厦門大學

本 科 毕 业 论 文 （设 计）

（主修 / 辅修专业）

面向 Web 服务的
微电网能量管理系统的设计与开发

Design and Development of Microgrid Energy Management System for Web Service

姓　　名：　　苏铃峰

学　　号：　　19920122203490

学　　院：　　航空航天学院

专　　业：　　仪表与电气系

年　　级：　　2012 级

校内指导教师：　　张景瑞　　助理教授

校外指导教师：　　(姓名)　　(职务)

二〇一六 年 五 月　　日

厦门大学本科学位论文诚信承诺书

本人呈交的学位论文是在导师指导下独立完成的研究成果。本人在论文写作中参考其他个人或集体已经发表的研究成果，均在文中以适当方式明确标明，并符合相关法律规范及《厦门大学本科毕业论文（设计）规范》。

该学位论文为（　　张景瑞　　）课题（组）的研究成果，获得（　　张景瑞　　）课题（组）经费或实验室的资助，在（　　嘉庚四 306　　）实验室完成（请在以上括号内填写课题或课题组负责人或实验室名称，未有此项声明内容的，可以不作特别声明）。

本人承诺辅修专业毕业论文（设计）（如有）的内容与主修专业不存在相同与相近情况。

学生声明（签名）：

年　　月　　日

致谢

四年的大学生活，数月的毕业设计准备，即将画上一个句号。在此，我要感谢所有曾经教导过我的老师和关心过我的同学，在四年的学习生活中，有各位老师孜孜不倦的教诲，有各位同学悉心的关照和包容。四年的大学生活不仅仅是丰厚知识的积累，更是思维方式、表达能力和为人处世能力的提高。感谢厦门大学给了我们舒适干净的生活学习环境，多元化的竞争平台，让我们更好地去锻炼自己，提升自己。

我要特别感谢张景瑞导师对我的关怀和指导。几个月来，经历了选题，开题答辩，中期答辩，到最后的毕业设计的完成，每个环节都缺少不了张老师的指导。毕业设计的完成过程，并不是那么顺利，其中碰到了许多困难。在碰到自己解决不了的问题时，我会和实验室的同学讨论，请教学长学姐。有时候问题超过了自己所掌握的知识范围，便只能向老师请教。每每这时，老师都能在身兼科研和教学压力下给我宝贵的建议。

这次毕业设计会使我终生受益，我也感受到了真真正正用心去专研一件事的收获是有多大。感谢张老师的指导，感谢实验室学长学姐还有同学对我的帮助。

摘要

随着微电网技术的不断发展，微电网能量管理系统也逐渐成为研究热点。能量管理系统的核心，就是基于系统设计对应的电力优化调度模型。而电力优化调度模型在数学上，是具有大量非线性约束条件以及包含连续和离散变量的大规模非线性优化问题。传统优化方法大都依赖于优化问题数学模型的可导性与可微性，在实际运用中具有一定的局限性。和传统算法相比，人工智能优化算法不要求目标函数和约束的连续性和凹凸性，具有更强的全局搜索能力，所以被广泛运用在电力优化求解中。然而，传统智能算法通常是串行的，不能充分利用多核现代高性能 CPU 和多任务操作系统。这对于能量管理系统来说，会造成运算时间的负担。

针对以上的问题，本文提出了一种基于 MPI 并行程序设计的粒子群优化算法，将原来的单进程串行任务，拆分为多进程的并行任务，大大提高了逻辑 CPU 的利用率，在不减弱算法全局搜索能力的情况下，提高了运算速度。

进而，设计了面向 Web 服务的能量管理系统。同时，通过客户端-服务器(Client/Server)主从式架构，将能量管理系统的电力优化程序和储存系统参数和算法参数的 MySQL 关系型数据库放在服务端，服务端通过基于 gSOAP 开发包的中间件，为客户端提供 Web 服务。

以 IEEE 九节点微电网系统模型，对风电、光伏、蓄电池互补的系统多时段电力优化问题进行分析，考虑潮流约束和功率约束，并与外部大电网进行并网交换。通过能量管理系统，实现各发电单元的出力分配，计算出系统最优的发电运行成本，并对出力分配和运行成本的函数图像进行简单的记录与分析。

[关键字] 能量管理系统 粒子群优化算法 MPI 并行计算

Abstract

With the development of microgrid technology, the energy management system of microgrid has gradually become a hot topic. The core of the energy management system is a power optimization schedule model, which is based on the proper system. The optimal scheduling for power system is a large-scale-nonlinear optimization problem, which contains a number of nonlinear constraints, continuous variables and discrete variables. Traditional optimization methods rely on the mathematical model of optimization problems that have derivative and differential, which may have some limitations in practical use. Comparing with traditional algorithms, artificial intelligence optimization algorithms don't require the objective function and constraints which have continuity and irregularities. They have stronger global search ability and are widely used in the power optimization. However, most intelligent algorithms tend to get better global search capability by increasing the size of the population. It may cost more time for the energy management system to solve the problem.

To solve the above problems, this paper presents a small population particle swarm optimization algorithm based on MPI parallel program. It split the original single process task into parallel tasks running on multiple processes. It greatly improves the utilization of servers' logic CPU. It can improve the program speed without weakening the ability of global search.

So, we design a energy management system for Web service. The program of the system is based on the Client-Server(C/S) model. The server contains electric power optimizer and MySQL relational database. It can provide web service for different clients with the gSOAP toolkits.

The program chooses a nine-node-microgrid system of IEEE. It can analysis the multi-period optimization problem which contains wind power, photovoltaic power and battery power. Considering current constraints and power constraints, the microgrid system switches power with the external electric power system. The system achieves the output of each power generating unit. It can also calculate the optimal cost for power generation. The output of each power generating unit and the optimal cost can be described with images.

Keywords: Energy Management System(EMS), Particle Swarm Optimization(PSO), Message Passing Interface(MPI)

目录

第 1 章 引言 1
1.1. 课题研究的背景和意义 1
1.2. 微电网能量管理系统的研究现状 1
1.3. 微电网经济调度模型的分析 1
1.3.1. 传统优化算法 2
1.3.2. 人工智能优化算法 2
1.4. 面向 Web 服务 2
1.5. MPI 并行程序设计 3
1.6. 本文所做的工作 3
第 2 章 考虑功率约束与潮流约束的经济调度模型 5
2.1. 约束条件 5
2.1.1. 潮流约束 5
2.1.2. 功率约束 10
2.2. 目标函数 10
2.3. 经济调度模型 12
第 3 章 基于 MPI 并行计算的粒子群优化算法 13
3.1. 粒子群(PSO)优化算法的定义 13
3.2. 单进程下运用粒子群算法解决微电网系统经济调度问题 13
3.3. MPI 并行计算下运用粒子群算法解决微电网系统经济调度问题 15
3.3.1. MPI 并行设计分析 15

3.3.2. 粒子群算法的并行设计 18
3.3.3. 粒子群并行算法的性能 19
第 4 章 面向 Web 服务的微电网能量管理系统 21
4.1. **能量管理系统的构建与组成部分** 21
4.1.1. 如何构建一个能量管理系统 21
4.1.2. 能量管理系统的基本组成 22
4.1.3. 能量管理系统的工作原理 25
4.2. **能量管理系统的实例分析** 26
4.2.1. 微电网能量管理系统客户端 26
4.2.2. 微电网能量管理系统服务端 34
第 5 章 IEEE 九节点模型算例分析 37
5.1. **IEEE 九节点微电网系统模型** 37
5.2. **IEEE 九节点微电网系统算法参数设置** 40
5.3. **IEEE 九节点微电网系统经济调度结果分析** 41
5.4. **粒子群算法参数和 MPI 并行算法参数对算例结果影响的分析** 45
5.4.1. MPI 算法参数对算例结果的影响 45
5.4.2. 粒子群算法参数对算例结果的分析 46
第 6 章 结论与展望 49
6.1. **本文工作总结** 49
6.2. **系统存在问题** 50
6.3. **未来的展望** 50
参考文献 51

附录.....53

附录 A　IEEE 九节点参数.....53

附录 B　部分程序.....55

Contents

Chapter 1 Introduction 1

1.1. Background and Significance of Research 1

1.2. Status of Microgrid Energy Management System 1

1.3. Analysis of Microgird Economic Dispatch Model 1

1.3.1. Traditional Optimization Algorithms 2

1.3.2. Artifical Intelligence Optimization Algorithms 2

1.4. Oriented Web Services 2

1.5. MPI Parallel Programming 3

1.6. Work of the Paper 3

Chapter 2 Economic Dispatch Model Considering Power Constraints and Flow Constraints 5

2.1. Costraints 5

2.1.1. Flow Constraints 5

2.1.2. Power Constraints 10

2.2. Object Function 10

2.3. Economic Dispatch Model 12

Chapter 3 Particle Swarm Optimization based on MPI 13

3.1. Definition of Particle Swarm Optimization(PSO) 13

3.2. Particle Swarm Optimization Using Single Process 13

3.3. Paricle Swarm Optimization Using MPI Parallel Computing 15

3.3.1. Analysis of MPI Parallel Design 15

3.3.2. Parallel Design of PSO 18

3.3.3. Algorithm Performance of Parallel Design 19

Chapter 4 Design and Development of Microgrid Energy Management System for Web Service 21

4.1. Construction of Energy Management System 21

4.1.1. the Way to Build an Energy Management System 21

4.1.2. Basic Components of Energe Management System 22

4.1.3. Priciple of Energe Management System 25

4.2. Example of Energy Management System 26

4.2.1. Client of Energy Management System 26

4.2.2. Server of Energy Management System 34

Chapter 5 Nine-node Case of IEEE 37

5.1. Model of Nine-node case 37

5.2. Parameter setting of Nine-node case 40

5.3. Economic Dispatch Analysis of Nine-node case 41

5.4. Impact of PSO Parameters and MPI Parallel Parameters 45

5.4.1. Impact of MPI Parallel Parameters 45

5.4.2. Impact of PSO Parameters 46

Chapter 6 Conclusion and Outlook 49

6.1. Work Summary 49

6.2. Problem of System 50

6.3. Future Outlook 50

References 51

Appendix 53

Appendix A Nine-node Parameters of IEEE 53

Appendix B Part of Program 55

第 1 章　引言

1.1. 课题研究的背景和意义

随着能源的短缺，可再生能源因其环保、经济的优点得到迅速的发展，而微电网由于其能够引入新能源电力并安全稳定的运行，也逐渐得到重视和发展。随着电网中分布式发电系统数量的日益增多，尤其是基于可再生能源的并网发电容量的增大，单一的分布式电源由于间歇性引起的功率波动及电能质量问题也日渐凸显[1]。由于微电网能够充分发挥各分布式能源的优势，并使之在系统中互补使用，因此被认为是未来电网的有效支撑。由于微电网具有提高可再生能源利用率和改善电能质量的作用，因此得到了学者的高度重视和积极研究[2]。

目前，微电网技术发展的可行性已得到验证，然而，可再生能源供电的波动性和随机性、系统结构的复杂性等都使得微电网的经济调度和能源优化管理面临许多的挑战。其中，微电网多时段潮流优化问题是微电网技术发展急需处理的关键问题之一。随着互联网产业的高速发展，设计和开发一款基于 Web 服务的微电网能量管理系统显得尤为关键[3]。

1.2. 微电网能量管理系统的研究现状

目前，关于微电网能量管理系统的研究主要集中在系统网络框架、调度控制策略以及单元级发电/储能的控制[4]。文献[5]主要针对微电网能量管理系统的基本设计框架进行介绍，主要分为硬件层和软件层两方面。文献[6]根据遗传算法建立模型，优化上游网络的系统运行成本。文献[7]通过分析最优潮流算法，建立了多目标分布式发电系统。但在函数归一化过程中运用二次规划，线性化了约束条件，这样与现实就存在了偏差。文献[8]虽然是多时段单目标的最优潮流方案，但研究对象为储存在微网中的各类能源，采用高度简化的网络模型，没有考虑风光热组合优化。

1.3. 微电网经济调度模型的分析

微电网经济调度模型是微电网能量管理系统（Energy Management System, EMS）的主要

内容，我们根据负荷变化情况进行动态机组组合，尽可能减少系统调度成本。通过常规理论不容易得到最优解，我们主要采用传统优化算法和人工智能优化算法进行运算。

1.3.1. 传统优化算法

传统优化算法一般针对结构性问题，有较为明确的问题和条件描述，包括：线性优化、二次规划、优先顺序和动态规划等。文献[9]中采用优先顺序和动态规划的方法，求解电力系统中的机组组合问题，在小机组中的时间复杂度与空间复杂度都较好。但在大机组中，计算量太大，需要采用近似方法加以简化，这样不可避免地要丢失最优解，通盘考虑整个系统，不太灵活。文献[10]采用局部加密动态规划与常规动态算法进行比较，计算精度和计算时间都有所提高，一定程度上解决了电力市场中，不同报价曲线的负荷分配问题。

1.3.2. 人工智能优化算法

随着科技的发展，越来越多的人工智能算法，逐渐进入到我们的视野中，如：进化算法、模拟退火算法、人工神经网络算法、蚁群算法和粒子群算法等。这些算法，大都是建立在生物智能或物理现象基础上的随机搜索算法。这类算法一般不要求目标函数和约束的连续性和凹凸性，和传统算法相比，具有更强的全局搜索能力[11]。

在本篇文章中，我们选用粒子群算法(Particle Swarm Optimization，PSO)进行求解。其具有容易实现、精度高、收敛快等优点，在电力系统分析中得到了广泛运用[12]。

1.4. 面向 Web 服务

Web 服务是一种服务导向架构技术，通过标准的 Web 协议提供服务，目的是保证不同平台的应用服务可以互操作。根据 W3C 的定义，Web 服务应当是一个软件系统，用以支持网络间不同机器的互动操作[13]。

一般情况下，我们基于主从式架构（Client-Server），根据 SOAP 协议进行传递 XML 格式消息，客户端通过 WSDL 描述识别服务器提供的 Web 服务。在本项目中，Web 服务端承担核心算法和数据储存工作，并通过 gSOAP 工具包提供 API 接口，供客户端使用。通过 Web 服务，提高了软件的复用性和跨平台性，也为程序的并行计算提供保障[14]。

1.5. MPI 并行程序设计

MPI（Message Passing Interface）是一个由众多计算机厂商、软件开发组织/单位、并行应用单位等共同维护的应用程序接口（API）标准，其实现主要包括 MPICH、LAMMPI、IBM MPL 等[15]。本项目使用 MPICH2 并行环境，运行于 Debian Linux 服务器版操作系统的 64 位集群（共 3 个服务器节点）上，编译环境为 MPIC++编译器，代码以 C++语言为主[16]。

通过搭建 MPI 分布式并行计算集群，相对与原有单进程串行任务，缩短了粒子群（PSO）算法的计算时间，提高了计算效率[17]。

1.6. 本文所做的工作

本文针对微电网能量管理系统的经济调度特点，考虑功率约束和潮流约束，建立了微电网系统单目标多时段优化模型，并将粒子群优化算法应用于模型的求解，主要研究工作如下：

第 1 章，介绍课题研究的背景和意义，针对微电网管理系统和经济调度模型现状进行简要分析，并介绍了 Web 服务和 MPI 并行计算技术。

第 2 章，介绍了考虑功率约束和潮流约束的微电网经济调度模型，其中，重点介绍了牛顿-拉夫逊(Newton-Raphson)潮流算法和其编程思路。

第 3 章，介绍了单进程和 MPI 并行框架下的粒子群优化算法(PSO)，并基于 MPI-PSO 算法解决微电网中经济调度问题。

第 4 章，介绍了面向 Web 服务的微电网能量管理系统的软件架构，包括 Linux 平台的服务端、基于微软.NET 平台的客户端和连接服务端和客户端的 gSOAP 调度中间件。

第 5 章，针对 IEEE 九节点微电网模型，基于本文的能量管理系统，计算微电网算例各分布式发电单元的出力分配和系统的最优运行成本。

第 6 章，对本文的工作内容进行总结，分析系统存在的问题，并对未来工作进行展望。

第 2 章　考虑功率约束与潮流约束的经济调度模型

2.1. 约束条件

2.1.1. 潮流约束

$$P_i = e_i \sum_{j=1}^{n} (G_{ij} e_j - B_{ij} f_j) + f_i \sum_{j=1}^{n} (G_{ij} f_j + B_{ij} e_j) \tag{2.1}$$

这里，P_i 是母线的有功功率，n 为节点数，e_i 和 f_i 分别是母线 i 的电压幅值和频率，G_{ij} 和 B_{ij} 分别为节点导纳矩阵中的电导和电纳元素。本文使用牛顿-拉夫逊法进行潮流计算。

2.1.1.1. 直角坐标形式下的牛顿-拉夫逊潮流算法

牛顿-拉夫逊(Newton-Raphson)法，是求解非线性代数方程的一种有效且收敛速度快的迭代计算方法。考虑 n 维非线性代数方程组[18]：

$$\left.\begin{array}{l} f_1(x_1, x_2, \ldots, x_n) = 0 \\ f_2(x_1, x_2, \ldots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) = 0 \end{array}\right\} \tag{2.2}$$

在直角坐标系中，节点电压表示为：

$$\dot{V}_i = e_i + jf_i \tag{2.3}$$

导纳矩阵元素则表示为：

$$Y_{ij} = G_{ij} + B_{ij} \tag{2.4}$$

我们知道，n 个节点电力系统的潮流方程的一般形式是：

$$P_i + jQ_i = \dot{V}_i \sum_{i=1}^{n} Y_{ij}^{*} V_j^{*} \quad (i = 1, 2, ..., n) \tag{2.5}$$

将式(2.3)和式(2.4)带入式(2.5)的右边，展开并分别得出实部或虚部，便得：

$$\left.\begin{aligned} P_i &= e_i\sum_{j=1}^{n}(G_{ij}e_j - B_{ij}f_j) + f_i\sum_{j=1}^{n}(G_{ij}f_j + B_{ij}e_j) \\ Q_i &= f_i\sum_{j=1}^{n}(G_{ij}e_j - B_{ij}f_j) - e_i\sum_{j=1}^{n}(G_{ij}f_j + B_{ij}e_j) \end{aligned}\right\} \tag{2.6}$$

假定系统中的第 1，2，…， m 号节点为 PQ 节点，第 i 个基点的给定功率设为 P_{is} 和 Q_{is}，对该节点可列写方程：

$$\left.\begin{aligned} \Delta P_i &= P_{is} - P_i = 0 \\ \Delta Q_i &= Q_{is} - Q_i = 0 \end{aligned}\right\} \quad (i = 1,2,\cdots,m) \tag{2.7}$$

