Using Numerical Optimization Methods to find Hamiltonian Cycle in Directed Graph

Danielle C. Maddix
CME 304 Numerical Optimization
Final Project

March 20, 2014

1 Introduction

We use the methods of continuous numerical optimizations, which assumes an objective function f has continuous first and second derivatives to solve a famous NP-complete problem from discrete mathematics, namely the Hamiltonian Cycle problem. A directed graph defined as $G(V, E)$, where V is the set of notes and E is the set of edges has a Hamiltonian cycle if it contains a cycle, which visits each node exactly once. Thus, the length of this simple cycle must be $N = |V|$. This is related to the famous Traveling Salesman problem, which states to find the minimum cost path of a salesman who visits every city exactly once. In later sections, we will discuss the specific problem formulation, numerical methods used and the computational/numerical challenges faced in the different problem formulations.

2 Problem Description

From the work in [1], it can be shown that minimizing the below objective function can be used to solve this Hamiltonian cycle, subject to certain linear constraints.

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \varphi(x) \\ \text{subject to} \quad & x \in S. \end{aligned} \tag{1}$$

$x \in \mathbb{R}^M$, where $M = |E|$, consisting of the edge transion probabilities $x_{ij}, (i, j) \in E$. The objective function φ is given below as:

$$\varphi = \det(I - P(x) + \frac{1}{N} ee^T), \tag{2}$$

where e is the one-vector and $P(x) = D^{-1}A$, where $D_{ii} = 1/\deg(v_i)\ \forall i$ and $A \in \mathbb{R}^{NxN}$ is the adjacency matrix of G. In terms of x

$$P(x)_{ij} = \begin{cases} x_{ij}, & (i, j) \in E \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

One challenge is that it requires the global minimum, x^*, to be found rather than the local minimum. One common problem is that depending on the initial problem, an algorithm can get stuck at a local minimum and not be able to find the global minimum.

As shown in [1], if $\varphi(x^*) = -N$, for global minimizer x^*, then we have found the Hamiltonian cycle. Moreover, if $\varphi(x^*) = -N$, then the graph has no Hamiltonian cycle. Of course, to solve this discrete problem using continuous relaxation, we must round our solution x, to get a feasible binary solution of 0's and 1's, where $x_{ij} = 1$ indicates that edge (i, j) is in the Hamiltonian cycle and $x_{ij} = 0$ indicates that edge (i, j) is not.

As discussed above, we are minimizing a nonlinear objective function, subject to a set of linear equality and inequality constraints. We test the problem for three different constraint sets:

1. *Stochastic Constraints*: $x \in S \iff x_{ij} \geq 0$ and rows of $P(x)$ sum to 1
2. *Doubly Stochastic Constraints*: $x \in DS \iff x_{ij} \geq 0$ and rows and columns of $P(x)$ sum to 1
3. *Additional Constraints* added to S

We will discuss the advantages and disadvantages of each set of these different constraints, as well as describe constraint 3 in the results sections.

3 Overview of Methods

3.1 Line Search

One key module needed in solving a nonlinear optimization problem is a line search module. Given a descent direction p_k, maximum step length, α_m, current iterate x_k and current function and gradient values, a line search computes a feasible step length α_k which will result in sufficient decrease for the objective function. Note that p_k is a descent direction if and only if $p_k^T g_k < 0$, where g_k is the gradient of the objective function, F evaluated at x_k. For appropriately chosen α this will guarantee that for an objective function, $F_{k+1} < F_k$. The next iterate is computed in the below update:

$$x_{k+1} = x_k + \alpha_k p_k \tag{4}$$

At first, a simple line search backtracking algorithm was implemented. It checked that the Armijo condition, defined below was satisfied and if not it reduced the current α_k by 0.5. The Armijo condition guarantees descent in the objective:

$$F(x_{k+1}) \leq F(x_k) + \mu \alpha_k g_k^T p_k \tag{5}$$

This is known as a crude line search, where $0 < \mu < 1$. Note μ was chosen to be 10^{-4}, as indicated in [3].

A good initial value for α_k is one. Thus, initially $\alpha_k = \min(1, \alpha_m)$, since it cannot be larger than the maximum step length, otherwise constraints can be violated. A unit step length is desirable, since Newton's method is when $\alpha_k = 1$ and $p_k = -H^{-1} g_k$, resulting from the quadratic model from the Taylor series $\min F = \min g^T p + p^T H p$. So, $\nabla F = g + Hp = 0$ if and only if p is defined as above. Newton's Method results in quadratic convergence, which is a very desirable property. Note that Newton's method has only local convergence properties, which requires a good initial point. However, Modified Newton, which we will utilize, combines Newton's Method with a linear search and constructs a positive definite modified Hessian, and is globally convergent. The method will be described in more detail in the later section on Modified Newton and Directions of Negative Curvature. Thus, in order to observe the quadratic convergence of Newton's when x is approaching x^* a unit step length should be used.

The flaws in this method is that it does not include the below strong curvature condition, which results from doing an exact line search, that is $\underset{\alpha}{\text{minimize}} \quad F(x_k + \alpha p_k)$

$$|g(x_k + \alpha_k p_k)^T p_k| \leq \eta g_k^T p_k, \tag{6}$$

where $0 \leq \eta < 1$ and $\eta = 0$ corresponds to an exact line search. This curvature condition ensures that α_k is not too small. Moreover, it is advantageous to implement for functionality with using the BFGS Quasi Newton Method, which requires $y^T s > 0$ for $y = g_{k+1} - g_k$ and $s = \alpha_k p_k$ [2]. The curvature condition from Equation 6 cannot be used alone, since it does not guarantee descent. Thus, the Wolfe conditions combine the above Armijo conditions from Equation 5 and the curvature condition from Equation 6. Denote α_k satisfying the Wolfe conditions, as $\alpha_k \in \Gamma(\mu, \eta)$.

Thus, a bisection and bracketing line search, which guarantees that the Wolfe conditions are satisfied is utilized. This method was throughly tested on simple unconstrained nonlinear functions, where the descent direction $p = -g$ for steepest descent and $p = -H^{-1}g$ for Newton. The key in guaranteeing that the algorithm works correctly is to test each module individually.

Also, additional features were added to the line search code to give warnings of potential errors in other functions, such as an exception being thrown if p_k is not actually a descent direction, by checking the sign of the inner product of p_k and g_k. Moreover, since the line search is the most likely part of the algorithm that could be break down, it should only be doing at most 10-20 iterations in early iterations. An exception is then thrown if a step requires more than 30 line search iterations.

Below we summarize the results using Newton and steepest descent, where the line search succeeds for both methods on the first simple convex function and breaks down and exits for steepest descent on the Rosenbrock function $f(x_1, x_2) = (1-x_1)^2 + 100(x_2 - x_1^2)^2$ [4], which is known as a quadratic function for which steepest descent has difficulties. Both functionalities of the line search module must be tested for use within the larger program. Note that in the unconstrained case, α_m is set to be infinite.

Our simple simple quadratic test function is $f(x_1, x_2) = (x_1 - x_2)^4 + 2x_1^2 + x_2^2 - x_1 + 2x_2$ [6]. Recall the first order and second order optimality conditions for unconstrained minimizer is that $g(x^*) = 0$ and the Hessian evaluated at x^* is positive definite. Hence, termination when the gradient is less than a tolerance of 10^{-8}, which is approximately the square root of machine precision. Steepest decent takes 35 iterations to converge linearly within the above tolerance, whereas Newton only takes 4 iterations to converge quadratically.

