i f(y_i)|) \leq 10^{-4}$ and $||\nabla f(y)||_2 \leq 10^{-2}$. We combine this two conditions because using only the condition on the single block was too weak. Actually it happened that only one coordinate became significantly small, stopping the procedure far from an appropriate accuracy.

The second inequality as a bigger tolerance because we noticed that gradient's norm decrease more slowly than the other two methods.

3.1 Toy Model

For the Toy Model we have these results. The first two methods reach an accuracy of 1 in a few iterations. Also the randomized BCGD converges in accuracy, but with much more iterations, although is the fastest in terms of time. For the gradient norms the situation is different and can also be a little bit miss leading. The randomized BCGD method does not converge at all in term of the gradient norm but converges in terms of accuracy. This can be explained because the method takes a few times (and also iterations) to converge in sign but much more to the exact values.

More specifically in the case of α fixed, the GD algorithm reaches a norm of order 10^{-2}, the BCGD cyclic reaches a norm of 10^{-4} in 1000 iteration and the BCGD randomized reaches a norm of 10^4, but in 10^5 iteration (as shown in Figure 5).

In the case of α calculated with the Armijo rule (see Figure 6) the GD method stopped after 500 iterations and cyclic BCGD method after 150 iterations. We obtained similar results also with the exact line search using these two methods: GD stopped in 600 iterations and cyclic BCGD in 400 (see Figure 7). In all these cases, in fact, we noticed that the algorithms do not reach the maximum number of iterations and stopped thank to the gradient norm criterion.

The worst algorithm with respect to the iterations and to gradient norms appears to be the randomized BCGD, but it remains faster in terms of CPU-time.

In the following plots for all the accuracy vs time plots we use a logarithmic scale for time, and in all gradient's norm plots we use a logarithmic scale for the y-axes.

