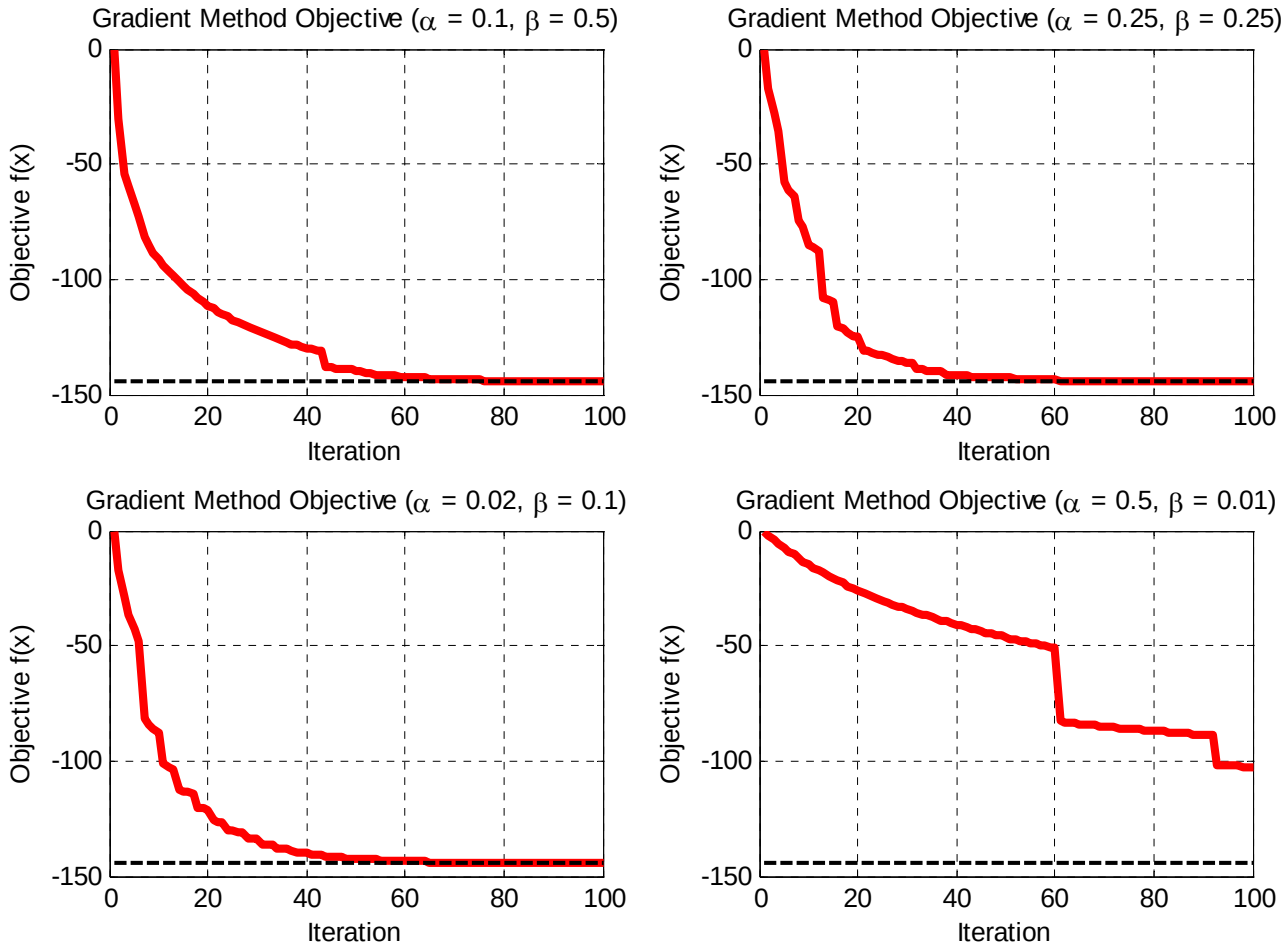
Christopher Tsai
February 28, 2008
EE 364 – Convex Optimization