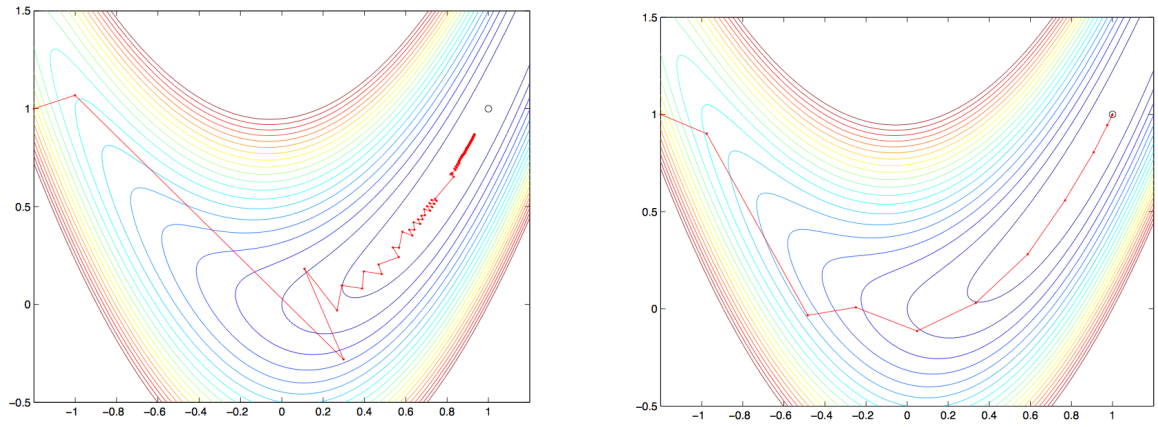
```
EDU>> xk = steepestdescent(x,8)

k          F      alpha     ||g||      ||p||
1       0.0625     0.5    2.5e+00    2.5e+00
2      -0.4219     0.1    1.6e+00    1.6e+00
3      -0.5439     0.2    1.6e+00    1.6e+00
4      -0.7036     0.1    3.7e-01    3.7e-01
5      -0.7100     0.1    2.1e-01    2.1e-01
6      -0.7125     0.1    1.1e-01    1.1e-01
7      -0.7126     0.2    1.6e-01    1.6e-01
8      -0.7135     0.1    7.2e-02    7.2e-02

xk =

    0.0352
   -0.5743
```

```
EDU>> xk = Newton(x,8)

k          F      alpha     ||g||      ||p||      cond(H)
1       0.3088     0.5    2.7e+00    8.2e-01      4.3
2      -0.7137     1.0    5.9e-03    5.5e-04      4.1
3      -0.7137     1.0    5.6e-06    5.2e-07      4.1
4      -0.7137     1.0    4.9e-12    4.6e-13      4.1

xk =

    0.0335
   -0.5670
```

In optimization, the key is not just the final solution x_k but also the sequence of iterates along the way, including the function values, step length, norm of gradient and norm of descent direction for unconstrained. It is important to check that the objective is actually decreasing. Below is a comparison of the output from the first eight iterations of steepest descent and Newton on the above test function. Note that the unit step length is chosen for Newton in every iteration, except for the first one. Below is the output of steepest descent and Newton, when tested on the Rosenbrock function with initial point $x_0 = [1.2; 1.2]$. The Rosenbrock function is known to have the exact global minimizer, $x^* = [1; 1]$ and so we can calculate the error at each step as $\|x_k - x^*\|$. To exhibit quadratic convergence, we must have $\|x_{k+1} - x^*\| \leq c\|x_k - x^*\|^2$ Again, note the unit step length selection for Newton. This case of steepest descent shows that the line search will terminate when too many iterations have occurred.

```
EDU>> xk = steepestdescent(x,4)

k          F      alpha     ||g||      ||p||
??? Error using ==> wLineSearch at 46
Too many line search iterations

Error in ==> steepestdescent at 36
    alpha_k = wLineSearch(f,g,pk,alpha_max,x_k);
```

```
EDU>> xk = Newton(x,100)

k        error       F       alpha     ||g||      ||p||      cond(H)
1       4.7e-01   0.0384      1.0    4.0e-01    5.1e-01      4513.5
2       2.2e-01   0.0188      0.5    4.8e+00    7.3e-02      1176.4
3       1.5e-01   0.0043      1.0    6.6e-01    1.2e-01      2500.8
4       2.5e-02   0.0009      1.0    1.3e+00    1.5e-02      1682.7
5       9.5e-03   0.0000      1.0    3.5e-02    9.4e-03      2512.4
6       9.7e-05   0.0000      1.0    8.0e-03    9.7e-05      2499.6
7       4.0e-07   0.0000      1.0    1.5e-06    4.0e-07      2508.0
8       1.7e-13   0.0000      1.0    1.4e-11    1.7e-13      2508.0

xk =

    1.0000
    1.0000
```



(a) Steepest Descent Iterates

(b) Newton Iterates

Figure 1: Contours of Rosenbrock function

We can create the above contour plots of the Rosenbrock function and plot the iterates, choosing not to stop the line search for steepest descent. The success of using Newton over steepest descent on this function is clear, as shown in the above Figure 1 [4]. Note that the condition number indicates that the Hessian is almost ill-conditioned, which is also reflected by the contours.

3.2 Calculating Search Direction: Directions of Negative Curvature and Modified Newton

When solving a problem based on a quadratic model, there are several possible choices for the matrix B_k, where the search direction $p_k = -Z_k B_k^{-1} Z_k^T g_k$. Note that this is for the linearly constrained case. For unconstrained, Z_k would be removed so equal to the identity matrix, I. For steepest decent, $B_k = I$, but as discussed above, steepest descent does not possess some of the desired properties of Newton. It is also possible to use a Quasi-Newton method, such as BFGS, in which B_k is an approximation to the Hessian. This is a desirable approach in general. However, when given second derivative information, it is generally a good idea to take advantage of it, even though it may be more computationally expensive. In particular, for the Hamiltonian Cycle problem, directions of negative curvature are extremely important for reducing the objective φ [1].

Thus, we utilize a Modified Newton type method, which computes the Newton search direction p_k and a direction of negative curvature d_k. By of indefiniteness, $\exists$ some nonzero vector d_k, such that $d_k^T H_k d_k < 0$. A direction of negative curvature can be used to move off of a stationary point, which is not a minimizer or at any other time in the algorithm, since it does define a descent direction, as long as the sign is chosen correctly. If $d_k^T g_k > 0$, we set $d_k = -d_k$ to guarantee that it is a descent direction. Thus, a good clear direction of negative curvature d_k would be the eigenvector corresponding the largest negative eigenvalue. This

can be found by simply doing a spectral decomposition of the Hessian, H_k.

So, to compute d_k we first check if the Hessian, H_k is indefinite, that is, if it has any negative eigenvalues. If it does, we set d_k as discussed above. It is possible to combine p_k and d_k by using a linear combination, such as $x_{k+1} = x_k + \alpha_k^2 p_k + \alpha_k d_k$ [2]. Thus, we also compute a modified Hessian matrix, B_k so that the search direction p_k can be computed uniquely. $H_k = U\Omega U^T$, since H_k is symmetric, its eigenvalue decomposition exists. We simply modify the diagonal eigenvalue matrix, Ω, by setting the diagonal elements to be $\bar{\Omega}_{ii} = \max(|\lambda_i|, \delta)$ for some positive δ on the order of 10^{-8}, denoting the smallest tolerance for positive. δ is critical in preventing the eigenvalues from being 0 and so preventing the resulting matrix from being positive semi-definite, since we need it to be invertible. Hence, $B_k = U\bar{\Omega}U^T$, which can then be used to compute the Modified Newton search direction.

