Final Hand In · Week VIII

CCO · Constraint Continuous Optimization

Wednesday 21st October, 2009

Johan Sejr Brinch Nielsen

Email: zerrez@diku.dk
Cpr.: 260886-2547

Dept. of Computer Science,
University of Copenhagen

Contents

Introduction

This is my final hand in for the course “Constraint Continuous Optimization”. I hope you will enjoy it :-)

Problem 1: Mandatory

I have filled out the Course Evaluation.

Problem 2: Taylor

Taylor Series

A Taylor series is a series expansion of a function $f(x) : \mathbb{R} \rightarrow \mathbb{R}$ in some point a. It is given by:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n$$

Taylor's Theorem

Taylor's Theorem states that any real function f(where we can derive the needed derivatives) can be expanded as ($a = 0$):

$$f(x) = f(0) + xf'(0) + \frac{x^2}{x!}f''(0) + \ldots + \frac{x^{n-1}}{(n-1)!}f^{n-1}(0) + \int_0^x \frac{(x-u)^{n-1}}{(n-1)!}f^{(n)}(u)du$$

Note the remainder denoted by the integral. Taylor series expansion are an important tool in optimization, since it can result in a simpler function (f.x. a quadratic function) which approximates the original objective function.

Problem 3: Line Search Methods

The principle behind line search methods is to minimize the objective function by first computing a descent direction and then an “appropriate” step size to jump. Examples of line search methods are steepest descent and Newton's method.

The Armijo and Wolfe conditions is a set of conditions that restrict the area the optimization method will consider as “good” choices. Line search methods can use the conditions to select a step size after the step direction has been chosen.

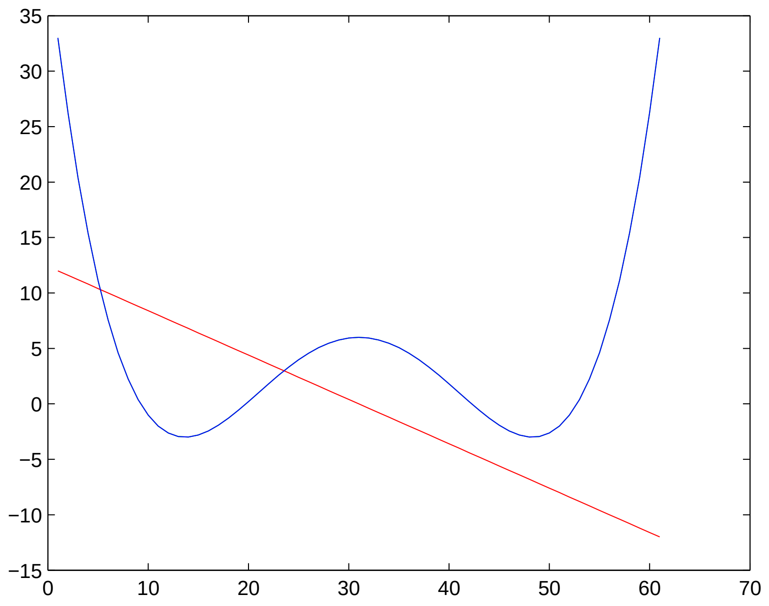
Armijo Conditions

The Armijo conditions are satisfied when:

$$f(x + \alpha p_k) \leq f(x) + c_1 \alpha p^T \nabla f(x)$$

Where x is the current point, p is the descent direction, α is the step size and $0 < c_1 < 1$ is a constant relaxing the condition. The result of this condition

Figure 1: Example of Armijo Back-tracking



Armijo back-tracking visualized as a line (red) going through a 2D objection function (blue). Only values beneath the red line is permitted. The first of the two intersections (to the left) is the current solution point x, while the second denotes the maximum distance reachable under the conditions.

is that the taken step should lower the objective value at least as much as what we would have gotten following a straight line in the same direction. c_1 relaxes the gradient of this direction.

Wolfe Conditions

The Wolfe conditions are satisfied when the Armijo conditions are satisfied and:

$$p^T \nabla f(x + \alpha p) \geq c_2 p^T \nabla f(x)$$

Where x is the current point, p is the search direction, α is the step size and $0 < c_1 < c_2 < 1$ is a constant relaxing the condition.