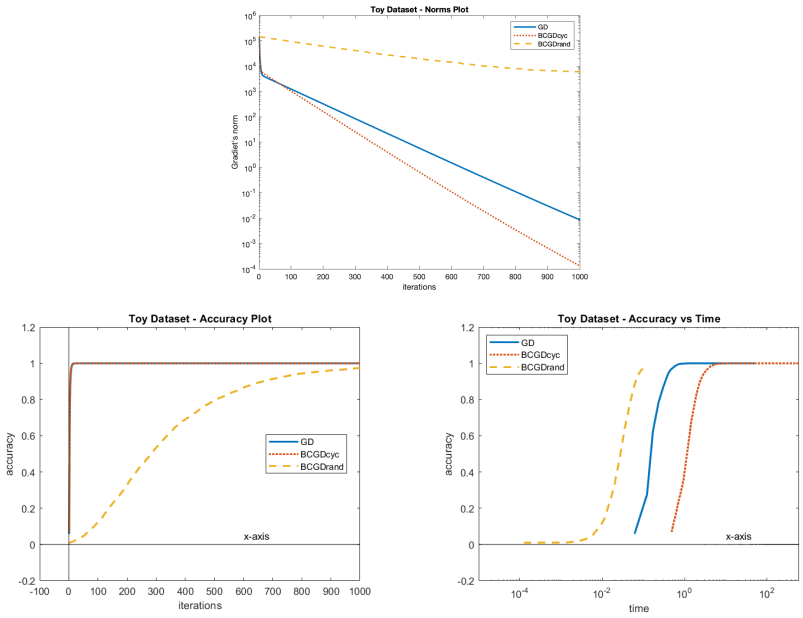

Figure 5: Results for the three methods with *alpha* fixed

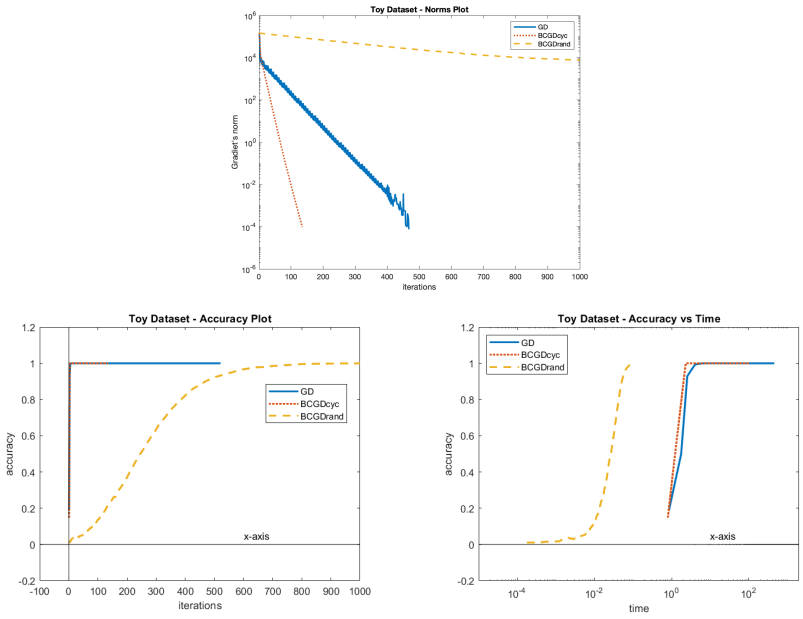

Figure 6: Results for the three methods with *alpha* calculated with Armijo rule

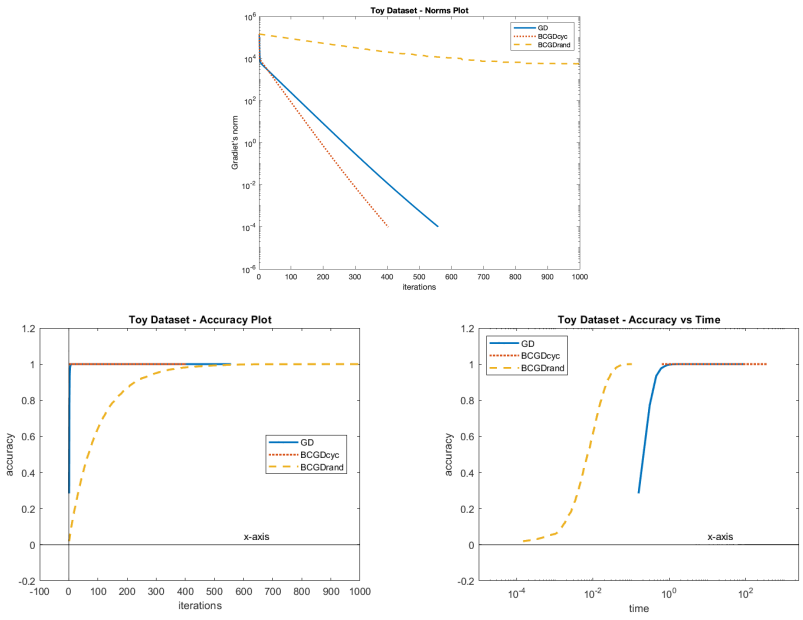

Figure 7: Results for the three methods with *alpha* calculated with exact line search rule

3.2 Real dataset

Similarly to what we did for the Toy Model, also for the real dataset we have tested the three methods using the three different step sizes listed in Section 2.

Fixed Step Size As we can see from Figure 8, both GD gradient's norm and the cyclic BCGD gradient's norm decrease with the increase of the iterations even if they do not meet the stopping criterion in 1000 iterations, while the randomized BCGD gradient's norm fluctuates around the same value and it does not decrease.

