Morenet Project

Xinliang Dai

Jan 2020 - Sept 2020

Contents

1 Introduction

MOReNet (Modellierung, Optimierung und Regelung von Netzwerken heterogener Energiesysteme mit volatiler erneuerbarer Energieerzeugung) is a German research project supported by the Federal Ministry of Education and Research (BMBF). Project execution organisation is DESY (Hamburg). Our task is to improve the existing algorithms to provide a prototype software for a decentralized load flow calculation on realistic CGMs. This solution will be accompanied by a concept that describes how such a research result can become operational in daily TSO-DSO business.

This report describes the modeling and simulation approach, during the HiWi-job in Institute for Automation and Applied Informatics (IAI) and internship in TransnetBW. The purpose here is to investigate a suitable algorithm to solve power flow problem within a reasonable time, base on the data file from MATPOWER toolbox[5]. The idea comes from Tillmann Mühlpfordt, to transform original power flow problem into a least-squares problem.

1.1 Challenges in large-scale problem

Within the Morenet project, KIT is focusing on distributed optimization algorithms to solve the non-convex nonlinear constrained problem. One of the challenges is the problem size. When the number of nodes in model becomes increasingly large, finding the solution becomes difficult: it either takes too long to run the algorithm or results in a convergence failure.

To solve this, common options are either to improve algorithm or select a more suitable solver. In decentralized power flow problem, its local objective is set to zero, and thus there is an third option: transform the feasibility problem into a least-squares problem. Doing so, the original constrained problem becomes an unconstrained problem, which is supposed to reduce computation cost dramatically.

1.2 Challenges in realistic model

Another challenge is building an realistic model. In real world, electrical power system consists of transmission system and distribution system. It is supposed to observe a significant large power flow from transmission side to distribution side, via transformer.

In Morenet project, a case file, built by data file from MATPOWER toolbox, represents a transmission/distribution system. Several case files are merged into one file to represent the whole eletrical power system. However, the data file describes a sort of self-sufficiency system, and its power flow between transmission and distribution is comparable small, which is not tally with reality.

Our solution is to create a new feature "generation shift key", so that the generated power in distribution systems could be "shifted" to transmission system.

2 Problem Formulation

This section focuses on mathematical formulations of the problem. In section 2.1, the original power flow problem is presented at first. In order to solve it in a distributed way, the original problem would be formulated in a feasibility problem in section 2.2. In section 2.3, by examining Least-squares formulation, we hope to transform the constrained problem into a unconstrained problem to save computation cost. Last but no least, in section 2.4, sensitivities, especially of least-squares problem, are investigated, to further reduce computation burden.

2.1 Original Problem

The original decentralized power flow problem could be written as:

$$\begin{aligned} g_i^{\text{pf}}(x_i, z_i) &= 0, && \text{(1a)} \\ g_i^{\text{bus}}(x_i) &= 0, && \text{(1b)} \\ \sum_{i=1}^{n^{\text{reg}}} A_i \begin{bmatrix} x_i \\ z_i \end{bmatrix} &= 0, && \text{(1c)} \end{aligned}$$

where $g^{pf} = 0$ constitute the power flow equation, while $g^{bus} = 0$ stands for the remaining bus specification. x_i represents the state of core nodes and z_i contains the state of copy nodes in i^{th} local region. A_i represents local consensus matrix.

2.2 Feasibility Problem

Firstly, the problem is solved as a distributed feasibility problem. The mathematical formulation reads:

$$\begin{aligned} \min_{x_i, z_i \ \forall i \in \{1, \dots, n^{\text{reg}}\}} \quad 0 \quad \text{s.t.} & && \text{(2a)} \\ g_i^{\text{pf}}(x_i, z_i) &= 0, && \text{(2b)} \\ \sum_{i=1}^{n^{\text{reg}}} A_i \begin{bmatrix} x_i \\ z_i \end{bmatrix} &= 0. && \text{(2c)} \end{aligned}$$

considering it is a feasibility problem, the objective is set to zero.

