Optimization for Learning - FRTN50

Assignment 3

Task 1 Derive the convex conjugate h^*.

The conjugate h^* :

We have: $h(z) = \sum_{i=1}^{N} h_i(z_i)$ with: $h_i(z_i) = \frac{1}{N}\max(0, 1 - z_i)$.
i.e. h is separable.
Hence:

$$h^*(s) = \sum_{i=1}^{N} h_i^*(s_i)\,, \quad s = (s_1, ..., , s_N)^T \in \mathbb{R}^n$$

Now take $i \in \{1, ..., n\}$ and $t \in \mathbb{R}$, we have:
For all $s_i \in \partial h_i(x_i)$, the conjugate satisfies $h_i^*(s_i) = x_i s_i - h_i(x_i)$ (Due to the Fenchel-Young).
The subdifferential is:

$$\partial h_i(x_i) = \begin{cases} -\frac{1}{N} & \text{if } x_i < 1 \\ [-\frac{1}{N}, 0] & \text{if } x_i = 1 \\ 0 & \text{if } x_i > 1 \end{cases}$$

- For $x_i < 1$, we have $s_i = -\frac{1}{N}$ and $h_i^*(-\frac{1}{N}) = -x_i\frac{1}{N} - \frac{1}{N}(1 - x_i) = -\frac{1}{N}$.
- For $x_i > 1$, we have $s_i = 0$ and $h_i^*(0) = 0 - h_i(0) = 0$.
- For $x_i = 1$, then $s_i \in [-\frac{1}{N}, 0]$ and $h_i^*(s_i) = s_i - h_i(1) = s_i$.

The other s_i are not subgradients to h_i at any x_i. We verify that $h_i^*(s_i) = \infty$.

For $s_i < -\frac{1}{N}$, let $x_i = t_- < -1$ with $t_- \to -\infty$:

$$h_i^*(s_i) \geq s_i t_- - h_i(t_-) = (s_i + \frac{1}{N})t_- - \frac{1}{N} \to \infty.$$

For $s_i > 0$, let $x_i = t_+ > 1$ with $t_+ \to +\infty$:

$$h_i^*(s_i) \geq s_i t_+ - h_i(t_+) = s_i t_+ \to \infty.$$

Therefore,

$$h_i^*(s_i) = \begin{cases} s_i & \text{if } s_i \in \left[-\frac{1}{N}, 0\right] \\ \infty & \text{else} \end{cases}$$

Hence:

$$h^*(s) = \sum_{i=1}^{N} \left(s_i + \iota_{[-\frac{1}{N}, 0]}(s_i) \right) , \quad s = (s_1, ..., , s_N)^T \in \mathbb{R}^n$$

Given a solution to the dual SVM problem, ν^*, find the primal SVM solution ω^* :

We know that the Primal problem is:

$$\min_{\omega} h(L\omega) + g(\omega)$$

With: $L = YX^T$ and $g(\omega) = \frac{\lambda}{2}||\omega||_2^2$
Then the Fenchel-dual problem is as follows:

$$\boxed{\min_{\nu} f(\nu) + h^*(\nu)}$$

With: $f(\nu) = g^*(-L^T\nu) = \frac{1}{2}\nu^T Q \nu$

Since the functions h and g are closed convex and the primal and the dual constraint qualifications hold, we can recover a primal solution from the primal-dual optimality condition, that satisfies;

$$\omega^* \in \partial g^*(-L^T\nu^*)$$

$$L\omega^* \in \partial h^*(\nu^*)$$

Note that g is closed convex and $relint(Dom(f \circ -L^T)) \neq \emptyset$, then:

$$\partial f(\nu^*) = -L\partial g^*(-L^T\nu^*)$$

Since f is differentiable and the primal problem is strongly convex, then the solution ω^* is unique:

$$\boxed{\omega^* = \nabla g^*(-L^T\nu^*) = -\frac{1}{\lambda}L^T\nu^*}$$

Task 2 For the dual SVM problem above, compare PG, APG and CG:

We know that f is twice differentiable with:

$$\nabla^2 f(x) = Q \qquad \forall x \in \mathbb{R}^n$$

Since $Q \succeq 0$, then:
$\forall i \in \{1, .., n\} \quad \lambda_i \geq 0$ where λ_i are the eigenvalues of the matrix Q.
Hence:

$$\boxed{\forall x \in \mathbb{R}^n \quad 0 \leq \nabla^2 f(x) \leq \lambda_{\max}(Q) I_n} \quad \text{with} \quad \lambda_{\max}(Q) = \max(\lambda_1, ..., \lambda_n)$$

Therefore, f is $\lambda_{\max}(Q)$-smooth convex function. i.e. $L = \lambda_{\max}(Q)$

- We choose the step-size for woth PG and APG as follows:

$$\gamma = \frac{1}{L} = \frac{1}{\lambda_{\max}(Q)} \quad \text{with } \lambda_{\max}(Q) \neq 0 \text{ since } Q \neq 0$$

- We have:

```
julia> minimum(eigvals(Q))
    0.07936489482711627
```

  Then:

  $\forall i \in \{1, .., n\} \quad \lambda_i \geq \lambda_{\min}(Q) > 0 \quad$ where λ_i are the eigenvalues of the matrix Q.

  i.e. $Q \succ 0$

- Using the proximal gradient method we find ν^* by solving the problem to extra high precision ($||\nu_{n+1} - \nu_n||$ around 10^{-15}) before generating the plots.
  The function **solution()** returns the solution ν^* to use for algorithms.