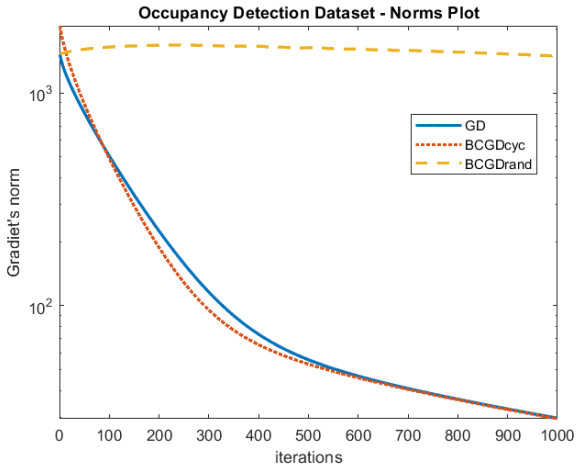

Figure 8: Norms of the gradient for the three methods with *alpha* fixed

Even looking to the accuracy plot (Figure 9), we can see that GD and cyclic BCGD achieve a very good accuracy in a few iterations, instead the random BCGD takes a lot more iterations to reach the same accuracy; upon reaching the maximum iterations we can see that all models achieved an accuracy between $96,5\%$ and 97%.

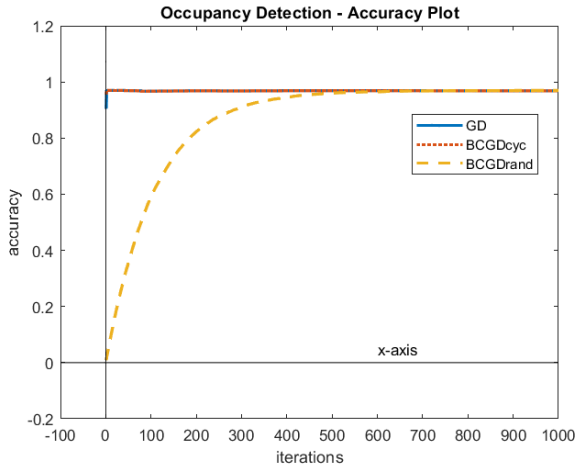

Figure 9: Accuracy of the three methods with *alpha* fixed

Looking at the accuracy versus time plot we can see that even if the random BCGD model take significantly more iterations it is still faster than the other two methods.

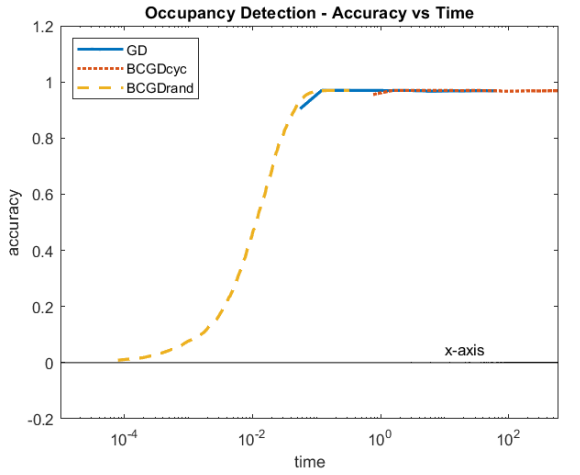

Figure 10: Accuracy over time of the three methods with *alpha* fixed

Armijo Rule We then tested the Armijo rule; from the plot of the gradient's norms we can see that the random BCGD behaves very similarly to the *alpha* fixed method and the cyclic BCGD model converged faster reaching the stopping criterion before the maximum number of iterations, meanwhile the GD gradient's norm started oscillating without converging.

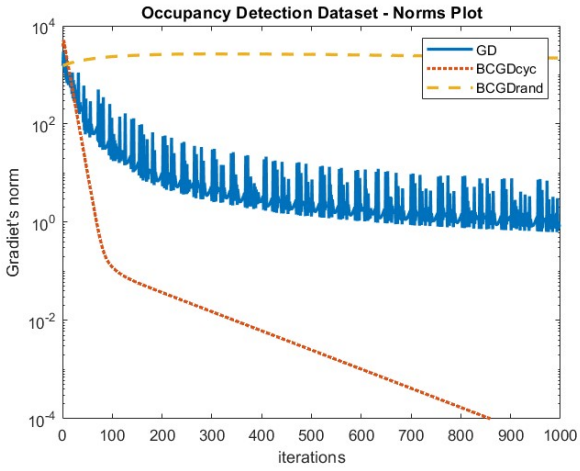

Figure 11: Norms of the gradient for the three methods with *alpha* calculated with Armijo rule

Watching at the accuracy plot (Figure 12) we can see that all the three methods behave very similarly to the *alpha* fixed methods.