2.3 Least-Square Problem

From our experience in previous research, feasibility problem might not be the best solution, because of the non-linear equality constraints, which would increase computation burden. Here we attempt to transform the problem into least-squares problem. The mathematical formulation reads:

$$\min_{x_i, z_i\, \forall i \in \{1,\dots,n^{\text{reg}}\}} \sum_{i=1}^{n^{\text{reg}}} \left\| \begin{bmatrix} g_i^{\text{pf}}(x_i, z_i) \\ g_i^{\text{bus}}(x_i) \end{bmatrix} \right\|^2 \quad \text{s.t.} \quad \sum_{i=1}^{n^{\text{reg}}} A_i \begin{bmatrix} x_i \\ z_i \end{bmatrix} = 0. \tag{3a}$$

Solution to **Problem 2** full-fills the **equality constrains 2b**. Clearly, it is also a solution to **Problem 3**. Therefore, we view solving the least-squares **Problem 3** as a necessary condition.

2.4 Sensitivities Computation

Due to **Sensitivity Documentation** in Morenet project, we focus on Gradeients, Jacobians and Hessians of each local power flow problem.

Gradient:	$g_i^k = \nabla f_i(x_i^k)$	(4a)
Jacobians	$\nabla h_i(x_i^k)$	(4b)
Hessian Approximations	$B_i^k \approx \nabla^2\{f_i(x_i^k) + \kappa_i^T h_i(x_i^k)\}$	(4c)

Here we focus on computing sensitivities in least-squares Problem:

1. Local Gradients

$$g_i^k = \nabla_{x_i} ||g_i^{LS}(x_i)||^2 = \nabla_{x_i}\{g_i^{LS}(x_i)^T g_i^{LS}(x_i)\} = 2J_i(x_i)^T g_i^{LS}(x_i) \tag{5}$$

   where $g_i^{LS}(x_i) = [g_i^{pf}(x_i), g_i^{bus}(x_i)]^T$ is a $(4N_{core} \times 1)$ vector. Notice that x_i here including states of both 'core' and 'copy' nodes, and $J_i(x_i)$ is the $(4N_{core} \times (4N_{core} + 2N_{copy}))$ matrix of first partial derivatives of $g_i^{LS}(x_i)$

2. Jacobian

   As the mathematical forulation of **Problem 3**, there is no equality constraints, which comes from local power flow problem. Only the consensus constraints remain. Thus, in each local NLP step, it is a unconstrained optimization problem.

3. Hessian Approximations

   Due to lack of equality constraints in least-squares problem, as described in previous section, the Hessian approximations would be simplified as:

$$B_i^k \approx \nabla^2\{f_i(x_i^k)\} \tag{6}$$

In a large-scale problem, obtaining an exact hessian is difficult. Thus inexact hessian approximations are preferred in morenet project. Four methods to approximate Hessian matrix are tested in this report. Firstly is BFGS, which is widely used in Hessian Approximation. However, in a large-scale problem, BFGS cannot compute the inexact hessian with a reasonable duration. Alternative, Limited-memory BFGS [3] reduces the computation burden and could save a lot of time.

The third method is Finte-Difference. The problem of this method is, if analytic derivatives are not able to be computed, the finite-difference approximations might not good enough to solve problem.

Also its performance in large-scale problem remains unknown.

Fortunately, by least-squares problem, we have an another option:

$$\nabla^2 f(x) = \sum_{j=1}^{m} \nabla r_j(x) \nabla r_j(x)^T + \sum_{j=1}^{m} r_j(x) \nabla^2 r_j(x) \tag{7a}$$

$$= J_k^T J_k + \sum_{j=1}^{m} r_j(x) \nabla^2 r_j(x) \tag{7b}$$

In our case:

$$r(x) = g_i^{LS}(x_i) \tag{8}$$

Because the first term $J_k^T J_k$ dominates the second term in **Eq 7b**, especially when x approach minimum x^* and the residual $r(x)$ are significant small [3]. Hence, we have:

$$\nabla^2 \{f_i(x_i^k)\} \approx J_i(x_i)^T J_i(x_i) \tag{9}$$

3 Algorithm & Solver

3.1 Algorithm

To solve the distributed power flow problem, two Algorithms have been implemented in this project, i.e. the Alternating Direction Method of Multipliers (ADMM) [4] and An Augmented Lagrangian Based Algorithm for Distributed Non-Convex Optimization (ALADIN) [1].

3.2 Solver

In the previous tasks, we have explored CasADi with Ipopt and small-scale problems, i.e. bus number less than 300, could be solved in a reasonable time. However, when it comes to medium-scale problems (Bus number less than 1000), it takes too long to load a problem by CasADi. Thus, we turn to other solvers here, to improve the ALADIN Toolbox in larger-scale problem.