假定方程中的第 $m+1$，$m+2$，…，$n-1$ 号节点为 PV 节点，则对其中每一个节点可以列写方程：

$$\left.\begin{aligned} \Delta P_i &= P_{is} - P_i = 0 \\ \Delta V_i^2 &= V_{is}^2 - V_i^2 = 0 \end{aligned}\right\} \quad (i = m+1, m+2,\cdots,n-1) \tag{2.8}$$

第 n 号节点为平衡节点，其电压 $V_n = e_n + jf_n$ 是给定的，故不参加迭代。

式(2.7)和式(2.8)总共包含了 $2(n-1)$ 个方程，待求得变量有 e_1，f_1，e_2，f_2，…，e_{n-1}，f_{n-1} 也是 $2(n-1)$ 个。我们还可以看到式(2.7)和式(2.8)已经具备了式(2.2)的形式。因此，不难写出如下的修正方程式：

$$\Delta W = -J\Delta V \tag{2.9}$$

式(2.9)中：

$$\Delta W = [\Delta P_1 \quad \Delta Q_1 \quad \dots \quad \Delta P_m \quad \Delta Q_m \quad \Delta P_{m+1} \quad \Delta V_{m+1}^2 \quad \dots \quad \Delta P_{n-1} \quad \Delta V_{n-1}^2]^T$$

$$\Delta V = [\Delta e_1 \quad \Delta f_1 \quad \dots \quad \Delta e_m \quad \Delta f_m \quad \Delta e_{m+1} \quad \Delta f_{m+1} \quad \dots \quad \Delta e_{n-1} \quad \Delta f_{n-1}]^T$$

$$J=\begin{bmatrix}
\frac{\partial\Delta P_1}{\partial\Delta e_1} & \frac{\partial\Delta P_1}{\partial\Delta f_1} & \cdots & \frac{\partial\Delta P_1}{\partial\Delta e_m} & \frac{\partial\Delta P_1}{\partial\Delta f_m} & \frac{\partial\Delta P_1}{\partial\Delta e_{m+1}} & \frac{\partial\Delta P_1}{\partial\Delta f_{m+1}} & \cdots & \frac{\partial\Delta P_1}{\partial\Delta e_{n-1}} & \frac{\partial\Delta P_1}{\partial\Delta f_{n-1}} \\
\frac{\partial\Delta Q_1}{\partial\Delta e_1} & \frac{\partial\Delta Q_1}{\partial\Delta f_1} & \cdots & \frac{\partial\Delta Q_1}{\partial\Delta e_m} & \frac{\partial\Delta Q_1}{\partial\Delta f_m} & \frac{\partial\Delta Q_1}{\partial\Delta e_{m+1}} & \frac{\partial\Delta Q_1}{\partial\Delta f_{m+1}} & \cdots & \frac{\partial\Delta Q_1}{\partial\Delta e_{n-1}} & \frac{\partial\Delta Q_1}{\partial\Delta f_{n-1}} \\
\cdots & \cdots & & \cdots & \cdots & \cdots & \cdots & & \cdots & \cdots \\
\frac{\partial\Delta P_m}{\partial\Delta e_1} & \frac{\partial\Delta P_m}{\partial\Delta f_1} & \cdots & \frac{\partial\Delta P_m}{\partial\Delta e_m} & \frac{\partial\Delta P_m}{\partial\Delta f_m} & \frac{\partial\Delta P_m}{\partial\Delta e_{m+1}} & \frac{\partial\Delta P_m}{\partial\Delta f_{m+1}} & \cdots & \frac{\partial\Delta P_m}{\partial\Delta e_{n-1}} & \frac{\partial\Delta P_m}{\partial\Delta f_{n-1}} \\
\frac{\partial\Delta Q_m}{\partial\Delta e_1} & \frac{\partial\Delta Q_m}{\partial\Delta f_1} & \cdots & \frac{\partial\Delta Q_m}{\partial\Delta e_m} & \frac{\partial\Delta Q_m}{\partial\Delta f_m} & \frac{\partial\Delta Q_m}{\partial\Delta e_{m+1}} & \frac{\partial\Delta Q_m}{\partial\Delta f_{m+1}} & \cdots & \frac{\partial\Delta Q_m}{\partial\Delta e_{n-1}} & \frac{\partial\Delta Q_m}{\partial\Delta f_{n-1}} \\
\frac{\partial\Delta P_{m+1}}{\partial\Delta e_1} & \frac{\partial\Delta P_{m+1}}{\partial\Delta f_1} & \cdots & \frac{\partial\Delta P_{m+1}}{\partial\Delta e_m} & \frac{\partial\Delta P_{m+1}}{\partial\Delta f_m} & \frac{\partial\Delta P_{m+1}}{\partial\Delta e_{m+1}} & \frac{\partial\Delta P_{m+1}}{\partial\Delta f_{m+1}} & \cdots & \frac{\partial\Delta P_{m+1}}{\partial\Delta e_{n-1}} & \frac{\partial\Delta P_{m+1}}{\partial\Delta f_{n-1}} \\
\frac{\partial\Delta Q_{m+1}}{\partial\Delta e_1} & \frac{\partial\Delta Q_{m+1}}{\partial\Delta f_1} & \cdots & \frac{\partial\Delta Q_{m+1}}{\partial\Delta e_m} & \frac{\partial\Delta Q_{m+1}}{\partial\Delta f_m} & \frac{\partial\Delta Q_{m+1}}{\partial\Delta e_{m+1}} & \frac{\partial\Delta Q_{m+1}}{\partial\Delta f_{m+1}} & \cdots & \frac{\partial\Delta Q_{m+1}}{\partial\Delta e_{n-1}} & \frac{\partial\Delta Q_{m+1}}{\partial\Delta f_{n-1}} \\
\cdots & \cdots & & \cdots & \cdots & \cdots & \cdots & & \cdots & \cdots \\
\frac{\partial\Delta P_{n-1}}{\partial\Delta e_1} & \frac{\partial\Delta P_{n-1}}{\partial\Delta f_1} & \cdots & \frac{\partial\Delta P_{n-1}}{\partial\Delta e_m} & \frac{\partial\Delta P_{n-1}}{\partial\Delta f_m} & \frac{\partial\Delta P_{n-1}}{\partial\Delta e_{m+1}} & \frac{\partial\Delta P_{n-1}}{\partial\Delta f_{m+1}} & \cdots & \frac{\partial\Delta P_{n-1}}{\partial\Delta e_{n-1}} & \frac{\partial\Delta P_{n-1}}{\partial\Delta f_{n-1}} \\
\frac{\partial\Delta Q_{n-1}}{\partial\Delta e_1} & \frac{\partial\Delta Q_{n-1}}{\partial\Delta f_1} & \cdots & \frac{\partial\Delta Q_{n-1}}{\partial\Delta e_m} & \frac{\partial\Delta Q_{n-1}}{\partial\Delta f_m} & \frac{\partial\Delta Q_{n-1}}{\partial\Delta e_{m+1}} & \frac{\partial\Delta Q_{n-1}}{\partial\Delta f_{m+1}} & \cdots & \frac{\partial\Delta Q_{n-1}}{\partial\Delta e_{n-1}} & \frac{\partial\Delta Q_{n-1}}{\partial\Delta f_{n-1}}
\end{bmatrix}$$

上述方程中雅可比矩阵的各元素，可以对式(2.7)和式(2.8)求偏导得到。

当$i\neq j$时：

$$\left.\begin{aligned}
&\frac{\partial\Delta P_i}{\partial e_j}=-\frac{\partial\Delta Q_i}{\partial f_j}=-(G_{ij}e_i+B_{ij}f_i)\\
&\frac{\partial\Delta P_i}{\partial f_j}=\frac{\partial\Delta Q_i}{\partial e_j}=B_{ij}e_i-G_{ij}f_i\\
&\frac{\partial\Delta V_i^2}{\partial e_j}=\frac{\partial\Delta V_i^2}{\partial f_j}=0
\end{aligned}\right\} \tag{2.10}$$

当$j=i$时：

$$\left.\begin{aligned}
&\frac{\partial \Delta P_i}{\partial e_i} = -\sum_{k=1}^{n}(G_{ik}e_k - B_{ik}f_k) - G_{ii}e_i - B_{ii}f_i \\
&\frac{\partial \Delta P_i}{\partial f_i} = -\sum_{k=1}^{n}(G_{ik}f_k + B_{ik}e_k) + B_{ii}e_i - G_{ii}f_i \\
&\frac{\partial \Delta Q_i}{\partial e_i} = \sum_{k=1}^{n}(G_{ik}f_k + B_{ik}e_k) + B_{ii}e_i - G_{ii}f_i \\
&\frac{\partial \Delta Q_i}{\partial f_i} = -\sum_{k=1}^{n}(G_{ik}e_k - B_{ik}f_k) + G_{ii}e_i + B_{ii}f_i \\
&\frac{\partial \Delta V_i^2}{\partial e_i} = -2e_i \\
&\frac{\partial \Delta V_i^2}{\partial f_i} = -2f_i
\end{aligned}\right\} \tag{2.11}$$

2.1.1.2. 牛顿-拉夫逊法潮流计算的流程

用牛顿-拉夫逊法计算潮流的流程框图如图 2.1 所示：

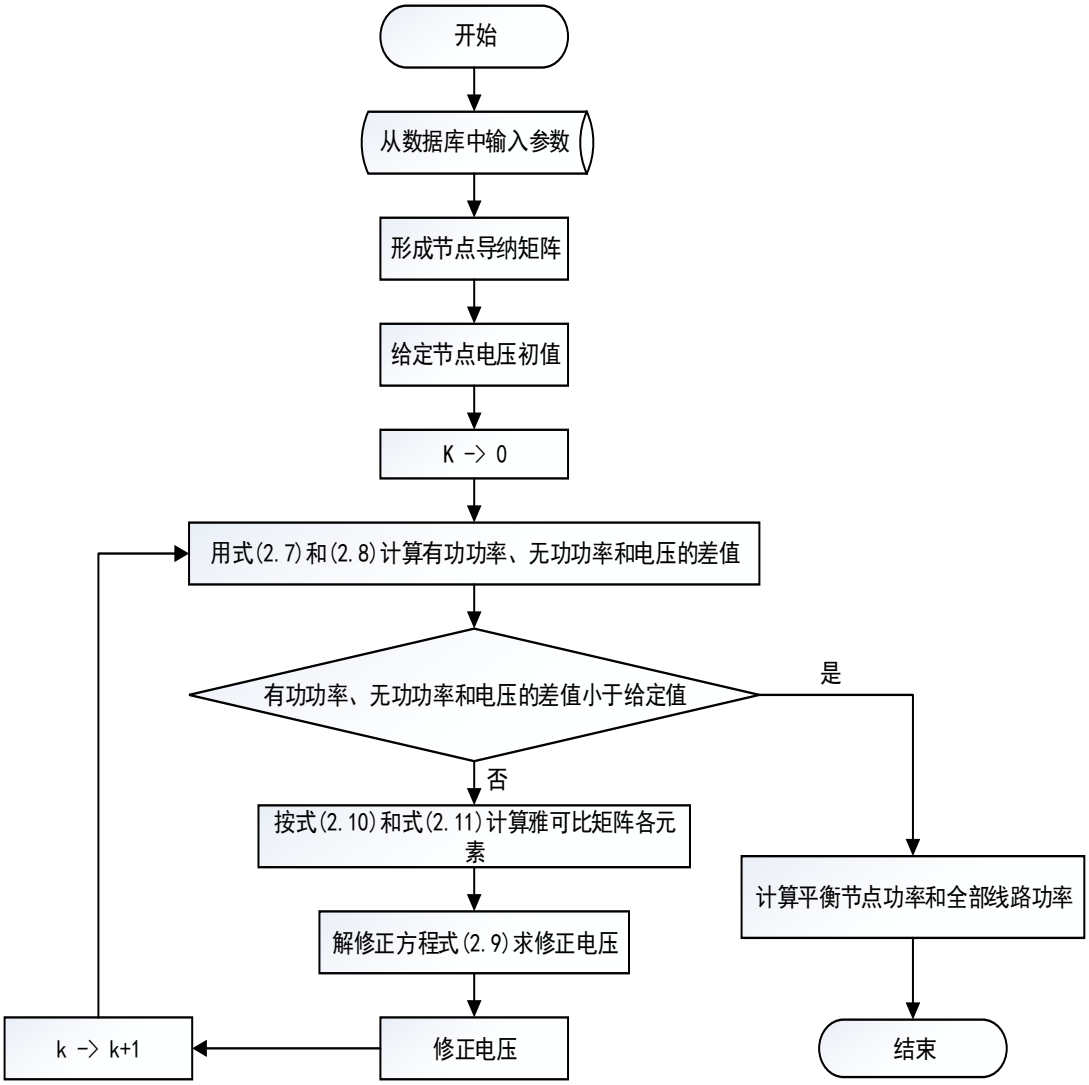

图 2.1 牛顿-拉夫逊法潮流计算程序框图

首先，我们需要从数据库读取数据形成节点导纳矩阵。输入节点电压初值，置迭代计数 $k=0$，然后开始进入牛顿迭代过程。在进行第 $k+1$ 次迭代时，计算步骤如下[18]：

1) 按上一次迭代算出的节点电压值，利用式(2.7)和式(2.8)计算各类节点的不平衡量。

2) 按条件校验收敛，即：

$$\max\{\left|\Delta P_i^{(k)},\Delta Q_i^{(k)},\Delta V_i^{2(k)}\right|\}<\varepsilon \tag{2.12}$$

如果收敛，迭代到此结束，转入计算各线路潮流和平衡节点功率，并打印输出计算

结果。如果不收敛则继续计算。

3) 利用式(2.10)和(2.11)计算雅可比矩阵各元素。

4) 解修正方程式(2.9)，求节点电压的修正量。

5) 修正个基点的电压：

$$e_i^{(k+1)} = e_i^{(k)} + \Delta e_i^{(k)} \tag{2.13}$$

$$f_i^{(k+1)} = f_i^{(k)} + \Delta f_i^{(k)} \tag{2.14}$$

6) 迭代计数 k 加 1，返回第一步继续迭代过程。

迭代结束后，还要算出平衡节点的功率和网络中的功率分布。输电线路的计算公式如下：

$$S_{ij} = P_{ij} + jQ_{ij} = \dot{V}_i \overset{*}{I}_{ij} = V_i^2 \overset{*}{y}_{i0} + \dot{V}_i(\overset{*}{V}_i - \overset{*}{V}_j)\overset{*}{y}_{ij} \tag{2.15}$$

2.1.2. 功率约束

$$\left.\begin{aligned} P_{DGj\min} \le P_{DGj} \le P_{DGj\max} \\ P_{grid.buy} \le P_{grid} \le P_{grid.sale} \end{aligned}\right\} \tag{2.16}$$

在式(2.16)中，P_{DGj} 是第 j 个微电源的出力；P_{DGjmin} 和 P_{DGjmax} 分别为出力的最小值和最大值；P_{grid} 是交换功率，购电(主网流向微电网)为正，售电(微电网流向主网)为负[19]。

2.2. 目标函数

$$F = \min COST \tag{2.17}$$

$$\left.\begin{aligned} C_{O\&Mj} &= P_{j(t)} K_{om,j} \\ COST &= \sum_{h=1}^{24}\sum_{j=1}^{m} C_{O\&Mj} + \sum_{h=1}^{24} C_h S_{grid} \end{aligned}\right\} \tag{2.18}$$

在以上两个公式中，m 是总电源数目；$C_{O\&Mj}$ 是微电网中第 j 个元件的运行维护成本；P_j 是第 j 个微电源在时段 t 的有功出力；$K_{om,j}$ 为运行维护系数；S_{grid} 是微电网与主网之间每小时的交换电量，以微电网向主网购电为主，售电为负；C_h 是分时电价[19]。

风电和光伏的运行维护成本和分时电价如表格 2.1 和表格 2.2 所述：

表格 2.1 光伏和风电运行维护成本

DER 类型	运行维护成本系数(元/(kW·h))
光伏电池	0.0096
风力发电	0.045

表格 2.2 购电和售电电价

项目	价格(元/(kW·h))		
	峰时段	平时段	谷时段
购电	0.83	0.49	0.17
售电	0.65	0.38	0.13
时段	11:00-15:00 19:00-21:00	08:00-10:00 16:00-18:00	00:00-07:00

2.3. 经济调度模型

本文的优化目标是参考风力发电机、光伏发电组件和蓄电池的运行维护成本和购电、售电价格，优化周期为 24 个小时，在满足潮流约束和功率约束的前提下，以经济运行成本为原则，建立以最小运行维护成本作为优化目标的优化函数。