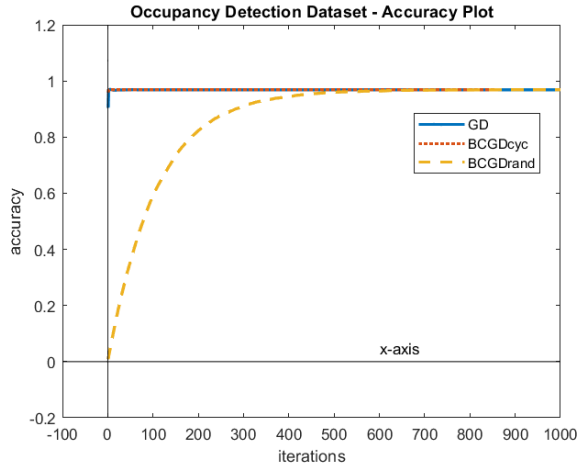

Figure 12: Accuracy of the three methods with *alpha* calculated with Armijo rule

Even if with the accuracy plot we cannot spot many differences with the previous methods, the following plot (Figure 13) shows that the calculation of the step size with the Armijo rule make the overall process slower.

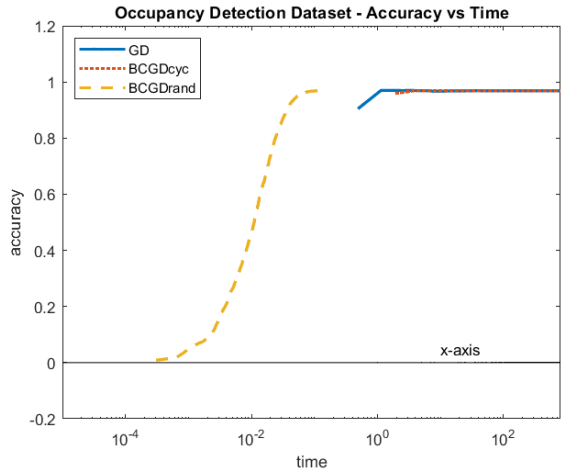

Figure 13: Accuracy over time of the three methods with *alpha* calculated with Armijo rule

Exact Line Search We finally tested the Exact Line Search; as we can see from Figure 14, no algorithm converges within 1000 iterations but, as before, the cyclic BCGD seems to be the fastest while the random BCGD does not converge in norm.

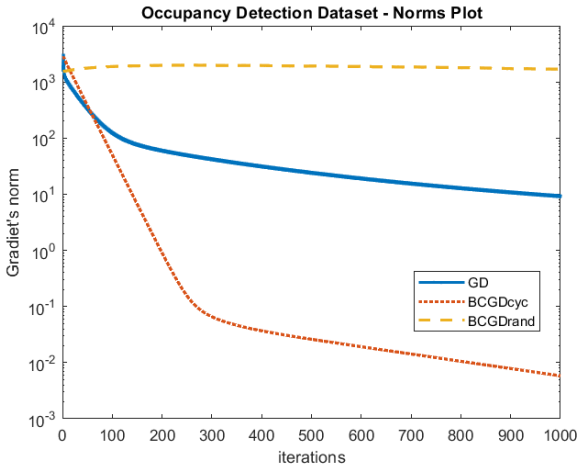

Figure 14: Norms of the gradient for the three methods with *alpha* calculated with exact line search rule

Looking at Figure 15 we can see the exact same behaviour for the accuracy as the previous choice of the step size rule.