Fmincon, a widely uesd matlab function, first comes to our minds, which is developed to solve constrained non-linear problem. WORHP, which is designed to solve large-scale problem, is another option. IPOPT is also worth a try. However, it is complicate to develop the environment in Windows OS. We would explore IPOPT in Linux OS in the future. KNITRO doesn't support academic-license, therefore it will not be covered here.

To have an intuitive sense, an algorithm tree is plotted in **Figure 1**.

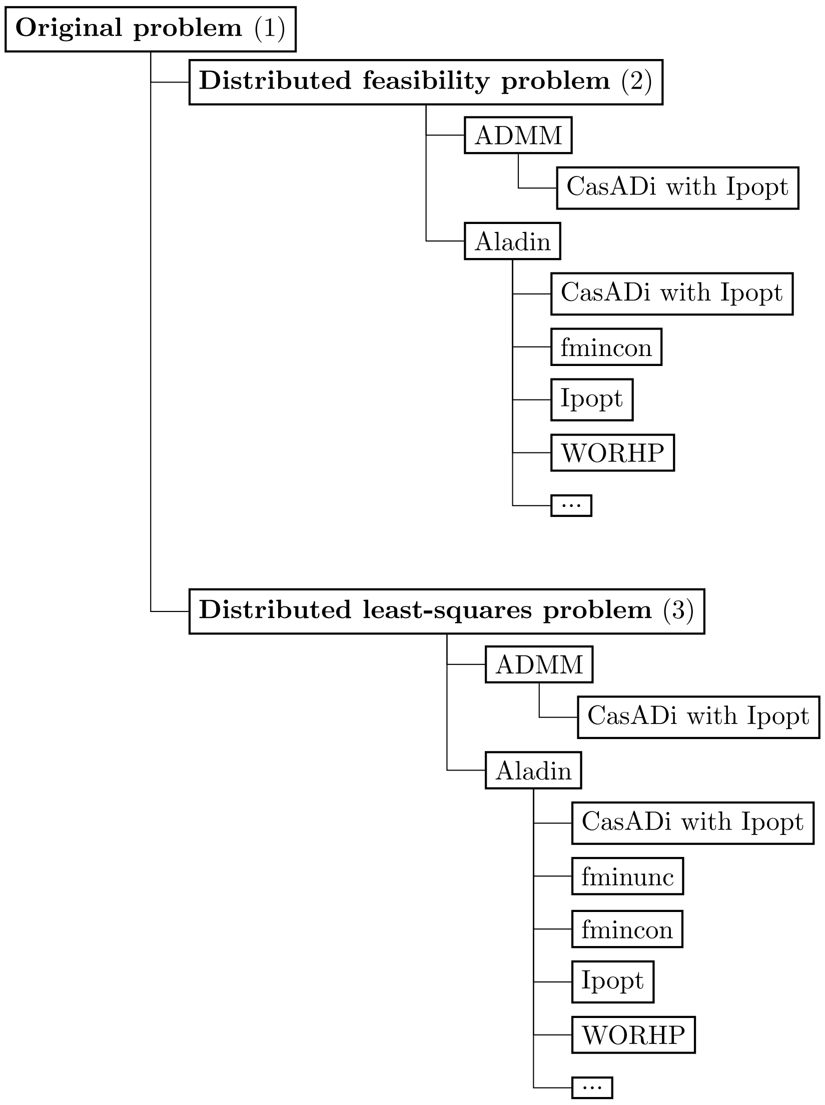
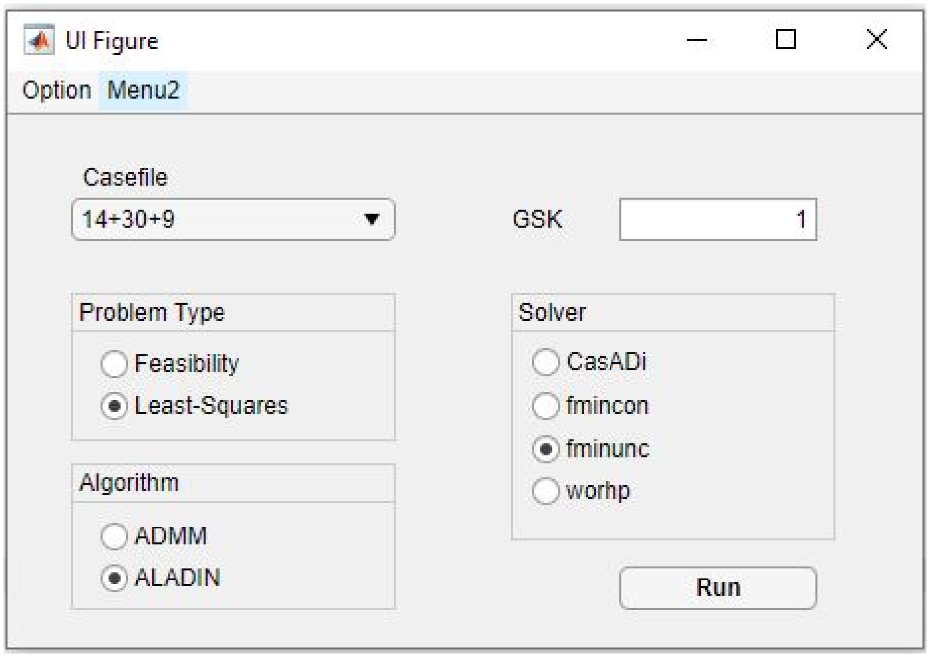