在并网情况下，光伏电池和风力发电机所发电量首先满足微电网内部负载，有剩余电量则存入蓄电池，若蓄电池充满，则向主网出售。

当光伏发电组件和风力发电机所提供电量，不够内部负载使用时，则蓄电池出力。若三者均无法满足要求时，则向主网购电。

以上经济策略如图 2.2 所示：

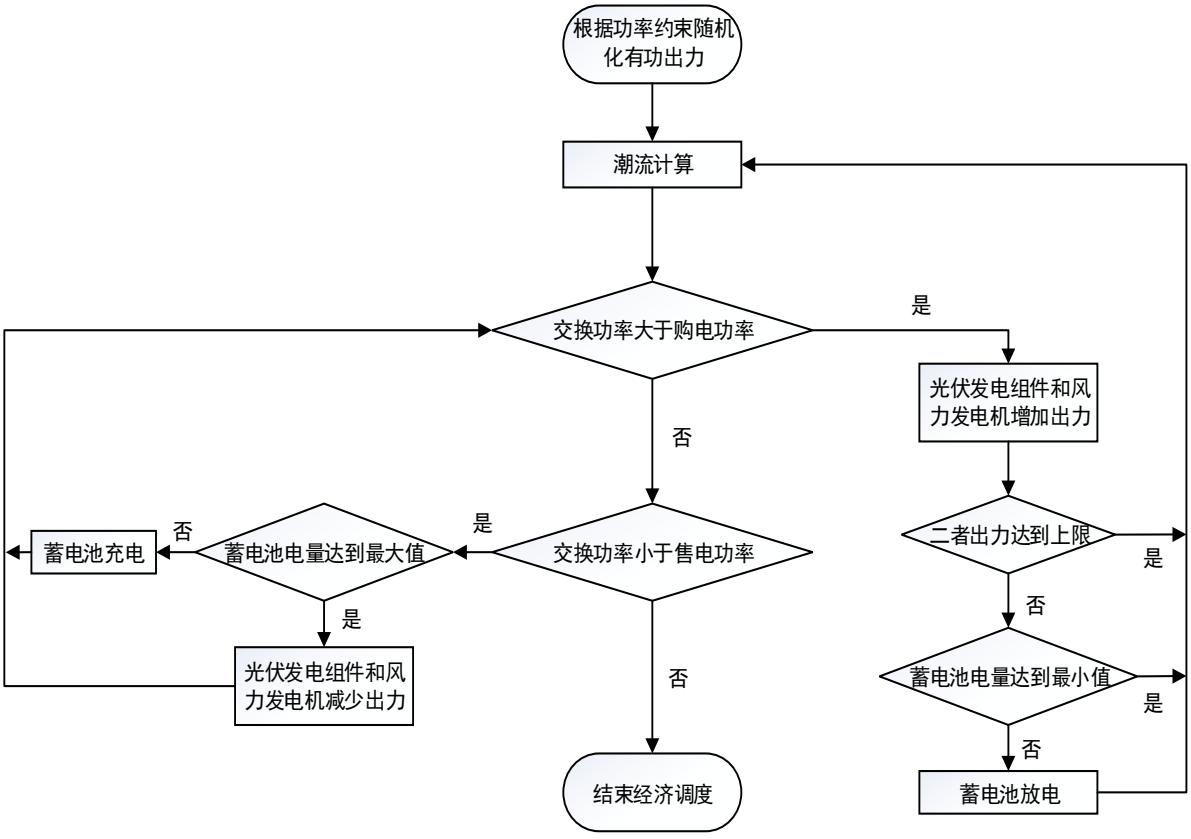

图 2.2 经济调度模型

第 3 章　基于 MPI 并行计算的粒子群优化算法

3.1. 粒子群(PSO)优化算法的定义

粒子群优化(Particle Swarm Optimization，PSO)是由 J.Kennedy 和 R.C.Eberhart 等在 1995 年开发的演化计算技术。其中“群(Swarm)”来源于微粒群模型，“粒子(particle)”是一种折衷选择，用来描述无质量、无体积，但同时又有速度和加速度的一种状态[20]。

PSO 算法基于群体，我们依据对环境的适应度将群体中的个体移动到较好的区域。它将个体看作 D 维搜索空间中的微粒，给予一定的飞行速度，这个速度需要根据自生的状态和同伴的状态进行动态调整[20]。

第 i 个微粒为 $X_i=(x_{i1},x_{i2},x_{i3},\cdots,x_{iD})$，其最好位置(最优适应值)为 $P_i=(p_{i1},p_{i2},p_{i3},\cdots,p_{iD})$，也记为 P_{best}。群体中所有粒子的最好位置则表示为 $P_g=(p_{g1},p_{g2},p_{g3},\cdots,p_{gD})$，也被称为 g_{best}。粒子 i 的速度用 $V_i=(v_{i1},v_{i2},v_{i3},\cdots,v_{iD})$ 表示。每一代粒子，他的第 d 维 $(1\le d\le D)$ 依据下面的式子进行计算[20]：

$$v_{id}=(w*v_{id})+c1*rand()*(p_{id}-x_{id})+c2*Rand()*(p_{gd}-x_{id}) \tag{3.1}$$

$$x_{id}=x_{id}+v_{id} \tag{3.2}$$

其中 w 为惯性权重(inertia weight)，$c1$ 和 $c2$ 为加速常数(acceleration constants)，$rand()$ 和 $Rand()$ 是两个范围为[0,1] 的随机数[20]。

同时，微粒需要设定一个最大速度 $V_{\max}$，若微粒某维速度 v_{id} 超过本维最大速度 $v_{\max d}$，则速度被限制为本维最大速度 $v_{\max d}$。

3.2. 单进程下运用粒子群算法解决微电网系统经济调度问题

单进程状态下，微电网系统的粒子群算法如图 3.1 所示。

首先，我们针对所选粒子群进行随机初始化(在约束范围内)，包括粒子的初始速度、初始位置、个体最优位置和群体最优位置。

随后，我们针对粒子群中每一个粒子，进行迭代，计算它们的适应度。这里，我们根据式(3.1)和式(3.2)，更新每一个粒子的速度和位置。判断范围约束后，依照经济调度模型，计算粒子的目标函数值。对于每个微粒，我们将其适应值(目标函数值)，分别和它经历的最好位置和全局最好位置进行比较，不断更新 P_{best} 和 g_{best}。

最后，粒子群满足迭代次数要求，输出最终得到的群体最优解 g_{best}。

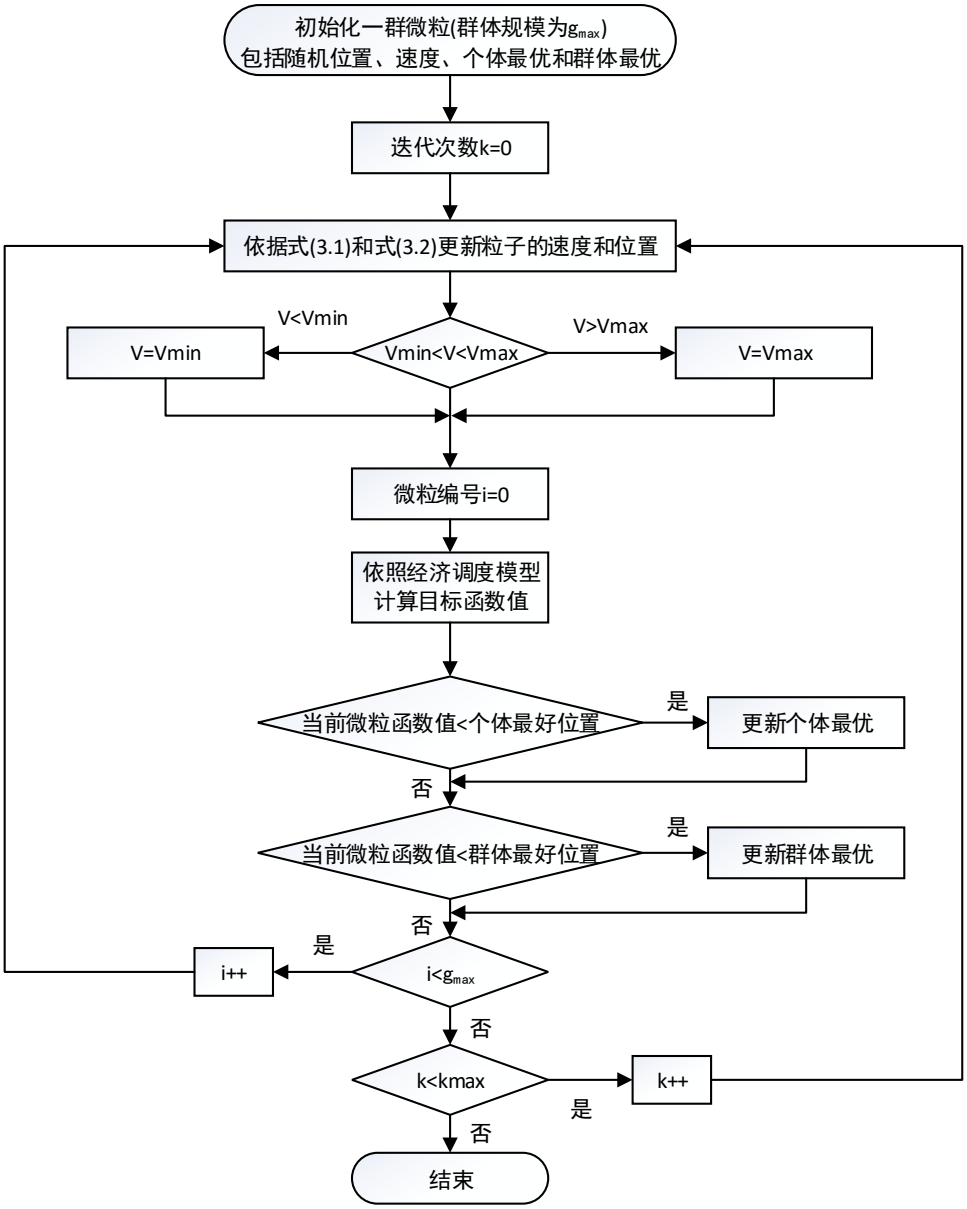

图 3.1 单进程粒子群算法流程图

3.3. MPI 并行计算下运用粒子群算法解决微电网系统经济调度问题

标准的粒子群(PSO)算法，一般通过设置足够大的粒子数增加算法的搜索效率，以提高算法获取群体最优解的能力。但粒子数量的增加会增加算法的迭代次数，由此带来的潮流计算负担，增加了电力系统优化运行的内存空间和运算时间。通过减小粒子群数目，有效地高了优化算法的求解速度[21,22]。

3.3.1. MPI 并行设计分析

MPI(Message Passing Interface)是一种消息传递编程模型。消息传递模型需要对常规串行任务进行分解，分配到不同的进程，我们只需要组织不同进程间的数据并行和消息传递。MPI 模型并行粒度打完，适用于大规模可拓展并行计算，正契合于粒子群算法的特性[23]。

作为一种并行计算标准，MPI 拥有众多实现版本，其中典型的有 MPICH、LAMMPI、IBM MPL 等。由于 MPICH 的开发与 MPI 规范的制定是同步进行的，所以其最能反映 MPI 的发展与变化。本项目采用 MPICH2 作为 MPI 并行程序的开发工具[23]。

基于 MPICH2 进行开发，需要遵循一定的工作模式。我们以串行模式编写程序，但运行时，程序分别执行不同的代码块。所有的程序元素(函数、全局变量和局部变量等)，在没有进行显式区分的情况下，均为所有进程共同拥有。虽然在不同进程中名字相同，但“物理”上却互不联系，属于进程的私有变量[23]。

MPI 中的“显式区分”，就是我们在串行编写程序时，通过显式的条件判断，让不同的进程，执行不同的代码块。常规的 MPI 代码模式如图 3.2 所示：

```
//common data structures & functions

//initialization code block

if (rank == process0)
{
    //define works to be carried out by process0
}

else if (rank == process1)
{
    //define works to be carried out by process1
}

//...

else if (rank == processi)
{
    //define works to be carried out by processi
}

//...

else if (rank == processn)
{
    //define works to be carried out by processn
}

else
{
    //define works to be carried out by all processes
}
```

图 3.2 常规 MPI 代码模式

同时，基于 MPICH2 进行并行程序开发，需要遵循一定的设计流程，图 3.3 给出了基于 MPICH2 编写程序的通用流程。

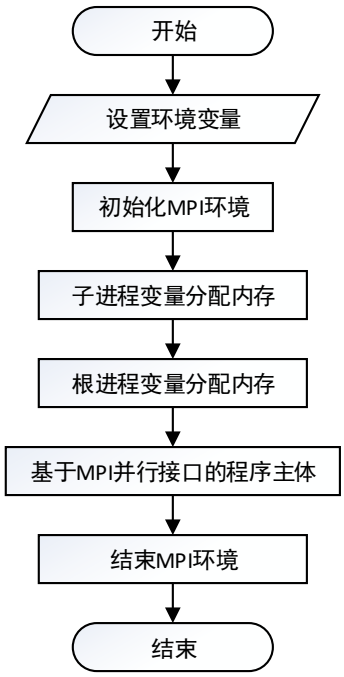

图 3.3 基于 MPICH2 的并行程序流程

根据图 3.3，进行 MPI 程序设计，首先需要调用 MPI_Init 初始化 MPI 环境，之后才可调用其它 API 接口；调用 MPI_Get_processor_name 获取当前进程运行主机名；调用 MPI_Comm_rank 获得当前进程编号；调用 MPI_Comm_size 获得程序总进程数。

MPI 并行机制通过各个进程间的通信操作，如点对点通信操作、集合通信操作和分散通信操作等来共享数据。各个进程在处理完数据后，调用 MPI_Gather 和 MPI_Reduce 将数据集合到某进程，这个进程称为根进程。根进程处理完数据后，再通过 MPI_Scatter 将数据发散到各个进程(包括根进程本身)。在集合和分散操作前后，需要调用 MPI_Barrier 同步各进程，这样，我们就完成了一个基本的 MPI 并行程序。

在 MPI 的缓冲通信模式中，我们需要分配内存作为发送缓冲区和接收缓冲区。因此，合理分配内存空间，在 MPI 集合和分散操作中，是非常有必要的。对于多维变量，我们还需要考虑内存空间的连续分配。给出了 PSO 算法中粒子群集合分配操作的通信示意图[24]。

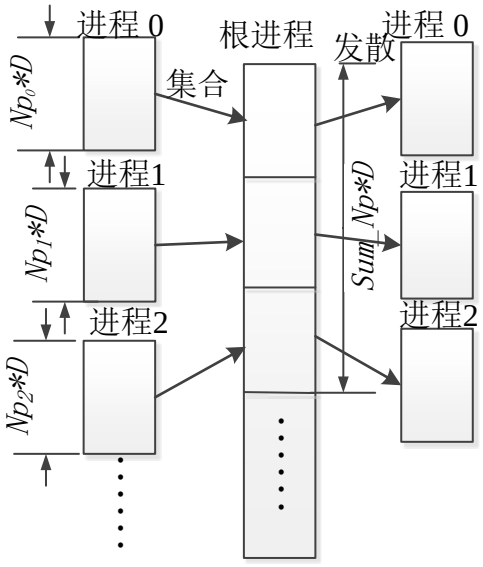

图 3.4 粒子群集合发散操作通信示意图

由图 3.4 可知，粒子群参数在每个进程中所占内存粒子群规模和个体位数的积。以三个进程为例，编号分别为进程 0、进程 1 和进程 2。当进行集合操作时，这三个进程在根进程中的内存是按一定顺序连续分配的，同时在根进程中占用的内存空间是这三个集成粒子群参数占用内存的和。当进行分散操作时，根进程按照同样顺序，将消息发送到对应进程，接收参数的内存大小和发送参数的内存大小一致。

完成程序主体后，我们通过 MPI_Finalized 接口结束 MPI 并行环境。此后，我们不得再调用任何一个 MPI 接口函数。

3.3.2. 粒子群算法的并行设计

通过并行计算，我们将粒子群算法的大规模群体划分成一定数量的子群，每个子群占用 CPU 中的一个进程及一定内存空间。同过集合操作，我们将每个进程的子群收集到根进程，这样得到的新粒子群合理利用了每个进程的子群信息，增加了个体多样性，提高了算法的运行效率[25]。

基于 MPI 并行程序设计的粒子群算法，需要结合粒子群算法的特性，进行集合和分散操作，结合上述图 3.3，我们可以设计如图 3.5 的并行算法流程图。

我们在初始化 MPI 环境后，将子进程和根进程参与集合操作所需内存分配好，这里需要注意多维变量的内存连续性。随后，仍参照标准粒子群算法进行迭代，并对每个粒子进行适应度评价。当迭代次数满足集合操作条件时，则进行对应的集合操作，计算出总最优适应值，并将其分配到各子进程。在迭代结束后，输出总群体的最优解，作为最终的目标函数值。

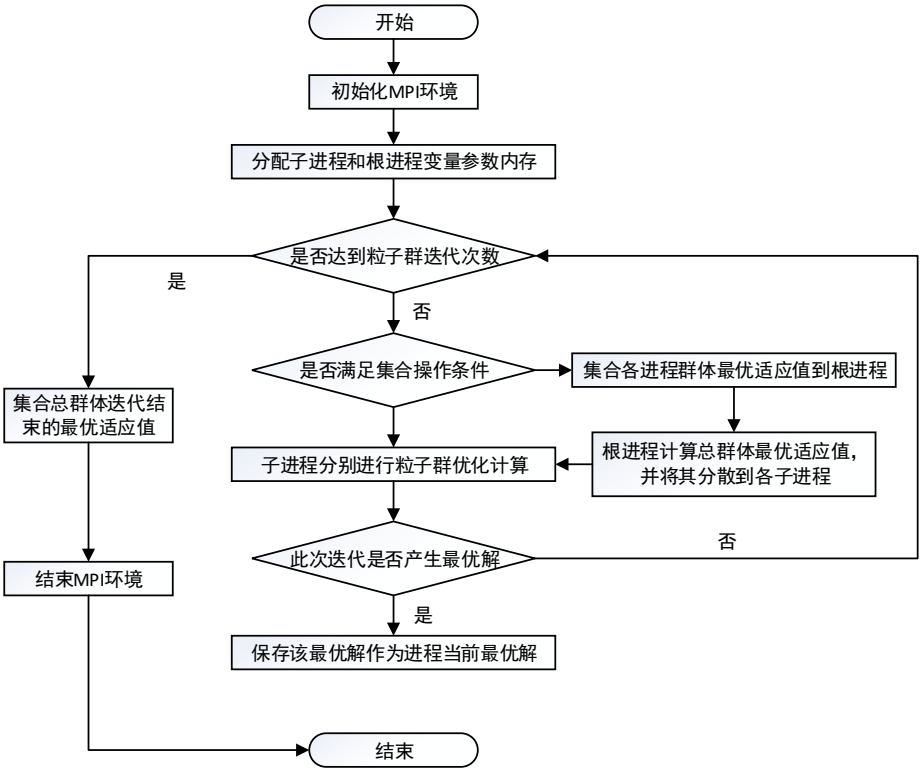

图 3.5 粒子群算法的并行算法流程图

3.3.3. 粒子群并行算法的性能

粒子群并行算法的性能，主要取决于我们设定的进程数、进行集合分散操作的频率和每个进程子群的规模。这一节结合 MPI 并行计算原理，分析这几个影响粒子群并行计算性能的主要参数。

图 3.6 介绍了本文 MPI 粒子群算法程序的运行原理，我们这里设置 3 个进程来同时运行。在串行编写的程序中，只有进程 0 可以执行根进程部分的代码(红色字体部分)，而黑色字体部分的代码，每个进程都需要执行。