The condition states that the slope of the function in the chosen point must at least that of the starting point.

Armijo Back-tracking

The projected Armijo back-tracking will, given a search direction p, find a step size α for which $f(x + \alpha p)$ satisfies the Armijo conditions.

The algorithm starts with a large step length (e.g. 1). The step length is then lowered until a point that meets the condition is found.

If the chosen search direction is a descent direction, such a point must necessarily exist (the condition is satisfied when $\alpha = 0$). However, it may be very close to the initial x. If so, the step taken can be very short, nearly nothing at all. It is important to ensure a minimum step size to avoid repeating computations in x.

The algorithm can be described with the following pseudo code:

```
while not (Armijo-condition of (x+a*c1*pk)) do
  a = a * c
end
```

Where $0 < c < 1$.

Problem 4: Trust Region Methods

Trust region methods works by first deciding on a step size and then compute the a decent direction within the step size (the "region").

The idea is to use a model function m that approximates the objective function f. Knowing the step size λ, the goal is to find the best search direction in the model function (which is easy to minimize) and then follow this direction in f. Of course one needs to verify that the model is in fact precise enough to yield a decent in f. If not, the λ is lowered. λ is a measure for the precision of the model function. The region containing points with a distance lower than λ away from x_k is called the "trust" region.

Levenberg-Marquardt

The Levenberg-Marquardt uses an interpolation between Gauss-Newton and steepest descent. Intuitively, it tries the direction of Gauss-Newton and then falls back to steepest descent when unsuccessful.

The Levenberg-Marquardt update value is defined as:

$$\Delta = -(J^T J - \lambda I)^{-1} J^T r$$

Where J is the jacobian matrix, λ is the step modifier, r is the residual vector and Δ is the next modification of the approximation. When the step modifier λ gets large, it will dominate the inverted matrix, such that:

$$\Delta \approx -(-\lambda I)^{-1} J^T r =$$

$$-\frac{1}{\lambda} I J^T r$$

It is now clear why λ becomes a step modifier and how Levenberg-Marquardt graduately switches to steepest descent. λ determines whether $J^T J$ or $-\lambda I$ should dominate the direction. The method adapts both step size and direction during the optimization, however the step size is always lowered. Hence, it makes sense to classify this method as a trust region method (bound by the starting λ value).

Dogleg

The Dog-Leg method is based on a combination of the Cauchy point and a Quasi-Newton method. It uses a second degree Taylor expansion as its model. It will choose its search direction based on the following three cases:

1. When the point suggested by Newton's method is within the trust region, this point is selected.
2. If this is not the case and the trust region requests a smaller step than that of the Cauchy point, the direction towards this point is followed as far as the region permits.
3. In the third case, when the region boundary is between the Cauchy point and the Newton point, the intersection between the limiting boundary and the line between these two points are chosen.

As can be seen, the possible directions is given by two joined lines (from p_k to the Cauchy point and further to Newton's point). This gives a bending line (a "dog leg"). Note that since the model function is quadratic, the Newton method will minimize it in a single iteration. So there is never a need to go further than the point found by Newton's method.

Specifically, p_k may take one of the following forms:

1. $$p_{k+1} = p^B$$
2. $$p_{k+1} = \frac{\Delta}{\mid p^U \mid} p^U$$
3. $$p_{k+1} = p^U + \frac{\Delta - \mid p^U \mid}{\mid p^B \mid + 2p^{U\prime} p^B} p^B$$

Where Δ is the size of the trust region, p^B is the point suggested by Newton's method and p^U is the Cauchy point. The two latter points are defined as:

$$p^B = -B^{-1} \cdot g$$

$$p^C = \frac{g'g}{g'Bg} \cdot g$$

Where g is the gradient in the point x_k, B is an approximation of the Hessian and B^{-1} is an approximation of its inverse; both B and B^{-1} are positive definite matrices.