Since the problem we are testing it on is relatively small, that is, a graph with about 10-20 nodes and a determinant is used to calculate the objective, which is very computationally expensive, it is not necessarily important in this case to optimize the other linear algebra. It is clear that a majority of the computations will be spent in evaluating the objective, its gradient and Hessian at each iteration. Thus, in this problem it is sufficient to use an eigenvalue decomposition. In larger problems, however, the Modified Cholesky algorithm [2] to compute B_k would be more appropriate, where $B_k = H_k + E_k = R_k^T R_k$, for upper triangular matrix R_k, which will produce a bounded matrix and so not ill-conditioned. Modified Cholesky can also be used to compute directions of negative curvature by simply doing an upper triangular solve, $R_k d_k = r_{ss} e_s$, where e_s is the one vector and r_{ss} corresponds to negative $\hat{a}_{ss}$ [2].

3.3 Active Set Method

Now that the line search module has been properly tested, the Feasible Point Active Set Method module must be tested on a nonlinear objective with linear constraints $\mathcal{A}x \geq \ell$. This is an equality quadratic program (EQP) method, since it uses a quadratic model and is based off the method for linear equality constraints. The task is to approximate the active set, which is the set of equality constraints, $\mathcal{A}x = \ell$. This may never be known exactly, but we select a working set of constraints A to be treated as equality constraints, such that $Ax = b$, which b is the corresponding entries in ℓ. Thus, we must have methods for adding and deleting constraints from the active set [2].

In feasible point active set methods, the solution has two phases, that is, the first phase determines a feasible point and the second phase finds a sequence of feasible points that converge to a solution. Note that feasible means that an iterate x_k does not violate any constraints. At each stage, the current iterate x_k satisfies the constraints in the working set exactly, $A_k x_k = b_k$ and the quadratic program subproblem is defined below:

$$\begin{aligned} \underset{p}{\text{minimize}} \quad & g_k^T p_k + \frac{1}{2} p_k^T B_k p_k \\ \text{subject to} \quad & A_k p_k = 0. \end{aligned} \tag{7}$$

Let Z be a matrix consisting a basis for the null space of A. Then, the optimality conditions

are similar to that as in the unconstrained case, except now they are in terms of the reduced gradient, $Z^T g$ and reduced Hessian $Z^T HZ$. Clearly, x^* must be feasible, so $\mathcal{A}x^* \geq \ell$, with $Ax^* = b$. Now the reduced gradient must be 0, which we will use of subspace termination convergence criteria . Thus, $Z^T g^* = 0 \iff A^T \lambda^* = g^*$, since the null space of A and row space are orthogonal. The vector λ^* is the lagrange multiplier and we require $\lambda^* \geq 0$. The sign on the lagrange multiplier is key when a constraint is being considered for deletion from the working set. Lastly, for sufficient condition the reduced Hessian must be positive definite. Note that this is actually a weaker condition than in the unconstrained case, since it requires the Hessian to only be positive definite in the null space of the constraints, rather than in the entire space.

3.3.1 Finding Initial Feasible Point and Initial Working Set

It is first important to find an initial feasible point x_0. Such an x_0 must satisfy $Ax_0 = b$ and $x \geq 0$.

$$\begin{aligned} &\text{minimize } c^T x \\ &\text{subject to } Ax = b \\ &\qquad\qquad l \leq x \leq u, \end{aligned}$$

where c is the 0 zero vector and there is only a lower bound $l = 0$ on x. Then we simply use Matlab's *linprog* function to solve it. Note that for numerical reasons and in the doubly stochastic case, we want x to be bounded away from 0, so l is set to be ϵ for some small $\epsilon > 0$. Requiring that $x \geq \epsilon$ bounds x away from 0 and also the boundary. Note that an initial feasible point can also be found by an active set strategy, by minimizing the set of violated constraints.

Once we have a feasible point, we can use it to construct the initial working set A to find the equality constraints. Clearly the matrix A will first consist of all the guaranteed equality constraints and so the number of equality constraints, *numeq* must be stored. These equality constraints cannot be deleted, since these are the constraints that we are certain to be active. As will be seen in the case of doubly stochastic constraints, DS, it is important that this matrix have full row rank, since when we solve for the lagrange multiplier estimates, as described in the deleted constraints section, we use the matrix A^T and would like a compatible system. Thus, we go throughout he matrix A row by row and delete the linearly dependent rows, so that the resulting matrix has full rank and has the same row space as the original matrix.

Let I be a vector, which consists of the indices of active constraints. To determine which rows we should add to the initial working set from the inequality constraints, we simply check if $A_0 x_0 = b_0$. Since, we are solving this numerically we need to check that they are equal within some tolerance close to machine precision, $\epsilon_m = 10^{-16}$. For these corresponding indices we add these indices to I and the corresponding rows to A and elements to the right

hand side matrix b. It is important that we add these below the actual permanent equality constraints.

Note that a function has been created to calculate the initial feasible point, another function has been created to calculate A_0, I_0 and b_0 and one to ensure that the matrix of equalities has full rank. Each were tested on simple matrices to guarantee that the initial processing is correct.

3.3.2 Computing Maximum Steplength

Contrary to unconstrained problems, a maximum step length, α_m is required for the line search to ensure that the step taken will not violated one of the constraints and so will remain feasible. The problem is to ensure that $x_{k+1} = x_k + \alpha_k p_k$ remains feasible. Let i be an index of constraint that is not in the current working set, that is, $i \notin I_k$. The important part is that check the sign of $a_i^T p_k$, since we require $\mathcal{A}x \geq l$. If $a_i^T p_k \geq 0$, the constraints will not be violated by linearity. However, if $a_i^T p_k \leq 0$, there is a maximum step α_m where the constraint becomes binding, that is $a_i^T(x_k + \alpha_m p_k) = 0$. Thus,

$$\alpha_m = \begin{cases} \min(\dfrac{\ell_i - a_i^T x_k}{a_i^T p_k}), & \text{if } a_i^T p_k < 0 \ \exists i \notin I_k \\ +\infty, & \text{if } a_i^T p_k \geq 0 \ \forall i \notin I_k \end{cases} \tag{8}$$

This equation was used in the the simple function called *computeAlphamax*, which returns α_m, which can be termed as the step to the nearest constraint. The index of this constraint is also returned and will be used in the adding constraint section [2]. Once the maximum step length and search direction have been computed, the line search module is called to compute α_k. This function was tested separately on simple matrices to ensure that it worked properly.

3.3.3 Adding constraints

After computing x_{k+1} as described in the above section, it is necessary to check whether an inequality constraint has been hit and if so to add it to the working set, A_k. It is clear that if $\alpha = \alpha_m$, a constraint has been hit. If more than one has been hit, we arbitrarily choose which one to add to A. The order in which the constraint is added is important. Whenever, a new constraint is added to A_k, it is added to the last row, not to interfere with the first *numeq* equality constraints. Clearly, the number of rows of A_k, denoted by m increases by one. The corresponding index of this constraint is added to I_k and corresponding value added to b_k. A new nullspace, Z_k, of A_k must also be computed. The computation of Z_k can be done efficiently since $Z_k = [Z_k \ z]$, where z is the nullspace of $[A; Z_k^T]$.

As noted in *Lemma* 2.2.3 from [2], the working set will stay linearly independent when a constraint is added because for a_i, $i \notin I_k$, such that a_i is a linear combination of the rows of A_k. Then, by definition of linear dependence $\exists$ a nonzero vector v such that $a_i = A_k^T v$. So if $A_k p_k = 0$, then $a_i p_k = v^T A_k p_k = 0 \not< 0$ and so by the definition of α_m given in the previous section, the dependent constraint a_i will never be added to the working set.