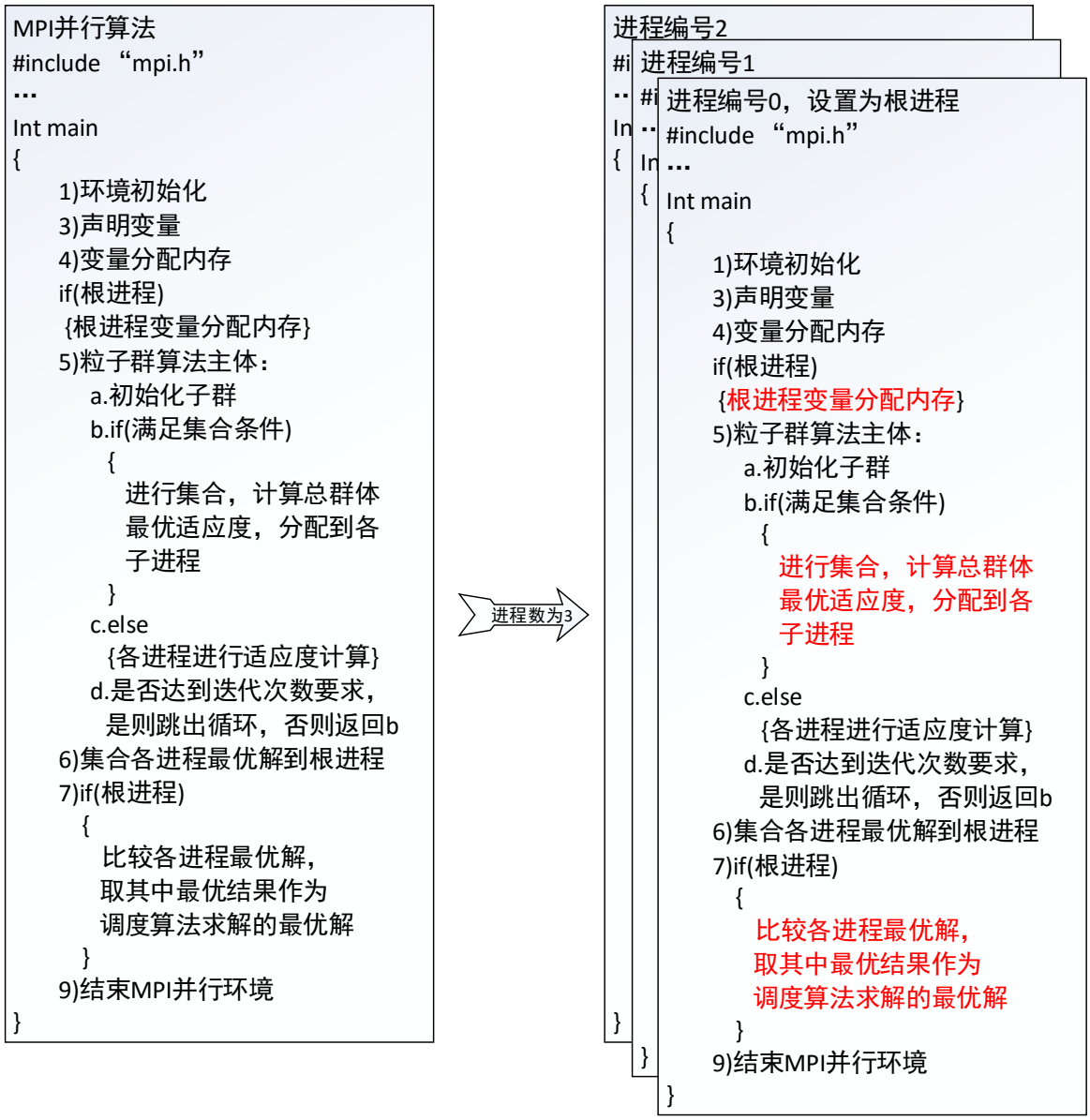

图 3.6 粒子群算法并行原理图

因此，我们分配多少个进程，相应的 PSO 算法代码也会复制到每个对应的进程中运行。我们在每次集合分散操作前后，需要同步各进程，这需要花费一定时间，同时，进行集合和分散通信，也需要适量运行时间，这两个时间，基本与进程数量正相关。如果进程数量过多，子群分得太细，则会造成集合发散的频繁操作，降低粒子群算法运行效率。反之，若进程数设置太少，每个进程的子群规模过大，各进程独自运算时间过长。这样，就不能有效地利用 CPU 资源和服务器集群资源，减弱了并行计算的实际功效。

所以，我们在设计 PSO 并行计算程序时，需要综合考虑进程数目、集合分散操作频率和子群规模，以谋求最优的运算性能。

第 4 章　面向 Web 服务的微电网能量管理系统

微电网能量管理系统是一套具有出力优化调度、负荷管理和分时段实时更新的能量管理软件[5]。通过该系统，我们实现了负荷在系统内分布式发电单元中的合理分配。在满足系统总的用电需求的前提下，使系统的运行成本最低。本文中，微电网能量管理系统主要由以下三个部分组成：运行后台调度算法和数据存储的远程服务端、承担人机交互功能的前台客户端和连接客户端与服务端的调度中间件。

4.1. 能量管理系统的构建与组成部分

4.1.1. 如何构建一个能量管理系统

一个能量管理系统的需求如下：首先，我们需要分析调度需求，收集我们应该整合的需求信息；其次，研究系统中的分布式部件，与其正常运行时考虑的约束条件；紧接着，我们根据需求选择合适的优化调度算法，以此为基础，构建相应的数学模型，设计模型函数接口，编写相应的软件程序；最后，选择合适的数据库存储程序参数。

4.1.1.1. 用户需求分析

用户需求是构建微电网系统的前提，针对微电网能量管理而言，最基本的功能是，用户能够根据其管理目标，针对某种特定算例，通过合理分配分布式发电单元的出力状态，使其目标函数达到最优解。所以，我们需要管理系统能够满足这几个条件：

1) 能够记录特定微电网系统的基本参数，并完成参数的增加、删除和修改功能。
2) 通过优化调度算法调节负荷的经济分配，制定分布式发电单元的出力计划，使目标函数最优。同时，能够对算法参数进行调整。
3) 通过数据表格和图像展示优化调度结果，并进行一定的数据分析。

4.1.1.2. 部件参数信息和约束条件

能量管理系统的部件参数主要包括：线路参数、节点参数、分布式发电单元参数和负荷

参数等。其中的发电单元，在不同的系统中也有不同的构成方式。本文研究的微电网系统主要包括风电单元、光伏单元和蓄电池单元。各部件除了考虑自身物理约束(出力约束、书电池容量约束等)，还需要考虑组合运行的约束(像潮流约束、并网交换功率约束等)。

4.1.1.3. 选择合适的优化调度算法

在能量管理系统中，优化算法是实现最优经济调度的核心部分。即使是相同的微电网系统，采用不同的优化算法，也会取得不同的优化效果。我们需要根据模型，选择最合适的优化算法，以取得最优的运行效率和调度结果。本文，选用粒子群算法进行优化。

4.1.1.4. 数据储存单元

数据储存单元是执行能量管理的重要组成部分，整个系统的各类参数(如部件参数、算法参数和调度结果等)，均需要在该单元中储存。本文使用 MySQL 数据库作为能量管理系统的数据储存单元。其为优化算法的运行提供必要的部件参数和算法参数，运算完成后，优化调度结果也会存入数据库中，通过交互界面展示给用户。

4.1.2. 能量管理系统的基本组成

本文中的能量管理系统由远程服务端、前台客户端和连接二者的调度中间件三部分构成。一个良好的管理系统，离不开三者的协调工作。

4.1.2.1. 服务端

服务端承担优化算法数学模型的具体实现，是微电网能量管理系统的核心部分。其主要由优化算法程序和数据库组成。图 4.1 给出了本系统服务端的基本工作流程图。

首先，我们根据客户端请求，初始化系统各部件参数和算法参数，参数信息由数据库读入；紧接着，我们根据部件参数信息和算法参数信息，初始化群体信息，按照潮流约束和功率约束，使群体满足约束条件；随后，我们调用粒子群优化算法，根据优化调度模型，求解出力分配和经济成本；最后，将得到的最优解存入数据库，并通知客户端，服务端已经完成计算。

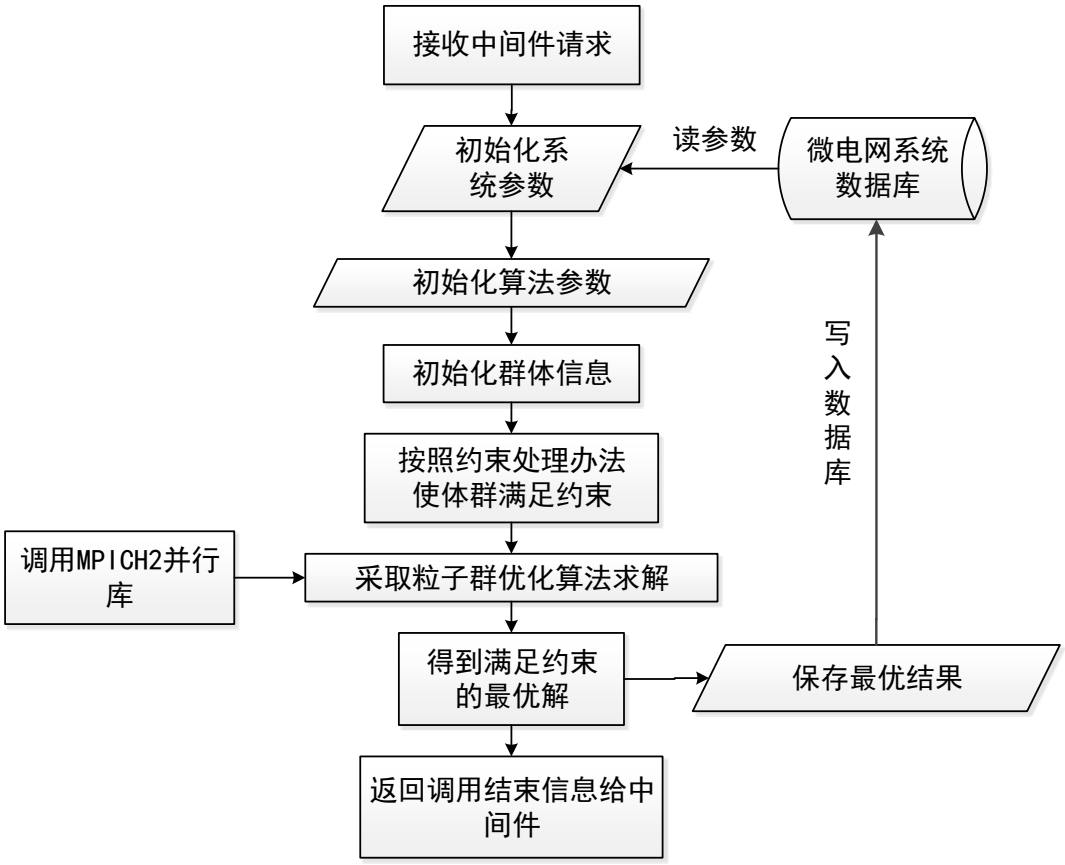

图 4.1 服务端工作流程图

4.1.2.2. 客户端

客户端作为为客户提供前台服务的程序，在能量管理系统中，主要承担人机交互的的功能。用户通过客户端界面，查看数据库中的微电网部件参数和算法参数。并能随时随地对这些数据进行插入、删除和修改。图 4.2 给出了客户端的工作流程图。

打开客户端，首先，我们需要进行用户信息验证，这些信息储存在远程数据库中，验证通过后打开用户交互界面。在交互界面中，通过客户端与数据库的绑定，用户可以对母线参数、各分布式电源(包括风电、光伏和蓄电池)参数、负载参数、线路参数与算法参数进行插入、删除和修改。随后，进入运行调度界面，选择合适的算法编号，向服务端发送调度请求。服务端计算完成存入数据库后，向客户端返回结果标志。接收到标志后，客户端从数据库读取出力分配和运行成本，并对结果数据进行图表分析。

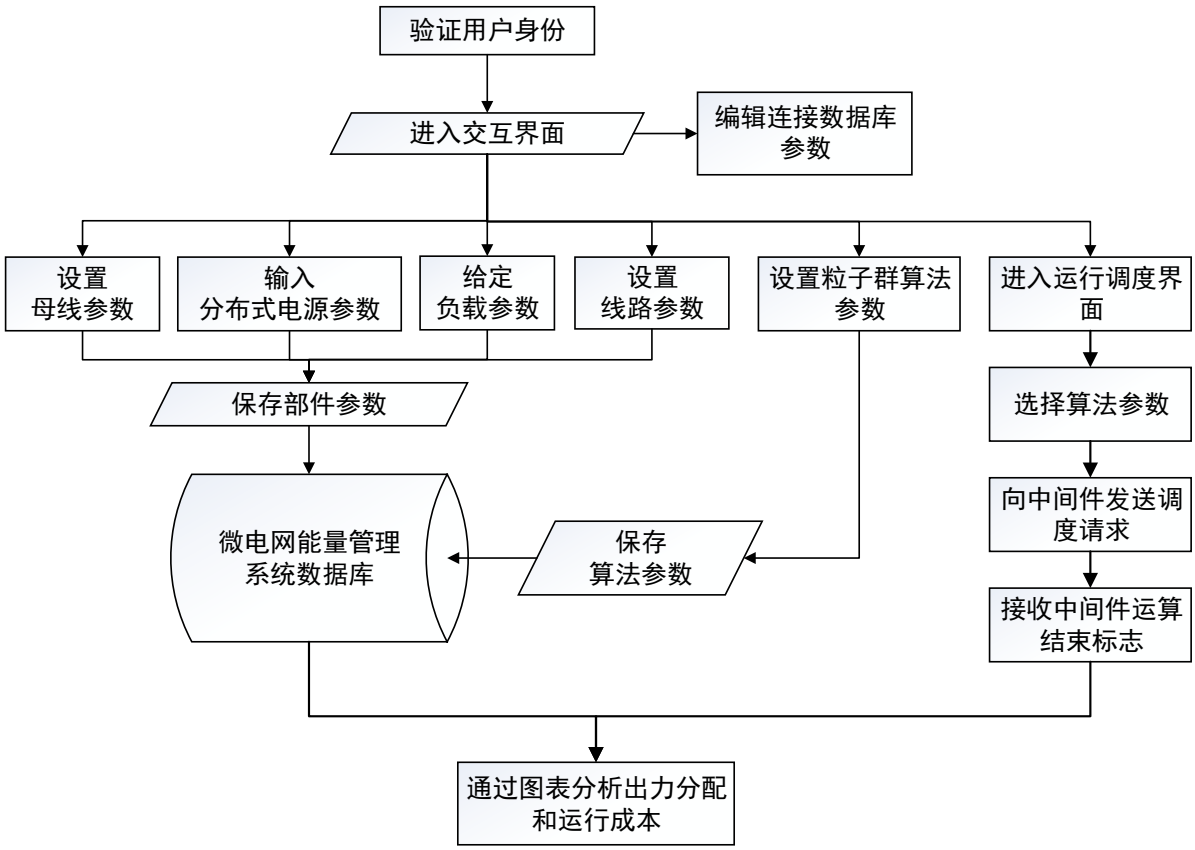

图 4.2 客户端工作流程图

4.1.2.3. 调度中间件

中间件(Middleware)，本来是用于连接系统软件和应用软件之间的软件，方便与软件各部件之间进行沟通[26]。这里，我们用来描述一种连接服务端和客户端之间的软件。本文选用 gSOAP 软件开发包作为调度中间件，用于 SOAP/XML Web 服务和 XML 的通用数据绑定，其主要具有这样的两个方面的功能[27,28]：

1) 调度中间件为客户端提供一定数量的接口函数，这些接口函数，分别对应中间件相应功能，客户端通过接口函数发送请求，让中间件调用各种功能。
2) 调度中间件能够正确处理客户端发来的请求信息，运行相应的函数，调用服务端优化调度算法。

本文使用的 gSOAP 是一款 C 和 C++语言开发工具包，其支持大多数开发平台，包括 Linux、Unix、Windows 和 Android、IOS 等。我们通过 gSOAP 内置转换器，将 C++源码转换为 WSDL 文件，客户端通过解析 WSDL 文件，与服务端进行通信操作[29]。

设计调度中间件时，我们需要完成一个头文件的编写。该文件中，主要包含服务命名空间、调用服务的 URL 地址、供客户端使用的接口函数和监听的端口等。在头文件中，一个接口函数对应服务端的一个服务函数。一般来说，接口函数的的参数主要分为两种：传入参数和状态参数，状态参数通常为接口函数的最后一个参数。我们可以利用这个参数，查看服务函数是否调用成功[30]。

图 4.3 是调度中间件的基本工作原理图：

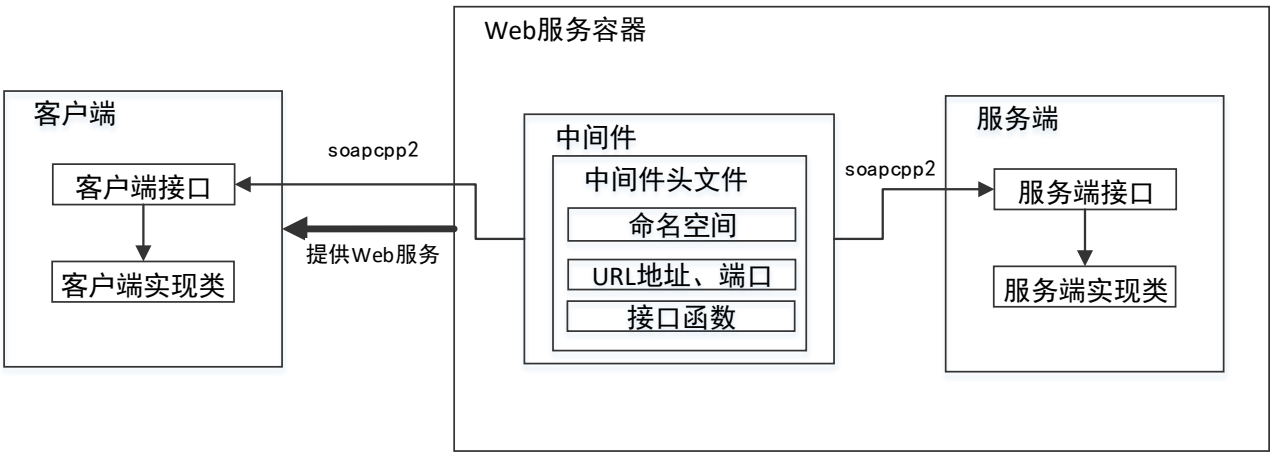

图 4.3 调度中间件原理

4.1.3. 能量管理系统的工作原理

微电网的能量管理系统需要远程服务端、前台客户端和调度中间件协同工作，才能完成一次完整的基于粒子群算法的优化调度任务，其工作原理如图 4.4 所示：

在图 4.4 中，我们可以看到整个系统的完整运行过程。用户在客户端输入系统部件参数和算法参数，选定数据库，将优化调度请求，发送给调度中间件。中间件接受客户端的信号，分析处理后，通过接口函数将信号传递给对应的服务端服务函数，该信号包括系统部件参数和算法参数。

