COM S 578X: Optimization for Machine Learning
Homework 5

Name: Steven Sleder Email: ssleder@iastate.edu

December 10, 2019

1 Notes on Algorithm Implementation

1.1 Gradient Descent

Gradient Descent's implementation did not feature any modifications or design choices due to its simplicity. The only thing of note is using the Inexact Line Search with Backtracking to find the step size, per the assignment instructions.

1.2 Heavy-Ball

Heavy-Ball's implementation also did not feature any modification. Again, the only notable aspect was using the Inexact Line Search with Backtracking to locate the step size. For each iteration, the momentum remained a fixed parameter.

1.3 Polak-Ribere Conjugate Gradient

Polak-Ribere Conjugate Gradient was implemented almost exactly as featured in the lecture notes. The main deviation was the number of iterations of the inner loop being set to the length of the input vector size $\mathbf{x}$. The early stopping value for this inner loop was also passed to the algorithm as a parameter. The implementation of the Exact Line Search to find the step size was implemented using the Golden-Selection Search (documented below in Subsection 1.8).

NOTE: This idea to use the Golden-Selection Search was shared with fellow student Pierce Adajar, it is unknown if he used it or not.

1.4 Nesterov Accelerated Gradient Descent

The original implementation for Nesterov Accelerated Gradient Descent followed directly form is presentation in the lecture notes. This lead to several issues which were then resolved by leveraging a different implementation.

The original gradient step relied upon the method being passed an estimate for the Lipschitz constant L in place of the step size. Unfortunately, experiments with various values of $L \in (0, 1]$ lead to overflows in the implementation. It was then recalled that one could estimate the Lipschitz constant by the step size by using the inexact line search with backtracking, since the exist condition for backtracking is satisfied when:

$$s \leq 1/L$$

This approach resolved my issues with overflows and lead to convergence with the algorithm. Taking the maximum of all these values produced over several executions of the program would yield a more accurate estimate of the Lipschitz constant. Sadly, time did not permit experimenting with this.

Calculating β_k in order to incorporate the momentum component of this method also presented an issue. Per the lecture slides, I passed in $a_0 \in (0, 1)$ and an estimate of the Condition Number κ as arguments to the algorithm to solve for $a_{k+1} \in (0, 1)$:

$$\alpha_{k+1}^2 + (\alpha_k^2 - (1/\kappa))\alpha_{k+1} - \alpha_k^2 = 0$$

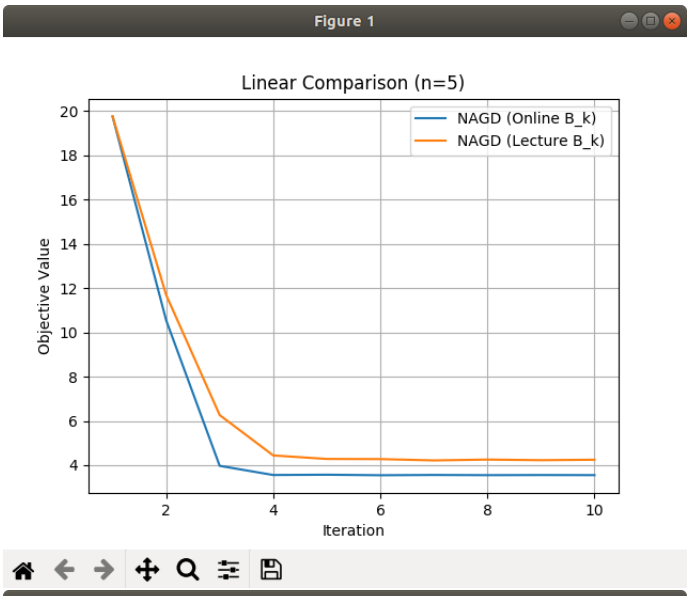
$$\beta_k = \frac{\alpha_k(1 - \alpha_{k+1})}{(\alpha_k^2 + \alpha_{k+1})}$$

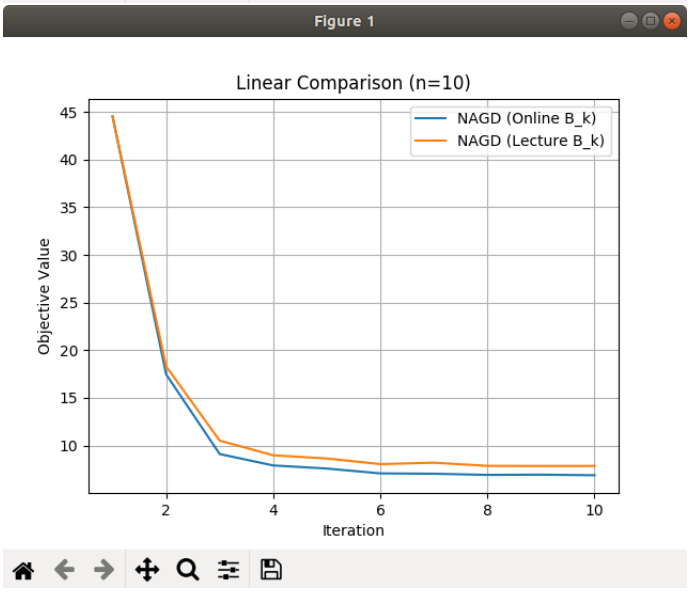
This method also lead to an overflow. Instead, the method from the previously linked page was used:

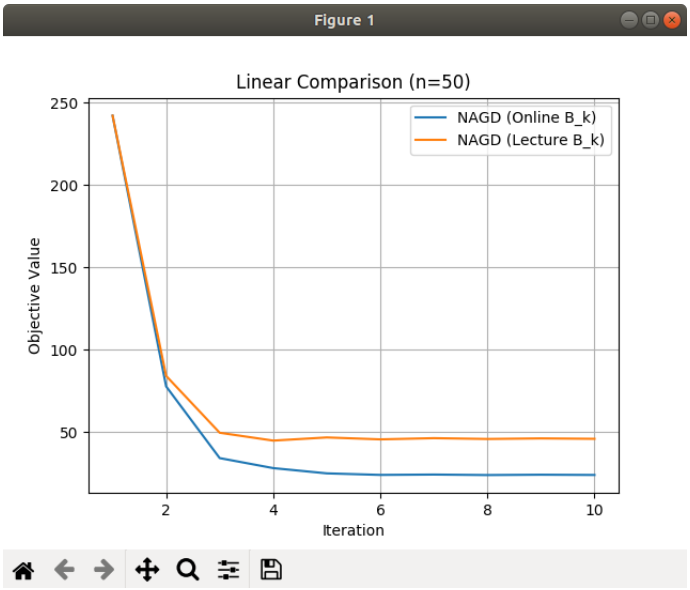
$$\alpha_{k+1} = \frac{1 + \sqrt{1 + 4\alpha_k^2}}{2}$$

$$\beta_k = \frac{1 - \alpha_k}{\alpha_{k+1}}$$

Doing this again eliminated the overflow and lead to the algorithm converging. However both of the two methods of actually calculating β_k once α_{k+1} was known could be utilized, but the online source's version performed slightly better empirically in terms of time to convergence and overall accuracy:







Which version is used can be toggled by commenting/uncommenting lines 137/138 in *methods.py*

1.5 FISTA

FISTA was implemented exactly as was shown in the lecture notes. Unfortunately, it was noticed in hindsight that the method used for finding a_{k+1} in Nesterov Accelerated Gradient Descent (from the online resource) was exactly the same as the method used to find t_{k+1}. The only differences between the two algorithms was the initialization of $t_0 = 1$ and a_0 being a chosen parameter and the momentum component having a different equation in FISTA:

$$\beta_k = \frac{t_k - 1}{t_{k+1}}$$

1.6 Barzilai-Borwein

Barzilai-Borwein was implemented exactly as in the lecture notes, except for the first iteration $k = 1$ as calculating s_k would mean division by zero if $x_{-1} = x_0$.

Therefore, the first iteration instead performed standard Gradient Descent with Inexact Line Search with Backtracking with the fixed parameters of $\alpha = 0.5$ and $\beta = 0.5$ for simplicity. As this would only affect the first iteration of the algorithm, its impact was expected to be minimal and was not experimented with due to time constraints.

1.7 Inexact Line Search with Backtracking

This method of locating the step size was implemented exactly as in the lecture notes. It featured a tunable value of α and β which are passed with each method that calls it (with the exception of Barzilai-Borwein).

1.8 Exact Line Search with Golden-Section Search

The Exact Line Search was utilized by this implementation of the Conjugate Gradient Method. Unfortunately, the exact line search presented yet another non-convex optimization problem to solve. Investigating this lead to finding the Golden Section-Search which attempts to find the minimum value of a strictly unimodal function with a local minima inside of a provided interval (chosen to be $(0, high)$] where $high$ is a passed parameter). The lack of convexity (local minima) and unimodality of the function was overcome by restricting the search to a finite number of iterations.

1.9 Early Stopping

All of the algorithms were implemented with an early stopping condition that is triggered at the end of every iteration which stops the algorithm if the solution is found, or if the relative error between the last two points is below the following threshold:

$$||\mathbf{x}_k - \mathbf{x_{k-1}}||_2 < \epsilon * ||\mathbf{x_k}||_2$$

Since the stopping condition was rarely reached, the value of $\epsilon = 1E-4$ remained static for the experiments.

2 Experiments

The following graphs were produced for each experiment:

1. **Linear Comparison (n=#)**: Line graph of the objective value $f(\mathbf{x})$ at each iteration.
2. **Log Comparison (n=#)**: Line graph of the log value of the objective value $f(\mathbf{x})$ at each iteration.
3. **Time Until Stop (n=#)**: Bar graph of the average amount of time taken for each method to complete execution
4. **Avg Iteration Time (n=#)**: Bar graph of the average amount of time taken for each iteration of the algorithm.