3.3.4 Deleting constraints

The EQP method consists of two loops, the outer loop continues until we have reached subspace convergence as indicated by the norm of the reduced gradient, $Z_k^T g_k$ and where the signs of the all lagrange multipliers λ corresponding to constraints that can be deleted are positive. The inner loop has been described in the previous sections, where the search direction p_k is computed, then α_m, followed by a Wolfe line search for α_k and the iterate is updated. If a constraint is hit, then it is added as described above. The subspaces convergence criteria is that $\|Z_k^T g_k\| \leq \epsilon \|Z_0^T g_0\|$ for $\epsilon = 10^{-2}$.

Once subspace convergence has been reached, the outer loop is entered and the importance of checking whether or not to delete a constraint is essential. The lagrange multipliers are utilized in the process. By the first order optimality conditions, we know that $A^T \lambda^* = g^*$, so we can approximate it by solving the system $A_k^T \lambda = g_k$ for the first order multipliers or to get the second order multipliers we solve $A_k^T \lambda = H p_k + g_k$. The first order multipliers are compatible with the search direction from steepest descent and BFGS, whereas the second order multiplier are compatible with Newton. Once the lagrange multipliers have been computed, the constraint corresponding to most negative multiplier is deleted. Note that we only consider the entires of $\lambda(numeq + 1 : end)$, since the first $numeq$ equality constraints cannot be deleted. If all of the elements in $\lambda(numeq+1 : end)$ are positive, the algorithm terminates with a minimum, as guaranteed by the optimality conditions. Otherwise, the constraint corresponding to the most negative lagrange multiplier estimate is deleted, and corresponding index and values from I_k and b_k, respectively. Thus, the number of rows of A_k, m decreases by one and the dimension of the null space of A_k increases. The new nullspace, $Z_k = [Z_k \ z]$, is computed as in the adding constraints section, for z is defined in that section. We then set $\|Z_0^T g_0\| = \|Z_k^T g_k\|$, as the new reference for the next subspace convergence subproblem in the inner loop.

This is a part of the algorithm, where experiments can be conducted, in terms of how to delete a constraint. Since we are using a direction of negative curvature if one exists or modified Newton for our search direction, it is not proven that a direction of negative curvature will stay feasible with respect to the deleted constraint, when using either multiplier estimates. Our approach is to use the first order multipliers. Then, compute the new search direction p_k and check if it remains feasible with respect to the deleted constraint, a_i, that is if $a_i^T p_k \geq 0$. If it satisfies this condition, we use this search direction. On the other hand, if $a_i^T p_k < 0$, we use the steepest descent direction, where $p_k = -Z_k(Z_k^T g_k)$, which is proven to stay feasible with the first order multipliers. This search direction is then used to compute α_m and in the line search and process continues. One numerical difficulty that can occur is that elements of p_k that are supposed to be 0 are only essentially 0, due to floating point precision, i.e. on the order of 10^{-16} or significantly small. Thus, it is good to first set any small components of p_k , which are less than small tolerance, say tol $= 10^{-10}$ to 0 before checking if the search direction remains feasible with respect to the deleted constraint.

One downside to using the first order multipliers is that they may not be as accurate of an estimate as the second order estimates. Thus, when checking the sign of $\sigma = \min(\lambda)$, we

only delete if σ is significantly negative, more precisely if $\sigma < -10^{-6}$. Otherwise, numerical and floating point errors could result in a false positive to delete a constraint when it really shouldn't be deleted, especially it is close to 0.

It is possible to use the second order multipliers, but a more conservative approach would be taken, where a constraint is only deleted if H_k is positive definite. Unfortunately, for the Hamiltonian cycle problem, the Hessian from the objective function φ is significantly indefinite. Another approach would be to use the H_k from Modified Newton and compute the second order multipliers using the descent direction from Modified Newton and not the direction of negative curvature. Again, for the objective in the Hamiltonian Cycle problem, directions of negative curvature are important to follow and in most cases for this problem seem to lead to the largest decrease in function values. Thus, more work can be explored in these areas.

3.3.5 Testing Active Set Module

To guarantee that this module worked correctly, it was tested on the below simple quadratic function. This tests both the adding and deleting of the constraints. The results are generated below and perfectly match those in Example 16.4 from [3].

$$\begin{aligned} \text{minimize } & (x_1 - 1)^2 + (x_2 - 2.5)^2 \\ \text{subject to } & x_1 - 2x_2 + 2 \geq 0, \\ & -x_1 - 2x_2 + 6 \geq 0, \\ & -x_1 + 2x_2 + 2 \geq 0, \\ & x_1 \geq 0, \\ & x_2 \geq 0, \end{aligned}$$

$$\mathcal{A} = \begin{pmatrix} 1 & -2 \\ -1 & -2 \\ -1 & 2 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad \ell = \begin{pmatrix} -2 \\ -6 \\ -2 \\ 0 \\ 0 \end{pmatrix}$$

Starting with initial feasible point $x_0 = (2, 0)^T$, $I_0 = (3, 5)$, which indicates the indices of the rows of $\mathcal{A}$ and ℓ to be in the working set.

```
EDU>> xk = ActiveSetEQPtest(x0, 100, 0)

k        phi       alpha    alpha_m       ||p||      ||Z^Tg||     m    null   cond(H)
1       7.2500      1.0     1.0e+100     0.0e+00     0.0e+00      2     0      1.0
2       6.2500      1.0     2.0e+00     0.0e+00     0.0e+00      1     1      1.0
3       1.0000      0.6     6.0e-01     2.5e+00     8.9e-01      1     1      1.0
4       0.8000      1.0     2.5e+00     4.5e-01     1.1e-16      1     1      1.0

xk =

    1.4000
    1.7000
```

3.4 Use of Finite Differences to Guarantee Correct Derivatives

A key source of error in optimization problems or even problems in general is doing a simple mistake when coding the objective function, its gradient, g and its matrix of second derivatives, the Hessian, H. For complicated formulas, it is easy to make a simple error when typing them. Thus, a good initial test that the gradient and Hessian have been computed properly is to use finite differences. Three simple submodules were written, one to compute, one to compute the gradient and one to compute the Hessian. It is especially important for the Hamiltonian cycle problem, since the gradient and Hessian include complicated formulas of calculating determinants of sub matrices with certain rows and columns removed. The forward difference formula for finite differences is given below as derived from Taylor series:

$$\nabla F(x)_i \approx \frac{F(x + he_i) - F(x)}{h}, \tag{9}$$

where e_i is the vector with 1 in the i^{th} position and zeros elsewhere. Thus, after writing the code to compute the function and the code to compute the gradient, a code check is written to loop through all the rows in the gradient vector and check that the absolute difference between the above finite difference calculation and the computed gradient is less than $10-^8$ for finite difference interval h chosen to be on the order of 10^{-8}, that is on the order of the square root of machine precision to balance the truncation error, τ and the numerical calculation error. If the absolute difference is greater than this tolerance, an exception is thrown. This gives confidence that the gradient was computed correctly.

Once we have confirmed that the gradient was computed correctly, we can again use forward differences on the gradient to confirm that the Hessian is computed correctly. Let

$$y_i = \frac{\nabla F(x + he_i) - \nabla F(x)}{h} \tag{10}$$

For every row we compute the finite difference above y and then loop through all the columns of H and check whether the absolute difference, $|y_j - H_{ij}|$ is within the tolerance of 10^{-8}. If not, an exception is thrown.

Hence, it was shown that finite differences can be used not only to approximate the second derivatives in second derivative methods, but also as a check that the derivatives have been

coded correctly. For the Hamiltonian cycle problem, the gradient and Hessian passed these finite differences tests.