服务端收到信号后，调用相应的粒子群优化算法，在并行环境下，求解最优经济运行成本和分布式电源出力分配。计算完成后，服务端通过调度中间件，将完成信号返回给客户端，这时，我们刷新客户端结果界面，就能发现最新的优化调度结果。

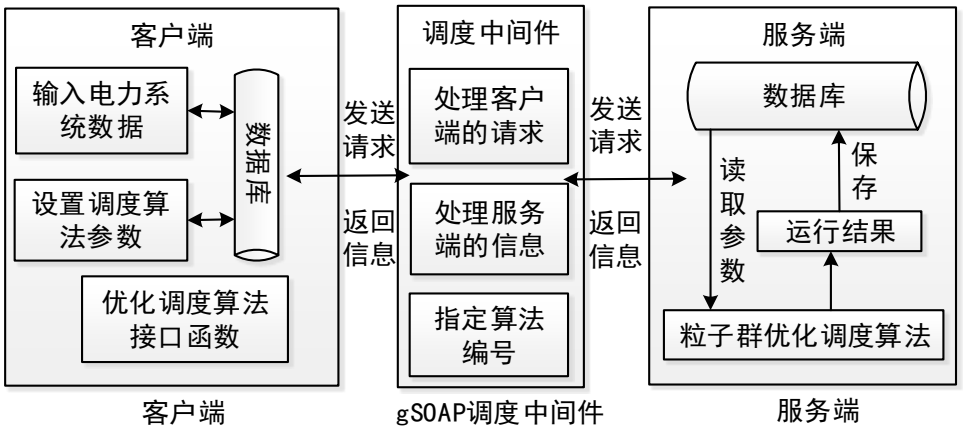

图 4.4 能量管理系统工作原理图

4.2. 能量管理系统的实例分析

4.2.1. 微电网能量管理系统客户端

该能量管理系统基于微软(Microsoft)的.NET Framenwork 4.5.2 平台进行开发，运用 Windows Forms 图形用户界面。Windows Forms 通过将现有的 Windows API 封装为托管代码，实现了对 Windows 本地组件的访问[31]。Windows Forms 客户端界面主要分为如图 4.5 这几个部分:

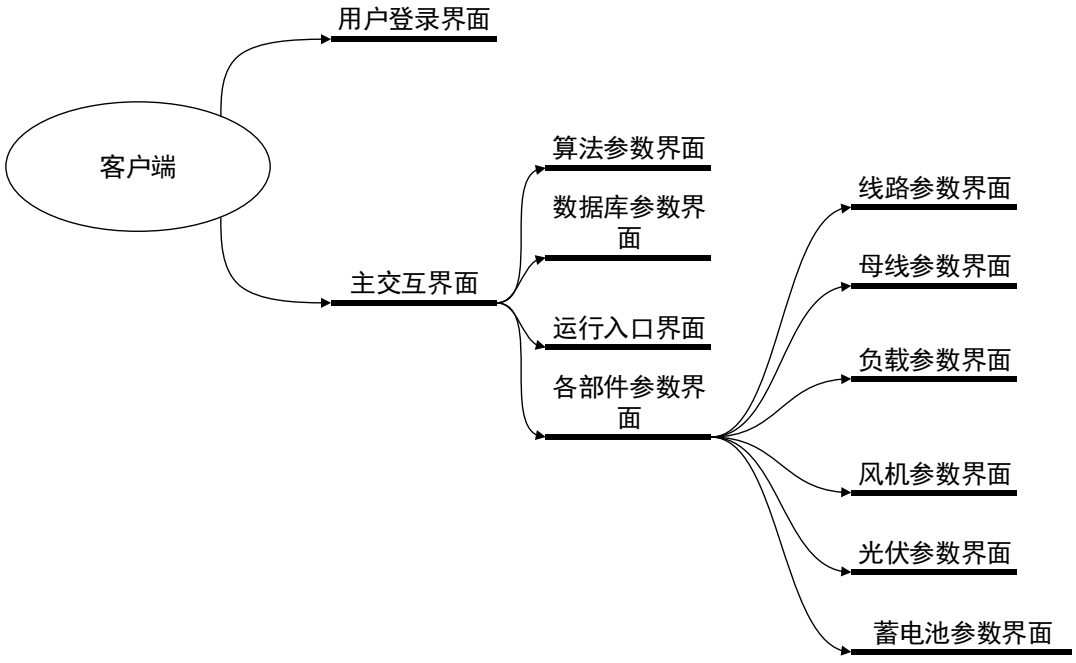

图 4.5 客户端架构图

图 4.6 为用户登录界面，提供了能量管理系统用户名和密码的登录功能。系统的用户信息均储存在服务器数据库中，只有通过系统验证的用户，才能进入主交互界面，否则，将提示“用户名或密码错误”。

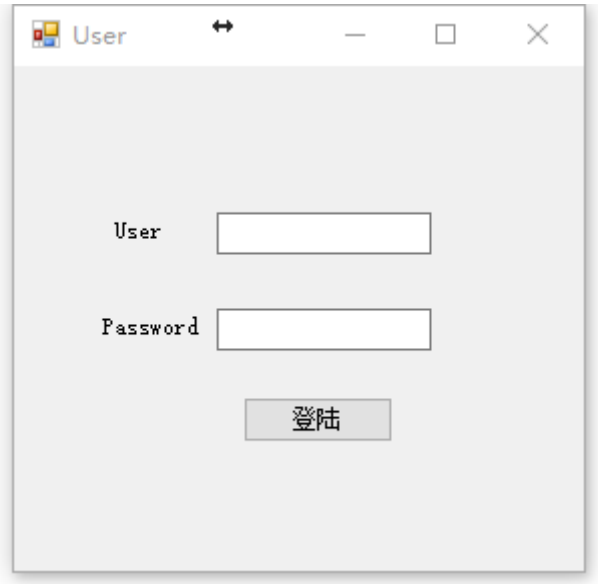

图 4.6 用户登录界面

通过用户名系统验证后，我们进入如图 4.7 的主交互界面。界面主要包括四个功能按钮：编辑各部件参数、更改算法参数、选定粒子群优化算法数据库和进入运行调度入口。

通过按钮，我们可以进入下一级子界面，对各种参数进行更细化的设置。

图 4.7 主交互界面

图 4.8 各部件参数界面

图 4.8 是各部件参数设置参数，这里我们可以对微电网系统的线路参数、母线参数、负载参数和各分布式发电单元(风电、光伏与蓄电池)参数进行设置。

图 4.9 线路参数界面

图 4.9 为线路参数界面，可以设置的项目有：线路编号、系统编号、上接母线标号、下接母线标号、母线电阻、母线电抗、对地电导、对地电纳、变压器变比和线路最大功率，线路编号为数据表主键。

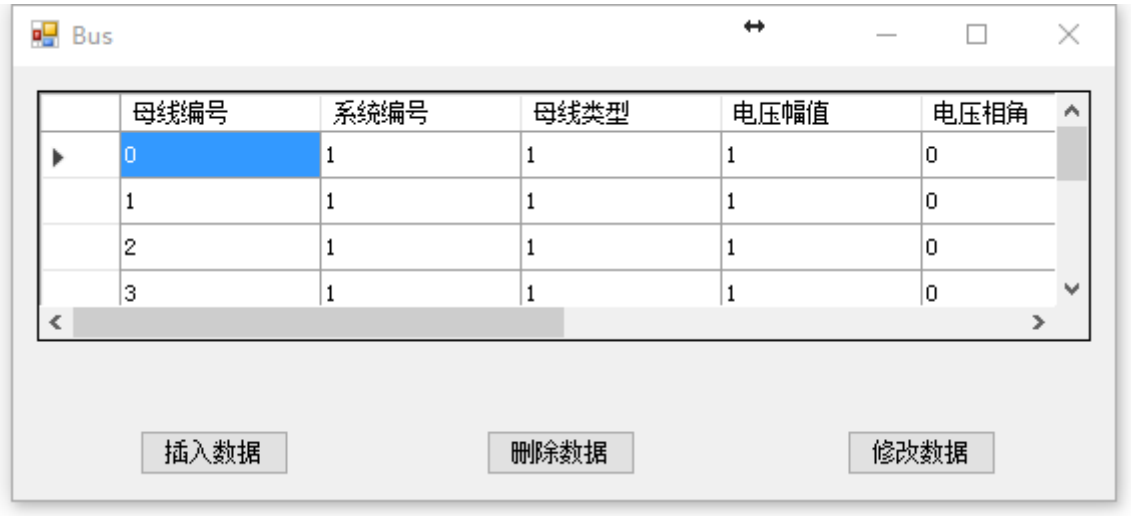

图 4.10 母线参数界面

图 4.10 为母线参数界面，可设置的项目为：母线编号、系统编号、母线类型、电压幅值、电压相角、有功供应、无功供应、电压上限和电压下限，母线编号为数据表主键。

图 4.11 负载参数界面

图 4.11 为负载参数界面，可设置的参数有：负载编号、母线编号、负载时段、有功负荷和无功负荷等，负载编号为数据表主键。

图 4.12 风电参数界面

图 4.12 是各分布式电源中的风电参数界面，其风机参数可变项为：风机编号、母线编号、风机时间段、风机预测发电功率、风机有功功率上限和风机有功功率下限，风机编号为数据表主键。

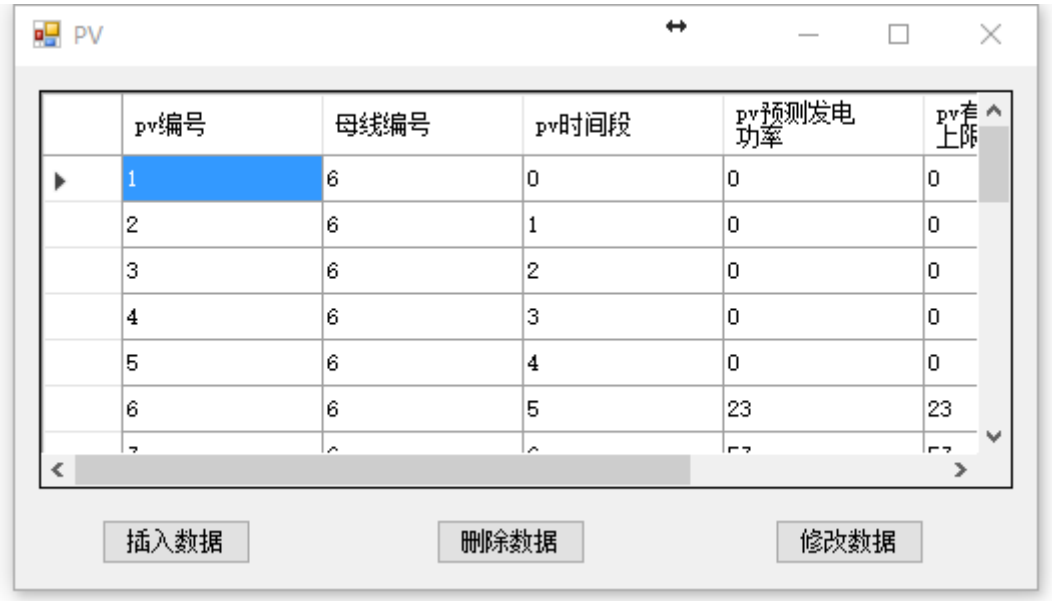

图 4.13 光伏参数界面

图 4.13 为光伏参数界面，其发电单元可变参数为：pv 编号、母线编号、pv 时间段、pv 预测发电功率、pv 有功功率上限和 pv 有功功率下限，pv 编号为数据表主键。

图 4.14 蓄电池参数界面

图 4.14 为蓄电池参数界面，其可变单元为：蓄电池编号、母线编号、蓄电池时间段、蓄电池预测功率、蓄电池有功功率上限和蓄电池有功功率下限，蓄电池编号为其主键。

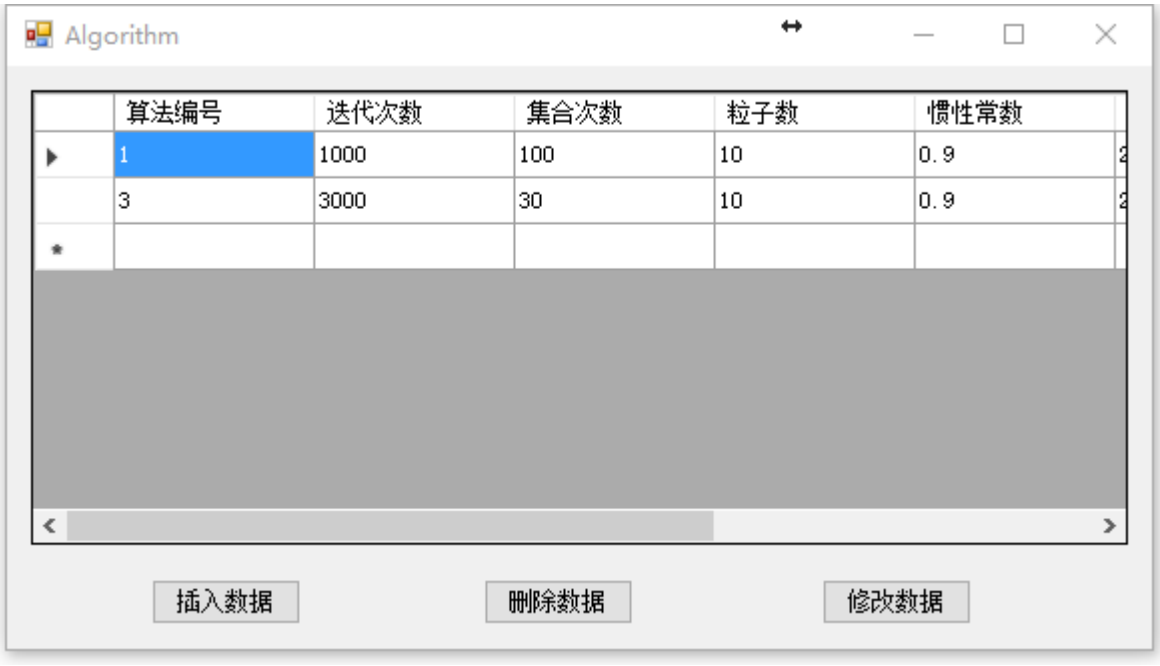

图 4.15 算法参数界面

图 4.15 为算法参数界面，我们可以在界面中添加各种粒子群算法参数和 MPI 并行参数，包括：算法编号、迭代次数、并行集合次数、子进程粒子数、粒子群的惯性常数、两个加速常数和并行计算开辟的进程数。算法标号为数据表主键，需要随信号发送给服务端。

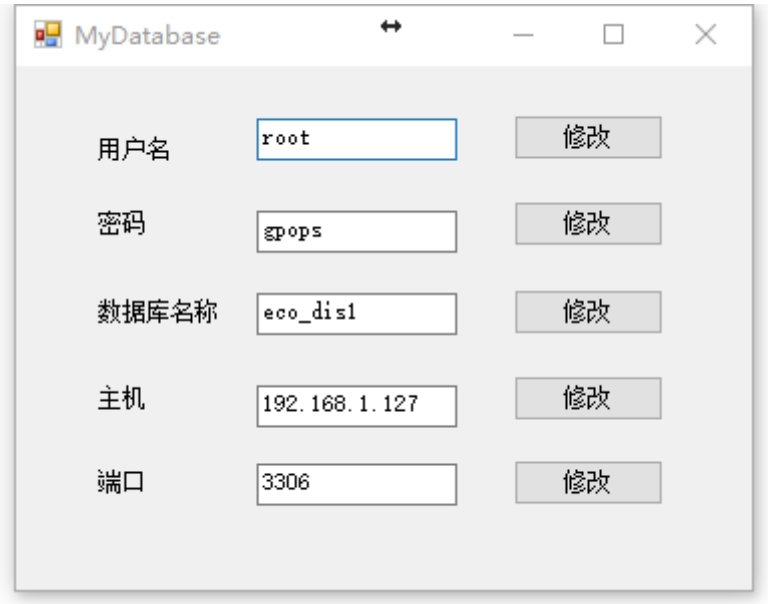

图 4.16 数据库参数界面

图 4.16 是数据库参数界面，我么你可以在其中设置需要传递给服务端的数据库参数，有：用户名、密码、数据库名称、主机和端口。服务端读取对应数据库的相关表格，进行粒子群优化调度运算。

图 4.17 运行入口界面

图 4.17 是客户端的运行入口界面，管理远程服务的调度和运算结果的数据分析，是客户端的核心部分。图 4.18 和图 4.19 分别为计算结果中的运行成本收敛曲线和各发电单元的出力分配曲线，是调用 ZedGraph 开源绘图工具包进行绘制的。