Figure 1: Algorithm Tree

4 Graphical User Interface (GUI)

To allow user interact with different algorithms and solvers in different problem formulation, a graphical user interface (GUI) is implemented, according the algorithm tree in **Figure 1**. **Figure 2** shows the GUI in morenet project.

Because the morenet project is on development, not all combinations listed are available. Check **Readme.md** for more details in current status.



Casefile: 14+30+9

GSK: 1

Problem Type: Feasibility [✓ Least-Squares]

Algorithm: ADMM [✓ ALADIN]

Solver: CasADi, fmincon, [✓ fminunc], worhp

Figure 2: :Graphical User Interface (GUI)

5 Simulation Result

5.1 Solve Feasibility Problem using ADMM

- Compare the results with different penalty parameter ρ

  ρ was set to 0.1,1,10,10^2,10^3,10^4,10^5 . The upper in **Figure 3** shows the step-length to the optimum point and lower one shows the violations in norm-inf:

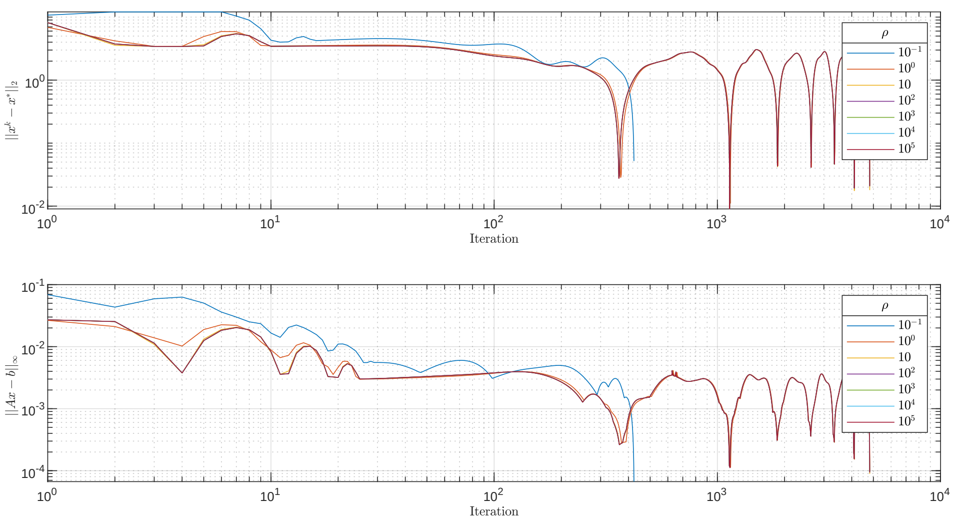


Figure 3: Comparison of different penalty parameter ρ

- Compare the results with different initial point

  Random initial points with fixed distance to the optimum, $\alpha = \|x_0 - x^*\|$, were created in this task. The fixed distance α were set to $0.01, 0.1, 3, 10$. Distance of each region was set to $\sqrt{\frac{\alpha}{N_{region}}}$, so that the distance of all state in norm-2 would be α

  The upper graphic in **Figure 4** shows the step-length to the optimum point and lower one shows the violations in norm-inf.