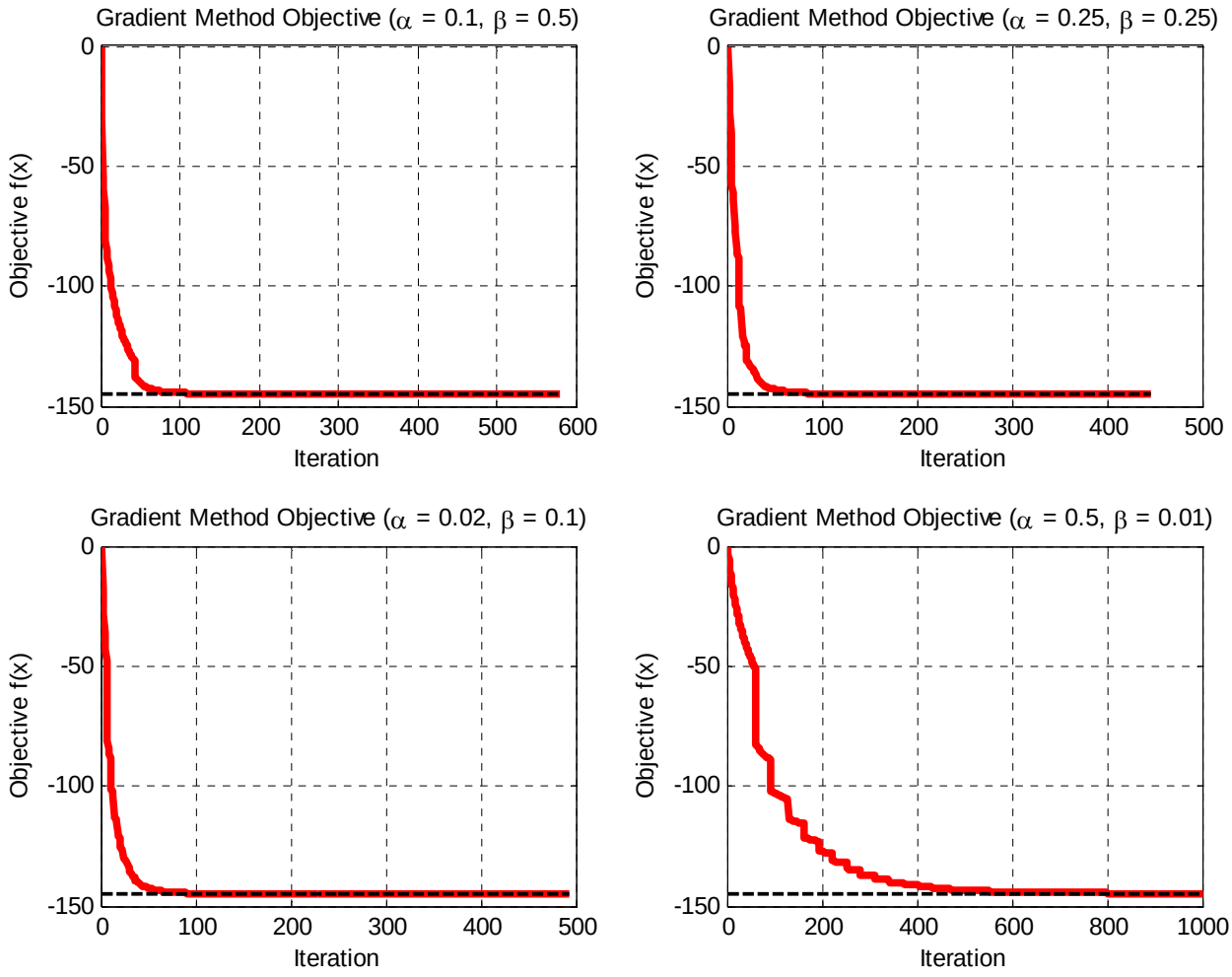
CVX Problem Set VII – Unconstrained Minimization

Problem 9.30 – Gradient and Newton Methods

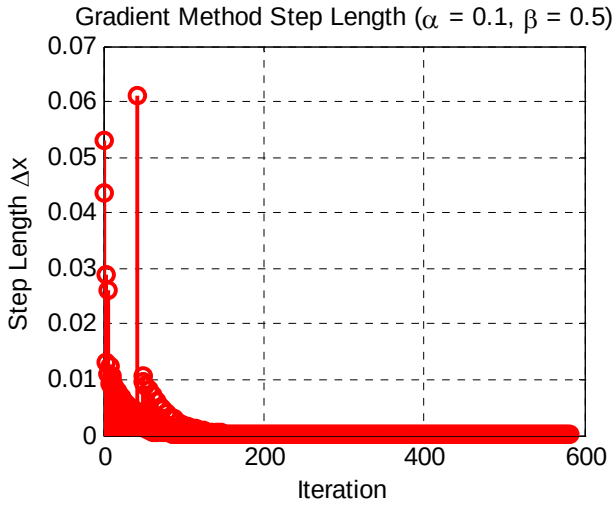


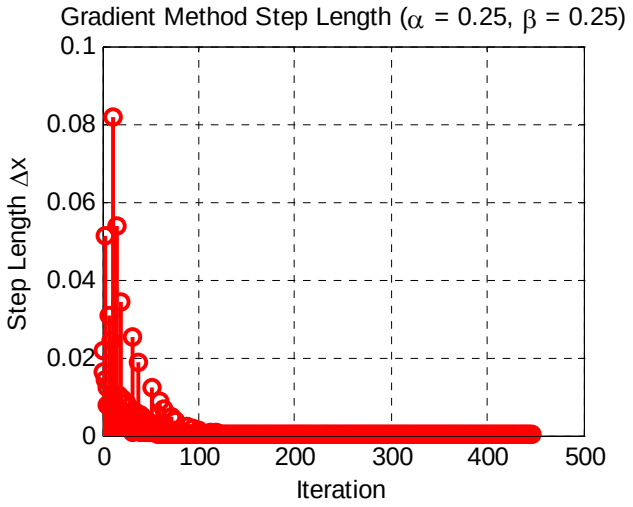
(a.) Although values of α and β on the order of 0.1 – 0.3 yield seemingly adequate convergence, all of our experiments pushed the number of iterations well beyond 100, meaning that the algorithm only ostensibly converges. When we decrease β too far (below 0.1), convergence requires even more iterations. Thus, while convergence remains likely robust at a variety of α values, we must keep β high enough to expedite convergence to the true optimal value of **-144.6979.** Note here that we implemented the gradient descent method with *gradient norm* backtracking line search rather than objective function backtracking.