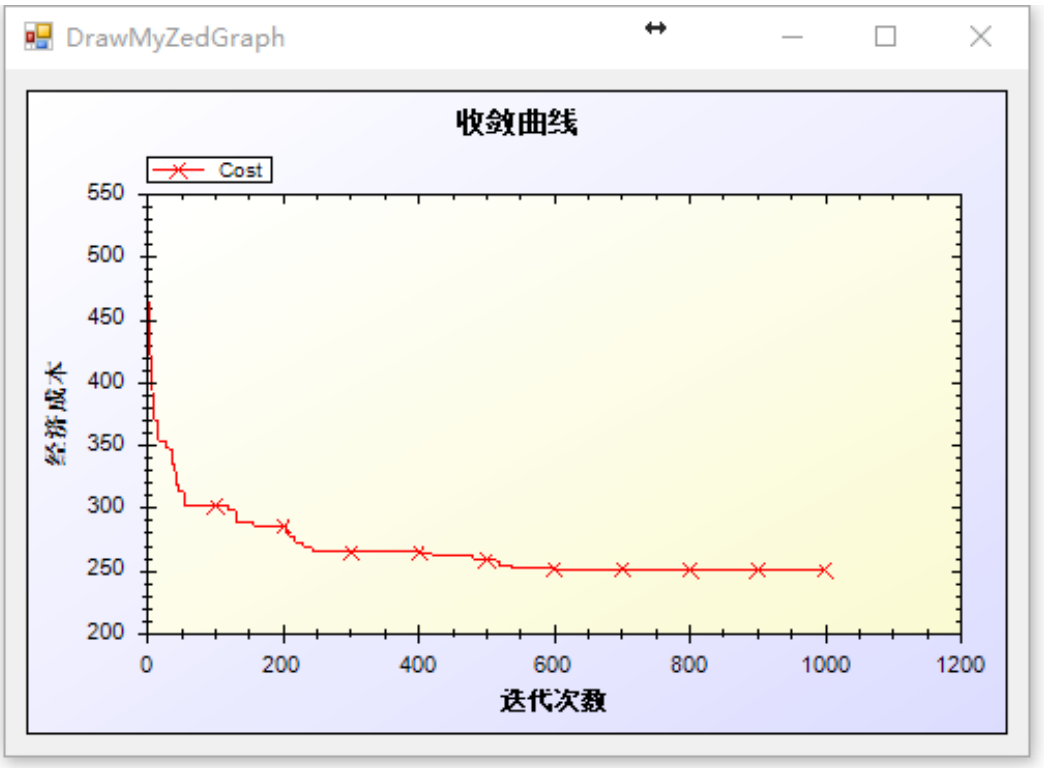

图 4.18 收敛曲线

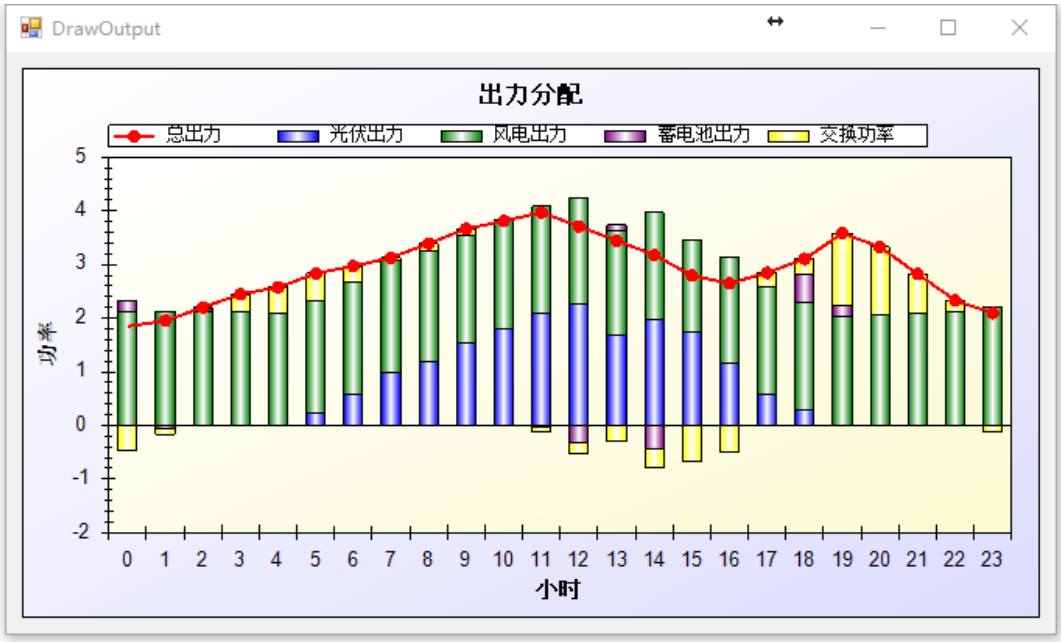

图 4.19 出力分配曲线

4.2.2. 微电网能量管理系统服务端

能量管理系统的服务端，主要分为粒子群优化调度程序和微电网能量管理系统数据库。下面就二者实例进行简要分析。

4.2.2.1. 粒子群优化调度程序

服务端程序采用面向对象(Object-oriented programming，OOP)的程序设计方法。对象是类的实例，也是程序的基本单元。我们将程序和数据封装在对象中，提高了软件的复用性、灵活性和拓展性[32]。

服务端程序的类主要分为如图 4.20 的三个种类，分别为：各部件参数类、核心运算类和矩阵辅助类。

各部件参数类中主要为组成微电网系统的各种电路参数，主要包括各自参数的构造和析构函数。核心运算类由读数据(ReadData)、潮流计算(PowerFlow)、约束调整(Adjustment)和粒子群优化(PSO)四个部分组成。MPI 并行计算主要在粒子群优化部分体现。而辅助类中，我们加入了矩阵求逆(InverseMatrix)的运算单元，方便在潮流计算类中进行调用。

通过类的封装，保证了服务端程序各个模块的数据独立性和逻辑抽象性，也使整个程序的架构变得清晰明了[33]。

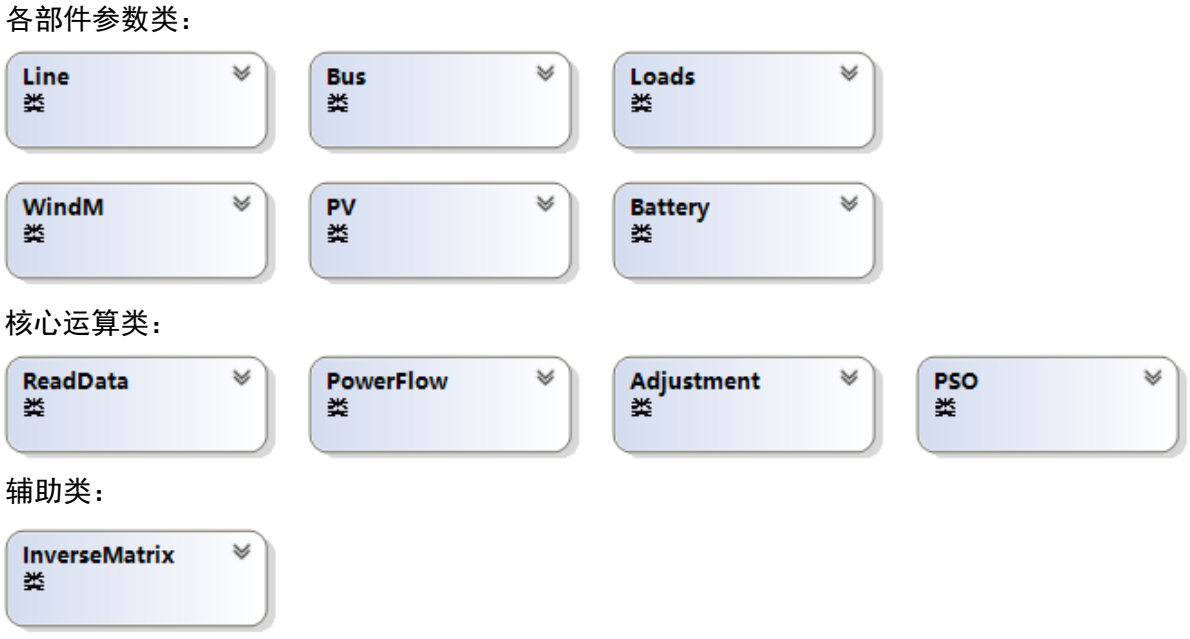

图 4.20 粒子群优化调度程序的类

4.2.2.2. 能量管理系统数据库

(1) 数据库设计流程

我们在构建微电网能量管理系统时，首先需要进行需求分析，包括数据需求分析与功能需求分析。一个结构良好的数据库，能够支撑能量管理系统各模块工作的需要。一个内容完整、结构清晰、数据表相对独立、冗余度小的数据库，是能量管理系统的基石。

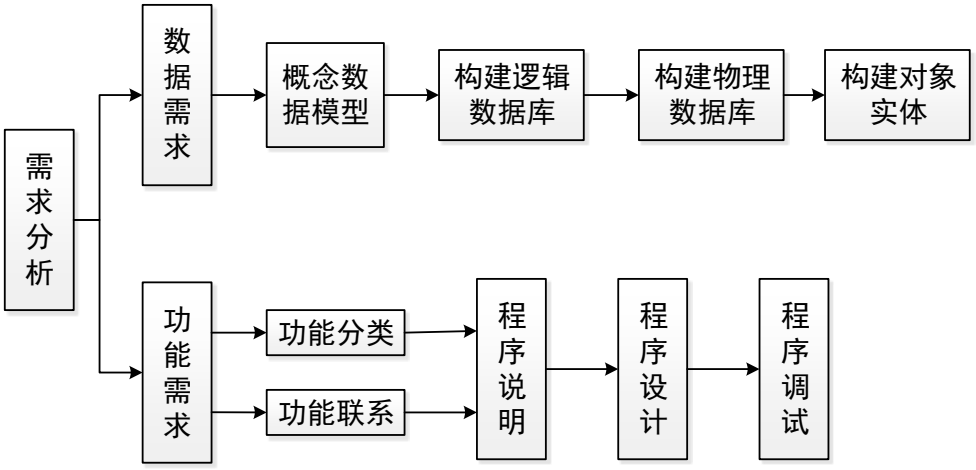

图 4.21 能量管理系统数据库开发流程图

图 4.21 是微电网能量管理系统的开发流程图，我们需要明确数据库开发中的数据需求和功能需求。数据需求需要我们了解各项数据之间的关系，构建数据模型；随后，在 PowerDesigner 中根据数据模型构建逻辑数据库，进而构建物理数据库，最后构建实例对象。而对于功能需求，我们则需要为数据库设计相应的连接程序，如客户端中，添加、删除和修改数据表格内容的功能模块。

(2) 微电网能量管理系统的数据库模型

图 4.22 展示了微电网能量管理系统的数据库表格。

line 表、bus 表和 loads 表中储存电路参数信息，windm 表、battery 表和 pv 表，分别储存了各分布式发电单元(风电、蓄电池和光伏)的参数信息。mydatabase 表储存数据库端口、主机地址、数据库名称等信息。algorithm 表储存粒子群算法和 mpi 并行算法的相关参数。power_distribution 表和 results 表则分别储存运算完成后得到的各发电单元出力分配和最优运行成本。

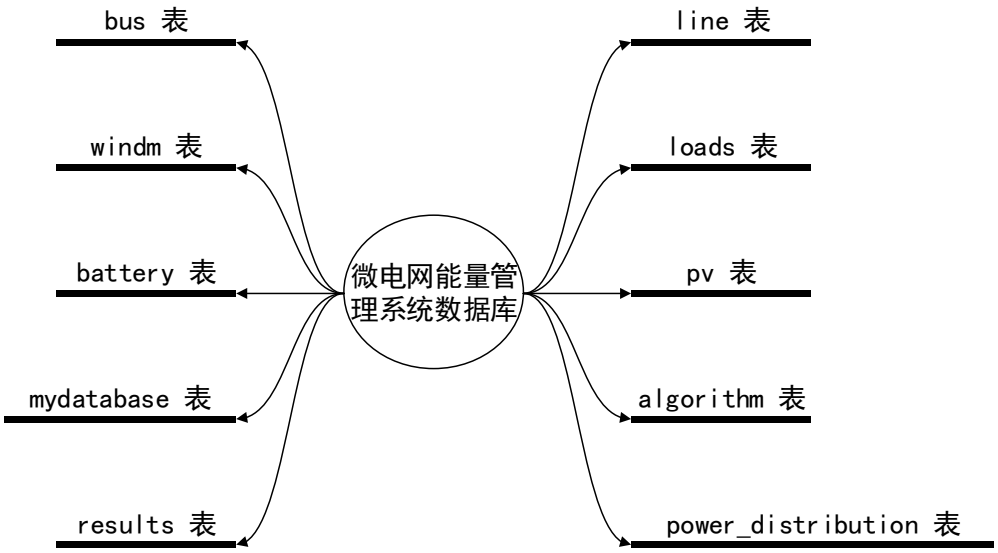

图 4.22 微电网能量管理系统的数据库表格

第 5 章　IEEE 九节点模型算例分析

基于 IEEE 九节点系统的微电网模型，主要包括这些参数：线路参数、母线参数、一天 24 各小时的负载参数、风电单元参数、光伏单元参数、蓄电池单元参数。

5.1. IEEE 九节点微电网系统模型

我们构建如图 5.1 的系统拓扑，系统中包含九个节点：六个 PQ 节点、两个 PV 节点和一个平衡(slack)节点。六个节点中，有三个节点接入了负载，两个 PV 节点，一个接入了光伏发电单元和蓄电池，一个接入了风力发电单元。在网络系统中，平衡节点亦为微电网与外界大电网的购电、售电交换节点。

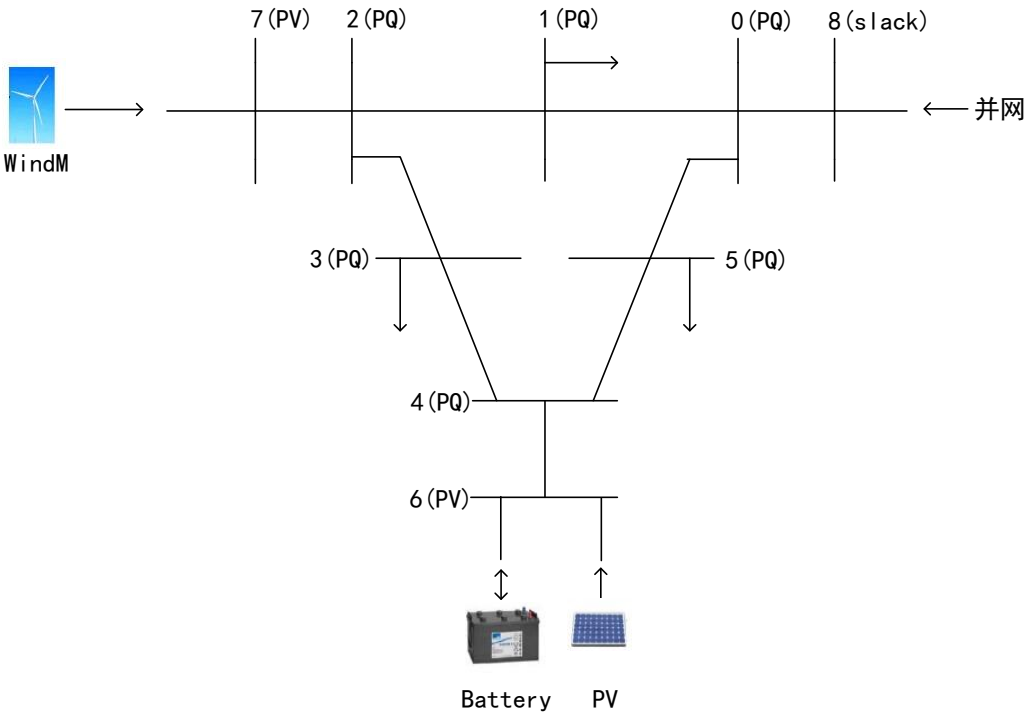

图 5.1 九节点算例网络结构图

本系统的线路参数和母线参数均通过 matpower 的 case9 算例获得，如表格 5.1 和表格 5.2 所示：

表格 5.1 算例母线参数

母线编号	母线类型	Gs	Bs	Vm	Va	Vmax	Vmin
0	PQ	0	0	1	0	1.1	0.9
1	PQ	0	0	1	0	1.1	0.9
2	PQ	0	0	1	0	1.1	0.9
3	PQ	0	0	1	0	1.1	0.9
4	PQ	0	0	1	0	1.1	0.9
5	PQ	0	0	1	0	1.1	0.9
6	PV	0	0	1	0	1.1	0.9
7	PV	0	0	1	0	1.1	0.9
8	Slack	0	0	1	0	1.1	0.9

表格 5.2 算例线路参数

fbus	tbus	r	x	b	g	Ratio
8	0	0	0.0576	0	0	0
0	1	0.017	0.092	0.158	0	0
1	2	0.039	0.17	0.358	0	0
7	2	0	0.0586	0	0	0
2	3	0.0119	0.1008	0.209	0	0
3	4	0.0085	0.072	0.149	0	0
4	6	0	0.0625	0	0	0
4	5	0.032	0.161	0.306	0	0
5	0	0.01	0.085	0.176	0	0

三个 PQ 节点(1、3、5)均接入了负载，分别为负载 1、负载 2 和负载 3，其中负载 1 和负载 3 数据一致[34]。负载需求如表格 5.1 所示，图 5.2 为负载曲线，由曲线我们可以看出，一天内有两个用电高峰(中午和晚上)。

表格 5.3 算例 24 小时负载需求

时间	负荷 1	负荷 2	负荷 3	时间	负荷 1	负荷 2	负荷 3
1	70	35	70	13	140	70	140
2	75	37	75	14	130	65	130
3	85	42	85	15	120	60	120
4	95	47	95	16	105	52	105
5	100	50	100	17	100	50	100
6	110	55	110	18	110	55	110
7	115	57	115	19	120	60	120
8	120	60	120	20	140	70	140
9	130	65	130	21	130	65	130
10	140	70	140	22	110	55	110
11	145	72	145	23	90	45	90
12	150	75	150	24	80	40	80

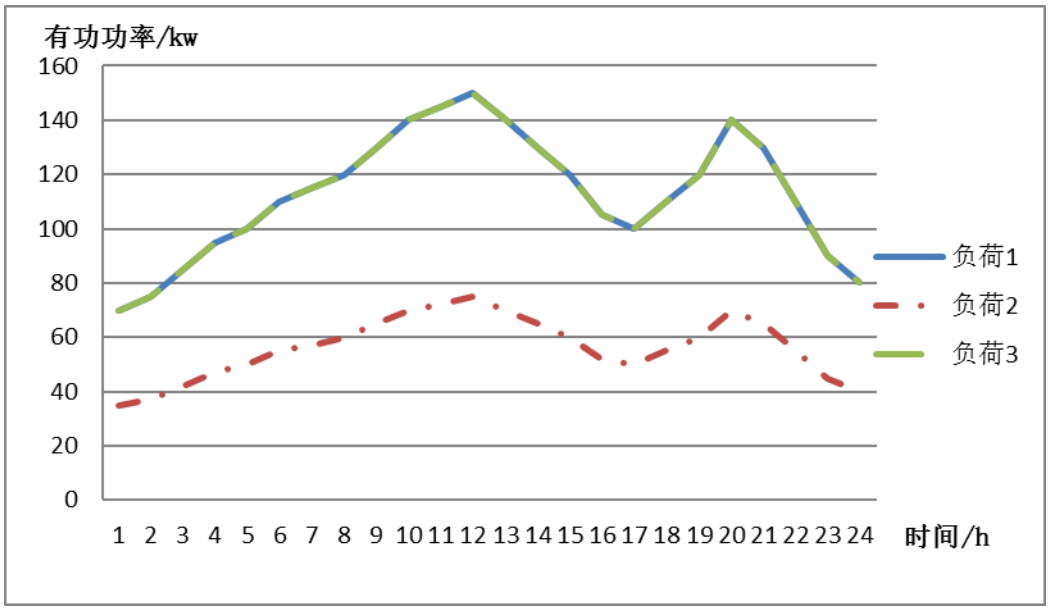

图 5.2 算例 24 小时负载曲线

在本算例中，PV 节点 6 接入了光伏发电单元和蓄电池单元，PV 节点 7 接入了风电单元。光伏单元和风电单元的最大有功功率如表格 5.5 所示，图 5.3 为二者 24 小时内最大有功功率曲线。