The proximal gradient method:
The algorithm is:

$$\nu_{k+1} = \underset{\gamma h^*}{\text{prox}}\left(\nu_k - \gamma \nabla f(\nu_k)\right)$$

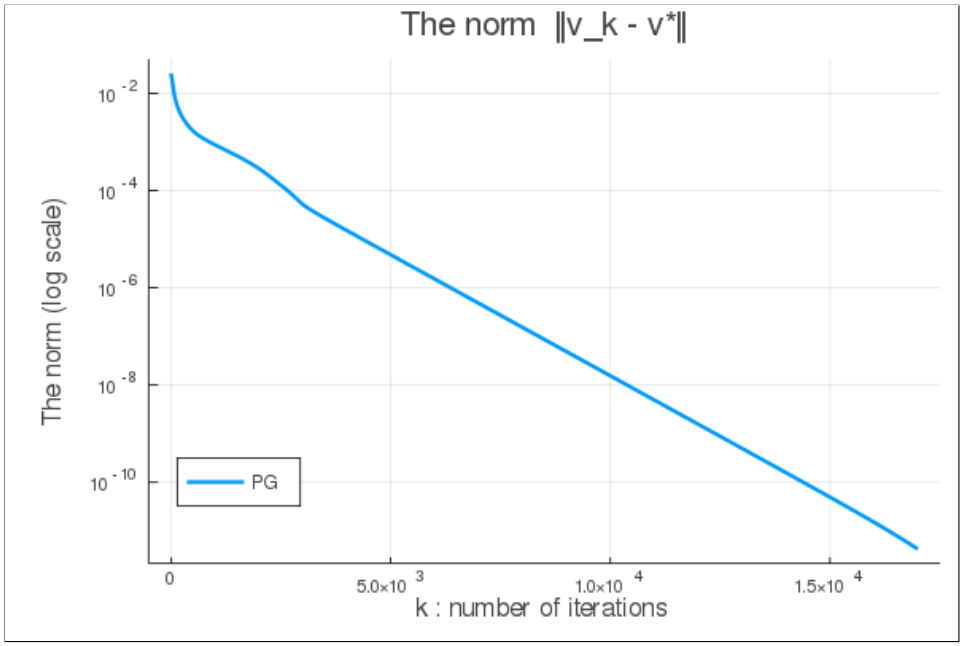


Figure 1: **The distance to the solution $||\nu_k - \nu^*||$ using the Proximal Gradient Method**

Remark:
As we can see, the convergence rate is linear and this is due to the fact that all assumptions of strong convexity setting are valid.

The accelerated proximal gradient method:
Compare both APG with (3) and (4), we compare APG using the common choice of β_k and APG using the β_k of (4) since f is **strongly convex**:

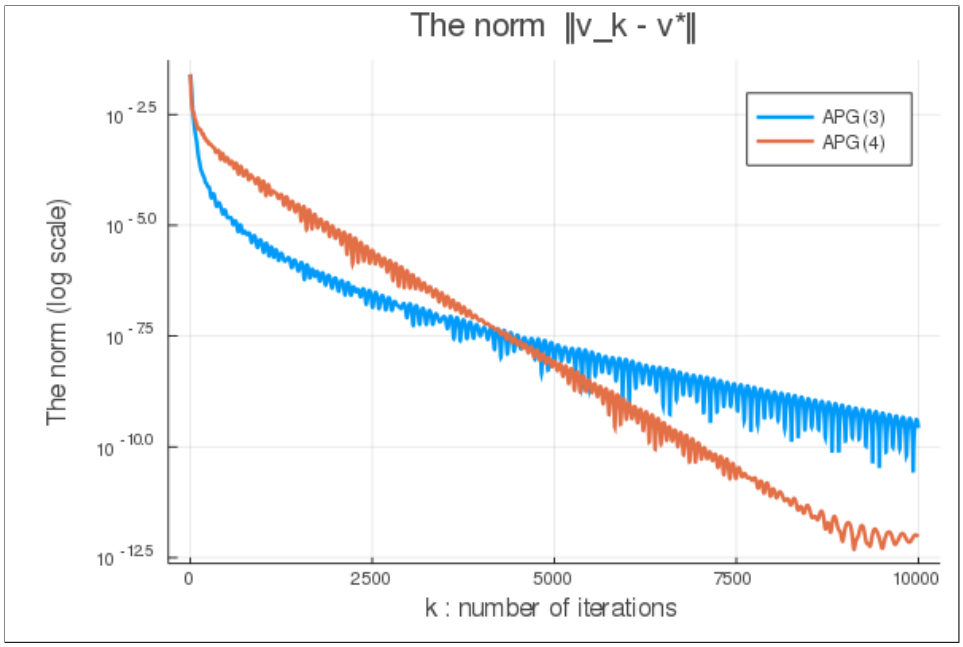


Figure 2: **The distance $||\nu_k - \nu^*||$ using the Accelerated Proximal Gradient Method**

For APG, did the choice of beta sequence have a large or small effect on the result

The choice of beta sequence has a large effect on the result:

- The convergence rate in the first common choice (3) tends to be **sublinear**, since we use only the convexity of the problem.
- The convergence rate in the second choice (4) tends to be **linear**, since we exploit the strong convexity of the problem.

Moreover, APG in (4) is faster than APG in (3) then the fewer iterations needed to reach the solution.

The coordinate gradient method:

We cmpare both the coordinate-wise step-size of $\gamma_i = \frac{1}{Q_{ii}}$ and the uniform choice $\gamma_i = \frac{1}{L}$.

We run the algorithm and we scale the iteration axis with $\frac{1}{N}$ to normalize the computational cost: We let the number of iterations be N times as many and we plot the cost for every N iterations.

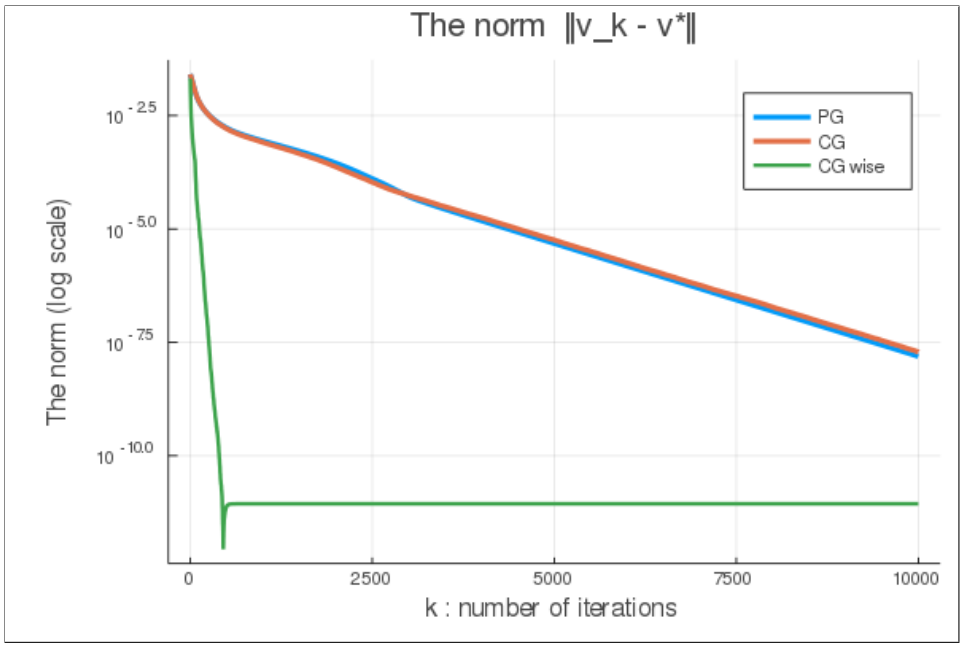


Figure 3: **The distance to the solution $||\nu_k - \nu^*||$ using the coordinate Gradient Method**

For CG, was it beneficial to usecoordinate wise step-sizes?

We see that CD and PG converge at approximately the same speed for the same amount of computations. However, by selecting a wise-coordinate step-size for each coordinate according to the individual smoothness constants as $\gamma_i = \frac{1}{Q_{ii}}$, we get considerably faster convergence.

Which method required the least amount of computational effort?

The gradient of f is $\nabla f(\nu) = Q\nu$ with h^* is seperable.

- One iteration of proximal gradient descent method requires vector operations ($\mathcal{O}(N)$), one matrix multiplication ($\mathcal{O}(N^2)$) as well as the cost of proximal operator which is negligible.
  PG therefore has complexity ($\mathcal{O}(N^2)$) per iteration.
- One iteration of accelerated proximal gradient method requires vector operations ($\mathcal{O}(N)$), one matrix multiplication ($\mathcal{O}(N^2)$) as well as the cost of proximal operator which is negligible.
  APG therefore has complexity ($\mathcal{O}(N^2)$) per iteration.
- One iteration of coordinate gradient method requires multiplication of one row in Q with ν, the prox on h_i^* is negligible.
  CG therefore has complexity ($\mathcal{O}(N)$) per iteration.

