NUMERICAL LINEAR ALGEBRA

Project 1: direct methods in optimization with constraints

T1: Show that the predictor steps reduces to solve a linear system with matrix MKKT.

$$\mathrm{F}: R^N \to R^N,\ \mathrm{N} = \mathrm{n} + \mathrm{p} + 2\mathrm{m}$$

where $\mathrm{F(z)} = \mathrm{F}(\mathrm{x}, \gamma, \lambda, \mathrm{s}) = \frac{1}{2}\mathrm{x}^{\mathrm{T}}\mathrm{Gx} + g^T\mathrm{x} - \gamma^{\mathrm{T}}(\mathrm{A}^{\mathrm{T}}\mathrm{x} - \mathrm{b}) - \lambda^{\mathrm{T}}(\mathrm{C}^{\mathrm{T}}\mathrm{x} - \mathrm{d} - \mathrm{s})$

The optimization problem $\mathrm{F(z)} = 0$, will be solved using the Newton method.

Random point $\mathrm{z}_0 = (\mathrm{x}_0, \gamma_0, \lambda_0, \mathrm{s}_0)$

In every iteration we calculate a correction term δ_z to iteratively improve the guess for z - by using the below formula:

$$\mathrm{F}(\mathrm{z}_0 + \delta_z) \approx \mathrm{F}(\mathrm{z}_0) + \mathrm{J}_{\mathrm{F}}(\mathrm{z}_0)\,\delta_z$$

where $\mathrm{J}_{\mathrm{F}(\mathrm{z}_0)}$ is the Jacobean matrix of F with respect to z.

Hence, we need to solve the following equation:

$$-\mathrm{F}(\mathrm{z}_0) = \mathrm{J}_{\mathrm{F}}(\mathrm{z}_0)\delta_z$$

This equation is the KKT conditions for the optimization problem.

The matrix MKKT id defined as:

$$M_{KKT} = \mathrm{J}_{\mathrm{F}}(\mathrm{z}_0) = \begin{bmatrix} \frac{\partial F_1}{\partial x} & \frac{\partial F_1}{\partial \gamma} & \frac{\partial F_1}{\partial \lambda} & \frac{\partial F_1}{\partial s} \\ \frac{\partial F_2}{\partial x} & \frac{\partial F_2}{\partial \gamma} & \frac{\partial F_2}{\partial \lambda} & \frac{\partial F_2}{\partial s} \\ \frac{\partial F_3}{\partial x} & \frac{\partial F_3}{\partial \gamma} & \frac{\partial F_3}{\partial \lambda} & \frac{\partial F_3}{\partial s} \\ \frac{\partial F_4}{\partial x} & \frac{\partial F_4}{\partial \gamma} & \frac{\partial F_4}{\partial \lambda} & \frac{\partial F_4}{\partial s} \end{bmatrix} = \begin{bmatrix} G & -A & -C & 0 \\ -A^T & 0 & 0 & 0 \\ -C^T & 0 & 0 & I \\ 0 & 0 & S & \Lambda \end{bmatrix}$$

It is necessary to solve the linear system:

$$M_{KKT} * \delta_z = -F(z_0)$$

where $\delta_z = x_{k+1} - x_k$ the step between the points of the iterations.

We update the guess for z, after obtaining δ_z:

$$z_{k+1} = z_k + 0.95 * \alpha_k * \delta_z$$

a_k is a step size that we can adapt or reduce in each iteration to ensure convergence.

M_{KKT} captures how constraints and objectives respond to variable changes - hence it is important in the Newton method.

We come to a solution that meets KKT conditions and optimises the problem after we solve the linear system $M_{KKT} * \delta_z = -F(z_0)$.

C1: Write down a routine function that implements the step-size substep**.**

In the Python file included with the submission (under 'C1').

C2: Write down a program that, for a given n, implements the full algorithm for the test problem. Use the `numpy.linalg.solve` function to solve the KKT linear systems of the predictor and corrector substeps directly.