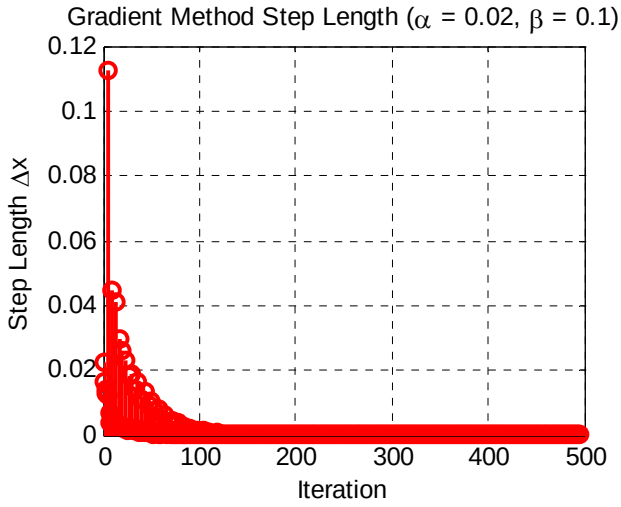
However, if we plumb deeper into the iteration cycle, we find convergence well into the five hundreds. However, the method does verifiably converge:

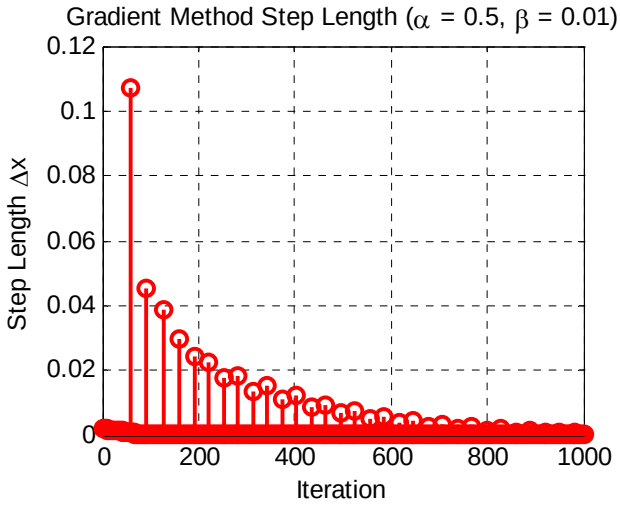


Regrettably, it seems like the gradient method, though convergent, typically requires around 500 iterations to settle on its final value and terminate operation. However, we *could* terminate earlier and settle for a slightly suboptimal value. With our tolerance of 10^{-6}, we find reasonably close optimal values after only 40-50 iterations. The other noteworthy feature of the gradient method is its steadily decreasing step size; initially, while most steps seem constant, the gradual approach toward the true solution means that the steps grow ever shorter, with the final steps shortest. However, the step size could also vary in periodic fashion throughout the algorithm, but, for the most part, the final steps are the smallest.