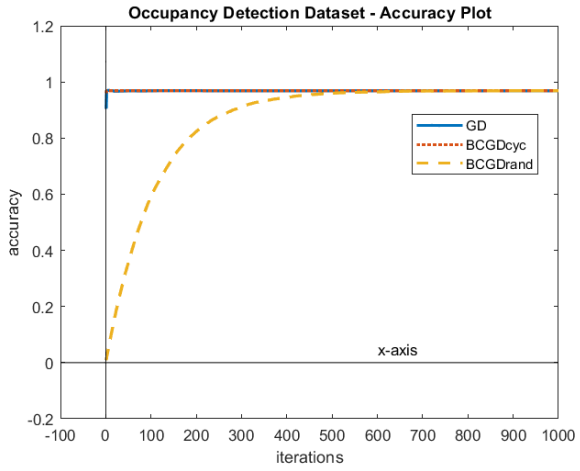

Figure 15: Accuracy of the three methods with *alpha* calculated with exact line search rule

Again the accuracy versus time plot (16) shows that the random BCGD is the fastest, we can also notice that this method is slightly slower than fixed step size (on the cyclic and random BCGD the difference can be very small because exact line search for those models almost does no computations).

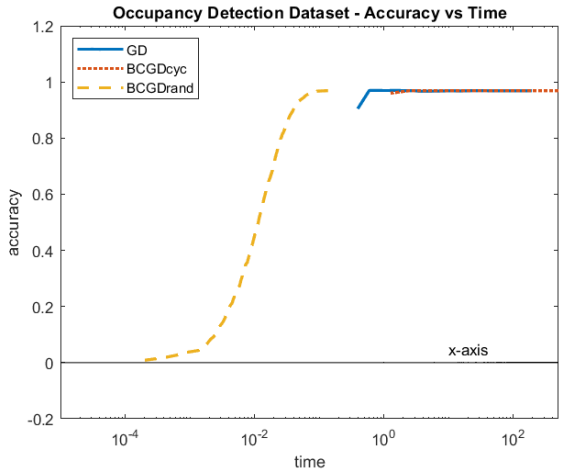

Figure 16: Accuracy over time of the three methods with *alpha* calculated with exact line search rule

Classification Now we can look how the different methods classify our data; the choice of the step size does not change the result so we show one plot

for each method.

We can see from Figure 17 that all classifications are very close one to the other; the accuracy is pretty high and the points that are miss classified are the ones that are close to examples of the other class and far from examples of their own class.

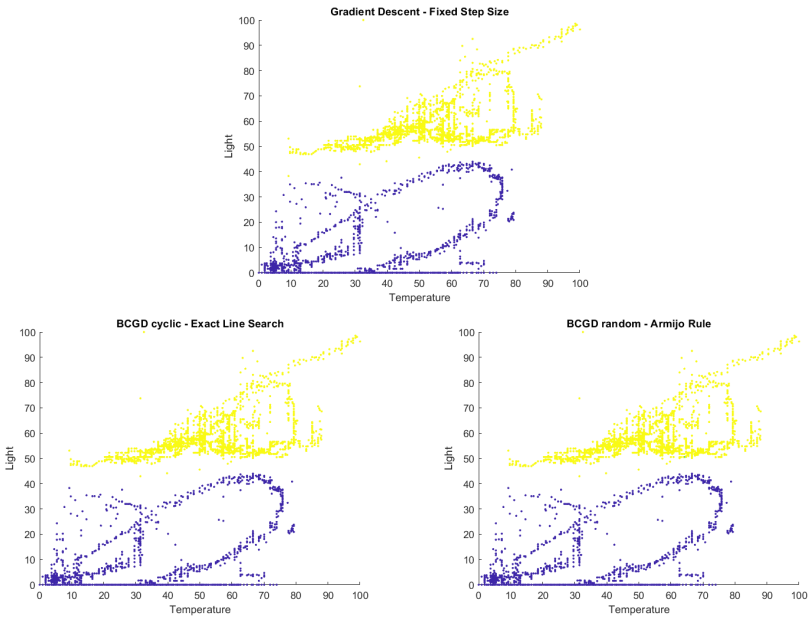

Figure 17: Classification for the three methods

Appendix

We now show the most important calculations we have done.