  The convergence rate depends on how far the initial point from the optimum is.

In Morenet cases, we face some trouble using ADMM to solve feasibility problem. Its performance depends on penalty parameter and it doesn't converge in most cases. Even it does converge, convergence rate is highly dependent from distance between optimum and initial point / initial guess. These increase challenges towards the realization in practice.

Hence, we stop investigating the feasibility problem with ADMM. Instead, we turns to ALADIN. In the following sections, problems are solved using ALADIN, which is proved to be very efficient in solving distributed non-convex optimization problem.
break

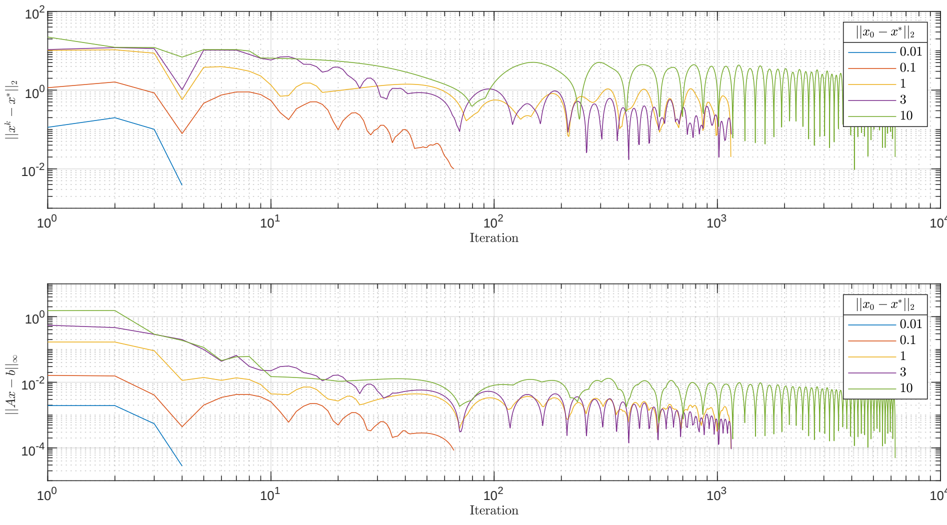


Figure 4: Comparison of different distance α between x_0 and x^*

5.2 Solve Least-Squares Problem using ALADIN

In earlier research, CasADi and fmincon are implemented to solve feasibility problem. However, it takes too long to run the algorithms, when the problem size goes from medium to large. To solve the problem withn a reasonable time, the problem type is transformed to least-squares problem. In **section 5.2.1**, several solvers are implemented in least-squares problem and their performance would be discuss. Later in **section 5.2.1**

5.2.1 Comparison of different Solvers

1. CasADi & IPOPT

   In earlier research, CasADi & IPOPT shows an ability to solve a small-to-medium size feasibility problems, namely the number of nodes is less than 300. With increasing number of nodes, setup time of a problem increase dramatically, while the duration of computing still remain short in contrast. Thus we believe CasADi is the main bottleneck. However, IPOPT is still worth a try.

2. fmincon & fminunc

   Figure 5 shows computing time of fmincon, with different scale cases. Unlike CasADi, the computing time has a linear growth, even when the number of nodes near 5000. Also notice that sensitivities are computing according to **Section 2.4** with least-squres formulation. This means Gradient, Jacobian and Hessian are user-defined function-handle in fmincon solver.