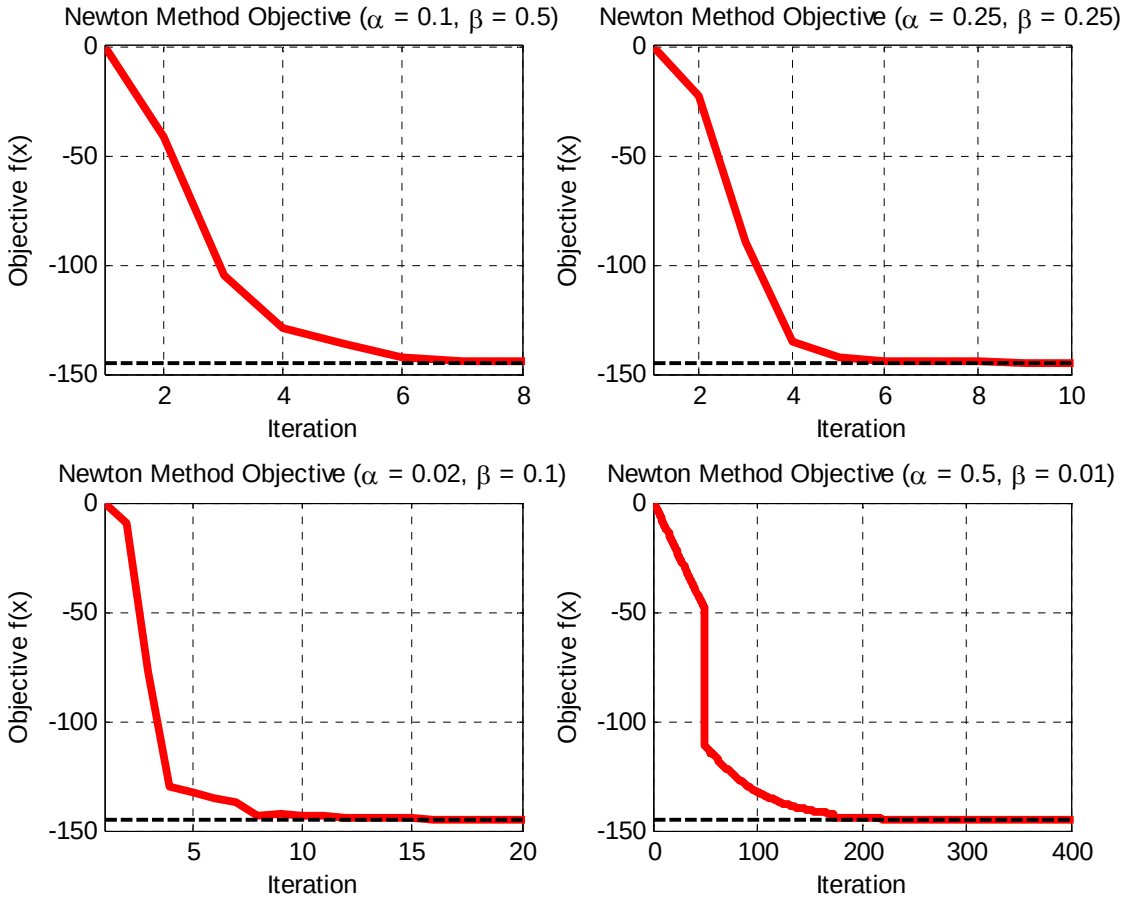


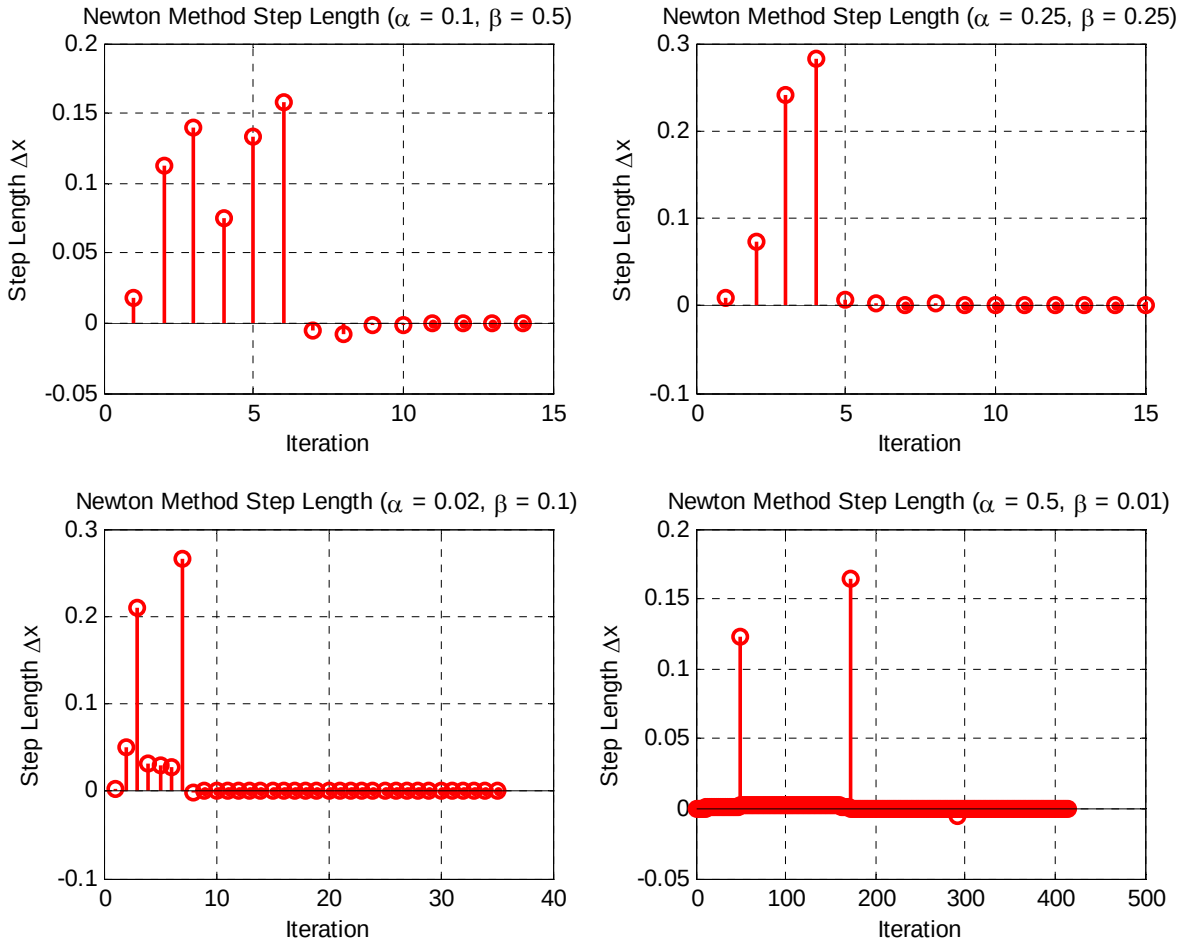




The gradient method incrementally converges to the optimizing value of x with a steadily decreasing step size. Although many of the iterations display zero decrement, the nonzero steps toward the optimizing value decrease inversely with the iteration, as expected.

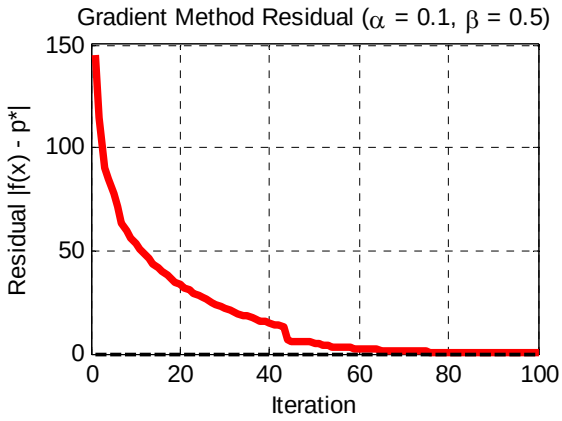
(b.) Newton's Method clearly converges to the proper optimum at an approximately quadratic pace. In fact, the algorithm terminates on its own after 8-10 iterations with the properly chosen parameters $\alpha = 0.1$ and $\beta = 0.5$. Meanwhile, other choices of (α, β) yield similarly satisfactory convergence; the $\alpha = 0.02$, $\beta = 0.1$ curve, in particular, tracks the optimal value almost immediately (after only **six** iterations!), but its precise convergence to within the specified error tolerance (10^{-6}) actually requires more iterations (nearly 20). Meanwhile, the pathologically low $\beta = 0.01$ retards convergence by more than one hundred iterations, but it makes much more progress than the gradient method managed, still beating it by over a hundred iterations!