Hence, **CG** is the only method which requires the least amount of computational effort per iteration.

Which method required the least amount of iterations?

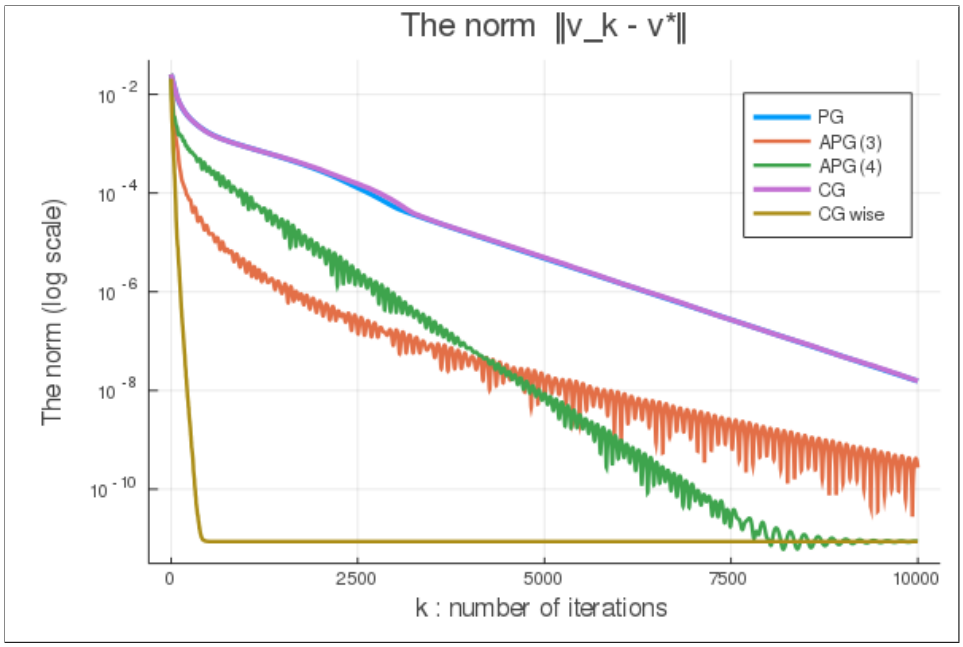
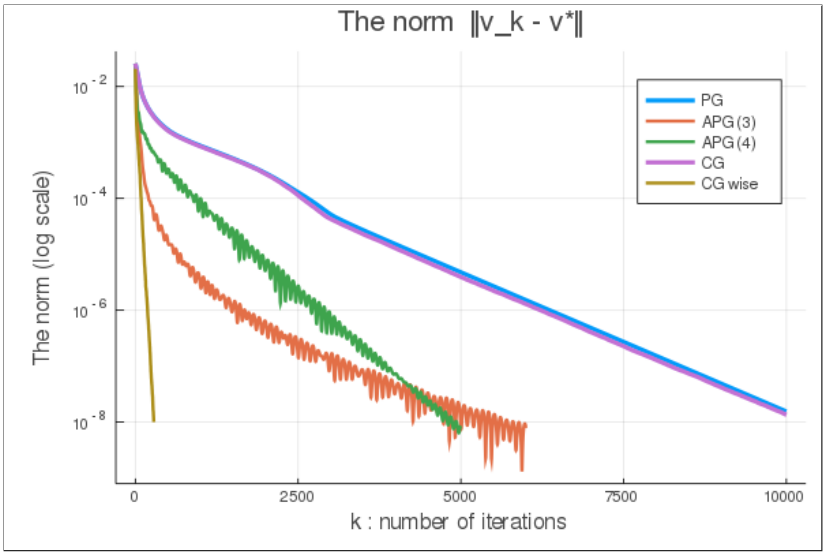


Figure 4: **The distance $||\nu_k - \nu^*||$ using PG, APG and CG**

- We will compare iterations using the same amount of computations, in that case the wise coordinate gradient method is the fastest and it requires the least amount of iterations (scaled iterations i.e. same amount of computations as the other methods).
- If we would like to talk about the real meaning of iteration, then the Accelerated proximal gradient method is the one which requires the least amount of iterations.

As you can see this is not a fair way to compare real time performance.
Which was fastest in real time?
Can you comment on real time performance and number of iterations needed?
How fair is it to compare real time performance? Can it be easily affected?