3. WORHP

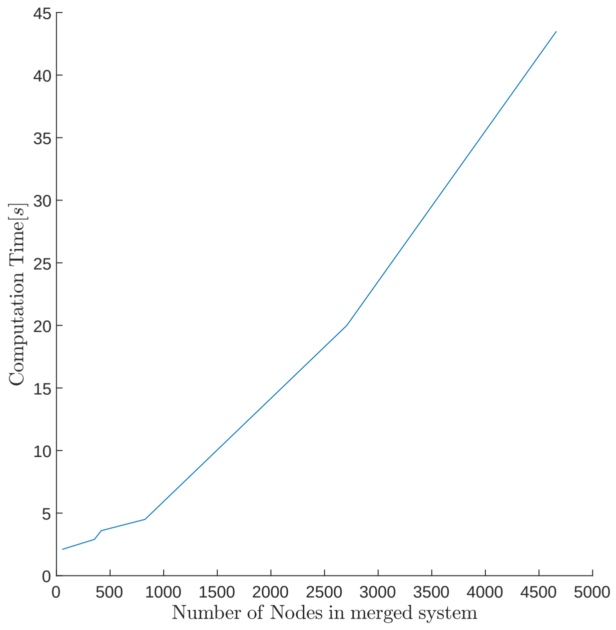


Figure 5: Performance of fmincon with Different Problem Size

WORHP is designed to solve large-scale problem and input data should be transformed in a specific sparse form. First challenges of WORHP is how to feed the Hessian-information. It requires a special sorting of the sparse entries. All entries on the diagonal must be given, even structural zeros. Furthermore, the non-diagonal entries must be given first. [2] So the correct order of elements is supposed to be:

- Nonzero elements in *lower triangular matrix*;
- All diagonal elements, including zero element.

In our model, sensitivities are saved in function-handle, a type of matlab data. After transformation, function-handle becomes complicated, and it would results in an error by WORHP. An intuitive solution is, instead of function-handle, hessian information is supported by a numerical matrix, when it is called by solver. Following code is an example of how it is done:

```
/* wCallback    :          input struct for WORHP in matlab-interface
   build_Hess_vector():   the function to transform hessian matrix into vector   */

Callback.hm           = @(x)build_Hess_vector(Hess.Func(x)...);
```

4. IPOPT

   Installation of development environment for IPOPT in Windows OS is too complicated and the guides online are out-of-date. So we planned to implement it in Linux OS. Now, a Linux system, Ubuntu, has been created in VirtualBox and Maltab and WORHP has been installed. However, there are other prior tasks to do, this task still remain unfinished.

Simulation results of solving least-squares problem are showed in **Talbe 1**. By the problem with 418 nodes, system consists of a medium-size subsystem with 300 nodes. Meanwhile, the largest two problem, i.e. 2708 nodes and 4662 nodes, both include at least one large subsystem with more than 1000 nodes. It exhibited that size of subsystem has an critical influence of computation burden.

Number of Nodes	fminunc [s]	fmincon [s]	worhp [s]
53	2.5	2.2	2.4
354	2.5	3.1	4.8
418	4.5	5.2	7.0
826	3.7	5.3	7.2
1180	4.9	6.7	9.8
2708	212.7	41.9	53.6
4662	387.9	90.1	113.8

Table 1: Computing Time Compared with Different Solvers

In aspect of number of convergence rate, algorithm with all three different solver converges in the same iterations by a same problem. Compared with fmincon, fminunc is slightly faster. However, the computing time increases dramatically, when the local problem size is large. In general, fminunc is fastest solver for small-size problem, but is not able to handle large-scale problem.

On the contrary, fmincon and worhp are qualified even when the number of nodes increases to 1000+. During the simulation, it is observed that the setup-time of worhp is slower than fmincon. One of the reason could be that fmincon is a matlab-rooted solver, while worhp is coded in cpp and requires a worhp-matlab-interface to run in matlab. However, due to sparse form of data, worhp is supposed to have an ability to find minimum within a reasonable cost, when the problem size keep growing.