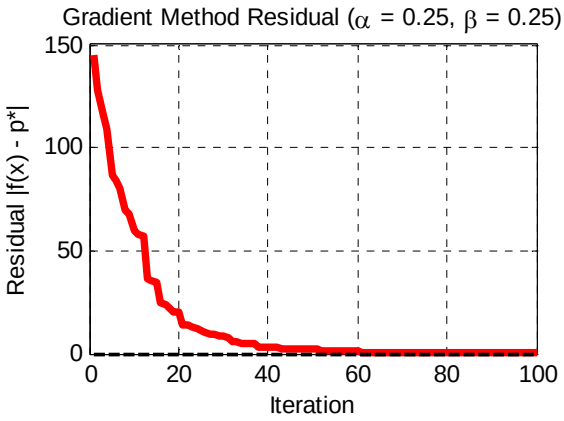


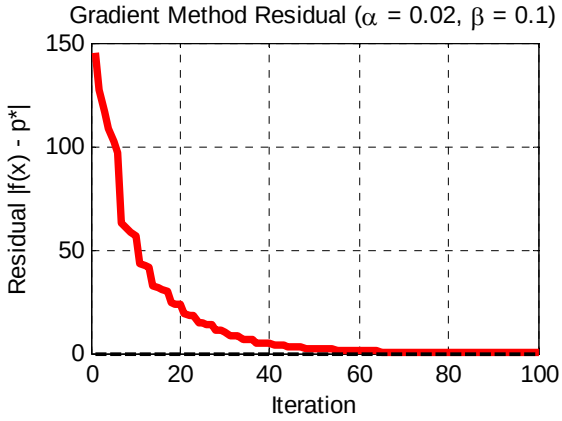


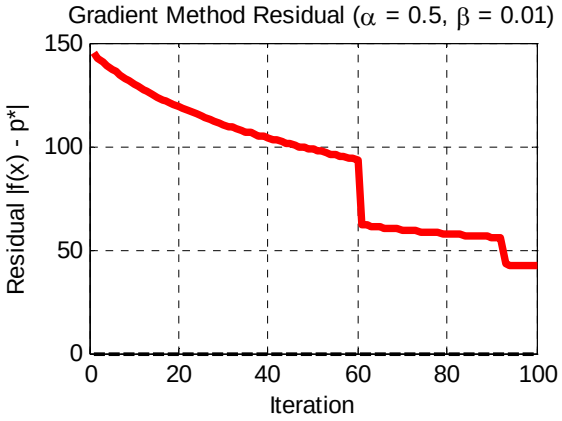
As with the gradient method, the moderate values of α and β yield steady convergence toward the optimizing x, with occasional large (1) steps spaced by smaller steps. However, unlike the gradient method steps, the Newton Method step sizes seem much more haphazard and unpredictable, with the only certainty being their eventual diminishment close to the optimizing value. Other than that, their variation with iteration appears much more random. Overall, compared to the gradient method, the Newton Method manages much smaller residuals much sooner, but the size trend is not monotonically decreasing.

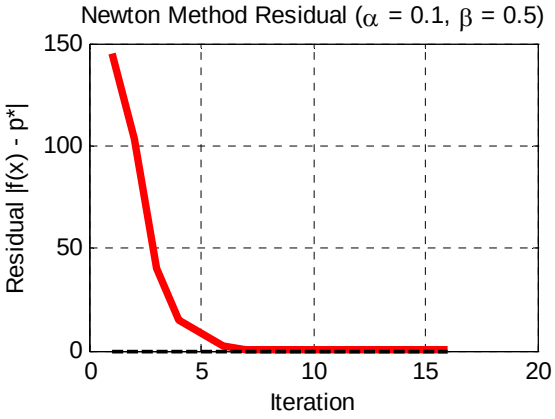
The residual plots on the following page track the difference between the true optimum and the perceived optimum. These curves are simply shifted versions of the objective curves, but they do reveal that the error approaches zero steadily. The more haphazard variations in the Newton Method step size do not appear because the slight fluctuations are much too small for the scale.

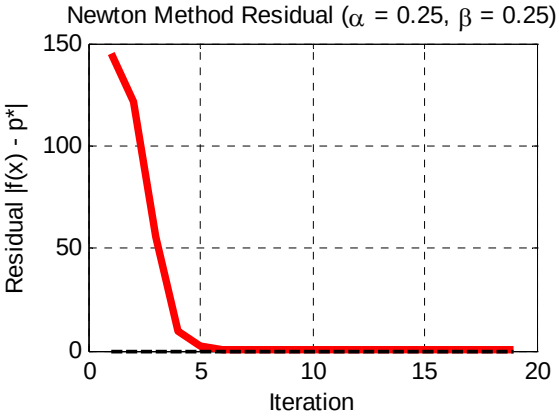

Gradient Method Residual (α = 0.1, β = 0.5)
Residual |f(x) - p*|
Iteration
0
50
100
150
0
20
40
60
80
100

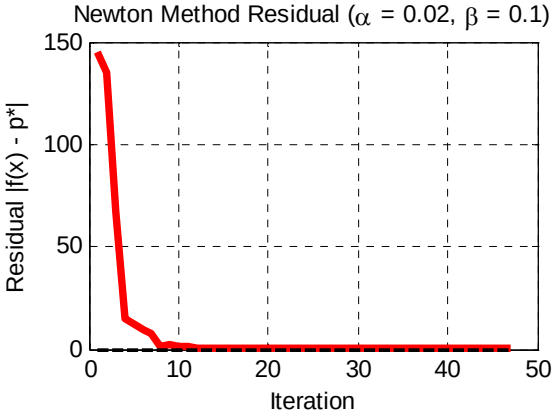

Gradient Method Residual (α = 0.25, β = 0.25)
Residual |f(x) - p*|
Iteration
0
50
100
150
0
20
40
60
80
100