In the Python file included with the submission (under 'C2').

C3: Write a modification of the previous program C2 to report the computation time of the solution of the test problem for different dimensions n.

In the Python file included with the submission (under 'C3').

With a threshold of 100 iterations, dimensions of matrix 5*5 and epsilon=10e-16, we get the following:

```
The computational time for the test problem is equal to:
0.0038001537322998047
```

```
The minimum of the function was found: -7.552407424433394
The real minimum is: -7.552407424433394
Iterations needed: 13
Condition number: 23.73915310201455
```

It can be observed that the computational time is small.

T2: Explain the previous derivations of the different strategies and justify under which assumptions they can be applied.

- *Strategy 1*

Isolating δ_s from the 3rd row of the $\mathrm{M_{KKT}}$ matrix:

$$\delta_s = \Lambda^{-1}(-r_3 - Sd_\lambda)$$

Hence:

$$\begin{pmatrix}-C^T & 0 & 1\end{pmatrix}(\delta_x\ \delta_\lambda\ \delta_s)^T = -r_2 \quad => \quad -C^T\delta_x + \delta_s = -r_2$$

Substituting δ_s from the previous equality:

$$-C^T\delta_x - \Lambda^{-1}(-r_3 - Sd_\lambda) = -r_2$$

$$=> \quad -C^T\delta_x - \Lambda^{-1}(Sd_\lambda) = (-r_2 - \Lambda^{-1}r_3)$$

Λ^{-1} must be available and invertible to be able to solve this equation.

- *Strategy 2*

Again isolating δ_s from the 2nd row of $\mathrm{M_{KKT}}$ matrix:

$$\delta_s = -r_2 + C^T\delta_x$$

Substituting this expression for δ_s into the 3rd row:

$$S\delta_s + \Lambda\delta_s = -r_3$$

$$=> \quad S\delta_s + \Lambda\left(-r_2 + C^T\delta_x\right) = -r_3$$

$$=> \quad \delta_s = S^{-1}\left(-r_3 + \Lambda r_2\right) - S^{-1}\Lambda C^T\delta_x$$

The last step is to substitute these expressions into the 1st row to have a smaller linear system:

$$\hat{G}\delta_x = -r_1 - \hat{r}$$

By using the Cholesky factorisation, the smaller system can be solved, assuming Λ is available and S is invertible.

C4: Write down two programs (modifications of C2) that solve the optimization problem for the test problem using the previous strategies. Report the computational time for different values of n and compare with the results in C3.

In the Python file included with the submission (under 'C4').

Algorithm	N (dimension)	Iterations	Time consumption	Condition number
C3	5	13	0.0025	23.74
C4_LDL	5	13	0.0042	25638650657811.87
C4_Cholesky	5	13	0.0019	1.00

Algorithm	N (dimension)	Iterations	Time consumption	Condition number
C3	50	15	0.0200	24.33
C4_LDL	50	15	0.01615	7728249045603.148
C4_Cholesky	50	15	0.00863	1.00

Algorithm	N (dimension)	Iterations	Time consumption	Condition number
C3	100	14	0.0588	24.77
C4_LDL	100	14	0.0711	23625920228780.55
C4_Cholesky	100	14	0.01415	1.00

It can be observed that the C4 Cholesky method has the smallest computation time regardless of the value of n.

Also, the C4 Cholesky method gives a condition number of 1 in all cases, meaning the Cholesky factorization method gives stable results.

C5: Write down a program that solves the optimization problem for the general case. Use numpy.linalg.solve function. Read the data of the optimization problems from the files (available at the Campus Virtual). Each problem consists on a collection of files: G.dad, g.dad, A.dad, b.dad, C.dad and d.dad. They contain the corresponding data in coordinate format.

In the Python file included with the submission (under 'C4').