There is one detail in the algorithm yet to be discussed. That is what size to choose for λ and how to adjust this dynamically. Generally, one should lower λ when no decent direction is found and only raise λ when the step taken is as long as λ (the step is restricted by λ). One way to achieve this is by comparing the actual improvement in the objective function with that of the model (the expected improvement).

Problem 5: First Order Optimality Conditions

The Karush-Kuhn-Tucker (KKT) conditions are a set of *necessary* conditions that must be satisfied before x can be a minimum of the function f. If the KKT conditions are not satisfied, one can be sure that x is not a minimum. However, whenever they are satisfied, one only knows that x *might* be a minimum.

Consider the minimization problem:

$$\begin{array}{rl} \min_x & f(x) \\ \text{subject to:} & c(x) \geq 0 \end{array}$$

Where $x \in R^n$, c are the constraints and $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$.

I will now derive the KKS conditions for this minimization problem. Let us first examine the first-order Taylor expansion of the objective and constraint function. To maintain feasibility we require that $c(x+d) = 0$, where d is the direction. Hence,

$$0 \leq c(x+d) \approx c(x) + \nabla c(x)^T d = c(x)^T d$$

The last step is due to $c(x) = 0$. When this condition is satisfied, we maintain feasibility after the step in direction d.

To ensure an improvement in the objective function, we seek to satisfy the condition:

$$0 > f(x+d) - f(x) \approx \nabla f(x)^T d$$

That is, the objective value should drop when moving in the direction of d (a descent direction). This is called an improvement to first order.

Consider the direction d pointing towards a constraint boundary. In such a case, where the optimal solution is bounded by one or more constraints, the gradient of f and that of c must be parrallel. Hence,

$$\nabla f(x) = \lambda \nabla c(x),\ \lambda > 0$$

When the descent direction is not restricted by any constraints, normal optimization techniques can be used. Note however, that the minimzer may lie far from any constrainting boundary that d is trying to cross.

Considor now the Lagrangian function:

$$\Lambda(x, \lambda) = f(x) - \lambda c(x)$$

And its derivate:

$$\nabla_x \Lambda(x, \lambda) = \nabla f(x) - \lambda \nabla c(x)$$

It is clear that the condition of ∇f can be written as:

$$\nabla f(x) = \lambda \nabla c(x) \Rightarrow \nabla_x \lambda(x, \lambda) = 0$$

This is interesting, since it shows a connection between stationary points in $\Lambda(x, \lambda)$ and possible minimizers in f.

A necessary condition for a minimizer $x^\star$ of f is then:

$$\nabla_x \Lambda(x^\star, \lambda^\star) = 0, \; \lambda^\star \geq 0$$

Furthermore, we know that the constraints should be fulfilled:

$$c(x^\star) \geq 0$$

However, since this was derived from *active* constraints (those that limit the search direction), we also require:

$$\lambda^\star c(x^\star) = 0$$

This condition ensures either $\lambda^\star$ or $c(x^\star)$ to be zero. The idea is, that whenever $c(x^\star) > 0$ (we are not near a constraint boundary) then $\lambda^\star = 0$ and the stationary point condition reduces to $\nabla f(x^\star) = 0$. On the other hand, having $c(x^\star) = 0$ allows $\lambda^\star$ to take a negative value.

The Karush-Kuhn-Tucker conditions describe these conditions using sets E and I, where E contains all active constraints and I contains all inactive constraints:

$$\begin{aligned} \nabla \Lambda(x^\star, \lambda^\star) &= 0, \\ c_i(x^\star) &= 0, \quad \text{for all } i \in E, \\ c_i(x^\star) &\geq 0, \quad \text{for all } i \in I, \\ \lambda_i^\star &\geq 0, \quad \text{for all } i \in I, \\ \lambda_i^\star c_i(x^\star) &= 0, \quad \text{for all } i \in E \cup I \end{aligned}$$

Problem 6: Calculus of Variation

Given the functional:

$$S = \int_a^b \Lambda(x, f, f')\, dx$$

The Euler-Lagrange equation becomes:

$$-\frac{d}{dx}\frac{\rho L}{\rho f'} + \frac{\rho L}{\rho f} = 0$$

Problem 7: Linear Complementary Problem

A linear complimentary problem (LCP) is a specific category of problems, which take the form:

$$\begin{aligned} A\lambda + b &\geq 0 \\ \lambda &\geq 0 \\ \lambda^T(A\lambda + b) &= 0 \end{aligned}$$

Where A is a given matrix, b is a given vector and λ is the solution vector that is to be found.