2.1 Vector Size *n=5*

This set of tests can be reproduced by running *python3 size_n5_tests.py*

All methods were run for 10 iterations and all parameters were tuned by hand.

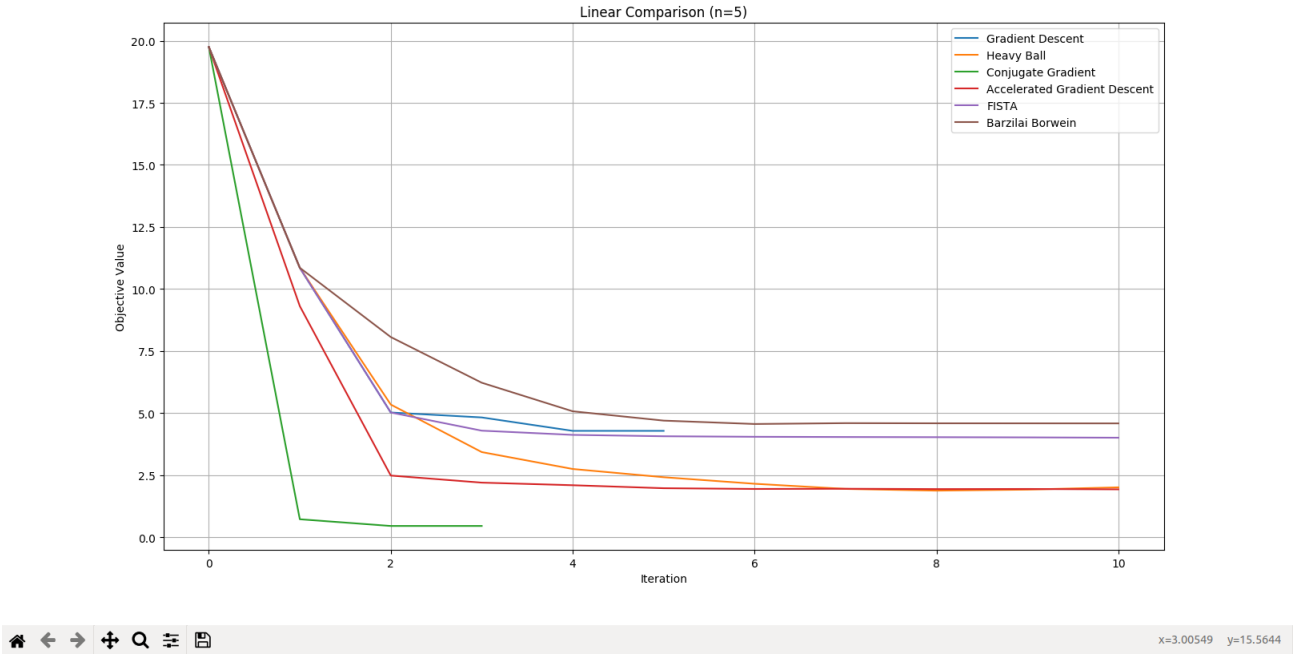
2.1.1 First Test

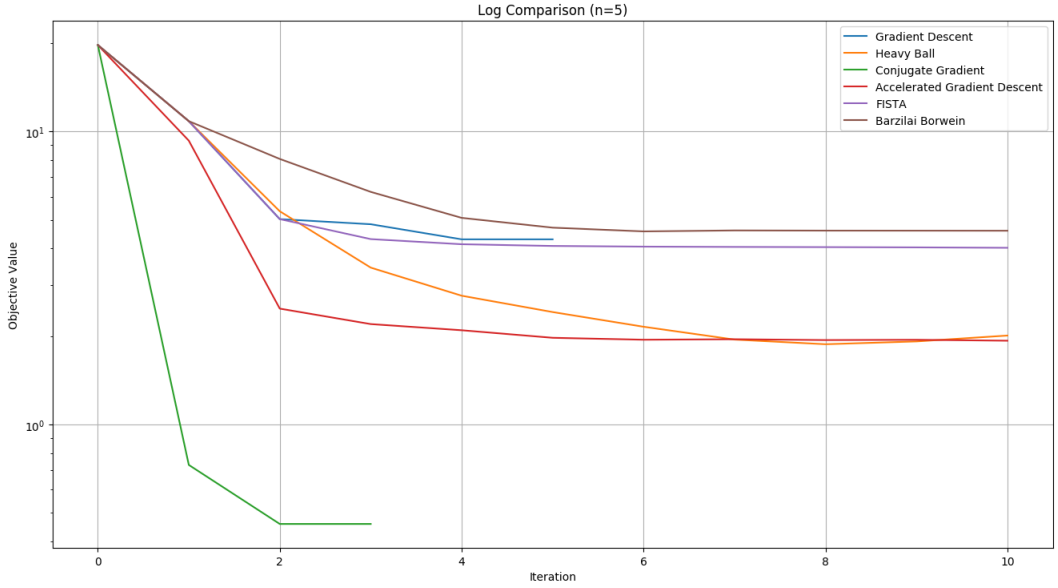
Uncertain which parameters to use for initial testing, the lecture slides for Inexact Search with Backtracking listed setting $\alpha = \beta = 1/2$. However, it was uncertain that the function was smooth and that this would work so it served as a good starting point. For simplicity, all other values were chosen set to this.

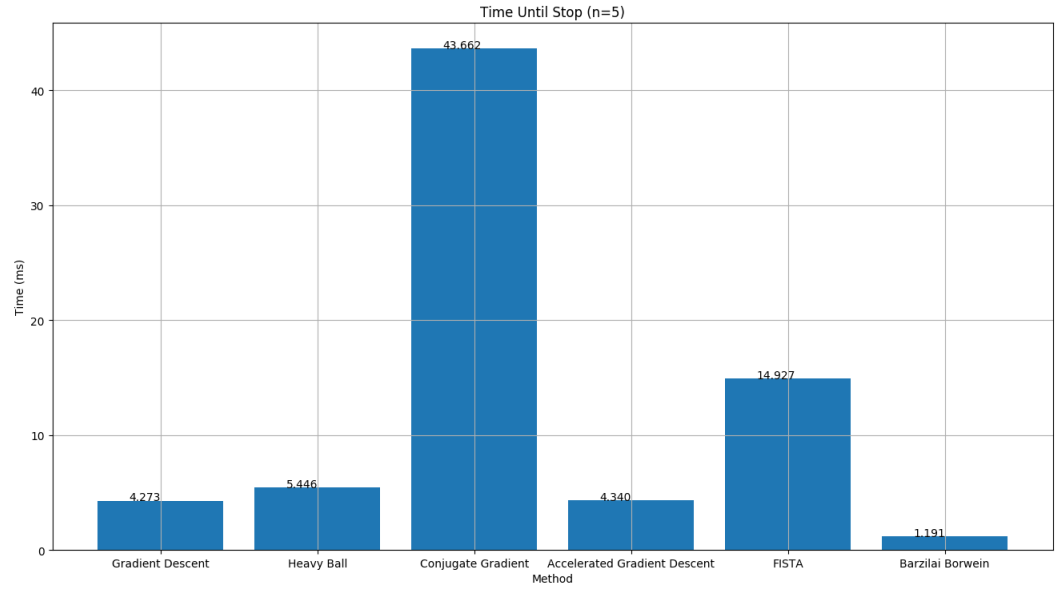
- **Parameters:**
  - **Gradient Descent**:
    - Inexact Line Search with Backtracking
      - $\alpha = 0.5$
      - $\beta = 0.5$
  - **Heavy-Ball**:
    - Inexact Line Search with Backtracking
      - $\alpha = 0.5$
      - $\beta = 0.5$
    - $\beta = 0.5$ (momentum)
  - **Conjugate Gradient**:
    - Exact Line Search with Golden-Selection Search
      - $Upper\_Bound = 0.5$
    - $\epsilon = 1E-5$
  - **Accelerated Gradient Descent**:
    - Inexact Line Search with Backtracking
      - $\alpha = 0.5$

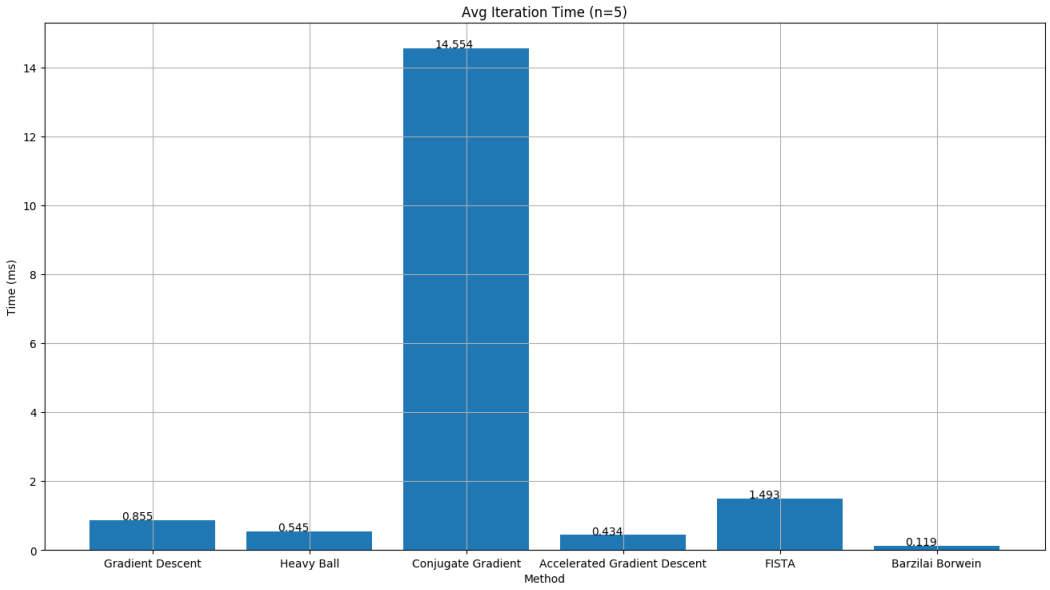
        - $\beta = 0.5$
    - $a_0 = 0.5$ (initial momentum)
  - **FISTA**:
    - Inexact Line Search with Backtracking
      - $\alpha = 0.5$
      - $\beta = 0.5$
  - **Barzilai-Borwein**:
    - NONE