由曲线我们可以看出，风电单元一天中发电基本平稳，光伏单元最大功率随太阳强度变化呈正相关。考虑图 5.2，用电最高峰和发电最高峰时刻并没有绝对重合，所以我们需要蓄电池单元进行调节。为保证蓄电池寿命，我们将需带你吃最低容量设为 30%，最高仍为 100%。

当蓄电池单元无法满足系统需求时，微电网系统就需要和外界大电网进行功率交换。

表格 5.4 算例 24 小时光伏单元和风电单元最大有功功率

时间	太阳能电池	风力发电机	时间	太阳能电池	风力发电机
1	0	213	13	226	198
2	0	212	14	239	197
3	0	211	15	257	199
4	0	211	16	173	200
5	0	210	17	144	200
6	23	209	18	57	201
7	57	209	19	28	202
8	100	207	20	0	204
9	120	205	21	0	207
10	154	201	22	0	209
11	180	202	23	0	213
12	209	199	24	0	222

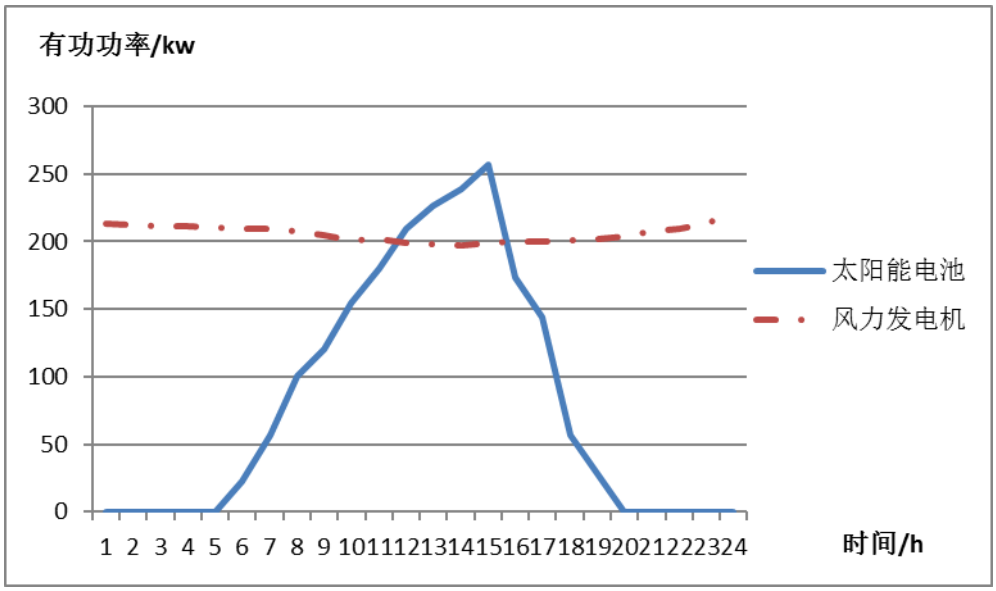

图 5.3 算例 24 小时光伏单元和风电单元最大有功功率曲线

5.2. IEEE 九节点微电网系统算法参数设置

本文使用的粒子群算法的主要算法参数有这几个：惯性权重 w、加速常数 $c1$ 和 $c2$。设置合理的粒子群算法参数可以加快运算速度，同时让收敛效果更加良好。

惯性权重 w，使粒子保持运动惯性，拓展粒子的空间搜索能力，让其有能力不断探索新的区域。我们通过调整 w 的大小，从而控制前一个时刻速度，对现在时刻速度的影响，使其平衡全局搜索(exploration)能力和局部搜索(exploitation)能力。w 较大时，速度 v 可以取得更大的值，有利于扩大搜索空间，增强全局搜索能力；而 w 较小时，速度 v 减小，有利于在当前适应值附近挖掘更优解，以增强粒子局部搜索能力[35]。

这里，我们使用线性递减惯性权重法(Linearly Decreaseing Weight,，LDM)，在迭代中确定惯性权重 w：

$$w = w_{\max} - \frac{w_{\max} - w_{\min}}{iter} * deltat \tag{4.1}$$

式(4.1)中，$w_{\max}$ 和 $w_{\min}$ 分别是 w 的最大值和最小值，$iter$ 和 $deltat$ 分别是最大迭代次数和当前迭代次数。由式(4.1)，我们可以看出在迭代开始时 $w = w_{\max}$，随着迭代过程的不断推进，w 单调递减，直到 $w = w_{\min}$。通过这种方式，在刚开始搜索时，能够快速确定合适的最优解范围，随着 w 逐渐减少，粒子速度变慢，能够在较小区域内进行更精细的搜索，加快了算法收敛速度，增强了 PSO 算法的收敛性[36]。

加速常数 $c1$ 和 $c2$ 用于调节粒子“自身经验”和“社会经验”在其运动中的参考权重。若 $c1 = 0$，粒子只有“社会经验”，会造成收敛过快，容易陷入局部最优解。若 $c2 = 0$，粒子只有“自身经验”，与社会没有交流，几乎不能得到较优解。综合考虑，在本算例中，我们选择 $c1 = c2 = 2$，以获得较优的个体经验和社会经验[37]。

粒子的种群规模 N，代表一个维度空间内，进行粒子群优化的粒子数量。当 N 较小时，粒子群运算速度加快，但种群多样性变差，可能造成过早收敛；而 N 较大时，PSO 的运算效率又会变慢。在 MPI 多进程并行计算中，粒子群规模 N 还需要综合考虑进程数量 $processNum$。综合考虑，在本算例中，我们选取种群规模 $N = 10$，$processNum = 20$，且 $processNum$ 平均分配在两个服务器节点上。

5.3. IEEE 九节点微电网系统经济调度结果分析

选择某一次运算结果进行分析，设定算法参数：迭代次数为 2000 次，集合次数 10 次，粒子数 10 个，进程数为 20 个(且平均分配在两个服务器节点上)。由图 5.4 可看

出，最后运行成本收敛于 235.91 元。

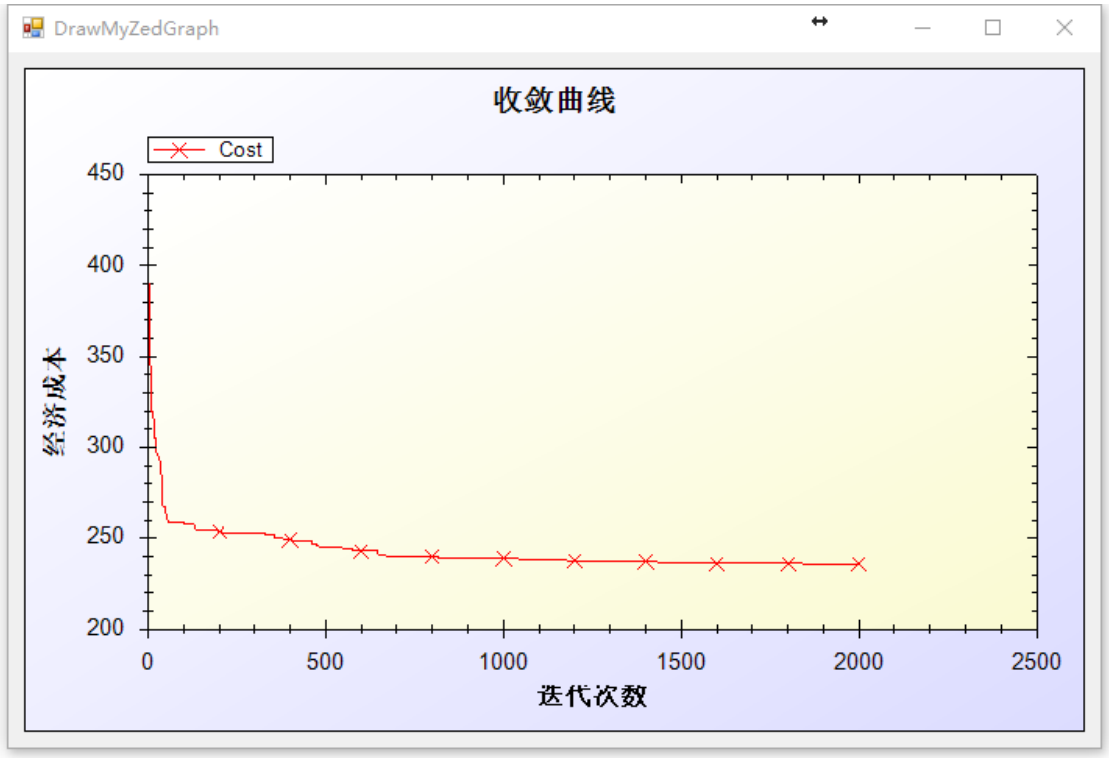

图 5.4 算例运行成本

从图 5.4，我们可以看出，在开始的 100 次迭代，我们的目标函数(经济成本)快速收敛，很快从 404 左右降低到 300 以下。迭代次数达到 800 以后，收敛变慢，接近我们需要的最优解，这样的特性，正是由粒子群算法的惯性权重所决定的。在迭代初期，粒子速度快，收敛快，便于快速定位全局最优解范围；迭代后期，粒子速度减缓，利于精确搜索最优解。

表格 5.5 各时段发电单元出力分配

时段	蓄电池出力	光伏出力	风电出力	交换功率	总出力
0	-0.311727	0	2.13	0.005452	1.82372
1	-0.185198	0	2.12	0.010238	1.94504
2	0.012176	0	2.11	0.075507	2.19768
3	0.15715	0	2.11	0.187481	2.45463
4	0.416481	0	2.1	0.077328	2.59381
5	0.111118	0.23	2.09	0.405953	2.83707
6	0	0.57	2.09	0.308988	2.96899
7	0	1	2.07	0.058026	3.12803
8	0	1.2	2.05	0.139737	3.38974
9	0	1.54	2.01	0.115064	3.66506
10	0	1.8	2.02	-0.008264	3.81174
11	0	2.07828	1.99	-0.096987	3.97129
12	-0.073924	2.0206	1.98	-0.215754	3.71093
13	-0.62459	2.38961	1.97	-0.291064	3.44394
14	2.1E-05	1.83317	1.82487	-0.467343	3.19072
15	5.5E-05	1.72394	1.73106	-0.660864	2.79418
16	0.000109	1.44	1.83098	-0.61283	2.65826
17	-0.001672	0.57	2.01	0.268143	2.84647
18	0.50585	0.28	2.02	0.299936	3.10579
19	0.19415	0	2.04	1.34712	3.58127
20	0	0	2.07	1.25709	3.32709
21	0	0	2.09	0.734126	2.82413
22	0	0	2.13	0.199599	2.3296
23	0	0	2.22	-0.127761	2.09224

表格 5.5 和图 5.5 分别为各时段发电单元的出力分配表格和柱状图，通过图 5.5 我们可以清晰地看出各时段发电单元的出力情况。

结合图 5.2 和图 5.5，我们可以看出一天内负载变化和总出力变化基本一致，我们可以用总出力变化趋势替代负载趋势进行分析。

从全局看，一天内的负载有两个高峰，分别为中午和傍晚。第一个高峰(中午)，正好也是光伏出力的高峰，光伏出力和风电出力加起来，完全可以负担整个微电网系统的用电需求，剩余电量可以向蓄电池充电。在蓄电池充满后，盈余的电量通过平衡节点，出售给大电网，以减少系统的经济成本。而第二个高峰(傍晚)，由于光伏出力减少，风机发电量和蓄电池储能，不足以供给整个微电网系统，所以，我们需要向大电网购电，这段时间的经济成本和中午相比是较高的。

其它时段，如午夜(23:00-2:00)，风电基本能够满足系负载需求，系统还能向蓄电池充电，但随着凌晨(3:00-6:00)负载逐渐增加，我们需要向主网购电以满足系统需求。但随着光伏出力的增加，购电需求逐渐减小至 0。

一天中，蓄电池充放电规律如所图 5.6 示，其充放电高峰均在夜间和下午，刚好与系统负载高峰相反，体现了蓄电池作为储能单元“峰用谷充”的良好特性。

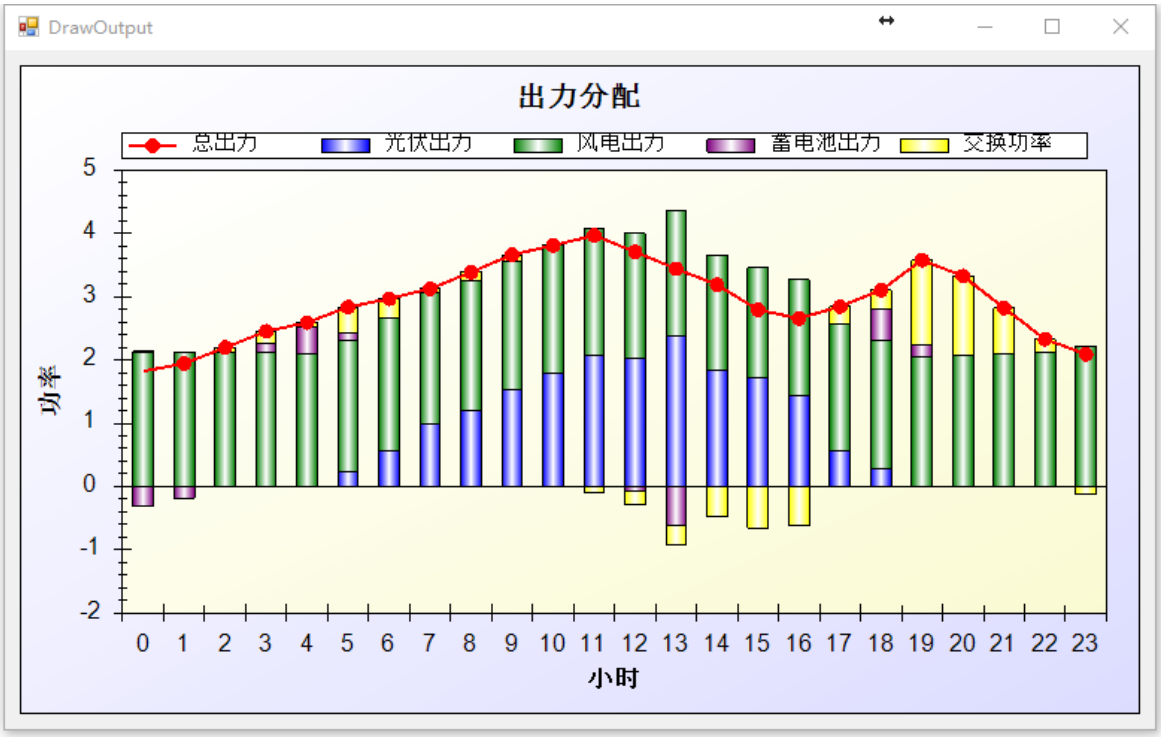

图 5.5 各时段发电单元出力分配柱状图

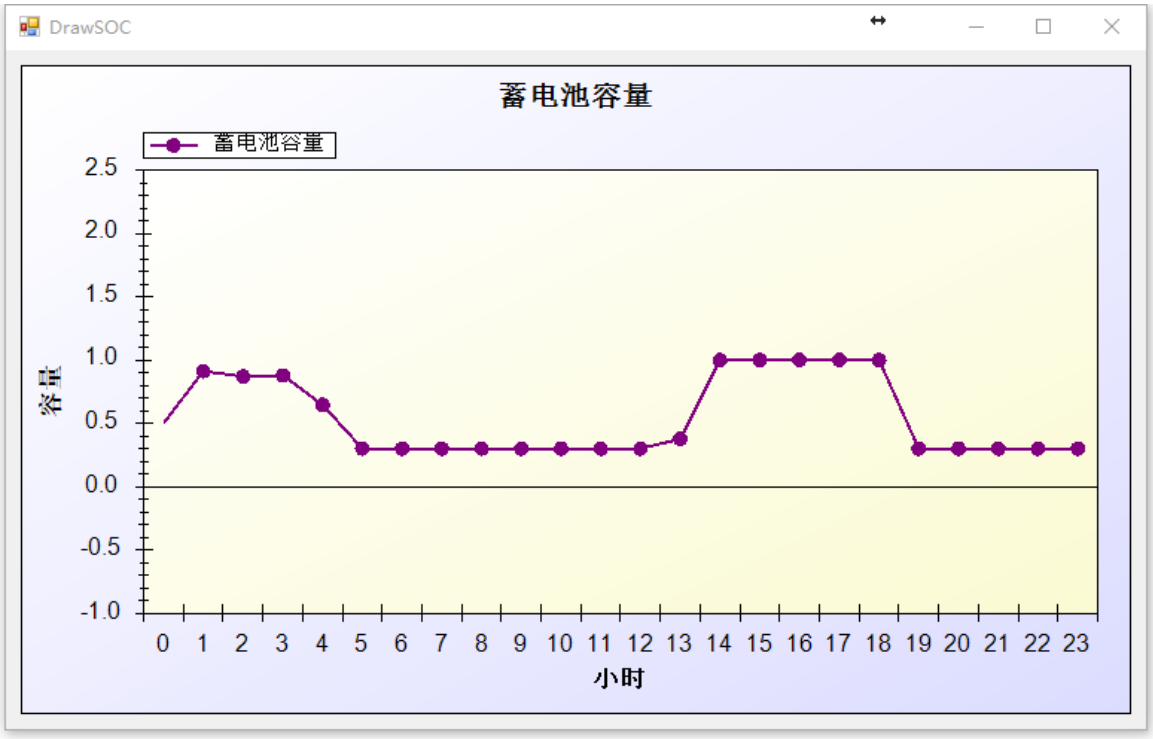

图 5.6 各时段蓄电池容量变化

5.4. 粒子群算法参数和 MPI 并行算法参数对算例结果影响的分析

本节，我们通过客户端设定不同的算法参数，包括不同进程数、不同集合次数、不同种群规模和不同迭代次数，对 MPI 并行粒子群算法性能进行分析。

5.4.1. MPI 算法参数对算例结果的影响

考虑 MPI 算法参数的影响时，我们控制粒子群相关参数(种群规模和迭代次数)为常量，从而进行分析。这里，我们设定迭代次数为 1000 次，种群规模为 2，针对每个性能参数取 3 个不同值分别进行 20 次运算，取运行成本和运行时间(不包括数据库 IO 读写时间)的平均值进行分析。