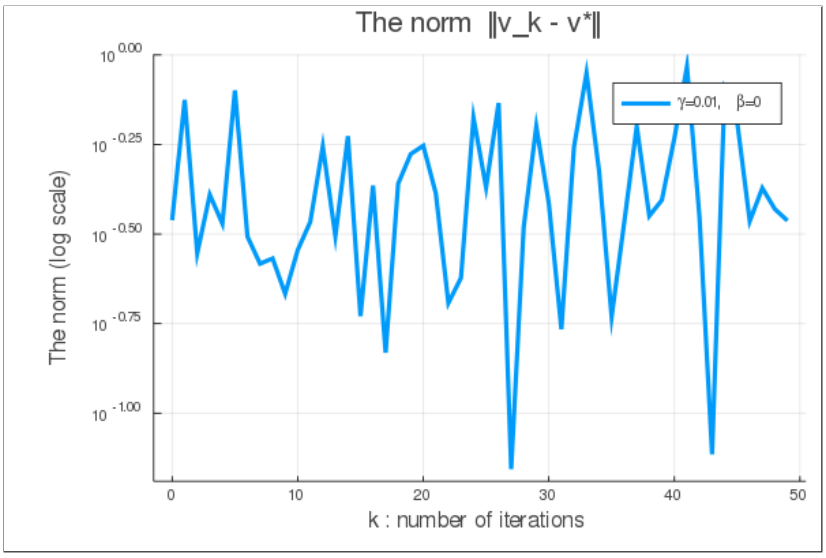
In order to compare real time performance using a fair way, we will compare all methods over needed iterations of the same amount of computations, in order to facilitate the analysis, we say that we reach the solution at (10^{-8}):



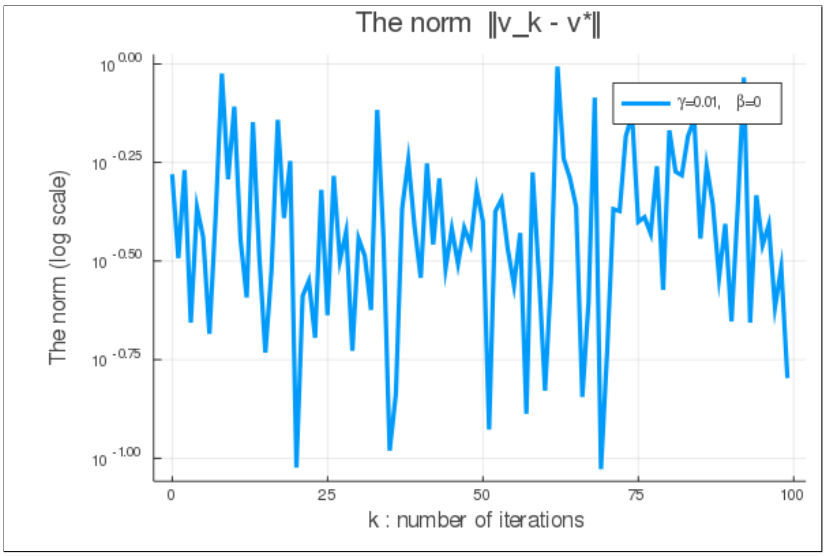
```
julia> @time PG(10^4)
  0.601415 seconds (465.34 k allocations: 394.505 MiB, 6.00% gc time)

julia> @time APG_3(6000)
  0.434628 seconds (405.22 k allocations: 321.613 MiB, 6.59% gc time)

julia> @time APG_4(5000)
  0.395786 seconds (428.84 k allocations: 278.672 MiB, 3.99% gc time)

julia> @time CG(10^4)
 12.814778 seconds (40.38 M allocations: 41.167 GiB, 20.00% gc time)

julia> @time CG_wise(281)
  0.376681 seconds (1.34 M allocations: 1.189 GiB, 19.18% gc time)
```

We see that the CG_wise is the fastest in real time and requires the least amount of iterations. Also the APG using (4) seems perfect since has approximately the same time of runnning and requires only 5000 iterations. If we consider iterations without the same amount of computations, then APG (4) is the fastest and requires the least amount of iterations since CG will require (281 x N » 5000).

Task 3 Solve the existing model fitting problem with the SG method:
We let the number of iterations be N times as many and we plot the cost for every N iterations. Using $\gamma = 0.01$, $\beta = 0$ and $50N$:

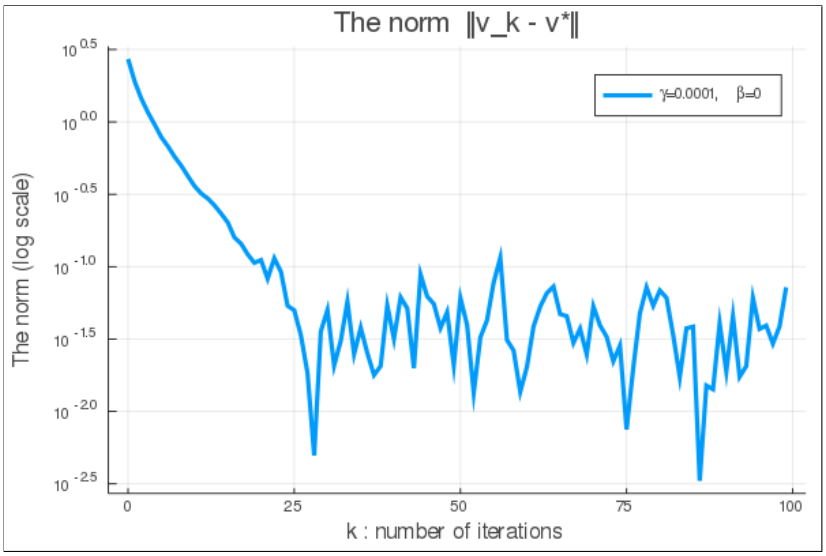


How close to the solution do you get?
After $50N$ iterations the distance to the solution does not decreases. It seems like it oscillates around the initiale point.
Can you get closer to the solution if you let the algorithm run longer?



As we see, even if we let the algorithm run longer the distance still noisy and does not show any progress.

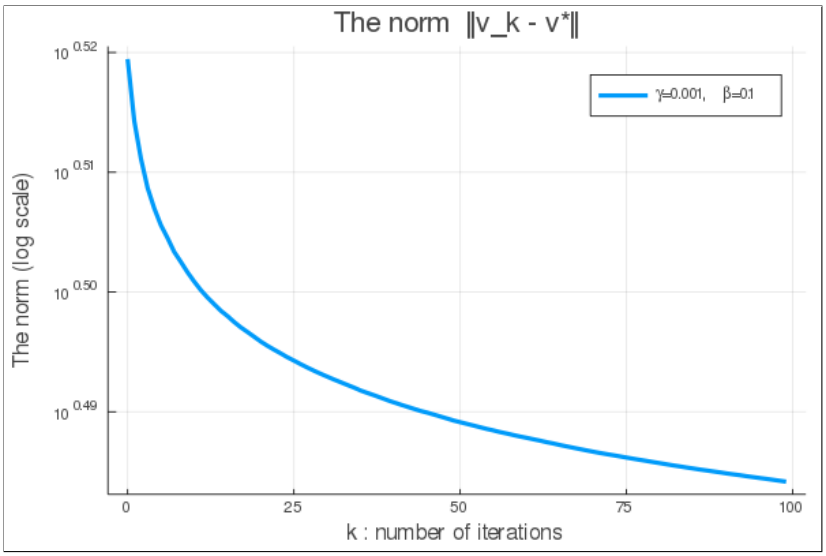
Redo the experiment with $\gamma = 0.0001$.



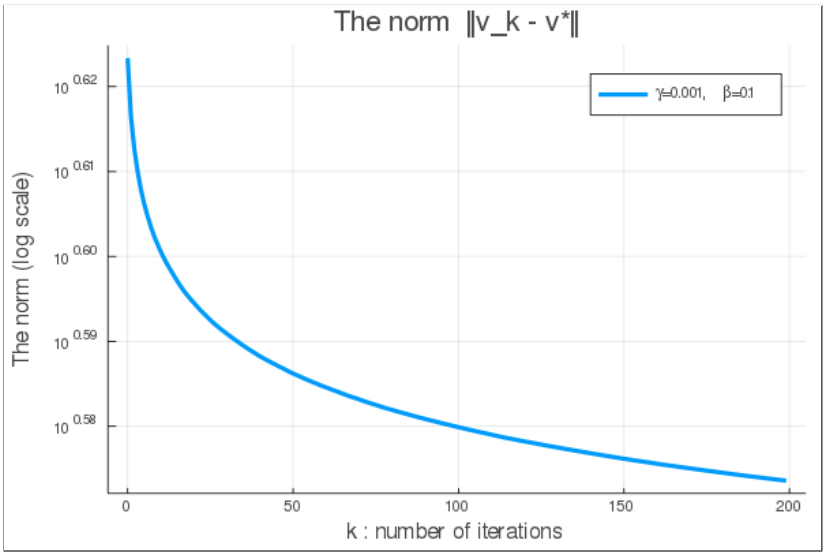
This step-size shows a fast decrease of the distance over the first 25 iterations, until we get a distance around 10^{-2} which is closer than the first case. However, the distance still be noisy after the last 75 iterations i.e. the algorithm does not show any progress.

Task 4 Solve the existing model fitting problem with the SG method:

Using $\gamma = 0.01$, $\beta = 0.1$ and $100N$ iterations:

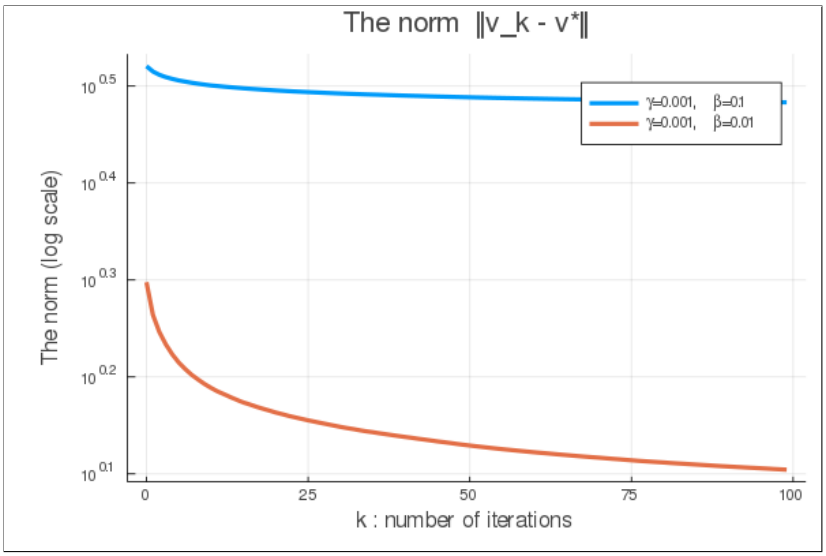


How close to the solution do you get?
After $100N$ iterations the distance to the solution decreases from $10^{0.52}$ to $10^{0.48}$, we don't reach the solution yet.
Run for $100N$ iterations more



We decrease with the same rate (sublinear), but a lot of iterations may required in order to reach the solution.

Rerun the experiment with a slower decay speed of $\beta = 0.01$.



From this figure we can say that $\beta = 0.01$ performs better than $\beta = 0.1$.
Both decaying step-sizes garantee convergence for smooth convex problem. However large decay ($\beta = 0.1$) will lead to slow convergence.

Task 5 Solve the 5th order model fitting problem with the SG method:

Using $\gamma = 0.001$, $\beta = 0$ and $100N$ iterations:

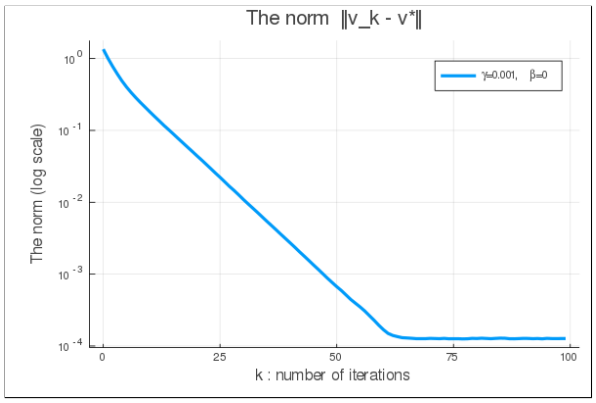


Figure 5: **The distance to the solution**

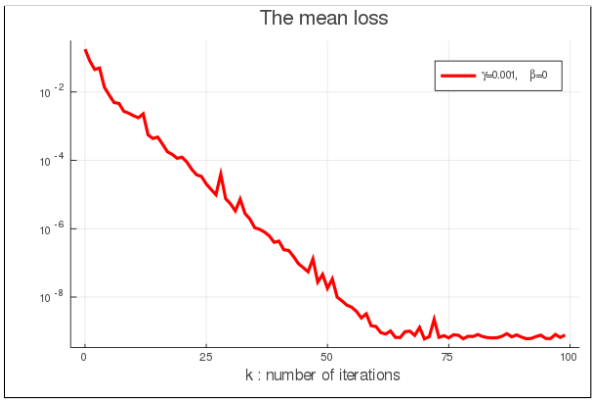


Figure 6: **The mean loss**

So, as we can see here SG algorithm converges in this case since the mean loss reaches the numerical precision 10^{-10}.
Reset the problem and instead run ordinary gradient descent for 100 iterations with $\gamma = 0.009$

```
julia> task5()
    ....
    Mean loss: 0.16298079826785927
    Distance to solution: 0.9200880724903178
    Mean loss: 0.13558599793794546
    Distance to solution: 0.8963633789978757
```

Both the mean loss and the distance to the solution of the GD did not reach the numerical precision over 100 iterations.

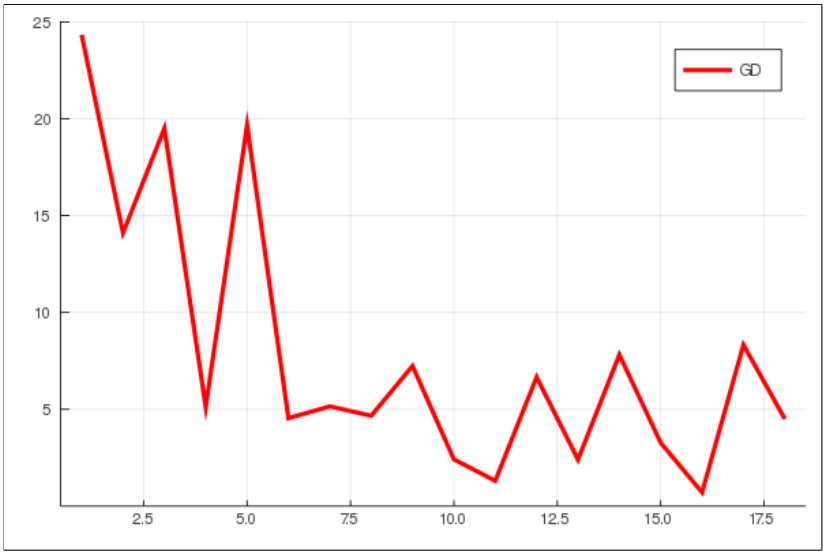
Both algorithms garantee convergence, but in this case SG is faster and performs better than the ordinary GD over 100 iterations since .