- **Output Graphs**




Log Comparison (n=5)
Gradient Descent
Heavy Ball
Conjugate Gradient
Accelerated Gradient Descent
FISTA
Barzilai Borwein
Objective Value
10^1
10^0
0
2
4
6
8
10
Iteration


Time Until Stop (n=5)
Time (ms)
0
10
20
30
40
4.273
5.446
43.662
4.340
14.927
1.191
Gradient Descent
Heavy Ball
Conjugate Gradient
Accelerated Gradient Descent
FISTA
Barzilai Borwein
Method




```
First Experiment for n=5
        Final Objective Values [f(x)] for Vector Size : 5
                Gradient Descent                    [4.28854828]
                Heavy Ball                          [2.01299213]
                Conjugate Gradient                  [0.45783219]
                Accelerated Gradient Descent        [1.9331544]
                FISTA                               [4.01260243]
                Barzilai Borwein                    [4.58865406]
```

- **Remarks**
  - **Gradient Descent**: The simplest method appeared to get stuck at a somewhat local minima rather quickly and met the stopping conditions by the sixth iteration. It appears that adjusting the step size might be a way to avoid this.
  - **Heavy-Ball**: Heavy Ball was a strong performer for accuracy and its convergence eventually beat Gradient Descent, which makes sense as it is an improvement on the method. The surprise for this method came from it converging to nearly the same point as Accelerated Gradient Descent, though at a slower rate. It may be possible to improve its performance with a higher momentum, though this may cause instability.
  - **Conjugate Gradient**: Conjugate Gradient had the best overall accuracy with very little tuning and the fastest convergence rate. It significantly reduced the error in just two iterations which is quite impressive. It also managed to reach the stopping condition very quickly.

    However, its iterations take a significant amount of time, no doubt due to the inner loop to set the step size and gradient. Increasing the value of the inner loop's stopping condition may reduce this but could also possibly decrease the accuracy by a large margin.
  - **Accelerated Gradient Descent**: The convergence rate of this method was quite impressive compared to the other methods, but its accuracy was expected to be higher than Heavy Ball's. That said, it is possible that this is simply due to the topology of the function's gradient decreasing the impact of the momentum.

- **FISTA**: FISTA performed quite poorly, especially considering its implementation similarities to Accelerated Gradient Descent. It seems that by increasing the initial step size (by increasing β), it may be possible to give it more of an initial push to replicate the success of Accelerated Gradient Descent.

  It also took a significant amount of time compared to the average time per iteration of the other methods, making it not seem suitable for a smaller vector size. No insight can be seen from the implementation as it is very similar to Accelerated Gradient Descent, but with a hard-coded initial $\alpha = 1.0$.
- **Barzilai-Borwein**: Surprisingly, this method performed the most poorly in terms of accuracy and convergence rate over the ten iterations. However, it absolutely had the lowest total time to completion by a significant margin for the ten iterations and the lowest overall time for completing individual iterations.

Overall, it seems that Conjugate Gradient may remain the method to beat, but it is expected to have recurring issues with the time it takes to complete each iteration being significantly more than any of the other methods.

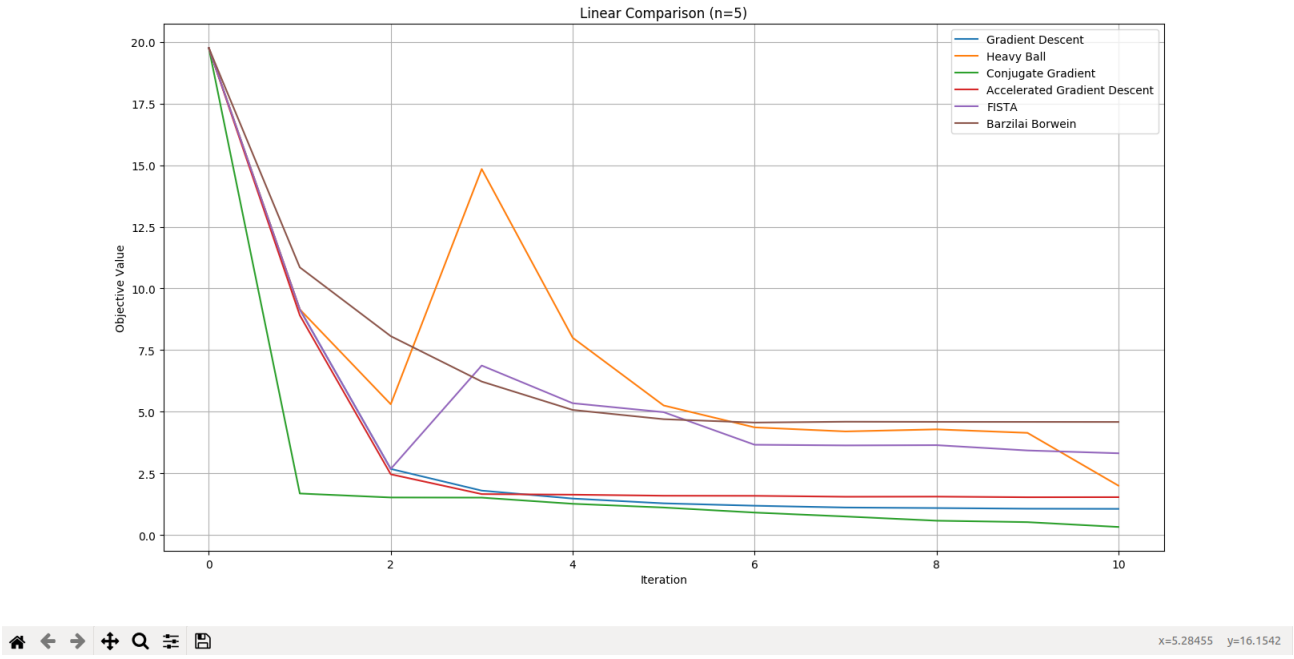
2.1.2 Second Test

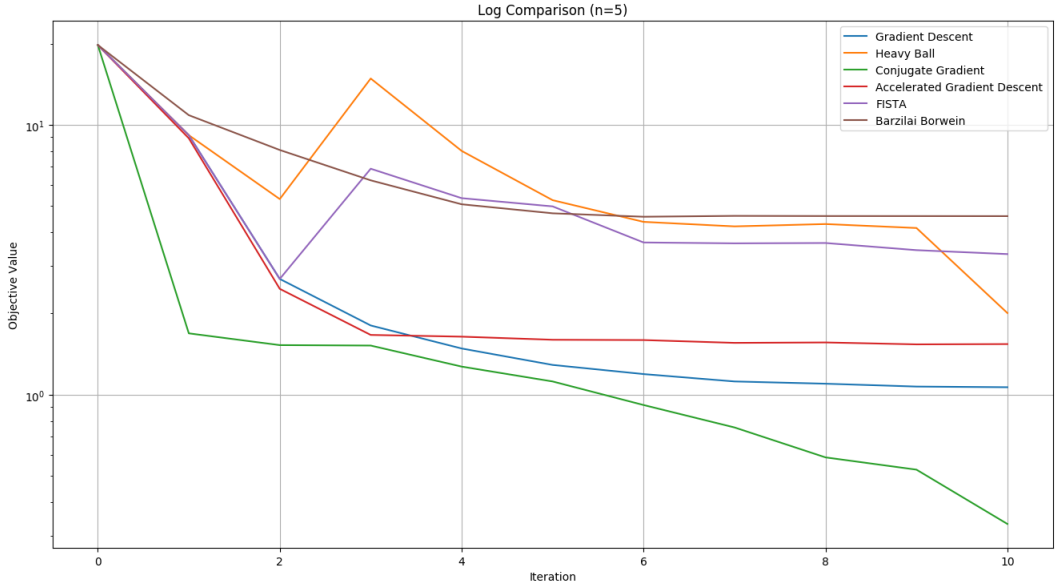
This experiment focused primarily on reducing the time taken for Conjugate Gradient to complete an iteration by causing an early stop for its inner loop. The other methods are primarily focused on increasing their overall accuracy.