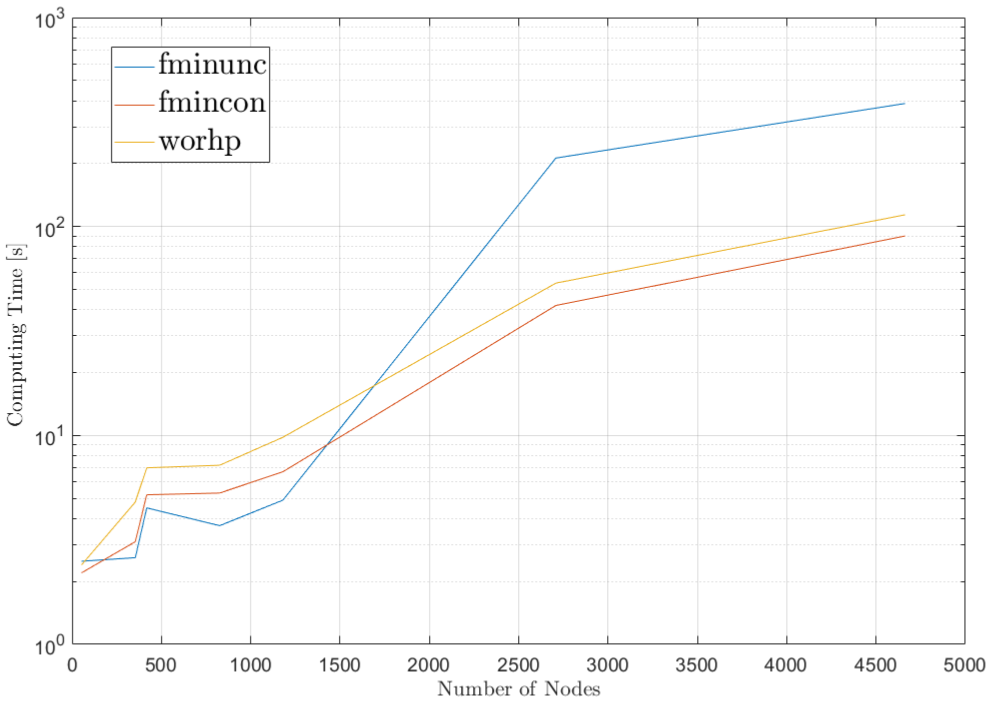


Figure 6: computing time by different solvers

5.2.2 Comparison of different Hessian Approximation Strategies

As described in **Section 2.4**, four methods are used to approximate Hessian, i.e. classic BFGS, limited-memory BFGS, Finite-Difference and approximation by Jacobian matrix $J_i(x_i)^T J_i(x_i)$

As shown in **Table 2** and **Figure 7**, approximation by Jacobian matrix has an amazing performance in all examples. In the meantime, the other three methods face convergence problem, when the problem size surpass 418 nodes.

Notice that, computing time of BFGS is intolerant, when the problem size of subsystem become medium (300 Nodes).

Number of Nodes	BFGS [s]	limit-memory BFGS [s]	Finite-difference [s]	user-defined [s]
53	28.6	22.9	10.0	2.2
354	287.8	107.4	61.5	3.1
418	1086.4	148.2	185.6	5.2

Table 2: Computing time compared with different Hessian approximations

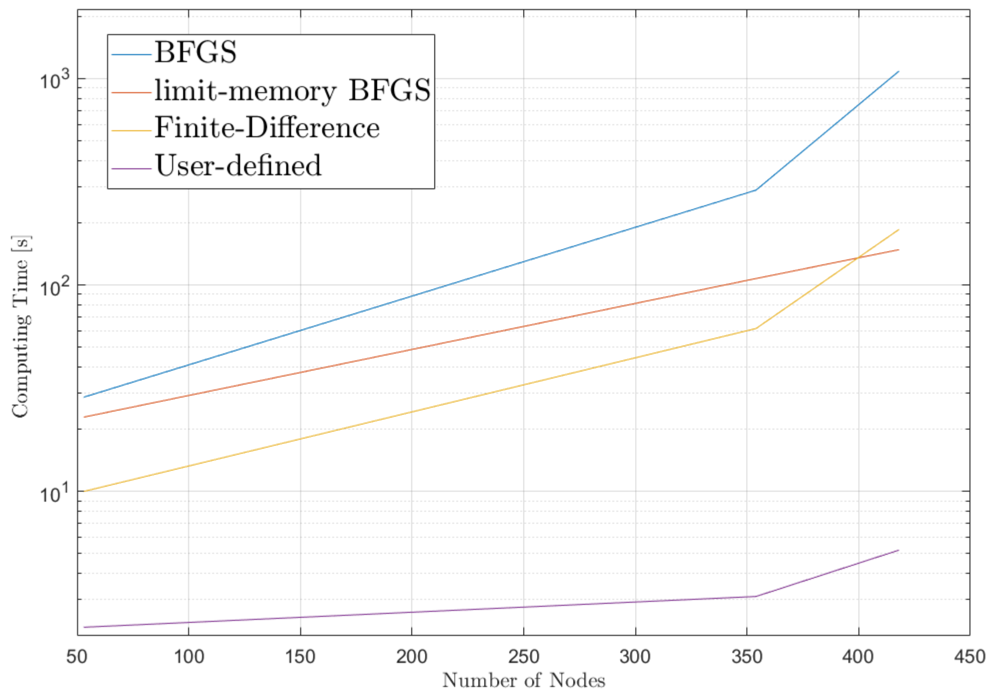
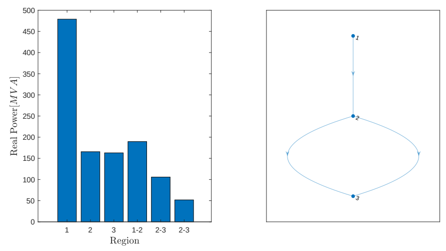