So it is fair to compare 100 iterations of GD against $100N$ iterations of SG since N iterations of stochastic gradient is at cost of 1 full gradient.

Task 6 Solve the neural network model:

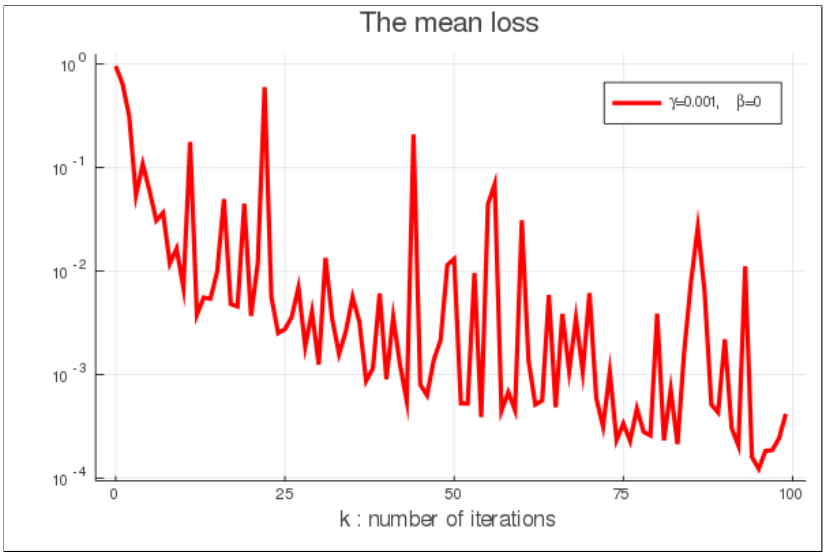
Using $\gamma = 0.02$, $\beta = 0$ and 100 iterations.

Using GD, does it appear to converge?

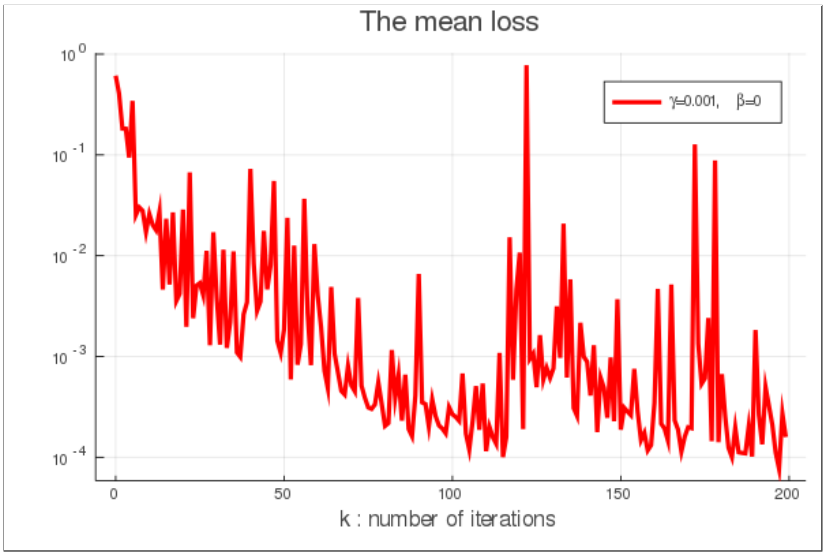


As we can see, the mean loss is not decreasing during learning.

Reset the problem and run SG for $100N$ iterations with $\gamma = 0.001$ and $\beta = 0.0$



So, using the SG method made the mean loss decreassing over iterations.
We get an average loss of 10^{-3} of precision.
Run for an additional $100N$ iterations. Does the mean loss improve?



Using neural network model will result a huge amount of parameters (large scale) giving us a normal convergence reaching a precision of 10^{-3} over 100 iterations. The 5th order is the right choice to fit the real model, that is why we get fast convergence. The second order works fine with SG but we need to choose wisely beta in order to avoid slow convergence. Finally, The real model is a 5th order problem, 2nd order can work fine but cannot fill the gap of the three left parameters. So using SG or GD may work fine on seen data but would risk to have overfitting or underfitting. Also moving from polynomial (in this case 5th order) model to more complex model which is neural network in this case, would have the same problem.