Connection to Constrained Optimization

Any LCP can be reformulated as a quadratic problem (QP). The QP corresponding to the LCP in the previous section becomes:

$$\begin{array}{rrl} \min_\lambda & \lambda^T(A\lambda + b) & \\ \text{such that:} & A\lambda + b & \geq 0 \\ & \lambda & \geq 0 \end{array}$$

Since $z \geq 0$ and $Mz + q \geq 0$, the product $z^T(Mz + q)$ must be positive. Hence, the minimum must be 0. The inequality constraints are still maintained, so the optimal solution will be equal to that of the original LPC.

Algebraic Transformation from LCP to QP

In this section I will convert the LCP solved in this weeks assignment to an QP. Let's start with the original LCP:

$$\begin{array}{rl} A\lambda + b & \geq 0 \\ \lambda & \geq 0 \\ \lambda^T(A\lambda + b) & = 0 \end{array}$$

Letting $h(\lambda) = A\lambda + b$, $x^\star = \lambda$ and $y^\star = h(\lambda)$ we obtain the following reformulation of the conditions:

$$\begin{array}{rl} h(\lambda) & \geq 0 \\ \lambda & \geq 0 \\ \lambda^T h(\lambda) & = 0 \\ (x^\star)^T h(\lambda) & = 0 \end{array}$$

The new conditions (last three) being repetitions of the original conditions (first three). Since $\min(\lambda, y^\star = A\lambda + b) = 0$ (see Splitting Method, page ix) we can add the constraint:

$$(y^\star)^T\lambda \quad = 0$$

Furthermore, since $x = \lambda$ and $y^\star = h(x)$ the following constraint is valid:

$$h(\lambda) + \nabla h(\lambda)^T\lambda - \nabla h(\lambda)^T\lambda - h(\lambda) = 0 \Rightarrow$$

$$h(\lambda) + \nabla h(\lambda)^T\lambda - \nabla h(\lambda)^T x^\star - y^\star = 0$$

The full set of conditions is now:

$$\begin{array}{rl} h(\lambda) & \geq 0 \\ \lambda & \geq 0 \\ \lambda^T h(\lambda) & = 0 \\ (x^\star)^T h(\lambda) & = 0 \\ (y^\star)^T\lambda & = 0 \\ h(\lambda) + \nabla h(\lambda)^T\lambda - \nabla h(\lambda)^T x^\star - y^\star & = 0 \end{array}$$

Which are the first order optimality conditions for the QP

$$\begin{array}{rrl} \min_\lambda & \lambda^T h(\lambda) & \\ \text{such that:} & h(\lambda) & \geq 0 \\ & \lambda & \geq 0 \end{array}$$

This transformation to QP shows that it is possible to solve LCP's using QP methods. An alternative way of showing this is to derive the KKT conditions from a QP and show that these reduce to an LCP.

Problem 8: Splitting Method

The Splitting method solves a LCP by first splitting the matrix A in two. Hereafter, it defines a series of sub-problems which also belongs to the LCP class. However, hopefully these are easy to solve.

The first step is to split A into to new matrices M and N such that:

$$A = M - N$$

The sub-problem is now defined as:

$$\begin{array}{rl} M\lambda^{k+1} + c^k & \geq 0 \\ \lambda^{k+1} & \geq 0 \\ (\lambda^{k+1})^T(M\lambda^{k+1} + c^k) & = 0 \end{array}$$

Where $c^k = b - N\lambda^k$. Letting $M\lambda^{k+1}$ be $M\lambda^k$ makes the left-hand side of the first inequality become:

$$\begin{aligned} M\lambda^k + c^k &= M\lambda^k + b - N\lambda^k \\ &= A\lambda^k + b \end{aligned}$$

And the sub-problem is back to the original LCP. Hence, the only difference lies in using λ^{k+1} together with M and λ^k together with N. Since λ^k is the approximation of $\lambda^\star$ at iteration k, and λ^{k+1} that at iteration $k+1$, this is a fixed point computation yielding a solution to the original LPC.