For the estimation of the Lipschitz constant L we have done the following inequalities. First we considered just one entry of the norm of the gradients difference

$$\begin{aligned}\frac{1}{4}\left(\nabla_j f(y) - \nabla_j f(y')\right)^2 &= \left(\sum_{i=1}^{l} \overline{w}_{ij}((y_j - \overline{y}_i) - (y'_j - \overline{y}_i)) + \sum_{i=1}^{u} w_{ij}((y_j - y_i) - (y'_j - y'_i))\right)^2 \\ &= \left(\sum_{i=1}^{l} \overline{w}_{ij}\left(y_j - y'_j\right) + \sum_{i=1}^{u} w_{ij}\left(y_j - y'_j\right) - \sum_{i=1}^{u} w_{ij}\left(y_i - y'_i\right)\right)^2 \\ &= \left(C_j\left((y_j - y'_j\right) - \mathbf{e}_j^t W\left(y_j - y'_j\right)\right)^2\end{aligned}$$

Where we call as before $C_j = \sum_{i=1}^{l} \overline{w}_{ij} + \sum_i^u w_{ij}$. Considering now the full norm we obtain

$$\begin{aligned}\frac{1}{4}\|\nabla f(y) - \nabla f(y')\|^2 &= \sum_{j=1}^{u}\left(C_j\left((y_j - y'_j\right) - \mathbf{e}_j^T W\left(y - y'\right)\right)^2 \\ &= \sum_{j=1}^{u} C_j^2\left(y_j - y'_j\right)^2 + \sum_{j=1}^{u}\left(\mathbf{e}_j^T W\left(y_j - y'_j\right)\right)^2 + \\ &- 2\sum_{j=1}^{u} C_j\left(y_j - y'_j\right)\mathbf{e}_j^t W\left(y_j - y'_j\right) \\ &\leq \left(C_{max}^2 + \lambda_{max}^2\right)\|y_j - y'_j\|^2 - 2C_{min}\left(y - y'\right)^t W\left(y - y'\right) \\ &\overset{(1)}{\leq} \left(C_{max}^2 + \lambda_{max}^2 - 2C_{min}\lambda_{min}\right)\|y_j - y'_j\|^2\end{aligned}$$

Where $C_{max} = \underset{j=1,\ldots,u}{max}\left\{\sum_i^u \overline{w}_{ij} + \sum_i^u w_{ij}\right\}$ and $C_{min} = \underset{j=1,\ldots,u}{min}\left\{\sum_i^u \overline{w}_{ij} + \sum_i^u w_{ij}\right\}$. The inequality (1) holds because: since W is symmetric, it is diagonalizable and so we can rewrite it in the following way

$$W = Q^t \Lambda Q$$

where Q is orthogonal and Λ is the diagonal matrix of the eigenvalues λ_i for $i = 1, .., u$. Then, by calling $\tilde{y} = Q\left(y - y'\right)$ we have

$$\begin{aligned}\left(y - y'\right)^t W\left(y - y'\right) &= \tilde{y}^t \Lambda \tilde{y} \\ &= \sum_{j=1}^{u} \lambda_j \tilde{y}_j^2 \\ &\geq \lambda_{min}\|\tilde{y}\|_2^2 \\ &= \lambda_{min}\|y - y'\|_2^2\end{aligned}$$