4 Hamiltonian Cycle Instance

We create a Hamiltonian Cycle function, which uses the above tools for specific instances of the initial working set, A_0, I_0, b_0 and initial feasible point, x_0. These will be used as input to the active set method solver, allowing for various cases for the different problem formulations. The initial feasible point is checked that the equality constraints are satisfied within a tolerance tolerance by taking their absolute difference and that every element of x_0 is positive greater than ϵ. If not, an exception is thrown. As described above, we then call the *fullrowrank* function on the set of equality constraints, which throws an exception at the end if the newly computed matrix with the same row space is not full rank. Note that if the matrix is full row rank, it is not modified, as in the stochastic constraints case. We then store the number of equality constraints *numeq*, which is equal to the number of rows of A. $\mathcal{A}$ is extended to include the inequality constraints, by including the MxM identity matrix and $\ell = [\ell; \epsilon e]$, where e is the one-vector. Then compute initial working set, Aw_0, b_0, I_0.

We then call a function called *generategraph*, which takes the number of nodes and degree for the random-regular graph case, for which the degree is 3 for cubic graphs. Lastly, we construct $\varphi, \nabla\varphi,$ and $\nabla^2\varphi$ and test them with finite differences, as described in the finite difference system. After this setup, the solver is called.

The resulting rounded x_k from the solver is then used to check if it satisfies the Hamilitonian Cycle condition and the resulting graph is plotted.

4.1 Generating Cubic and Random Non-regular Hamilitonian Graphs

The Hamiltonian Cycle Problem is tested on random cubic graphs, that is random 3-regular graphs and random non-regular Hamiltonian graphs. A cubic directed graph has in-degree equal to out-degree equal to 3 for each node. Almost all regular graphs are Hamiltonian and so are safe for testing. The latter type of graph is simply a random graph that has a Hamiltonian cycle. The size of these directed graphs is between 10 and 20 nodes. We use the matgraph package [5] to generate a random cubic graph, by calling *random_ regular*(g,N,3). Note that for the cubic graphs, an LP does not need to be solved, since the vector $\frac{1}{3}e$ is clearly feasible. To create a random Hamiltonian directed graph, we just start with a directed cycle and add edges to it, with probability 0.5, similar to constructing an Erdős Rényi random graph.

We utilize the *incidence_ matrix*(g,'signed') function which creates the incidence matrix of the directed graph. The incidence matrix, $ind \in \mathbb{R}^{NxM}$, of a graph is defined such that $(ind)_{ij}$ equals 1 for incoming edge e_j to node v_i, -1 for outgoing edge e_j to node v_i and 0 otherwise. One sample incidence matrix is shown below for Figure 4 in the Observations section, discussing the successes and failures of stochastic constraints on a 10 node cubic

graph. The incidence matrix is then used in the function, *computeP*, which is used to create the transition probability matrix $P(x)$.

```
ind =

  Columns 1 through 13

     1     0     0     1     0     0     0     0     0     0     0     0     1
     0     1     0     0     0     0     1     0     0     1     0     0     0
    -1     0     1     0     1     0     0     0     0     0     0     0     0
     0    -1     0     0     0     0     0     0     0     0     1     0     0
     0     0    -1     0     0     1     0     0     0     0     0     0     0
     0     0     0    -1     0     0     0     1     0     0     0     1     0
     0     0     0     0    -1    -1     0     0     1     0     0     0     0
     0     0     0     0     0     0    -1    -1    -1     0     0     0     0
     0     0     0     0     0     0     0     0     0    -1    -1    -1     0
     0     0     0     0     0     0     0     0     0     0     0     0    -1

  Columns 14 through 26

     0     0    -1     0     0    -1     0     0     0     0     0     0     0
     0     0     0    -1     0     0     0     0    -1     0     0    -1     0
     0     0     1     0    -1     0    -1     0     0     0     0     0     0
     1     0     0     1     0     0     0     0     0     0     0     0    -1
     0     1     0     0     1     0     0    -1     0     0     0     0     0
     0     0     0     0     0     1     0     0     0    -1     0     0     0
     0     0     0     0     0     0     1     1     0     0    -1     0     0
     0     0     0     0     0     0     0     0     1     1     1     0     0
     0     0     0     0     0     0     0     0     0     0     0     1     1
    -1    -1     0     0     0     0     0     0     0     0     0     0     0

  Columns 27 through 30

     0    -1     0     0
     0     0     0     0
     0     0     0     0
     0     0    -1     0
     0     0     0    -1
    -1     0     0     0
     0     0     0     0
     0     0     0     0
     1     0     0     0
     0     1     1     1
```

4.2 Computing matrices to enforce Stochastic and Doubly Stochastic constraints

A is designed so that its first rows correspond to the equality constraints, as defined by the stochastic or doubly stochastic constraints. To compute A we utilize the incidence matrix. For the doubly stochastic constraints, $A_{ij} = 1$ for all outgoing edges of node i, thus we find the -1's in the incidence matrix at position j and set $A_{ij} = 1$, with corresponding $b_i = 1$. For doubly stochastic, we also require that the sum of the incoming edges to node i is 1. Thus, we find the 1's in the incidence matrix at position j and set $A_{ij} = 1$ and do the same as above for the outgoing edges, setting the right hand side equal to 1 at those locations.

Note that x is a vector of edges and $Ax = b$ is requires the rows of P to sum to 1, which it must since it is a probability distribution, but A can be computed from the incidence matrix without P. For the stochastic case, these equality constraints define the first N rows of A and for the doubly stochastic constraints these equality constraints define the first $2N$ rows of A. One problem that can occur with the doubly stochastic constraints is that A will not have full row rank. It actually only has one linearly dependent row, which must be removed.

For both constraints, after computing an initial point x_0, we check that the rows of $P(x_0)$ sum to 1, by again checking that the absolute difference is on the order of machine precision and if they do not sum to 1, an exception is thrown. In addition, when the doubly stochastic constraints are used, we also check that the columns sum to 1 and if not throw an exception. This is checked on the initial point, as well as on the final solution x^*.

4.3 Constraints added to Stochastic Constraints to Achieve Better Results

As will be described in the results and observations sections, it is clear that the stochastic constraints, S have the lowest percentage of success, since they do not require that all incoming edges to a given node sum to 1, which is a property of a Hamiltonian cycle. They only require that all the outgoing edges sum to 1. Thus, its poor performance is reflected in it terminating in a local minimum, before reaching the global minimum. However, the stochastic constraints do have certain advantages over the doubly stochastic constraints, since the rows are always linearly independent and so the equality matrix A has full row rank and will terminate on a vertex, which is a square constraint matrix. Thus, we desire to add constraints to the stochastic constraints set to guide it out of this local minimum.

One point to consider is the initial starting point. If the starting point is close to a local minimum, the algorithm will converge to that local minimum, rather than to the global minimum, which is one reason why solving for global minimums is more challenging. A method is needed to prevent the method from getting stuck at the local minimum. Currently, the initial point is just required to be distance ϵ from the boundary, which could make it a biased initial point. We add an additional constraint on the initial feasible point in the LP solver that it must be equal distance from all the boundaries, making it unbiased and possibly not as close to a certain local minimum. As seen in the results section, this did improve the success of the stochastic constraints, but still did not make it as competitive as the doubly stochastic constraints.

Additional possible constraints to be added could involve specific properties of Hamiltonian cycles, which is a similar method as to how the doubly stochastic constraints were derived from the stochastic constraints. Such properties include adding constraints invoking the cut of a graph, where the cut $S \subseteq V$ is a subset of the vertices that partitions the vertices into two disjoint sets. The size of the cut is the number of edges crossing it. A minimum cut can be found in polynomial time and it is known that a cycle has exactly $\binom{n}{2}$ minimum cuts, using Karger's algorithm. However, these constraints may be too computationally expensive to enforce.