Figure 7: computing time by different hessian approximations

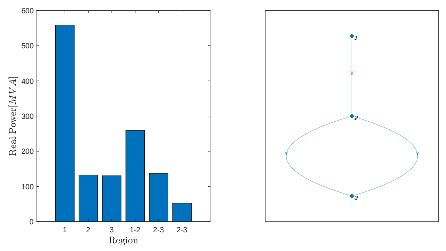
5.3 Generation Shift Key (GSK)

In the example case, there are three region, i.e. 1, 2 and 3. Region 1 represents transmission system, while Region 2/3 represent two distribution systems. There are a transformer connected between Region 1 and Region 2. Meanwhile, there are two transformer connected between Region 2 and Region 3.

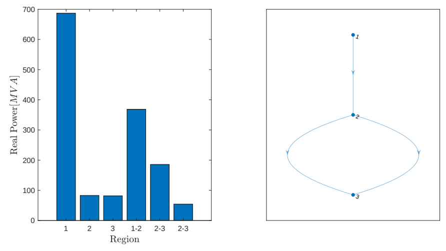
In each graphs in **Figure 8**, the first three bars represent a sum of generated real power in each region respectively, while 'i-j' bar represent the power flow between Region i and Region j. When the value is positive, power flow has the same direction as the arrow, vice versa.



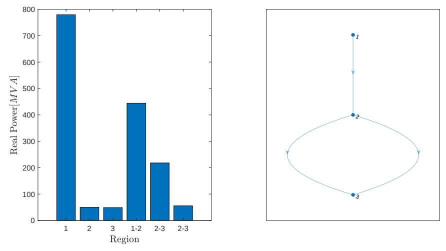
(a) Remain unchanged



(b) Reduce to 80%



(c) Reduce to 50%



(d) Reduce to 30%

Figure 8: Reduction of Generated Power in Transmissions

This new feature only works for small problem currently. For large-scale, i.e. number of nodes is more than 300, reduction will lead to infeasibility, when runpf() is called to validate the merged model. Even reduction to 90% doesn't work. It is supposed that, although reduction is small, the power flow between regions would increase dramatically because of larege number of nodes in total and violate constrains, e.g. line limit, which leads to infeasibility.

6 Conclusion & Future Work

Although ADMM is proved to be not suitable for solving feasibility problem, investigating ADMM solving least-squares problem is still worthy, because the local cost function has been changed and there is no more equality constraints in local NLP step.

In aspect of Solver, approximating Hessian by $J_i(x_i)^T J_i(x_i)$ is a better solution, when the problem formulation changed to least-sqaures. On the one hand, it could decrease computation burden. On the other hand the approximation becomes more and more accurate, when x approach minimum. However, for other forms of problem, computing or approximating Hessian is still a challenge, especially for a large-scale problem. In these case, limit-memory BFGS is worth trying. Another possible solution to large-scale problem, is to further partition the system, so that the local region size remains small.

Since Morenet project is based on matlab currently, fmincon, the matlab rooted funtion, is proved to be a more efficient solver and maintains a relative high performance for large-scale problem. However, this conclusion is highly platform dependent. If the project is planned to implement in practice, new research is needed to find out a suitable solver.

Meanwhile, there are still some tasks to be finished:

- solve Least-sqaures problem by ADMM and compare with the results by ALADIN
- implement solver IPOPT in Linux OS

References

[1] J. F. Boris Houska and M. Diehl. An augmented lagrangian based algorithm for distribution nonconvex optimization. *SIAM Journal on Optimization*, 2016.

[2] C. Büskens and D. Wassel. The esa nlp solver worhp. *Modeling and Optimization in Space Engineering*, pages 85–110, 2013.

[3] J. Nocedal and S. J. Wright. *Numerical Optimization.* 2006.

[4] E. C. S Boyd, N Parikh. *Distributed optimization and statistical learning via the alternating direction method of multipliers.* 2011.

[5] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas. Matpower: Steady-state operations, planning, and analysis tools for power systems research and education. *IEEE Transactions on Power Systems*, 26(1):12–19, 2011.