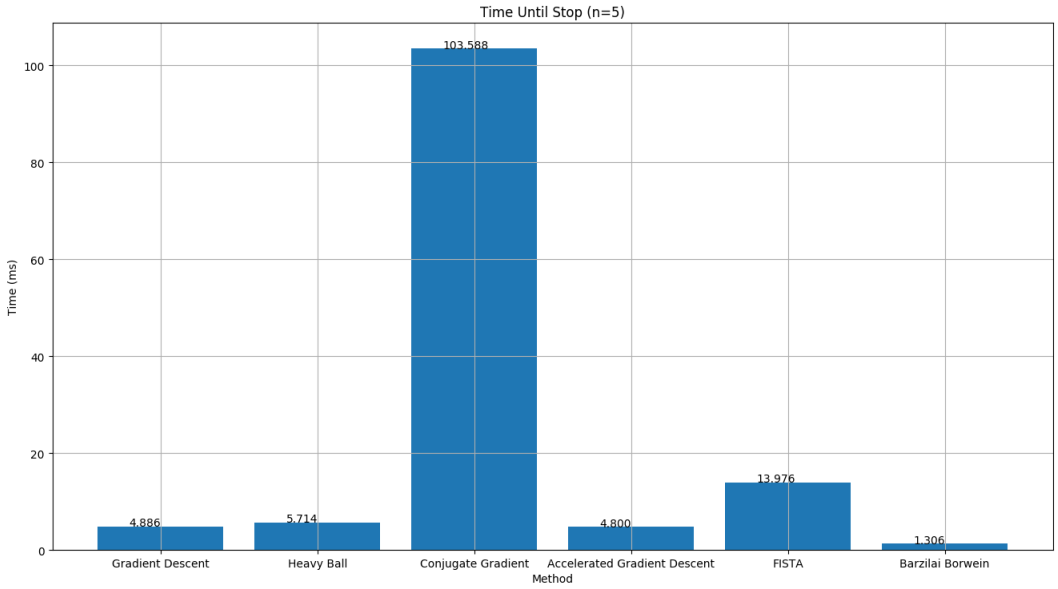
- **Parameters:**
  - **Gradient Descent**: The inexact line search was tweaked to hopefully overcome the early stopping by increasing β to preserve the higher step sizes.
    - Inexact Line Search with Backtracking
      - $\alpha = 0.5$
      - $\beta = 0.75$
  - **Heavy-Ball**: The inexact line search was tweaked to hopefully overcome the early stopping by increasing β to preserve the higher step sizes. The momentum was reduced to prevent oscillations from occurring due to the possibly larger step sizes
    - Inexact Line Search with Backtracking
      - $\alpha = 0.5$
      - $\beta = 0.75$
    - $\beta = 0.4$ (momentum)
  - **Conjugate Gradient**: The inner loop ϵ was increased to encourage earlier stopping to decrease the time needed for this method to complete a single iteration.
    - Exact Line Search with Golden-Selection Search
      - $Upper\_Bound = 0.5$
    - $\epsilon = 1E - 4$
  - **Accelerated Gradient Descent**: The inexact line search was tweaked to hopefully overcome the early stopping by increasing β to preserve the higher step sizes. The initial a_0 that is passed to the method was also increased to hopefully take advantage of the initial large accuracy increase as it will increase the value of the first momentum component.
    - Inexact Line Search with Backtracking
      - $\alpha = 0.5$
      - $\beta = 0.75$
    - $a_0 = 0.75$ (initial momentum)

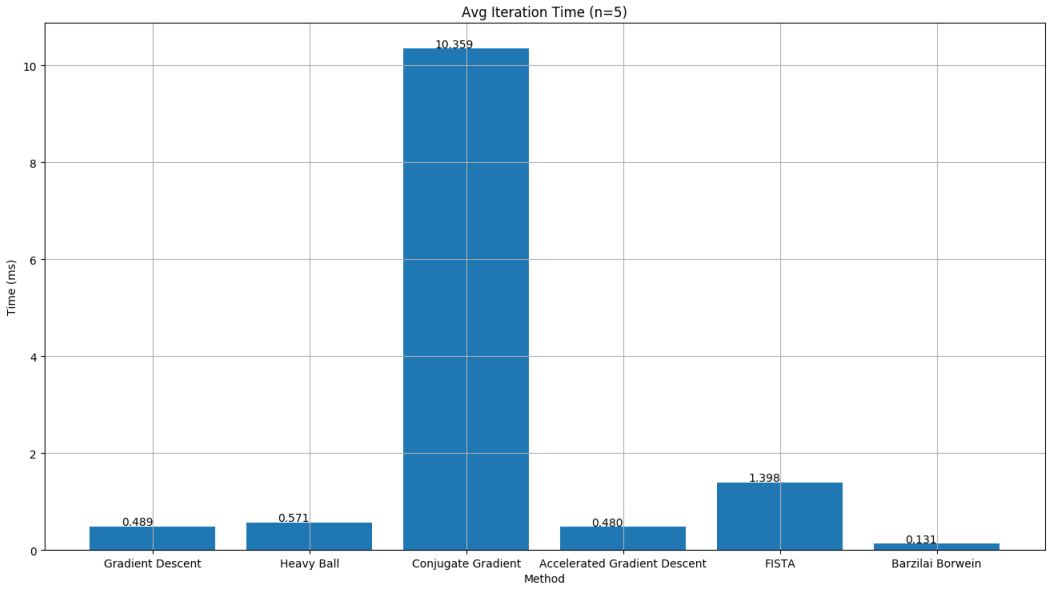
- **FISTA**: The inexact line search was tweaked to hopefully overcome the early stopping by increasing β to preserve the higher step sizes. Hopefully this will drive it to perform similar to Accelerated Gradient Descent
  - Inexact Line Search with Backtracking
    - $\alpha = 0.5$
    - $\beta = 0.75$
- **Barzilai-Borwein**:
  - NONE

- **Output Graphs**




Log Comparison (n=5)
Gradient Descent
Heavy Ball
Conjugate Gradient
Accelerated Gradient Descent
FISTA
Barzilai Borwein
Objective Value
Iteration


Time Until Stop (n=5)
103.588
4.886
5.714
4.800
13.976
1.306
Time (ms)
Gradient Descent
Heavy Ball
Conjugate Gradient
Accelerated Gradient Descent
FISTA
Barzilai Borwein
Method




```
Second Experiment for n=5
        Final Objective Values [f(x)] for Vector Size : 5
                Gradient Descent                        [1.06610724]
                Heavy Ball                              [2.00751465]
                Conjugate Gradient                      [0.3321043]
                Accelerated Gradient Descent            [1.54062459]
                FISTA                                   [3.31956292]
                Barzilai Borwein                        [4.58865406]
```

- **Remarks**
  - **Gradient Descent**: Modifying the line search lead to a significant improvement for Gradient Descent at these parameters, actually managing to beat several of the more complicated methods and rival Accelerated Gradient Descent.
  - **Heavy-Ball**: The momentum component of Heavy Ball seems to be too significant and must be reduced, as can be seen from the immediate spike. This is clearly visible from the linear and log graphs. Surprisingly, this actually did have a positive effect near the end of the set of iterations. It may be possible that this higher momentum would help it overcome the local minima that other methods seem to me stuck in towards the end of the experiment?
  - **Conjugate Gradient**: Increasing the inner early stopping ϵ lead to slightly lower average iteration times and also seemed to prevent the overall early stopping of the algorithm. Curiously, it also lead to a slightly improved accuracy over the last version. This seems counter-intuitive to what one would expect. Wouldn't it be better to go with a larger change in the gradient?

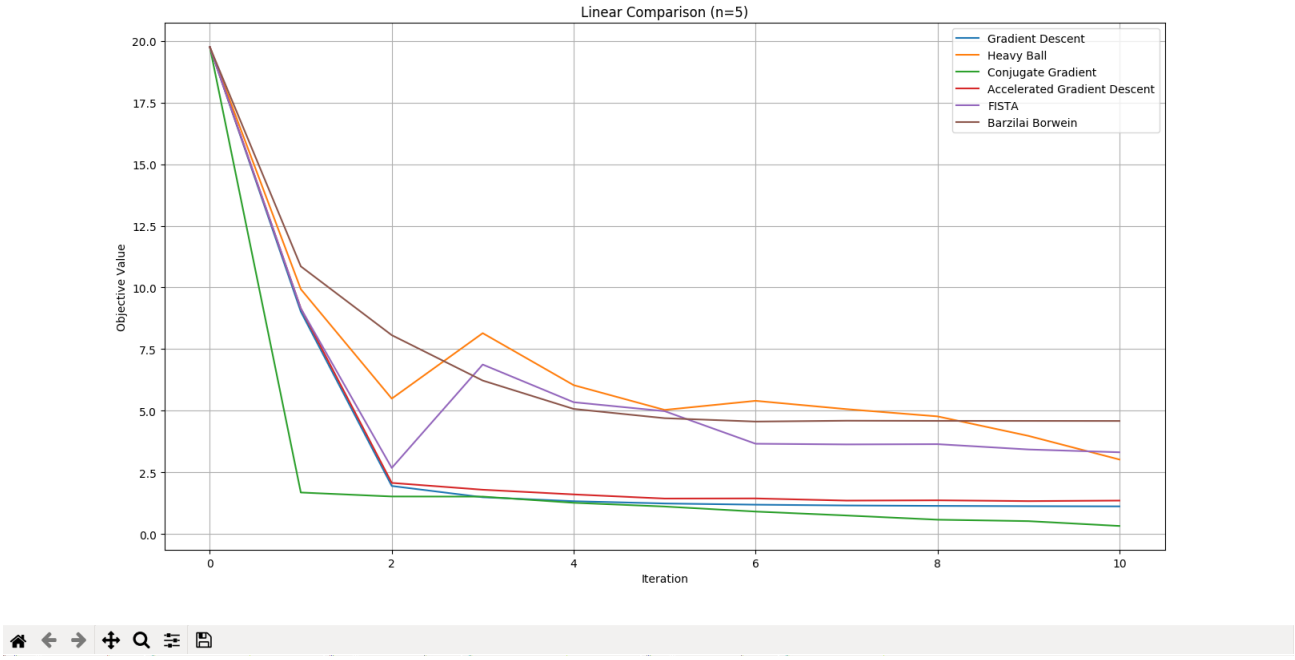
    However, it log graph seems to indicate that this method is still converging quite strongly and would continue to do so for additional iterations.
  - **Accelerated Gradient Descent**: This method was almost beat by Gradient Descent for its original convergence rate and then slightly beaten in terms of accuracy, which is surprising. It looks like the original value a_0 may need to be reduced after all to decrease the reliance on the momentum component.