4.4 Rounding

Since we are using a continuous method to solve a discrete problem, where the solution edge vector x should have entries $\in \{0, 1\}$, it is important to implement rounding of the x_i's in our module. Each element is converging to 1 or 0, but at each iteration we can create a new

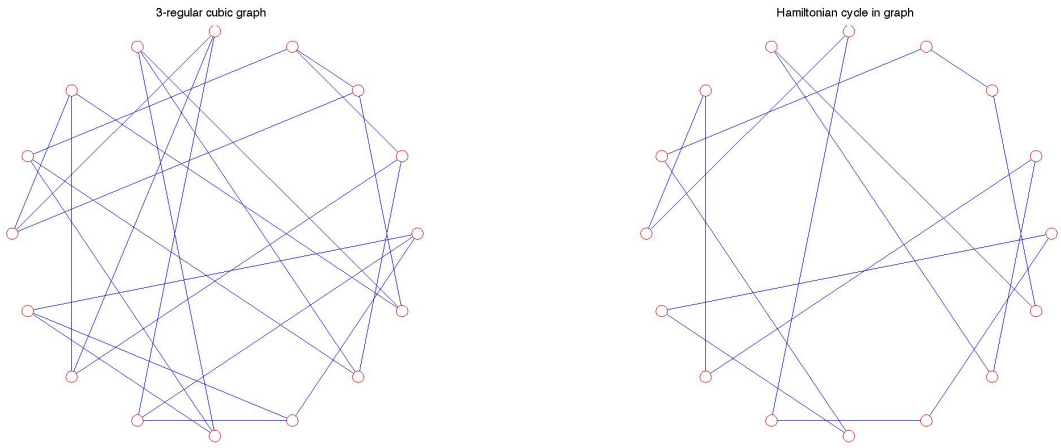
variable, $y = \text{round}(x)$, that is

$$y_i = \begin{cases} 1, & x_i \geq 0.5 \\ 0, & \text{otherwise} \end{cases}$$

We then check if y satisfies the condition that a Hamiltonian cycle has been found, that is that $\varphi(y) = -N$, for absolute difference within a certain tolerance and also check that y is feasible to the equality constraints, since it clearly satisfies the inequality constraints, $y \geq 0$. We observe that rounding can terminate the procedure quicker, since it can potentially find a rounded solution, before actual convergence occurs. Sometimes a rounded solution can be found after only 1 or 2 iterations only the smaller 10 node graphs.

4.5 Hamiltonian Cycle Results

We present results on the following 16 node cubic graph and 14 node random non-regular Hamiltonian graph, using the doubly stochastic constraints. Results from the stochastic constraints will be seen in later sections. The results indicate that the Hamiltonian cycles are properly found, as indicated in the plots and by the vectors of edges. Moreover, the given edge vector shows that the graph is traversing the edges in the correct order, that is, that the directed edges are all going in the same direction. It is important to note in the output that the objective function is always decreasing and that neither α_m nor α is too small, indicating infeasibility. Note that in the initial graph plot, each undirected edge represents two directed edges going in each direction, whereas for the Hamiltonian cycle plot, a edge represents a single directed edge, as indicated by the below edge set from node i to node j. This plotting is attributed to the fact that the draw graph in matgraph does not draw directed edges.

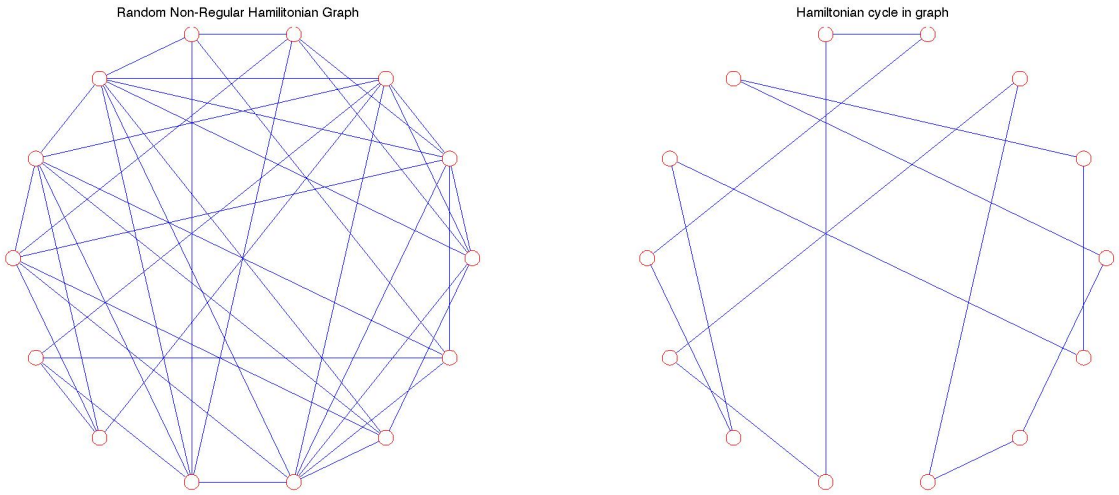


(a) Initial Cubic 16 Node Graph

(b) Corresponding Hamiltonian Cycle

Figure 2: Cubic graph and a corresponding Hamiltonian cycle from using the Doubly Stochastic Constraints with output and set of edges defining the Hamiltonian Cycle given below

```
k        phi      alpha      alpha_m       ||p||     ||Z^Tg||     m    null   cond(H)
1      -0.6138   1.0e+00    1.0e+00         0.2         0.9      32     16     20.4
2      -1.0127   6.1e-01    6.1e-01         1.0         1.4      33     15     12.1
3      -1.9237   5.3e-01    5.3e-01         1.4         2.1      34     14     12.1
4      -1.9237   1.8e-14    1.8e-14         3.6         1.8      35     13     16.7
5      -2.0255   3.1e-01    3.1e-01         3.3         1.8      36     12     16.7
6      -2.1995   1.4e-01    1.4e-01         5.5         1.8      37     11     16.7
7      -2.2395   3.5e-02    3.5e-02         3.1         1.7      38     10     17.7
8      -3.1282   5.3e-01    5.3e-01         3.5         3.1      39      9     18.0
9      -3.3253   6.6e-02    6.6e-02         2.5         3.3      40      8     23.9
10     -3.3253   1.2e-15    1.2e-15         4.5         2.2      41      7     25.0
11     -4.2632   3.4e-01    3.4e-01         1.0         3.2      42      6     25.0
12     -4.9840   2.1e-01    2.1e-01         8.5         3.7      43      5     28.7
13     -4.9840   7.5e-15    7.5e-15         4.4         3.7      44      4     41.0
14     -7.1467   4.9e-01    4.9e-01         4.4         3.3      45      3     41.0
Rounded iterate is solution
ans =

  -16.0000

Hamilitonian cycle exists!

e = (i,j)
6 = (9,7)
7 = (10,1)
8 = (11,2)
12 = (12,5)
15 = (13,10)
18 = (14,12)
20 = (15,6)
22 = (16,3)
26 = (3,4)
27 = (4,8)
29 = (5,9)
34 = (7,11)
38 = (8,13)
40 = (1,14)
43 = (2,15)
47 = (6,16)
```