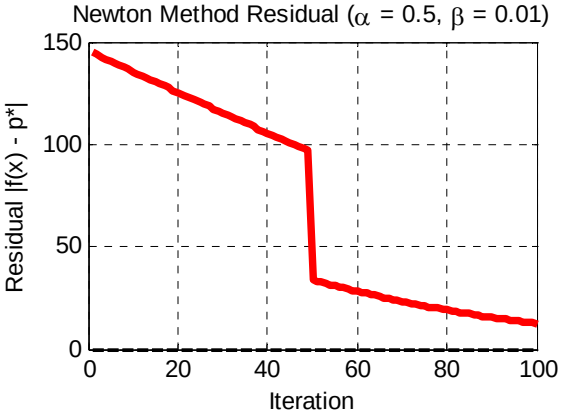

Gradient Method Residual (α = 0.02, β = 0.1)
Residual |f(x) - p*|
Iteration
0
50
100
150
0
20
40
60
80
100


Gradient Method Residual (α = 0.5, β = 0.01)
Residual |f(x) - p*|
Iteration
0
50
100
150
0
20
40
60
80
100


Newton Method Residual (α = 0.1, β = 0.5)
Residual |f(x) - p*|
Iteration
0
50
100
150
0
5
10
15
20

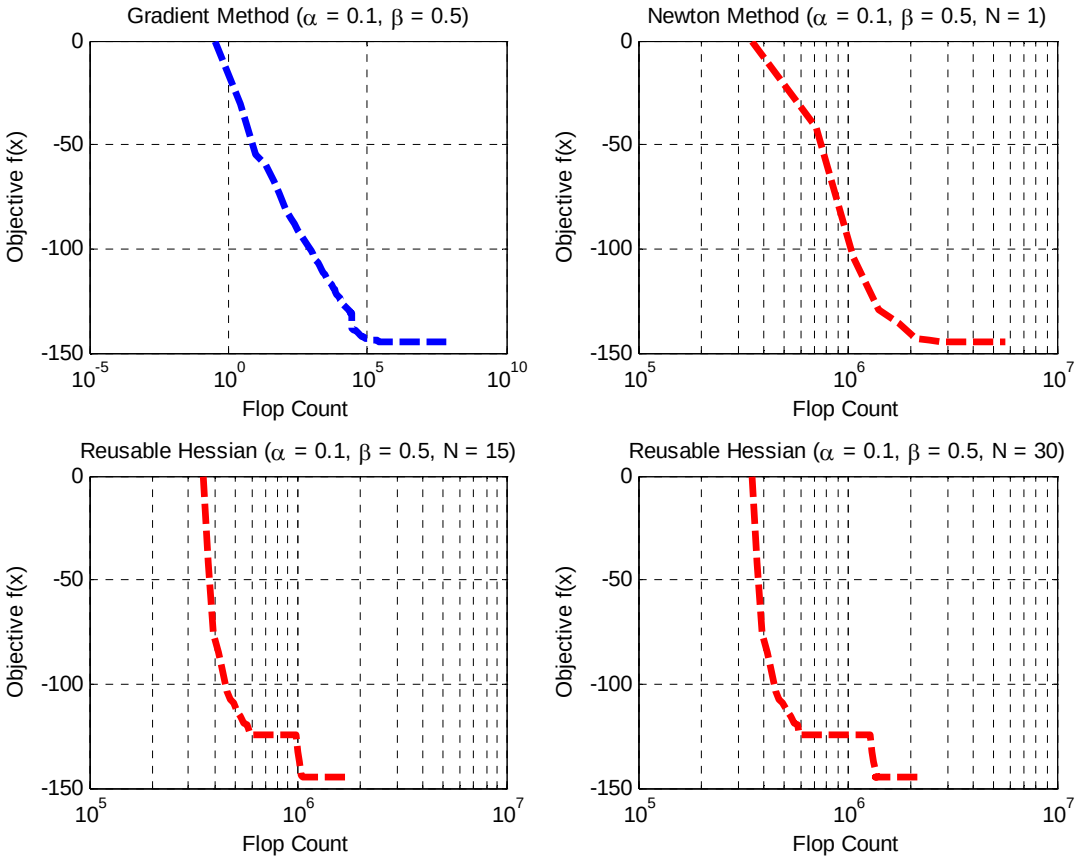

Newton Method Residual (α = 0.25, β = 0.25)
Residual |f(x) - p*|
Iteration
0
50
100
150
0
5
10
15
20


Newton Method Residual (α = 0.02, β = 0.1)
Residual |f(x) - p*|
Iteration
0
50
100
150
0
10
20
30
40
50


Newton Method Residual (α = 0.5, β = 0.01)
Residual |f(x) - p*|
Iteration
0
50
100
150
0
20
40
60
80
100