Results from the code when using matrices and vectors in 'optpr1' and 'optpr2':

optpr1

```
Computation time:  1.4867184162139893
Minimum was found: 11590.718119426767
Condition number: 1.9495161760426004e+18
Iterations needed: 25
```

optpr2

```
Computation time:  345.94063329696655
```

```
Minimum was found: 1087511.5673215014

Condition number: 5.554413718885413e+18

Iterations needed: 28
```

1. Optpr1 has a lower minimum objective function value than optpr2 => it is a more simple optimization problem to solve.
2. Optpr2 requires a lot more computational time than optpr1 => optpr2 is more heavy computationally.
3. The condition number of the KKT matrix for optpr2 is much higher than that of optpr1 => optpr2 is a more ill-conditioned problem thereby making the optimization more difficult.
4. optpr2 requires slightly more iterations to reach convergence than optpr1, even though both problems converge within the specified tolerance

T3: Isolate δs from the 4th row of MKKT and substitute into the 3rd row. Justify that this procedure leads to a linear system with a symmetric matrix.

Equations for the general case:

Eq 1: $G\,\delta_x - A\delta_\lambda - C\,\delta_\lambda = -r_1$

Eq 2: $-A\,\delta_x = -r_2$

Eq 3: $\delta_s - C^T\delta_x = -r_3$

Eq 4: $\Lambda\delta_s + S\,\delta_\lambda = -r_4$

Following the same procedure as in T2, except that δ_s is isolated from the 4th row instead of the 2nd and 3rd row. Resulting to the following equation for δ_s:

$$\delta_s = \; - \; \Lambda^{-1}(r_4 + S\delta_\lambda)$$

By substituting δ_s in the 3rd row, we get the following equality:

$$-C^T\delta_x - \; \Lambda^{-1}\left(r_4 + S\delta_\lambda\right) = \; - r_3$$

The following is the result when converting to a matrix form:

$$\begin{pmatrix} G & -A & -C \\ -A^T & 0 & 0 \\ -C^T & 0 & -S\Lambda^{-1} \end{pmatrix} (\delta_x \quad \delta_\lambda \quad \delta_s)^T = \; (-r_1 \quad -r_2 \quad \Lambda^{-1}r_4 - r_3)^T$$

Hence, the final matrix is symmetric since it is guaranteed by the properties of the matrices involved.

The matrix $-S\Lambda^{-1}$is always symmetric because it arises from the product of two diagonal matrices.

C6: Implement a routine that uses LDLT to solve the optimizations problems (in C5) and compare the results.

In the Python file included with the submission (under 'C4').

Results from the code when using matrices and vectors in 'optpr1' and 'optpr2':

optpr1

```
Computation time:  1.833892583847046
Minimum was found: 11590.718119426772
Condition number: 6.375699758041506e+21
Iterations needed: 31
```

optpr2

```
Computation time:  545.8264486789703
Minimum was found: 1087511.5673214972
Condition number: 8.859261568310938e+22
Iterations needed: 34
```

In the C5 algorithm, the numpy.linalg.solve factorization function is used.

In the C6 algorithm, the LDLT factorization function is used.

Comparing these two:

1. The computation times were shorted for C5 than C6 (since the extra LDLT factorisation steps are computationally costly)
2. The minimum values in C5 and C6 algorithms are similar, which shows that both methods converged to the same optima.
3. The condition number of the KKT matrix in these 2 algorithms is very different. C6 algorithm gives a higher condition number, which probably means that the matrix is ill-conditioned and that small changes in the input data can result to big changes in the solution. Hence, the C6 algorithm might produce less stable solutions in some cases.
4. The C6 algorithm needed more iterations to converge compared to C5 algorithm. This might be because of the extra LDLT factorization and step-size correction substeps in C6. With more iterations we get higher computation times.

In conclusion, numpy.linalg.solve factorization and LDLT factorization functions have different properties but are both able to solve optimization problems.

C5 algorithm is faster and it can be more suitable where computational speed is a priority. The C6 algorithm utilizes LDLT factorization, which appears to be more numerically stable, so it it may be more suitable for problems where the condition number of the KKT matrix is a concern.