Not surprisingly, this fixed point iteration can be solved iteratively. A bit more surprising is, that we need to apply the minimum map reformulation. This reformulation yields the following root-search problem:

$$\min(\lambda^{k+1}, M\lambda^{k+1} + c^k) = 0$$

This formulation ensures the inequality constraints are fulfilled, since neither can be negative (the smallest of the two must be 0). This is equivalent to the equality constraint, that ensures the product of the two is 0. Hence, the new formulation is the same as the sub-problem.

We can now work on the new formulation:

$$\begin{array}{llll} \min & (\lambda^{k+1}, M\lambda^{k+1} + c^k) & = 0 & \Rightarrow \\ \min & (0, M\lambda^{k+1} + c^k - \lambda^{k+1}) & = -\lambda^{k+1} & \Rightarrow \\ \max & (0, -M\lambda^{k+1} - c^k + \lambda^{k+1}) & = \lambda^{k+1} & \end{array}$$

The result is a fixed point problem (it contains λ^{k+1} on both sides). However, there is one more trick up the sleeve: case analysis on the sign of the ith element of $S = -M\lambda^{k+1} - c^k + \lambda^{k+1}$:

Case 1: $S_i = (-M\lambda^{k+1} - c^k + \lambda^{k+1})_i < 0$
If this is the case $\lambda_i^{k+1} = 0$, since $0 > S_i$.

Case 2: $S_i = (-M\lambda^{k+1} - c^k + \lambda^{k+1})_i \geq 0$
Clearly $S_i = \lambda_i^{k+1}$, since $S_i > 0$. That is:

$$(-M\lambda^{k+1} - c^k + \lambda^{k+1})_i = \lambda_i^{k+1} \Rightarrow$$

$$(-M\lambda^{k+1} - c^k)_i = 0 \Rightarrow$$

$$(-M\lambda^{k+1})_i = c_i^k$$

This situation is interesting. λ_i^{k+1} is now present on the left hand side only. Assuming that M is reversible, we can compute λ^{k+1} as:

$$\lambda_i^{k+1} = (-M^{-1}c^k)_i = (-M^{-1}(b - N\lambda^k))_i$$

Combining the two cases yields the following closed term for λ^{k+1}:

$$\lambda^{k+1} = \max(0, M^{-1}N\lambda^k - b)$$

In order to solve the LCP, one can approximate on $\lambda^\star$ using above equation until an acceptable error has been reached.

Problem 9: Reformulation

This problem reminds me of the tasks in Inverse Kinematics. Here we had a function $F : \mathbb{R}^n \to \mathbb{R}^m$, transforming n angles into a point of m dimensions. The goal was to minimize the distance to a specifif m-dimensional point e. A function $f : \mathbb{R}^n \to \mathbb{R}$ was defined by:

$$f(\theta) = (F(\theta) - e) \cdot (F(\theta) - e)$$

That is, f is a measure for the distance from the point given by the angles in θ to the wanted end point. The minization problem can now be formulated as:

$$\min_\theta \quad f(\theta)$$

I believe this is an example of the transformation wanted in this task.

Problem 11: Problem Modeling

In this section, I will go through the problems solved during this course and how they were modelled in the world of optimization.

The first framework we used was designed to solve Inverse Kinematics problems. In this case, the problem of finding n angles that would make an arm reach a chosen point. An $F : \mathbb{R}^n \to \mathbb{R}^m$ function from angles to coordinates was given. The goal was now to find θ that would solve the non-linear equation $F(\theta) - e = 0$. Using this the objective function $f : \mathbb{R}^n \to \mathbb{R}$

was defined as the distance from the point reached through the found angles to the chosen end point e. The function f could now be optimized using unconstained minimization methods.