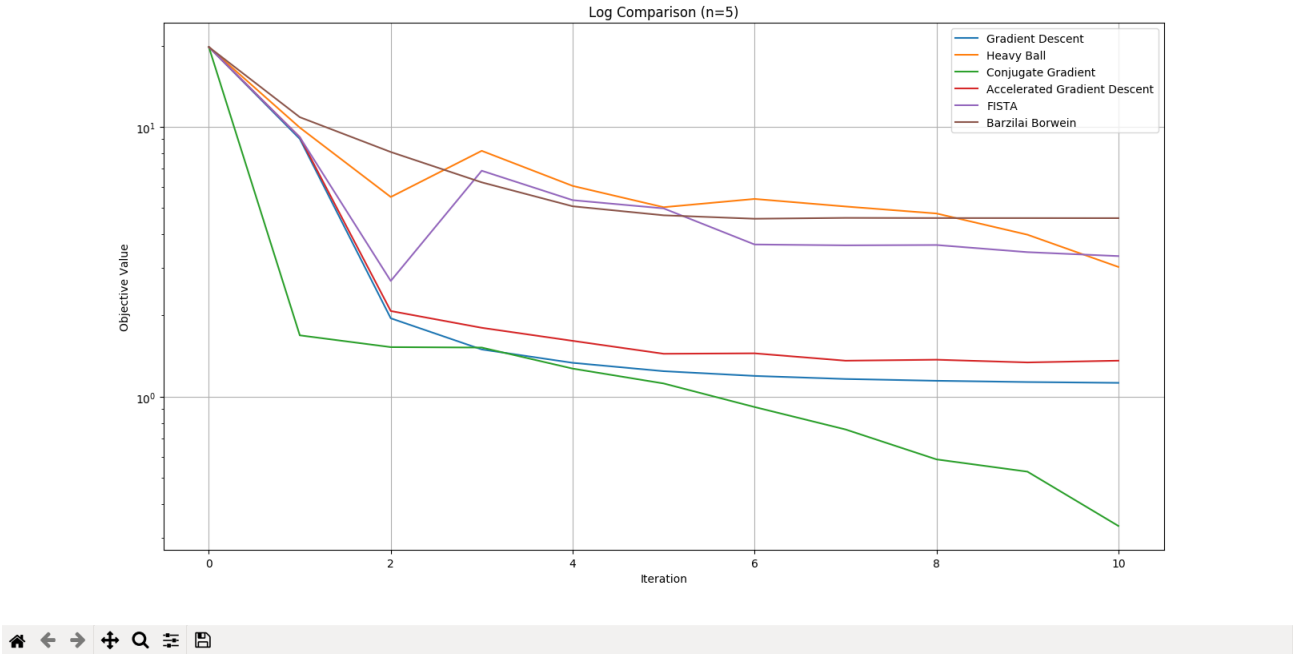
- **FISTA**: FISTA appeared to share the issues that Heavy Ball had and actually seems to indicate that my previous statement about Heavy Ball's momentum being too significant is not true. It appears that the line search may be the cause of these spikes that the two methods share at iteration #3.
- **Barzilai-Borwein**: No insight other than that this remains one of the fastest methods in terms of time per iteration.

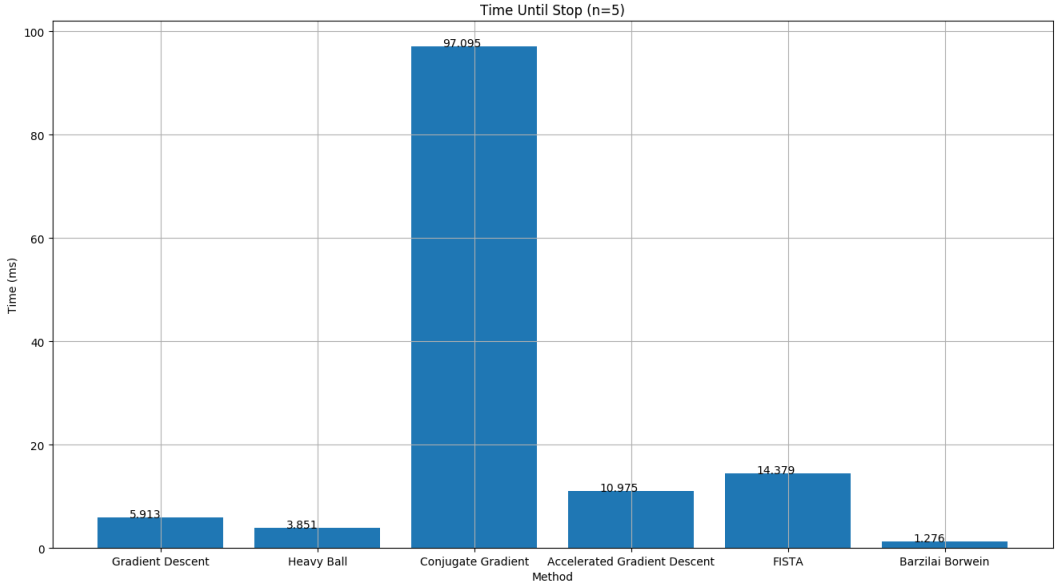
From this test set, it seems that manipulating the inexact line search for several of the algorithms actually led to worse results for the momentum based approaches (Heavy Ball, FISTA), but oddly not for others (Accelerated Gradient Descent. Once again, Conjugate Gradient remains the method to beat.

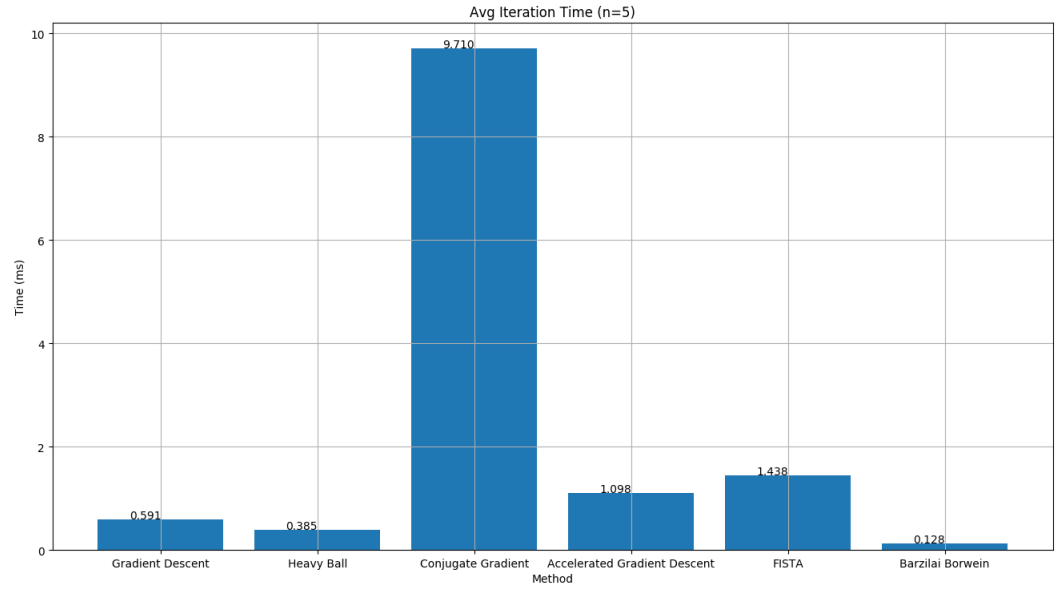
2.1.3 Third Test

- **Parameters:**
  - **Gradient Descent**: β was slightly increased to see if doing so would give the same improvement as in the last experiment.
    - Inexact Line Search with Backtracking
      - $\alpha = 0.5$
      - $\beta = 0.8$
  - **Heavy-Ball**: The inexact line search seemed to be the source of some issues so the decrease in the step size was increased by decreasing β somewhat. The momentum remained the same.
    - Inexact Line Search with Backtracking
      - $\alpha = 0.5$
      - $\beta = 0.6$
    - $\beta = 0.4$ (momentum)
  - **Conjugate Gradient**: The inner loop ϵ was increased once again, given its significant improvement to the speed and accuracy in the last test.
    - Exact Line Search with Golden-Selection Search
      - $Upper\_Bound = 0.5$
    - $\epsilon = 1E-2$
  - **Accelerated Gradient Descent**: The value of a_0 was set to exactly that of FISTA (hardcoded to 1.0) in the last iteration to see if they produce similar results, given their algorithmic similarity.
    - Inexact Line Search with Backtracking
      - $\alpha = 0.5$
      - $\beta = 0.75$
    - $a_0 = 1.05$ (initial momentum)
  - **FISTA**: FISTA remained the same to compare with Accelerated Gradient Descent.
    - Inexact Line Search with Backtracking
      - $\alpha = 0.5$
      - $\beta = 0.75$
  - **Barzilai-Borwein**:
    - NONE
- **Output Graphs**


Linear Comparison (n=5)
Gradient Descent
Heavy Ball
Conjugate Gradient
Accelerated Gradient Descent
FISTA
Barzilai Borwein
Objective Value
Iteration
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
2
4
6
8
10