Problem 9.31 – Some Approximate Newton Methods

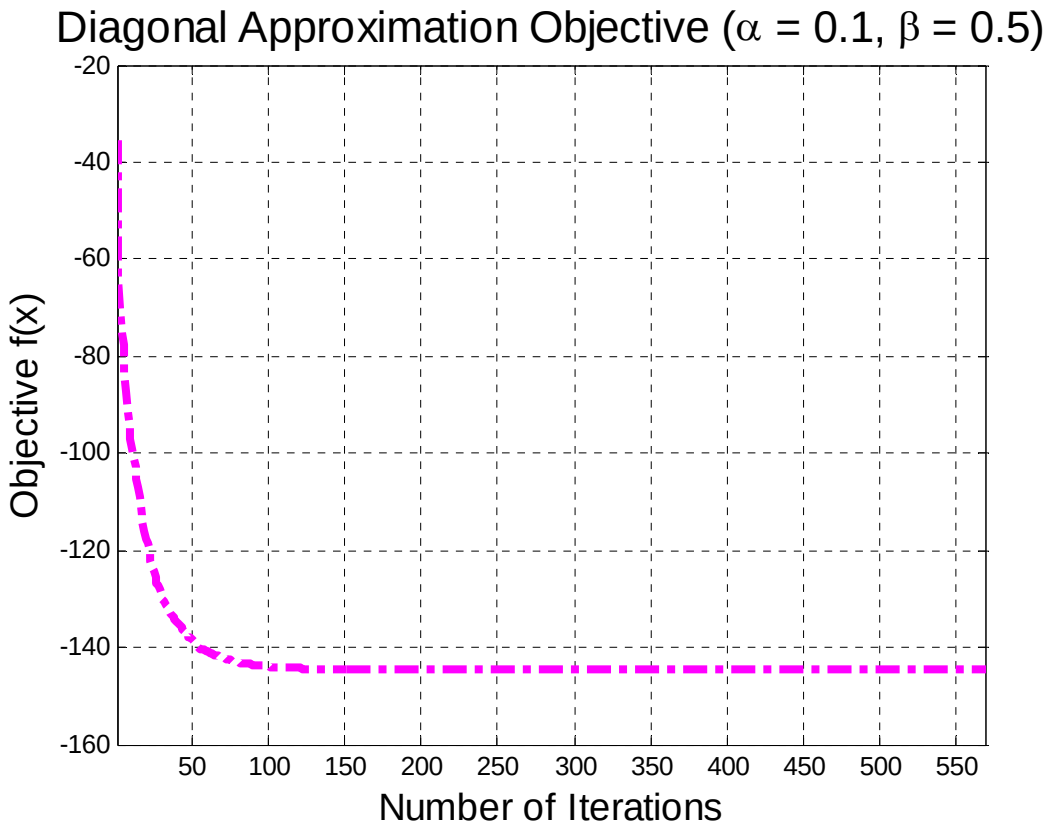
(a.) The cost of Newton's Method is dominated by the cost of evaluating the Hessian and the cost of solving the Newton system. Thus, we can save computation flops by evaluating and factoring the Hessian once only every N iterations, using the search step $\Delta x = -H^{-1} \nabla f(x)$ and avoiding copious unnecessary reevaluation. Results follow:

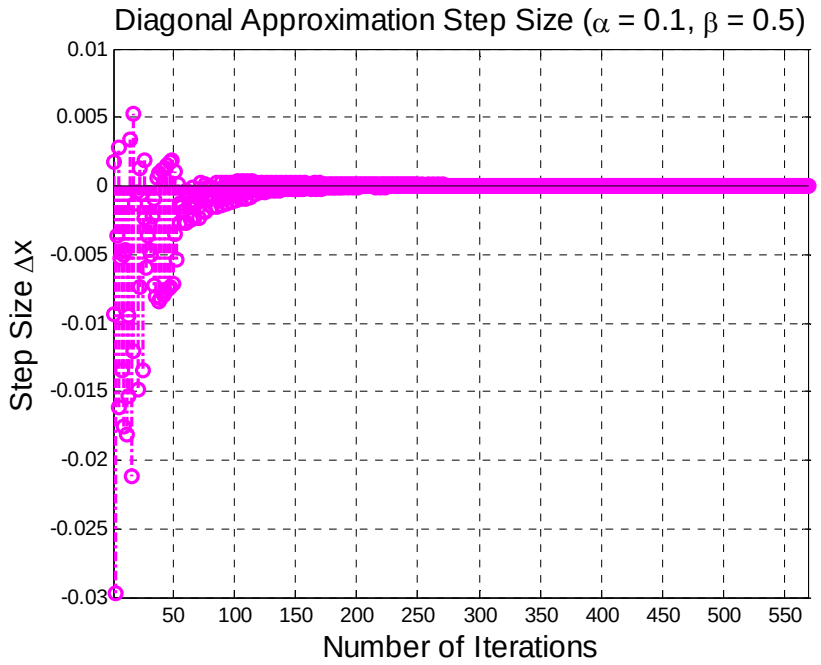


Apparently, Hessian reuse saves an entire order of magnitude of flops. Whereas the N =1 case requires a comparable number of iterations to the gradient method (no surprise, since N = 1 represents the ordinary fully evaluated Newton's Method), the N = 15 and N = 30 cases require far fewer flops. In terms of computation time, only the N = 15 case represents a significant time savings; while the N = 30 algorithm evaluates the Hessian far less frequently, its infrequent update retards the convergence, thereby forcing many more iterations than the N = 15 case to properly converge on the optimal value. The gradient method consumes the most time by far:

Thus, there exists a limit to which we can meaningfully exploit the savings gained by distancing our Hessian reevaluations; if we space the calculations too far apart, then our computation time could suffer. It is important to find the optimal (or near-optimal) value of N to profit from the method maximally!

(b.) The diagonal approximation yields nearly disastrous results, requiring a number of iterations (570) and computation time (10.64 seconds) commensurate with the gradient method. Apparently, by reducing the Hessian matrix to its main diagonal, we discard too much information about the fine variation of the objective and converge much more slowly than we would if we kept the off-diagonal elements. The flop count has decreased considerably, but this number is meaningless on a clock since the number of iterations required has skyrocketed. We conclude that Hessian reuse offers much better computation time improvements.