(a) 14 node random non-regular Ham Graph (b) Corresponding Hamiltonian Cycle

Figure 3: Random non-regular Hamiltonian graph and its corresponding Hamiltonian cycle from using Doubly Stochastic Constraints

k	phi	alpha	alpha_m	||p||	||z^Tg||	m	null	cond(H)
1	-1.8201	7.1e-01	7.1e-01	0.2	1.8	28	54	20.8
2	-1.9382	6.6e-02	6.6e-02	13.7	1.8	29	53	6.7
3	-2.0322	5.4e-02	5.4e-02	3.2	2.0	30	52	6.4
4	-2.0623	1.7e-02	1.7e-02	8.4	2.0	31	51	6.3
5	-2.0798	9.7e-03	9.7e-03	8.4	2.0	32	50	6.2
6	-2.1076	1.6e-02	1.6e-02	12.3	1.9	33	49	6.2
7	-2.1381	2.0e-02	2.0e-02	4.6	1.9	34	48	6.2
8	-2.1552	1.1e-02	1.1e-02	4.6	1.8	35	47	6.2
9	-2.5884	2.5e-01	2.5e-01	3.9	2.4	36	46	6.2
10	-2.7441	6.9e-02	6.9e-02	2.4	2.6	37	45	6.2
11	-2.8073	2.7e-02	2.7e-02	1.4	2.6	38	44	6.2
12	-2.8940	3.6e-02	3.6e-02	1.2	2.5	39	43	6.3
13	-2.9114	7.7e-03	7.7e-03	8.0	2.5	40	42	6.3
14	-3.1062	8.0e-02	8.0e-02	2.7	2.5	41	41	6.3
15	-3.1654	2.7e-02	2.7e-02	7.0	2.4	42	40	6.3
16	-3.1798	7.6e-03	7.6e-03	2.0	2.4	43	39	6.4
17	-3.2002	1.1e-02	1.1e-02	1.9	2.3	44	38	6.4
18	-3.5099	1.7e-01	1.7e-01	6.8	2.7	45	37	6.4
19	-3.5799	3.2e-02	3.2e-02	7.4	2.7	46	36	6.5
20	-3.7692	8.0e-02	8.0e-02	6.4	3.0	47	35	6.5
21	-3.9379	6.3e-02	6.3e-02	4.6	3.1	48	34	6.6
22	-4.0350	3.6e-02	3.6e-02	4.5	3.2	49	33	6.7
23	-4.0658	1.1e-02	1.1e-02	4.6	3.2	50	32	6.8
24	-4.0680	8.5e-04	8.5e-04	7.7	2.9	51	31	6.8
25	-4.2782	1.2e-01	1.2e-01	7.3	3.1	52	30	6.8
26	-4.3550	5.1e-02	5.1e-02	7.4	2.8	53	29	6.9
27	-4.3885	2.3e-02	2.3e-02	7.6	2.7	54	28	6.9
28	-4.4817	6.6e-02	6.6e-02	28.4	2.6	55	27	7.0
29	-4.8209	2.1e-01	2.1e-01	26.0	3.1	56	26	7.1
30	-5.0131	8.3e-02	8.3e-02	57.9	2.9	57	25	7.5
31	-5.0167	1.7e-03	1.7e-03	62.1	2.8	58	24	7.7
32	-5.0986	3.8e-02	3.8e-02	40.6	2.8	59	23	7.7
33	-5.1431	2.1e-02	2.1e-02	17.6	2.4	60	22	7.8
34	-5.2703	8.9e-02	8.9e-02	17.1	2.4	61	21	7.9
35	-5.2778	5.3e-03	5.3e-03	12.1	2.2	62	20	8.1
36	-5.4334	1.3e-01	1.3e-01	47.9	1.7	63	19	8.1
37	-5.5437	1.2e-01	1.2e-01	14.4	1.6	64	18	8.4
38	-5.5917	5.7e-02	5.7e-02	7.4	1.6	65	17	8.6
39	-5.6955	2.2e-01	2.2e-01	7.3	1.7	66	16	8.7
40	-6.3047	4.0e-01	4.0e-01	2.8	2.3	67	15	9.3
41	-6.3994	4.9e-02	4.9e-02	3.7	1.9	68	14	14.0
42	-6.6527	1.4e-01	1.4e-01	3.8	1.8	69	13	14.9
43	-6.8283	1.0e-01	1.0e-01	23.6	1.8	70	12	19.2
44	-6.9908	1.0e-01	1.0e-01	215.1	0.1	71	11	23.4
45	-6.9940	4.6e-02	4.6e-02	1.3	0.1	72	10	28.4
46	-7.0875	5.3e-01	5.3e-01	0.9	0.3	73	9	28.1
47	-7.1462	4.3e-01	4.3e-01	0.8	0.4	74	8	27.1
48	-7.2758	2.7e-01	2.7e-01	2.0	0.6	75	7	29.7

```
Rounded iterate is solution
ans =

  -14.0000
```

```
Hamilitonian cycle exists!

e = (i,j)
6 = (5,4)
7 = (6,1)
18 = (10,7)
22 = (11,5)
28 = (12,3)
36 = (13,12)
37 = (14,2)
49 = (2,6)
55 = (4,8)
57 = (3,9)
60 = (8,10)
66 = (9,11)
73 = (1,13)
80 = (7,14)
```

A script, *HCscript* was written to run the simulation $n = 20$ times and count the number of successes. Each time a new random non-regular Hamiltonian graph or random cubic graph is generated. It is evident from the below table that as the graph gets larger the problem becomes harder to solve. This is expected, since the number of ways it can go wrong increases and find local minimum for the stochastic constraints increases. It is important to node that we are solving an NP- complete problem.

.

Table 1: Various Constraints Results

Constraints	10 node cubic	16 node non-regular	20 node cubic
Stochastic	70%	65%	60%
Doubly Stochastic	85%	75%	70%
Additional Constraints	75%	70%	65%

4.6 Observations

4.6.1 Deletion of Constraints

It was noticed that in a majority of the cases run that the algorithm finds a solution without needing to delete a constraint. In the stochastic cases, where it fails to find the global minimum, it achieves subspace convergence and find that all the possible lagrange multipliers that can be deleted are positive and so it terminates. In cases, where constraints are deleted it is usually near the end of the simulation and only one or two constraints are deleted. Early termination also occurs due to rounding.

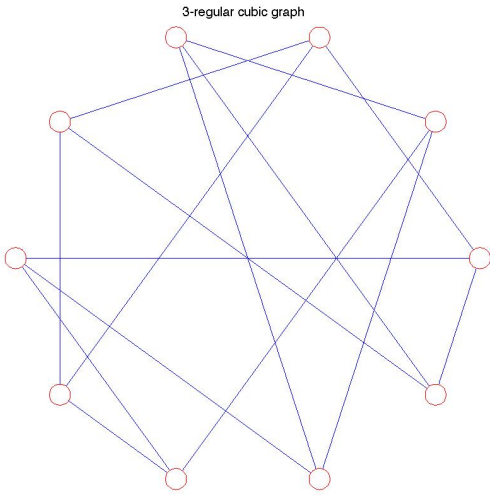
4.6.2 Choice of Using Directions of Negative Curvature and Modified Newton direction

The effect of using the direction of negative curvature ,d_k, as the search direction, the Modified Newton direction, q_k, or any linear combination of the two. Using just q_k will work, but it finds an answer more slowly. Contrary to the general problem, the best method seemed to be to use d_k whenever it was available and not even a linear combination of the two. Thus, the method used to generate the results always use d_k as the descent direction, whenever it is available.