Log Comparison (n=5)
Gradient Descent
Heavy Ball
Conjugate Gradient
Accelerated Gradient Descent
FISTA
Barzilai Borwein
Objective Value
Iteration
10^1
10^0
0
2
4
6
8
10


Time Until Stop (n=5)
Time (ms)
Method
Gradient Descent
Heavy Ball
Conjugate Gradient
Accelerated Gradient Descent
FISTA
Barzilai Borwein
5.913
3.851
97.095
10.975
14.379
1.276


Avg Iteration Time (n=5)
Time (ms)
Method
Gradient Descent
Heavy Ball
Conjugate Gradient
Accelerated Gradient Descent
FISTA
Barzilai Borwein
0.591
0.385
9.710
1.098
1.438
0.128

```
Second Experiment for n=5
        Final Objective Values [f(x)] for Vector Size : 5
                Gradient Descent                    [1.06610724]
                Heavy Ball                          [2.00751465]
                Conjugate Gradient                  [0.3321043]
                Accelerated Gradient Descent        [1.54062459]
                FISTA                               [3.31956292]
                Barzilai Borwein                    [4.58865406]
```

- **Remarks**
  - **Gradient Descent**: Unfortunately, the changes made to the step size search led to slightly worse convergence rates than last time as well as a slight drop inaccuracy.
  - **Heavy-Ball**: The changes made to the search by decreasing β seemed to have worked and reduced the visible spike in inaccuracy seen in the log and linear graphs of the objective value. This seems to indicate that the value of α and β could continue to be manipulated.
  - **Conjugate Gradient**: Conjugate gradient produced the exact same output objective value for the function, but featured a slight increase in the time it spent iterating. It seems that the previous values are actually the best so far.
  - **Accelerated Gradient Descent**: The accuracy and rate for this method remained almost entirely the same since the last test. However, there was a notable increase in the time taken to complete an iteration, on average.
  - **FISTA**: FISTA was a control for the experiment when comparing to Heavy Ball and Accelerated Gradient Descent, so there is nothing to note here.
  - **Barzilai-Borwein**: NONE

2.1.4 Conclusion for This Size

Overall, Conjugate Gradient Descent remained the most accurate method for minimizing the function. However, it consistently had very high times to complete each iteration, which seems will be a more significant issue as the size of the vectors increase.

Barzilai Borwein had the worst overall accuracy, but its was of implementation and incredibly low time taken per iteration makes it an attractive candidate, it is expected that this property will remain as the vector sizes increase.

Accelerated and normal Gradient Descent both were strong performers for the smaller vector size both or speed and overall accuracy, approaching much closer to an optimal solution than the other methods, with the exception of Conjugate Gradient. However, their low timer per iteration makes them more attractive than Conjugate Gradient would be for larger vector sizes if that pattern continues.

The overall lack of accuracy with the methods was somewhat surprising, but makes sense in retrospect. This is non-convex optimization and most methods will be unlikely to find an optimal solution because of this, but for this size, several of the methods were still able to come close. But it seems that this inaccuracy will increase with the vector size.

NOTE: A version to perform Grid Search with this vector size is featured in the code, but there was not enough time to document it before submission. It can be run with *python3 size_n5_search.py*

Parameters for Best Objective Values:

- **Gradient Descent**: $f(\mathbf{x}_{best}) = 1.06610724$
  - Inexact Line Search with Backtracking
    - $\alpha = 0.5$
    - $\beta = 0.75$

- **Heavy-Ball**: $f(\mathbf{x}_{best}) = 2.00751465$
  - Inexact Line Search with Backtracking
    - $\alpha = 0.5$
    - $\beta = 0.75$
  - $\beta = 0.4$ (momentum)
- **Conjugate Gradient**: $f(\mathbf{x}_{best}) = 0.3321043$
  - Exact Line Search with Golden-Selection Search
    - $Upper\_Bound = 0.5$
  - $\epsilon = 1E-3$
- **Accelerated Gradient Descent**: $f(\mathbf{x}_{best}) = 1.36037288$
  - Inexact Line Search with Backtracking
    - $\alpha = 0.5$
    - $\beta = 0.75$
  - $a_0 = 1.05$ (initial momentum)
- **FISTA**: $f(\mathbf{x}_{best}) = 3.31956292$
  - Inexact Line Search with Backtracking
    - $\alpha = 0.5$
    - $\beta = 0.75$
- **Barzilai-Borwein**: $f(\mathbf{x}_{best}) = 4.58865406$
  - NONE

2.2 Vector Size *n=10*

This set of parameter searches can be reproduced by running *python3 size_n10_search.py*

This set of tests can be reproduced by running *python3 size_n10_tests.py*

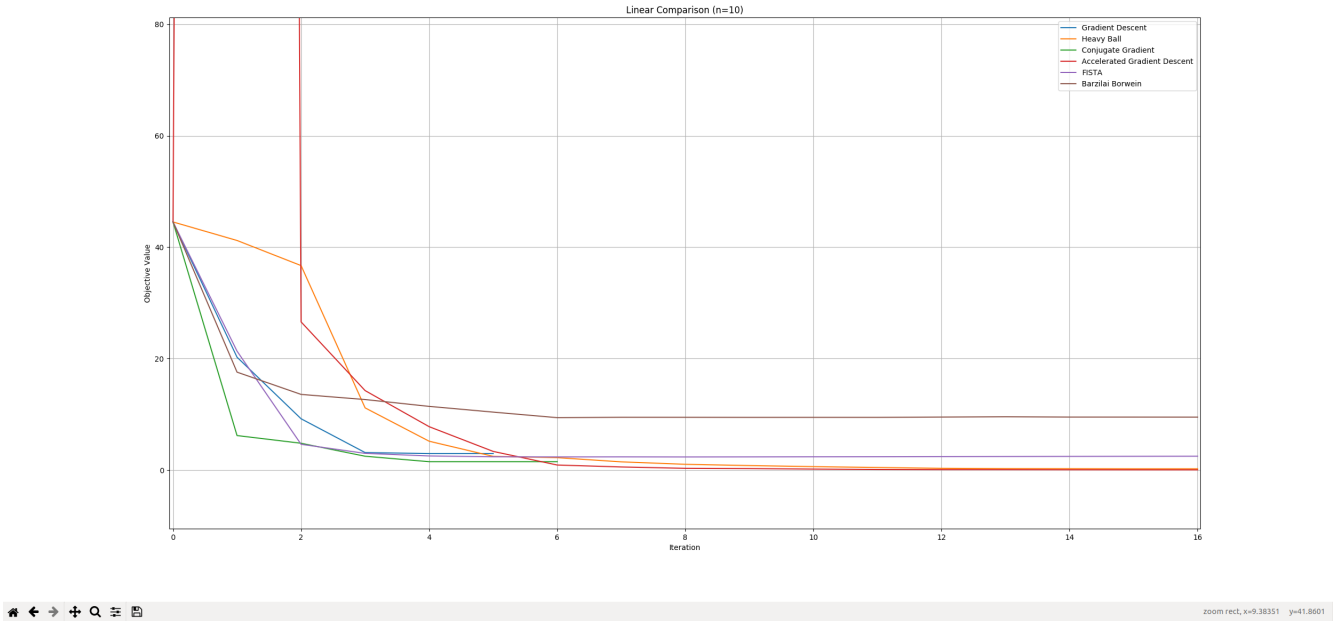
The previous vector size led to the realization that calibrating by hand simply takes too long. Instead, a parameter search was implemented to perform a grid search for each of the algorithms to locate parameters that led to the smallest objective value. For each method, 24 searches took place with the original parameters being the best performing set from the previous vector size for 10 iterations.

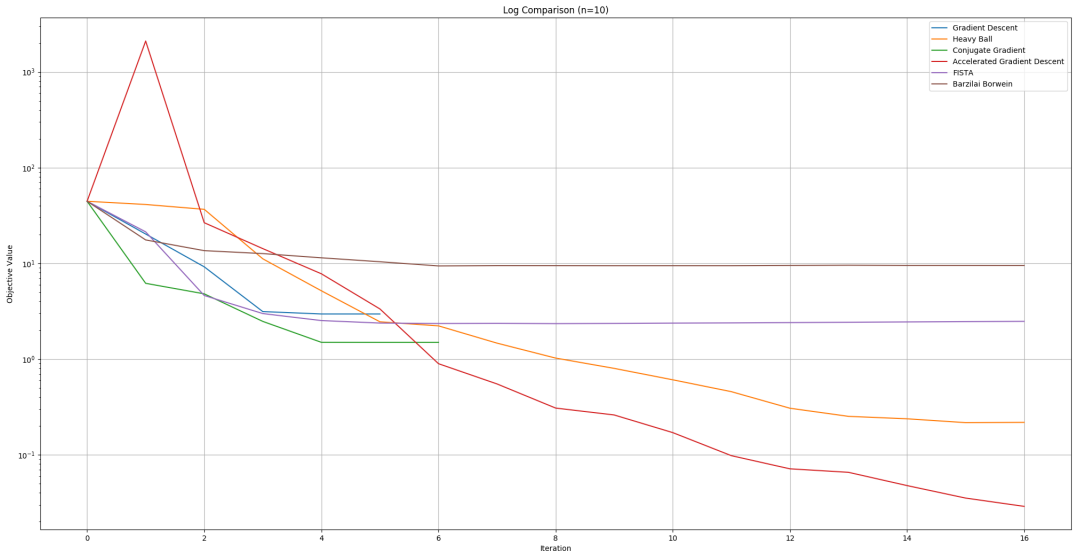
2.2.1 Best Parameters and Objective Values