To make things more interesting, a few constraints was added to the problem. The angles in θ was limited to a fixed interval, hence adding inequality constaints of the form $l \leq \theta \leq u$. The model would have to ensure, that the found solution was inside this boundary. Fortunately, this was quite simple, due to the box-property of the inequality (the constraints form a "box" around the feasible region). I experimented with allowing some of the angles to be loose, by not enforcing the constraints on these, and with tightening the permitted range bit by bit. This was the inspiration for some very interesting animations.

Hereafter, we moved on to physics simulation of bouncing balls. This was modelled mathematically with respect to the physical rules that were present in the simulation. The result was a linear complimentarity problem, that seeked to minimize the physical energy present in the system. Constraints were added whenever two balls collided and removed as soon as they were distanced again. Combined with rules of acceleration, this formed the basis for the simulation.

Lately, we have looked at Calculus of Variations, where the functions plays the role part instead of n-dimensional vectors. An example of such an optimization problem is to find the function, who's function-line between points a and b travel the shortest distance. The optimal solution is of course a straight line ($f(x) = ax + b$ in 2-dimensions), but it is a simple example in a series of complex optimization problems.

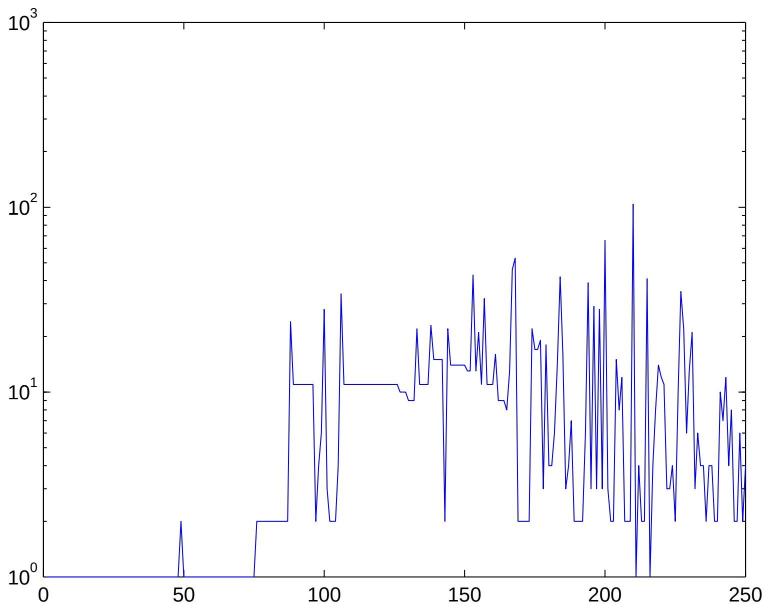
Problem 12: Comparison of Splittings

Sorry about no units! It is number of iterations (log) by iteration number. The plots and discussion are on the last page (no time to fix it) . . .

Notes on Absense

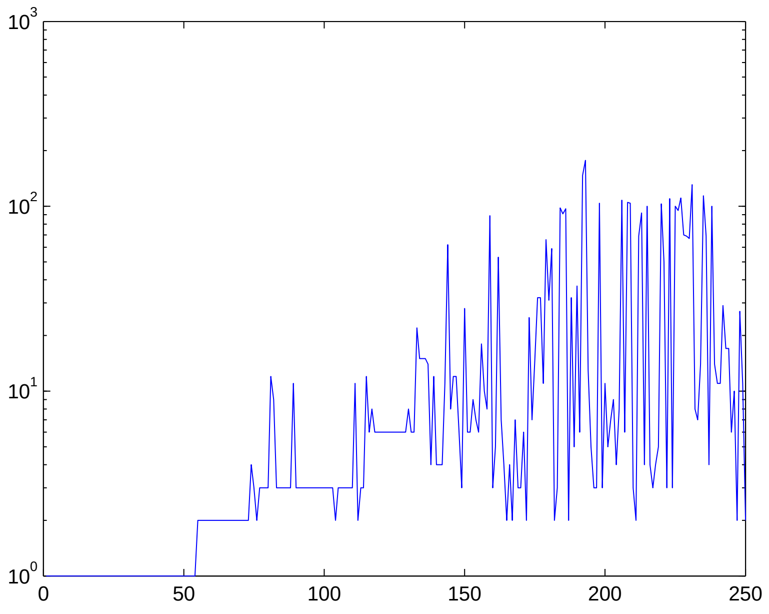
I have noted, that I have gained 1 absent mark. I do not recall being absent a full monday . . .