For the Armijo rule, consider the block coordinate case of size 1. As said before, we are looking for the largest α such that $f(y+\alpha d) \leq f(y)+\gamma\alpha\nabla f(y)^t d$ where $d = (d_1, \ldots, d_u) = (0, \ldots, -\partial f/\partial y_m, \ldots 0)$. We have

$$f(y+\alpha d) = \sum_{i=1}^{l}\sum_{j=1}^{u} \overline{w}_{ij}((y+ad)_j - \overline{y}_i)^2 + \frac{1}{2}\sum_{i,j=1}^{u} w_{ij}((y_i - y_j) + \alpha(d_i - d_j))^2$$

$$= \sum_{i=1}^{l}\sum_{j=1}^{u} \overline{w}_{ij}(y_j - \overline{y}_i)^2 + \frac{1}{2}\sum_{i,j=1}^{u} w_{ij}(y_i - y_j) + \sum_{i=1}^{l}\sum_{j=1}^{u} \overline{w}_{ij}\left[\alpha^2 d_j^2 + (2\alpha y_j - 2\alpha\overline{y}_i)d_j\right] +$$

$$+ \frac{1}{2}\sum_{i,j=1}^{u} w_{ij}(2\alpha(d_i - d_j)(y_i - y_j) + \alpha^2(d_i - d_j)^2)$$

$$= f(y) + \sum_{i=1}^{l}\sum_{j=1}^{u} \overline{w}_{ij}\alpha^2 d_j^2 + \sum_{i=1}^{l}\sum_{j=1}^{u} 2\overline{w}_{ij}\alpha d_j(y_j - \overline{y}_i) +$$

$$\frac{1}{2}\sum_{i,j=1}^{u} 2w_{ij}\alpha d_i(y_i - y_j) - \frac{1}{2}\sum_{i,j=1}^{u} 2w_{ij}\alpha d_j(y_i - y_j) + \frac{1}{2}\sum_{i,j=1}^{u} w_{ij}\alpha^2(d_i \cdot d_j)^2$$

$$= f(y) + 2d_m\left[\sum_{i=1}^{l} \overline{w}_{im}\alpha(y_m - \overline{y}_i) + \frac{1}{2}\sum_{j=1}^{u} w_{mj}\alpha(y_m - y_j) - \frac{1}{2}\sum_{i=1}^{u} w_{im}\alpha(y_i - y_m)\right]$$

$$+ \sum_{i=1}^{l} \overline{w}_{im}\alpha^2 d_m^2 + \frac{1}{2}\sum_{i,j=1}^{j} w_{ij}\alpha^2(d_i^2 + d_j^2 - 2d_i d_j)$$

$$= f(y) + \alpha d_m \cdot (-d_m) + \sum_{i,j=1}^{u} w_{ij}\alpha^2 d_i^2 - \alpha^2\sum_{i,j=1}^{u} w_{ij}d_i d_j + \sum_{i=1}^{l} \overline{w}_{im}\alpha^2 d_m^2$$

$$= f(y) - \alpha d_m^2 + \alpha^2 d_m^2 \sum_{j=i}^{u} w_{mj} + -\alpha^2 w_{mm}d_m^2 + \alpha d_m^2 \sum \overline{w}_{im}$$

$$f(y+\alpha d) = f(y) - \alpha d_m^2 + \alpha^2 d_m^2 C_m - \alpha^2 d_m^2 w_{mm}$$

So the condition above, reminding that $\nabla f(y) \cdot d = -d_m^2$, becomes

$$\begin{aligned} &- \alpha d_m^2 + \alpha^2 d_m^2 C_m - \alpha^2 d_m^2 w_{mm} \leq -\gamma\alpha d_m^2 \\ &- 1 + \alpha C_m - \alpha w_{mm} \leq -\gamma \\ &\alpha \leq \frac{1-\gamma}{C_m - w_{mm}} \end{aligned}$$