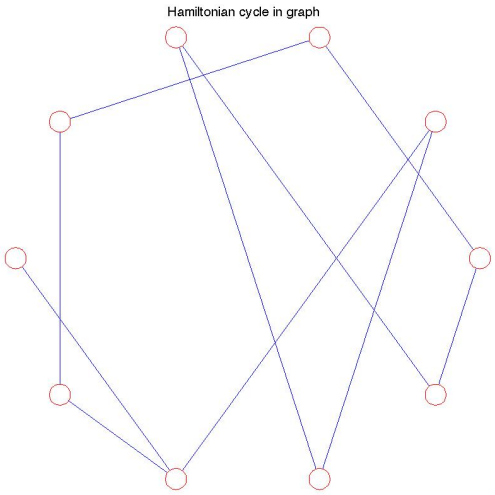
4.6.3 Variations in Performance with Different Constraint Sets

We can observe that the stochastic constraints can get stuck at a local minimum, as evident in the simple 10 node example for the graph corresponding to the incidence matrix in the Generating Graphs section. Here we see that the column sum of P is not 0 and so the sum of the incoming edges to a given node is not necessarily 1. Without this enforcement, certain situations, like the one below can occur.

It is clear that node 8 is the problem in the first case, since it has two incoming edges, leaving node 6 without any incoming edges, as revealed in the below plots and directed edge sets.



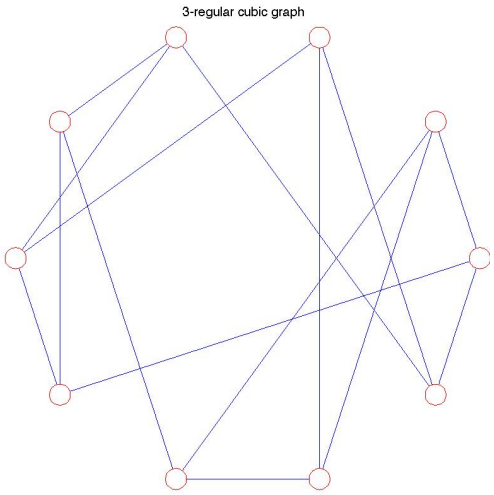
(a) Initial cubic graph with 10 nodes



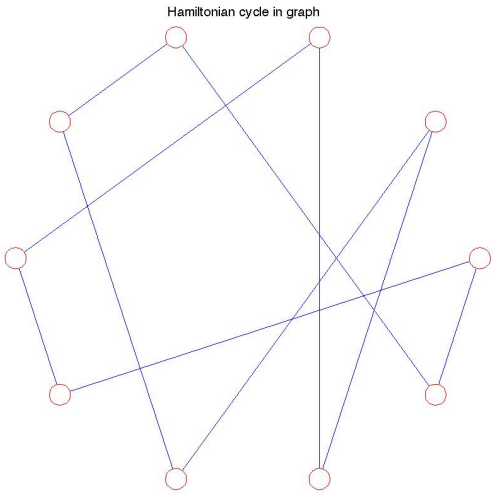
(b) Corresponding Hamiltonian Cycle

Figure 4: 10 Node Cubic Graph Failure Case using the Stochastic Constraints

```
ans =

   -9.0000

Did not find a Hamilitonian cycle

e = (i,j)
1 = (3,1)
3 = (5,3)
6 = (7,5)
9 = (8,7)
10 = (9,2)
14 = (10,4)
22 = (2,8)
23 = (6,8)
26 = (4,9)
28 = (1,10)
columns of P do not sum to 1
```



(a) Initial cubic graph with 10 nodes



(b) Corresponding Hamiltonian Cycle

Figure 5: 10 Node Cubic Graph Success Case using the Stochastic Constraints

```
k        phi      alpha      alpha_m       ||p||     ||Z^Tg||     m    null   cond(H)
1      -1.7862   9.2e-01    9.2e-01        0.4        2.1       11     19      6.7
2      -1.9193   7.2e-02    7.2e-02        2.4        2.0       12     18     27.1
3      -3.2552   6.0e-01    6.0e-01        9.0        3.8       13     17     21.2
4      -3.3198   1.8e-02    1.8e-02        8.1        3.7       14     16     12.8
5      -3.3780   1.8e-02    1.8e-02        6.2        3.5       15     15     12.6
6      -3.4288   1.6e-02    1.6e-02        9.3        3.2       16     14     12.5
7      -3.9888   3.7e-01    3.7e-01        7.4        3.6       17     13     12.4
8      -4.2554   1.6e-01    1.6e-01       14.6        3.6       18     12     11.4
9      -4.8245   4.0e-01    4.0e-01        5.7        3.8       19     11     11.1
10     -4.9050   3.2e-02    3.2e-02       11.7        3.6       20     10     10.6
Rounded iterate is solution
ans =

  -10.0000

Hamilitonian cycle exists!

e = (i,j)
2 = (5,4)
3 = (6,3)
7 = (7,6)
9 = (8,5)
10 = (9,2)
13 = (10,1)
20 = (1,7)
23 = (2,8)
26 = (3,9)
30 = (4,10)
```

In the stochastic case, we are always terminating at a vertex, whereas in the doubly stochastic case that is not the case. Note that rounding in the doubly stochastic case is not always defined. A clear benefit to the doubly stochastic case is that it rules out cases like the above failure example, by forcing the column sum to be 1, it does not get stuck at any local minima.

Moreover, one reason why the doubly stochastic constraints have higher performance and are more efficient is that a different formulation of the objective can be obtained for the doubly stochastic constraints. Evaluating the determinants in φ makes it run very slow and difficult for testing.

It can be shown for $x \in DS$ and its components from from zero, that is, in the relative interior rather than on the boundary, as we have enforced that only the leading principal minor of $I - P$ is needed to compute the objective function and so we may rewrite the objective function as given below:

$$\varphi(x) = -N\det(G^{NN}(x)), \tag{11}$$

where $G(x) = I - P(x)$ and G^{NN} is G with row N and column N removed. This can be computed using an LU factorization $G = LU$ without permutations and so we disable the permutations in matlab's LU solver and so we get:

$$\varphi = -N \prod_{i=1}^{N-1} u_{ii} \tag{12}$$

The gradients and Hessians are also given in the problem assignment and can be computed efficiently, without determinants, by doing triangular solves. Note that finite differences was again utilized to confirm that they were computed correctly.

5 Conclusions

In conclusion, this report and [1] show an exciting and promising applications of nonlinear optimization in solving certain discrete NP-complete problems. The Hamiltonian Cycle problem has an interesting connection to the famous traveling salesman (TSP). As detailed in the report, along the way certain numerical difficulties and challenges were encountered that would not be represented in the theory that assumes a machine with infinite precision. Problems from machine precision and floating point error had to be handled, such as in the termination requirement and size of the smallest lagrange multiplier. The key for testing that the entire problem works was to break it up into small and simple submodules and test those individually, as we have done. Thus, these small submodules and also the use of throwing exceptions are essential in the debugging process. It was very interesting to use the methods we learned in class to solve a practical problem and see the complications that arise and how to handle them.

References

[1] Michael Haythorpe. *Markov Chain Based Algorithms for the Hamiltonian Cycle Problem*, PhD thesis, University of South Austrailia, July 2012. http://www.stanford.edu/group/SOL/dissertations/michael-haythorpe-thesis.pdf

[2] P. E. Gill, W. Murray, and M. H. Wright, Practical Optimization, Academic Press.

[3] J. Nocedal, S. J. Wright, Numerical Optimization, Springer Verlag.

[4] Raphael Hauser. *Line Search Methods for Unconstrained Optimisation.*, Lecture 8, Numerical Linear Algebra and Optimisation, Oxford University Computing Laboratory, MT 2007. https://people.maths.ox.ac.uk/hauser/hauser_lecture2.pdf

[5] Edward R. Scheinerman. *Matgraph: A Matlab Toolbox for Graph Theory.* http://www.ams.jhu.edu/ ers/matgraph/matgraph.pdf

[6] Gerhard Dangelmay. *2D Newton's and Steepest Descent Methods in Matlab.* http://www.math.colostate.edu/ gerhard/classes/331/lab/newton2d.html