- **Gradient Descent**: $f(\mathbf{x}_{best}) = 2.95888701$
  - Inexact Line Search with Backtracking
    - $\alpha = 0.270848749$
    - $\beta = 0.874143749$
- **Heavy-Ball**: $f(\mathbf{x}_{best}) = 0.21802041$
  - Inexact Line Search with Backtracking
    - $\alpha = 0.020843749$,
    - $\beta = 0.291387916$
  - $\beta = 0.291679583$ (momentum)

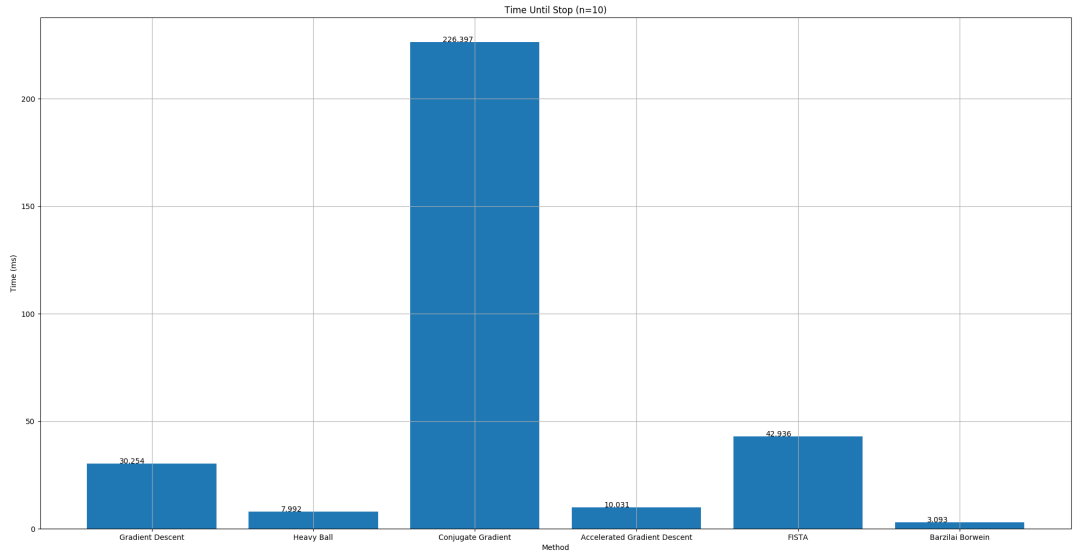
- **Conjugate Gradient**: $f(\mathbf{x}_{best}) = 1.49495188$
  - Exact Line Search with Golden-Selection Search
    - $Upper\_Bound = 0.458347916$
  - $\epsilon = 0.1$
- **Accelerated Gradient Descent**: $f(\mathbf{x}_{best}) = 0.02890335$
  - Inexact Line Search with Backtracking
    - $\alpha = 0.0833449$
    - $\beta = 0.2497625$
  - $a_0 = 0.0416$ (initial momentum)
- **FISTA**: $f(\mathbf{x}_{best}) = 2.4777001$
  - Inexact Line Search with Backtracking
    - $\alpha = 0.250015$
    - $\beta = 0.457889583$
- **Barzilai-Borwein**: $f(\mathbf{x}_{best}) = 9.49481081$
  - NONE

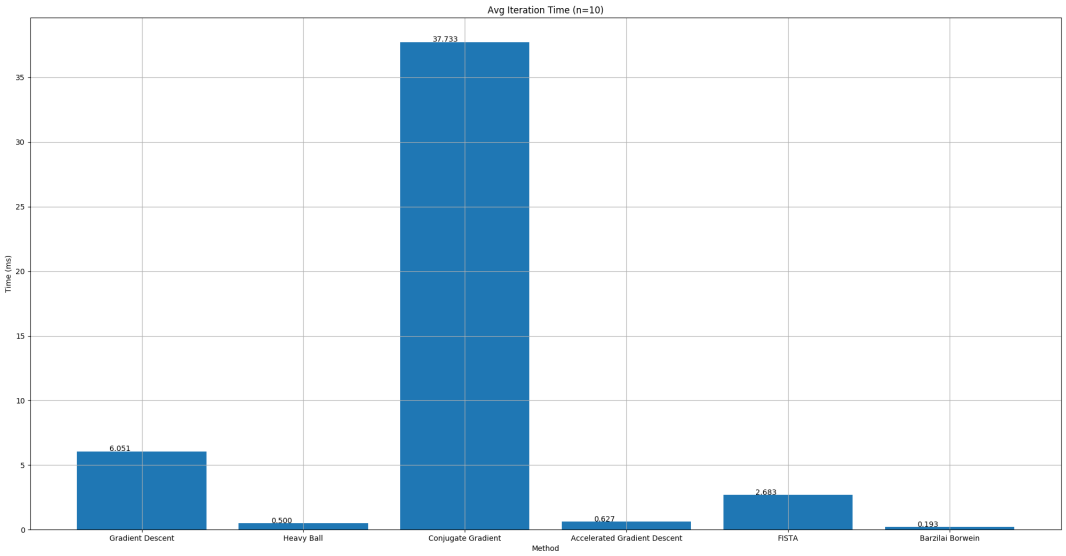
2.2.2 Output Graphs

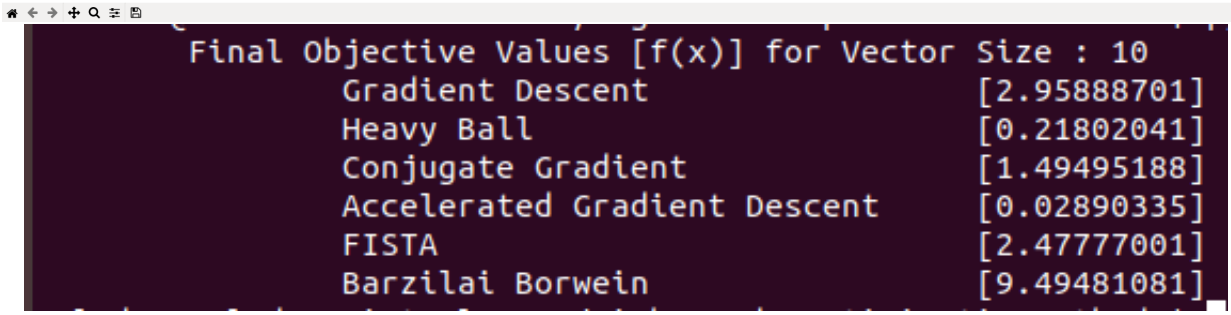



Log Comparison (n=10)
Gradient Descent
Heavy Ball
Conjugate Gradient
Accelerated Gradient Descent
FISTA
Barzilai Borwein
Objective Value
Iteration

x=5.82593 y=35.1791


Time Until Stop (n=10)
226.397
30.254
7.992
10.031
42.936
3.093
Time (ms)
Gradient Descent
Heavy Ball
Conjugate Gradient
Accelerated Gradient Descent
FISTA
Barzilai Borwein
Method





2.2.3 Remarks

The grid search of the parameters consumed nearly to hours but produced significantly better results that initial experiments of calibrating by hand.

Surprisingly, Accelerated Gradient Descent and Heavy Ball managed to beat Conjugate Gradient while also consuming significantly lower amounts of time per iteration. Though it is suspected that this is a result of simply not performing a thorough enough of a grid search for Conjugate Gradient. However, in terms of convergence in number of iterations, Conjugate Gradient still was the winner.

All other methods performed reasonably well as a result of the search, with Barzilai-Borwein once again serving as the baseline, but an incredibly simple to implement and fast to execute baseline.

2.3 Vector Size *n=50*

This set of parameter searches can be reproduced by running *python3 size_n50_search.py*

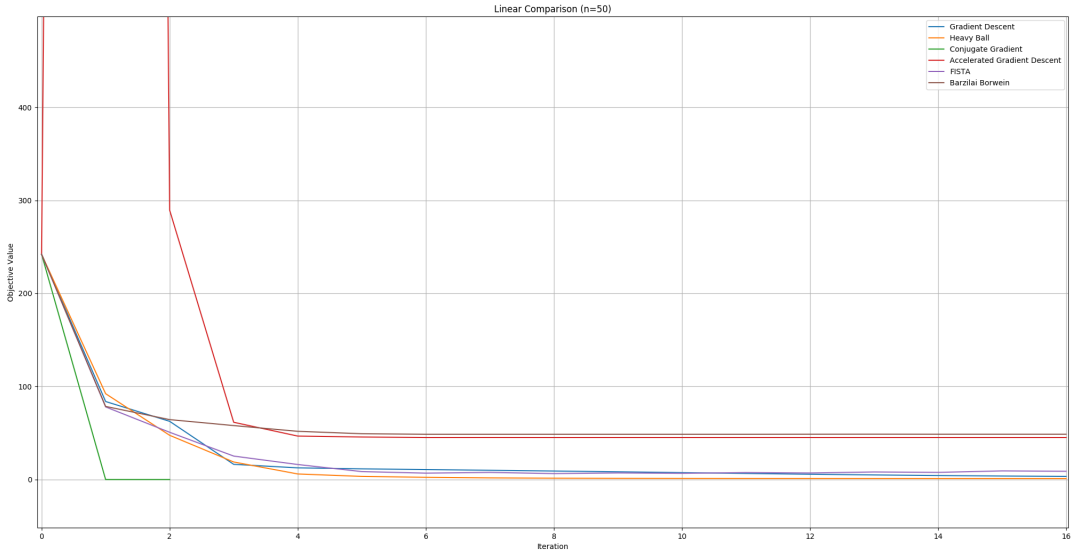
This set of tests can be reproduced by running *python3 size_n50_tests.py*