We now solve the equation for finding the exact line search step size. Consider the direction $d = (d_1, \ldots, d_u) = -\nabla f(y)$. Calling $\Phi(\alpha) := f(y + \alpha d)$ we have

$$
\begin{aligned}
\Phi(\alpha) &= \sum_{i=1}^{l}\sum_{j=1}^{u} \overline{w}_{ij}(y_j + \alpha d_j - \overline{y}_i)^2 + \\
&+ \frac{1}{2}\sum_{i=1}^{u}\sum_{j=1}^{u} w_{ij}((y_i - y_j) + \alpha(d_i - d_j))^2 \\
\Phi'(\alpha) &= 2\sum_{i=1}^{l}\sum_{j=1}^{u} \overline{w}_{ij}(y_j + \alpha d_j - \overline{y}_i)d_j + \\
&+ \sum_{i=1}^{u}\sum_{j=1}^{u} w_{ij}((y_i - y_j) + \alpha(d_i - d_j))(d_i - d_j) \\
&\overset{(2)}{=} 2\sum_{i=1}^{l}\sum_{j=1}^{u} \overline{w}_{ij}(y_j + \alpha d_j - \overline{y}_i)d_j + \\
&+ 2\sum_{i=1}^{u}\sum_{j=1}^{u} w_{ij}((y_i - y_j) + \alpha(d_i - d_j))d_i \\
&= \alpha\left[2\sum_{i=1}^{l}\sum_{j=1}^{u} \overline{w}_{ij}d_j^2 + 2\sum_{i=1}^{u}\sum_{j=1}^{u} w_{ij}(d_i - d_j)d_i\right] + \\
&+ 2\sum_{i=1}^{l}\sum_{j=1}^{u} \overline{w}_{ij}(y_j - \overline{y}_i)d_j + 2\sum_{i=1}^{u}\sum_{j=1}^{u} w_{ij}(y_i - y_j)d_i \\
&= \alpha\left[2\sum_{i=1}^{l}\sum_{j=1}^{u} \overline{w}_{ij}d_j^2 + 2\sum_{i=1}^{u}\sum_{j=1}^{u} w_{ij}(d_i - d_j)d_i\right] + \\
&+ \sum_{j=1}^{u}\left[2\sum_{i=1}^{l} \overline{w}_{ij}(y_j - \overline{y}_i) + 2\sum_{i=1}^{u} w_{ij}(y_j - y_i)\right]d_j \\
&= \alpha\left[2\sum_{i=1}^{l}\sum_{j=1}^{u} \overline{w}_{ij}d_j^2 + 2\sum_{i=1}^{u}\sum_{j=1}^{u} w_{ij}(d_i - d_j)d_i\right] + \\
&+ \sum_{j=1}^{u}(-d_j)d_j
\end{aligned}
$$

Where in (2) we used $W = W^t$ and the fact that $\sum_{i=1}^{u}\sum_{j=1}^{u} Z_{ij}(d_i - d_j) = 2\sum_{i=1}^{u}\sum_{j=1}^{u} Z_{ij}d_i$ because $Z_{ij} = -Z_{ji}$, with $Z_{ij} := w_{ij}((y_i - y_j) + \alpha(d_i - d_j))$. And in conclusion

$$
\Phi'(\alpha) = 0 \Longrightarrow \alpha = \frac{\sum_{j=1}^{u} d_j^2}{2\sum_{i=1}^{l}\sum_{j=1}^{u} \overline{w}_{ij}d_j^2 + 2\sum_{i=1}^{u}\sum_{j=1}^{u} w_{ij}(d_i - d_j)d_i}
$$

Which is the result we report in (2). In the case of BCGD algorithms we

have that, since the block sizes is 1, $d = (0, \cdots, d_m, \cdots, 0)$ and therefore the last result becomes

$$\alpha = d_m^{-2} \left(2 \sum_{i=1}^{l} \overline{w}_{im} + 2 \sum_{i=1}^{u} w_{im} d_m^2 - 2 w_{mm} d_m^2 \right) = \frac{1}{2(C_m - w_{mm})}.$$

Interesting to notice is that, since the function f is convex, we can replace d with $r \nabla f(y)$ for $\forall r \in \mathbb{R}$ and the point $y + \alpha d$ we obtain remain the same.