Figure 2: Jacobi



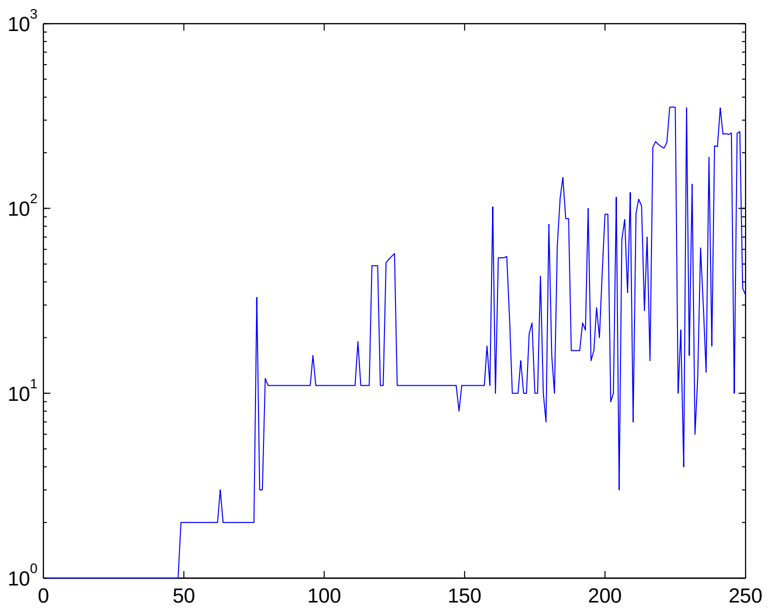
As can be seen the Jacobi splitting stays below 10^2, which is quite nice.

Figure 3: Gauss-Seidel



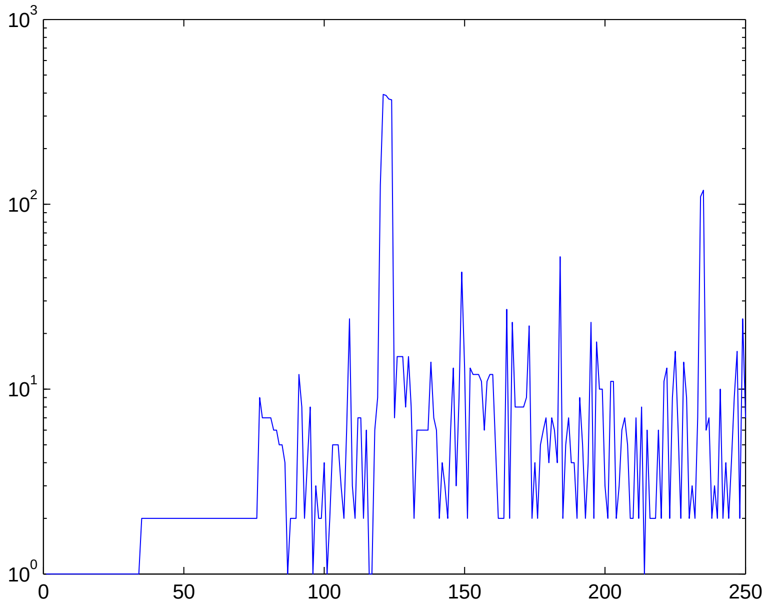
The Gauss-Seidel splitting variates a bit more than the Jacobi splitting, but looks good too.

Figure 4: SOR with $\omega = 0.25$



According to WikiPedia's text on the SOR splitting $\omega > 1$ should converge faster while $\omega < 1$ should be more robust. I wanted to test this so I chose $\omega = 0.25$ and $\omega = 1.75$. This plot of 0.25 seems to use a lot of iterations in the last part of the simulation.

Figure 5: SOR with $\omega = 1.75$



Wow! This beats all the other splittings in performance. The number of iterations used are on average much lower than 100.