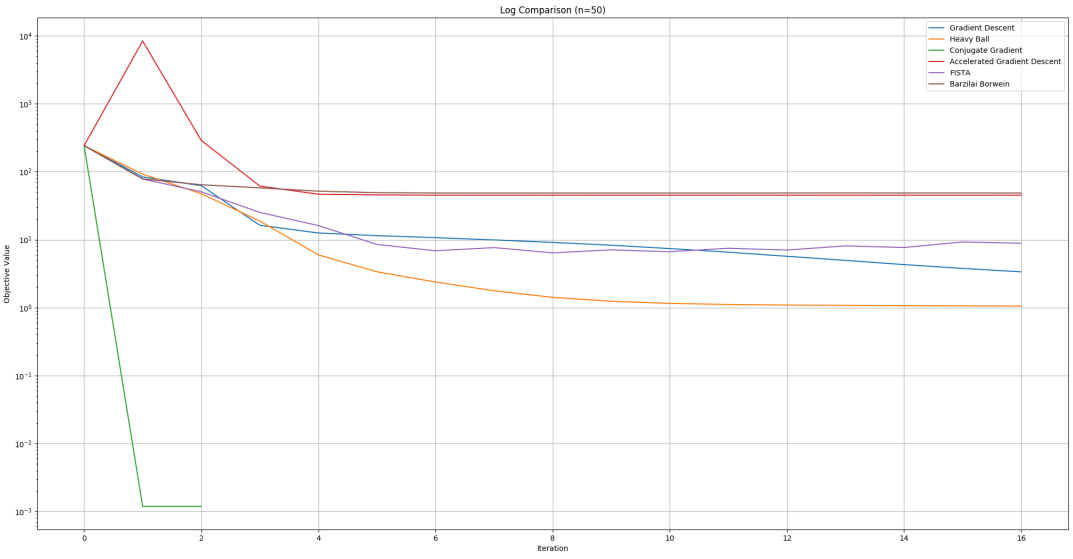
Given the success of the grid search for the last vector size, it was applied again, this time for only 5 searches due to the large increase in computation time from the increased vector size and the deadline for project submission being only an hour away. Again, the algorithms were all applied for 16 iterations on the resulting parameters.

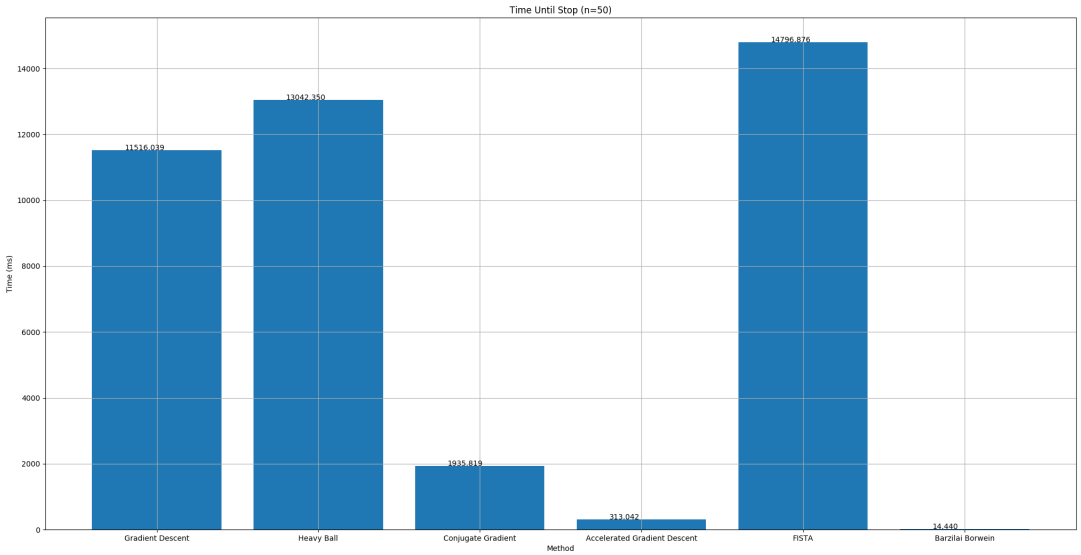
2.3.1 Best Parameters and Objective Values

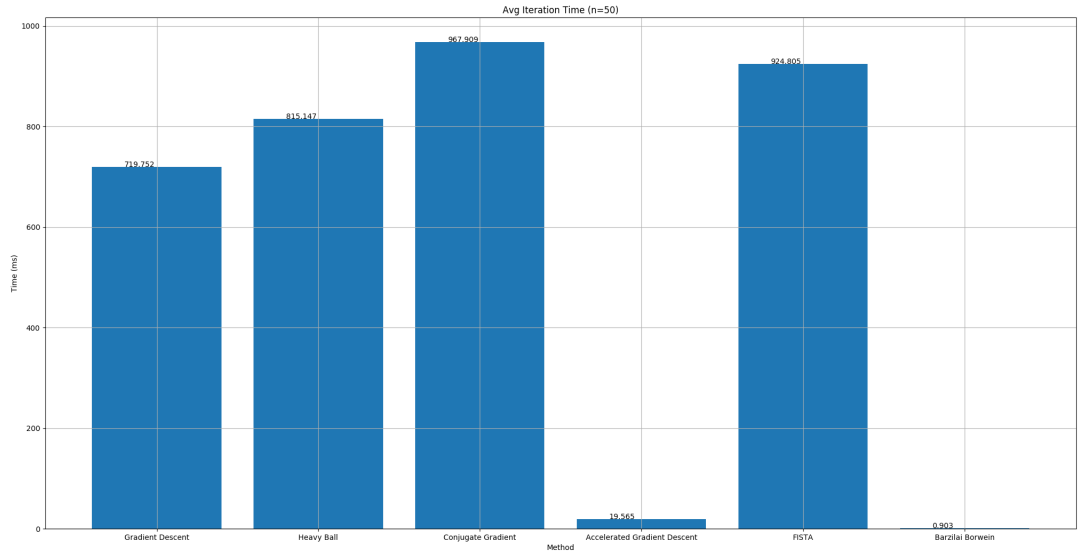
- **Gradient Descent**: $f(\mathbf{x}_{best}) = 3.36475855$
  - Inexact Line Search with Backtracking
    - $\alpha = 0.3500169$
    - $\beta = 0.999$
- **Heavy-Ball**: $f(\mathbf{x}_{best}) = 1.05501313$
  - Inexact Line Search with Backtracking
    - $\alpha = 0.300016$
    - $\beta = 0.999$
  - $\beta = 0.2200012$ (momentum)
- **Conjugate Gradient**: $f(\mathbf{x}_{best}) = 0.00118889$
  - Exact Line Search with Golden-Selection Search
    - $Upper\_Bound = 1.0$
  - $\epsilon = 1E-5$
- **Accelerated Gradient Descent**: $f(\mathbf{x}_{best}) = 45.13836696$
  - Inexact Line Search with Backtracking
    - $\alpha = 0.200014$
    - $\beta = 0.594159$
  - $a_0 = 0.0$ (initial momentum)
- **FISTA**: $f(\mathbf{x}_{best}) = 8.85981559$
  - Inexact Line Search with Backtracking
    - $\alpha = 0.5$
    - $\beta = 0.999$
- **Barzilai-Borwein**: $f(\mathbf{x}_{best}) = 48.68859037$
  - NONE

2.3.2 Output Graphs










```
Final Objective Values [f(x)] for Vector Size : 50
        Gradient Descent                    [3.36475855]
        Heavy Ball                          [1.05501313]
        Conjugate Gradient                  [0.00118889]
        Accelerated Gradient Descent        [45.13836696]
        FISTA                               [8.85981559]
        Barzilai Borwein                    [48.68859037]
```

2.3.3 Remarks

Accelerated Gradient Descent appeared to suffer from the large decrease in the search space of the grid search. However, all the other method performed quite well and several of them even beat their performance for hand-tuned parameters on the vector size of $n = 5$. This goes to show that the grid search was a good choice over tuning parameters by hand.

3 Conclusion

3.1 Parameter Tuning

The most significant insight to be gained from this project was the importance of parameter selection for optimization algorithms. With poor parameter selection (See Subsection 2.1), an algorithm may return a highly-suboptimal solution where it may produce a only slightly-suboptimal solution when provided more appropriate parameters. Yet determining the parameters is non-trivial and intuition can be met with limited success and still leads to significant amounts of time being committed to their selection. In many ways, parameter selection appears to be more of an art than a science.

However, while approaches such as grid search provide a helpful method for searching, but are adversely impacted by an exponential runtime, making the approach not feasible for more than a few values per parameter.

3.2 Accuracy and Convergence of Methods

Throughout the project, Conjugate Gradient remained one of the most accurate methods for estimating the function, despite the size. It also remained the fastest converging methods in terms of number of iterations.

3.3 Runtime of the Methods

Though Conjugate Gradient was one of the fastest converging and most accurate estimates of the optimal value of the objective function, it was plagued by its significant runtime. This seems to be directly controlled by the early stopping value in the inner loop. The lower the value, the higher the accuracy since, but it worse the time per iteration.

While it had the lowest estimate of the function in every scenario, Barzilai-Borwein was one of the simplest algorithms to implement, featured not parameter selection, and had an incredibly low runtime per iteration. So while it may note be the best possible approach for estimating a function, it appears to be a simple, low-cost, and fast method that does not require a tremendous amount of commitment. If speed (per iteration) and simplicity are more important than accuracy, Barzilai-Borwein is a good option. It took less than 15 milliseconds per iteration on all three vector sizes!