SUMMARY OF RESULTS

Strategy	Number of Iterations	Number of Flops	CPU Time
Gradient Method	581 iterations	6.5374×10^7 flops	10.704848 seconds
Newton's Method ($N = 1$)	8 iterations	2.8267×10^6 flops	0.284346 second
Hessian Reuse ($N = 15$)	32 iterations	1.6401×10^6 flops	0.113939 second
Hessian Reuse ($N = 30$)	49 iterations	1.6467×10^6 flops	0.109546 second
Diagonal Approximation	570 iterations	1.14×10^5 flops	10.640586 seconds

Additional Problem #1 – Three-Way Linear Classification

Using disciplined convex programming, we solve the following feasibility problem:

$$minimize \;\; 0 \quad subject\ to$$

$$(a_1 - a_2)^T [x^{(1)} \quad \dots \quad x^{(N)}] - (b_1 - b_2) \succcurlyeq 1$$

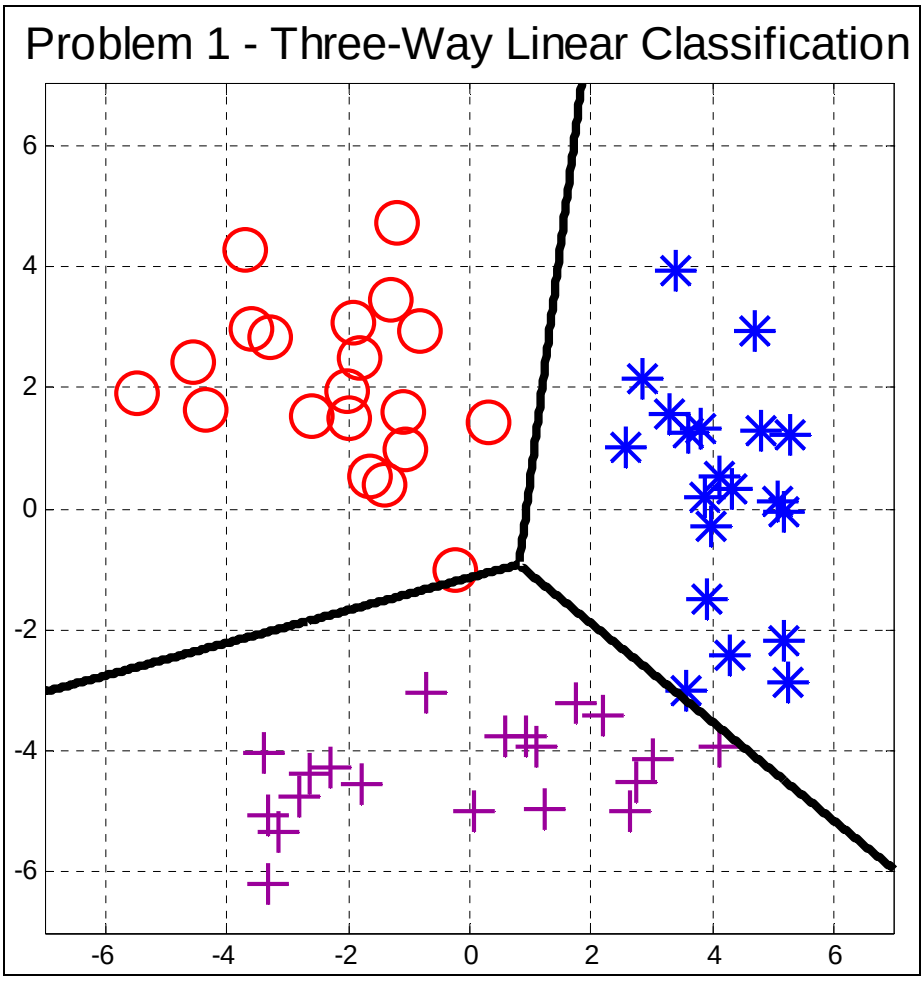
$$(a_1 - a_3)^T [x^{(1)} \quad \dots \quad x^{(N)}] - (b_1 - b_3) \succcurlyeq 1$$

$$(a_2 - a_1)^T [y^{(1)} \quad \dots \quad y^{(M)}] - (b_2 - b_1) \succcurlyeq 1$$

$$(a_2 - a_3)^T [y^{(1)} \quad \dots \quad y^{(M)}] - (b_2 - b_3) \succcurlyeq 1$$

$$(a_3 - a_1)^T [z^{(1)} \quad \dots \quad z^{(P)}] - (b_3 - b_1) \succcurlyeq 1$$

$$(a_3 - a_2)^T [z^{(1)} \quad \dots \quad z^{(P)}] - (b_3 - b_2) \succcurlyeq 1$$



Problem #2 – Efficient Numerical Method for Regularized Least-Squares

$$minimize \ \sum_{i=1}^{k}\left(a_i^T x - b_i\right)^2 + \delta \sum_{i=1}^{n-1}(x_i - x_{i+1})^2 + \eta \sum_{i=1}^{n} x_i^2$$

We implement both the ponderously slow direct method as well as our efficient strategy in code:

```
% DIRECT METHOD:
% --------------

tic;
xDirect = (A.'*A + delta*D + eta*I)\(A.'*b);
toc;

% SOLVING WITH THE DIRECT METHOD. . .
% Elapsed time is 5.234242 seconds.
```

```
% EFFICIENT METHOD:
% -----------------

tic;

% [i.] - Solving for q and Q:
X = triD\[g A.'];
q = X(:,1);
Q = X(:,2:(k+1));

% [ii.] - Forming w and W:
w = A*q;
W = A*Q;

% [iii.] - Solving for z:
z = (eye(k) + W) \ w;

% [iv.] - Forming solution:
xEfficient = q - Q*z;

toc;

% SOLVING WITH THE EFFICIENT METHOD. . .
% Elapsed time is 0.120394 seconds.
```

Clearly, the efficient method prevails, outpacing the direct method by more than an entire order of magnitude: 0.12 sec = 120 milliseconds << 5.234 seconds! We compare the two solutions and ascertain that the maximal difference is less than 0.00000015821%; the solutions are identical.