集合次数不同：进程数为 2，集合次数分别为：5、10、30。

进程数不同：集合次数为 10，进程数分别为 2、5、10。

表格 5.6 不同集合次数的计算结果

集合次数	成本(元)	运行时间(s)	进程数
5	269.51	12.1	2
10	262.04	13.2	2
30	253.23	14.5	2

由表格 5.6 可以看出，集合次数越多，算法计算结算结果变好。但由于集合发散次数增加，通过消息传递实现的并行算法消耗时间也会增加，所以我们需要合理设定集合次数。

表格 5.7 不同进程数的计算结果

进程数	成本(元)	运行时间(s)	集合次数
2	262.04	13.2	10
5	247.18	14.01	10
10	240.04	15.53	10

表格 5.7 展示了不同进程数的计算结果，随着算法进程数的增加，微电网运行成本结果更优，但随着进程数增加，CPU 开销和集合发散所需时间都更多，所以需要耗费更多运行时间。

我们在指定 MPI 并行程序的进程数时，需要考虑计算机的逻辑 CPU 数量。本文中使用两台 dell 服务器，每台服务器节点的逻辑 CPU 数分别为 40 和 32 个，我们在设定进程数量时，尽量不要超过服务器的逻辑 CPU 数。否则，可能会出现多个进程在一个逻辑核上运行的问题，这样就会增加 CPU 的调度时间，造成执行效果变差[38,39]。

5.4.2. 粒子群算法参数对算例结果的分析

考虑粒子群算法的参数的影响时，我们设置 MPI 并行算法参数(集合次数和进程数)为常量。我们这里设置集合次数为 5，进程数为 2，针对每个性能参数取 3 个不同值分别进行 20 次运算，也取运行成本和运行时间的平均值进行分析。

迭代次数不同：种群规模为 4，迭代次数分别为 1000、2000、3000。

种群规模不同：迭代次数为 1000，种群规模分别为 4、10、20。

从表格 5.8 中，我们可以看出，随着迭代次数的增加，运算结果主键趋向最优解，但运算时间自然会随之增加，而且在迭代后期，优化效果不会太明显。所以我们应该为算法选择合适的迭代次数，从而加快优化调度的运算速度。

表格 5.8 不同迭代次数的计算结果

迭代次数	成本(元)	运行时间(s)	种群规模
1000	247.99	26.1	4
2000	244.70	49.7	4
3000	243.32	72.4	4

考虑表格 5.9，我们可以清晰地看到，随着种群规模的扩大，微电网运行成本结果更优，但随着种群规模的扩大，算法耗费时间会增加，优化效果也不会特别显著，所以需要合理选择种群规模。

表格 5.9 不同种群规模的计算结果

种群规模	成本(元)	运行时间(s)	迭代次数
4	247.99	26.1	1000
10	239.25	65.9	1000
20	238.26	130.5	1000

第 6 章　结论与展望

6.1. 本文工作总结

电力系统优化调度在数学模型上，是具有大量非线性约束条件，且包含连续与离散变量的大规模非线性优化问题。传统算法大多针对优化模型的可导性和可微性进行分析，具有较大的局限性。由于人工智能算法，大都是建立在生物智能或物理现象基础上的随机搜索算法。这类算法一般不要求目标函数和约束的连续性和凸性，和传统算法相比，具有更强的全局搜索能力，所以，被广泛运用于电力系统优化问题的求解。

本文以智能算法中的粒子群优化算法为基础，考虑潮流约束和功率约束，建立一个考虑风能、光伏、蓄电池互补的与大电网并网的微电网能量管理系统，并引入了 MPI 并行计算，以解决粒子群算法，靠增大种群规模而获得较优全局搜索能力，所造成的运算时间问题，提高了能量管理系统电力优化的运行速度。本文主要做了如下工作：

(1) 提出了一种基于 MPI 程序设计的小种群粒子群优化算法。基于 MPI 并行设计的程序，充分利用了服务器集群的性能，将单进程大种群粒子群算法，拆分为多进程小种群粒子群算法，分配给不同的逻辑 CPU 核进行运算，提高了系统电力优化的运行速度。

(2) 以 IEEE 九节点微电网模型为对象，构建了微电网 24 小时时段内的优化调度数学模型，并采用小种群粒子群算法进行求解，计算出每时刻的各发电单元的出力分配和总的最佳运行成本，并对数据进行图像分析。

(3) 以(1)和(2)的优化调度算法和微电网模型为基础，编码实现了面向 Web 服务的微电网能量管理系统。本系统由远程服务端、前台客户端和连接二者的调度中间件三部分构成，系统参数和算法参数均储存在服务端的 MySQL 数据中。

(4) 本系统服务端采用 C++编写，设计为面向对象的 MPI 并行程序模型，编译器选用 MPICH2 并行库的 mpic++，负责系统电力优化的核心计算。客户端采用 C#编写，基于微软的.NET Framenwork 4.5.2 平台进行开发，运用 Windows Forms 图形用户界面，负责系统的人机交互功能，实现系统参数和算法参数的删改，并运用 ZedGraph 图形控件，对出力结果和运行成本进行分析。调度中间件使用 gSOAP 软件开发包开发，用于

SOAP/XML Web 服务和 XML 的通用数据绑定，负责系统服务端和客户端之间的通信。

6.2. 系统存在问题

本文建立的微电网能量管理系统，目的是为了满足一天个时段的优化调度需求。但在实际应用中，还存在这些问题：

(1) 能量管理系统的最小计算时段为小时，无法针对更精细的时段(如 15min)，进行操作。

(2) 管理系统的历史纪录能力待加强，暂时只能收集运行成本的最优解，无法记录上次运算的出力分配。

(3) 用户交互体验有待提高，系统参数和各电源参数均用表格显示，以后可以考虑用网络拓扑结构进行展示。

(4) 目前构建的模型尚未考虑迁徙操作，可能有陷入局部最优解的风险。之后可以通过设定种群聚集度的容忍上限，通过迁徙操作在当前最优解附近重新生成新种群，增加种群多样性，避免局部最优的陷阱。

(5) 目前构建的模型中还未考虑到无功补偿的作用，以后需要对模型进行完善。

6.3. 未来的展望

为了在今后的研究中不断积累和完善，笔者认为未来还可以进行这几个方面的工作：

(1) 丰富客户端系统的功能，提高客户端数据分析的能力。目前客户端仅仅在 Windows 平台下实现，未来可以在其它平台解析 WSDL 描述，以实现与服务端通信，如网页浏览器、Android 平台等。

(2) 系统的 MPI-PSO 算法是基于 CPU 平台的并行技术，进程并行数受到 CPU 逻辑核心数的限制。近年来，GPU 在通用计算领域大放异彩。和 CPU 相比，GPU 的处理核心更多，能够承受大量粒子线程的并行，线程间共享内存的通信速度也快于 MPI 模型的进程间通信。我们可以考虑将系统的粒子群进程，转化为单粒子线程，在大规模电力系统优化中，可以获得更快的运算速度[40]。

参考文献

[1] ETO J, LASSETER R, SCHENKMAN B 等. Overview of the CERTS Microgrid laboratory Test Bed[C]//Integration of Wide-Scale Renewable Resources Into the Power Delivery System, 2009 CIGRE/IEEE PES Joint Symposium. 2009: 1–1.

[2] 杨新法, 苏剑, 吕志鹏等. 微电网技术综述[J]. 中国电机工程学报, 2014(1): 57–70.

[3] CRISTALDI L, FERRERO A, MUSCAS C 等. The impact of Internet transmission on the uncertainty in the electric power quality estimation by means of a distributed measurement system[J]. IEEE Transactions on Instrumentation and Measurement, 2003, 52(4): 1073–1078.

[4] 崔琼, 舒杰, 吴志锋. 微电网能量管理系统功能结构研究[J]. 电测与仪表, 2015(05 vo v.52;No.606): 118–122.

[5] 吴雄, 王秀丽, 刘世民等. 微电网能量管理系统研究综述[J]. 电力自动化设备, 2014(10 vo v.34;No.246): 7–14.

[6] LIANG H Z, GOOI H B. Unit commitment in microgrids by improved genetic algorithm[C]//IPEC, 2010 Conference Proceedings. IEEE, 2010: 842–847.

[7] SHENG W, LIU K, CHENG S. Optimal power flow algorithm and analysis in distribution system considering distributed generation[J]. Generation, Transmission & Distribution, IET, 2014, 8(2): 261–272.

[8] LEVRON Y, GUERRERO J M, BECK Y. Optimal power flow in microgrids with energy storage[J]. Power Systems, IEEE Transactions on, 2013, 28(3): 3226–3234.

[9] 任垚, 张小青. 基于改进动态规划法的电力系统机组组合问题研究[J]. 自动化技术与应用, 2010(05 vo v.29;No.179): 6–8+12.

[10] 王承民. 电力市场中一种基于动态规划法的经济负荷分配算法[J]. 自动化技术与应用, 2010(05 vo v.29;No.179): 6–8+12.

[11] 寇晓丽. 群智能算法及其应用研究[D]. 西安电子科技大学, 2009.

[12] MOHAMMADI M, HOSSEINIAN S H, GHAREHPETIAN G B. Optimization of hybrid solar energy sources/wind turbine systems integrated to utility grids as microgrid (MG) under pool/bilateral/hybrid electricity market using PSO[J]. Solar Energy, 2012, 86(1): 112–125.

[13] 岳昆, 王晓玲, 周傲英. Web 服务核心支撑技术:研究综述[J]. 软件学报, 2004(3): 428–442.

[14] VAN ENGELEN R A, GALLIVAN K A. The gSOAP toolkit for web services and peer-to-peer computing networks[C]//Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on. IEEE, 2002: 128–128.

[15] 张翠莲, 刘方爱, 王亚楠. 基于 MPI 的并行程序设计[J]. 计算机技术与发展, 2006(8): 72–74+76.

[16] MPICH User’s Guide[J]. Mathematics and Computer Science Division Argonne National Laboratory, 2015.

[17] DEMAINE E. First class communication in MPI[C]//MPI Developer’s Conference, 1996. Proceedings., Second. 1996: 189–194.

[18] 何仰赞 温增银. 电力系统分析（下册）[M]. 华中理工大学出版社, 2002.

[19] 周晓燕, 刘天琪, 沈浩东等. 含多种分布式电源的微电网经济调度研究[J]. 电工电能新技术, 2013, 32(1): 5–8.
[20] 粒子群优化[J]. 维基百科，自由的百科全书, 2015.
[21] 张景瑞. 梯级水电站和水火电站群优化调度的 PSO 算法[D]. 华中科技大学, 2012.
[22] ZHANG J, WANG J, YUE C. Small Population-Based Particle Swarm Optimization for Short-Term Hydrothermal Scheduling[J]. IEEE Transactions on Power Systems, 2012, 27(1): 142–152.
[23] 张武生, 薛巍, 李建江等. MPI 并行程序设计实例教程[M]. 清华大学出版社, 2009.
[24] TRAFF J L, GROPP W D, THAKUR R. Self-Consistent MPI Performance Guidelines[J]. IEEE Transactions on Parallel and Distributed Systems, 2010, 21(5): 698–709.
[25] LUSK E. MPI in 2002: has it been ten years already?[C]//2002 IEEE International Conference on Cluster Computing, 2002. Proceedings. 2002: 435-.
[26] 中间件[J]. 维基百科，自由的百科全书, 2016.
[27] DETTORI P, FRANK D, SEELAM S R 等. Blueprint for Business Middleware as a Managed Cloud Service[C]//2014 IEEE International Conference on Cloud Engineering (IC2E). 2014: 261–270.
[28] WOHLSTADTER E, TAI S, MIKALSEN T 等. A Service-oriented Middleware for Runtime Web Services Interoperability[C]//2006 IEEE International Conference on Web Services (ICWS'06). 2006: 393–400.
[29] 罗学刚. 在 C/C++环境下基于 gSoap 实现 Web Services 调用[J]. 电脑编程技巧与维护, 2011(11): 64–65+68.
[30] 宗起振, 王丹丹, 赵琴. 基于 GSOAP 的状态监测设备系统设计与实现[J]. 软件导刊, 2016(1): 73–75.
[31] Windows Forms[J]. 维基百科，自由的百科全书, 2015.
[32] RENTSCH T. Object Oriented Programming[J]. SIGPLAN Not., 1982, 17(9): 51–57.
[33] 面向对象程序设计[J]. 维基百科，自由的百科全书, 2016.
[34] 洪博文, 郭力, 王成山等. 微电网多目标动态优化调度模型与方法[J]. 电力自动化设备, 2013, 33(3): 100–107.
[35] 张文. 基于粒子群体优化算法的电力系统无功优化研究[D]. 山东大学, 2006.
[36] 陈蕊, 夏安邦, 马玉龙. 电力系统无功优化算法综述[J]. 东北电力技术, 2006, 27(6): 38–41.
[37] 徐鹤鸣. 多目标粒子群优化算法的研究[D]. 上海交通大学, 2013.
[38] 张治宏. 基于 MPI 的并行计算研究[D]. 中国地质大学（北京）, 2006.
[39] 郭羽成. MPI 高性能云计算平台关键技术研究[D]. 武汉理工大学, 2013.
[40] 李景超. 基于 CUDA 的并行粒子群优化算法研究及应用[D]. 广东工业大学, 2014.

附录

附录 A　IEEE 九节点参数

表格 A.1 IEEE 九节点母线参数

母线编号	类型	负荷有功	负荷无功	并联导纳	并联电纳	电压幅值	电压相角
0	PQ	0	0	0	0	1	0
1	PQ	90	30	0	0	1	0
2	PQ	0	0	0	0	1	0
3	PQ	100	35	0	0	1	0
4	PQ	0	0	0	0	1	0
5	PQ	125	50	0	0	1	0
6	PV	0	0	0	0	1	0
7	PV	0	0	0	0	1	0
8	Slack	0	0	0	0	1	0

表格 A.2 IEEE 九节点电机参数

母线	有功输出	无功输出	无功上限	无功下限	有功上限	有功下限
6	163	0	300	-300	300	10
7	85	0	300	-300	270	10
8	0	0	300	-300	250	10

表格 A.3 IEEE 九节点线路参数

上接母线	下接母线	电阻	电抗	电纳	电导	变比
8	0	0	0.0576	0	0	0
0	1	0.017	0.092	0.158	0	0
1	2	0.039	0.17	0.358	0	0
7	2	0	0.0586	0	0	0
2	3	0.0119	0.1008	0.209	0	0
3	4	0.0085	0.072	0.149	0	0
4	6	0	0.0625	0	0	0
4	5	0.032	0.161	0.306	0	0
5	0	0.01	0.085	0.176	0	0

附录 B　部分程序

```
void PSO::Initial_PSO(int _numProcs)
{
    //vMax 初始化
    for (int i = 0; i < amountBattery + amountPV + amountWindM + 1; i++)
    {
        for (int t = 0; t < timeDivide; t++)
        {
            if (i < amountBattery)
            {
                vMax[i][t] = (0.7 - (-0.7)) / 5; //蓄电池容量在 0.3-1 之间
            }
            else if (i < amountBattery + amountPV)
            {
                vMax[i][t] = (pv[i - amountBattery].pvPMax[t] / pn - pv[i -
amountBattery].pvPMin[t] / pn) / 5;//pv
            }
            else if (i < amountBattery + amountPV + amountWindM)
            {
                vMax[i][t] = (windm[i - amountBattery - amountPV].windmPMax[t] / pn -
windm[i - amountBattery - amountPV].windmPMin[t] / pn) / 5; //wind
            }
            else
            {
                vMax[i][t] = (pSlackMax - pSlackMin) / 5;
            }
        }
```

```
    }

    //初始化 x,v,pBest
    for (int m = 0; m < g; m++)
    {
        for (int i = 0; i < amountBattery + amountPV + amountWindM + 1; i++)
        {
            for (int j = 0; j < timeDivide; j++)
            {
                if (i < amountBattery)
                {
                    x[m][i][j] = -0.7 + Random01() * (0.7 - (-0.7));
                }
                else if (i < amountBattery + amountPV)
                {
                    x[m][i][j] = pv[i - amountBattery].pvPMin[j] / pn + Random01() * (pv[i -
amountBattery].pvPMax[j] / pn - pv[i - amountBattery].pvPMin[j] / pn);
                }
                else if (i < amountBattery + amountPV + amountWindM)
                {
                    x[m][i][j] = windm[i - amountBattery - amountPV].windmPMin[j] / pn +
Random01() * (windm[i - amountBattery - amountPV].windmPMax[j] / pn - windm[i -
amountBattery - amountPV].windmPMin[j] / pn);
                }
                else
                {
                    x[m][i][j] = pSlackMin + Random01() * (pSlackMax - pSlackMin);
                }
```

```
                tmp[i][j] = Random01() * vMax[i][j];
                v[m][i][j] = tmp[i][j];
                if (Random01() < 0.5)
                {
                    v[m][i][j] = -tmp[i][j];
                }
                pBest[m][i][j] = x[m][i][j]; //初始化个体最好
            }
        }
    }

    //初始化 fx, fpBest
    for (int m = 0; m < g; m++)
    {
        for (int i = 0; i < amountBattery + amountPV + amountWindM + 1; i++)        {
            for (int j = 0; j < timeDivide; j++)
            {
                adj_p1[i][j] = x[m][i][j];
            }
        }

        //将 x 值赋给 adjust_p;
        ad->Adjustment_p(adj_p1);
        adj_p9 = ad->pReturn;
        adj_vm = ad->adj_vm;
        adj_pBat = ad->pBat;
        adj_pPV = ad->pPV;
        adj_pWindM = ad->pWindM;
```

```
adj_SOC = ad->SOC;
best_SOC = adj_SOC;

for (int i = 0; i < amountBattery + amountPV + amountWindM + 1; i++)
{
    for (int j = 0; j < timeDivide; j++)
    {
        if (i < amountBattery)
        {
            x[m][i][j] = adj_pBat[i][j];
        }
        else if (i < amountBattery + amountPV)
        {
            x[m][i][j] = adj_pPV[i - amountBattery][j];                    }
        else if (i < amountBattery + amountPV + amountWindM)
        {
            x[m][i][j] = adj_pWindM[i - amountBattery - amountPV][j];
        }
        else
        {
            x[m][i][j] = adj_p9[numBus - 1][j];
        }
    }
}

//pv 维护成本
f1 = 0;
for (int i = 0; i < amountPV; i++)
```

```
{
    for (int j = 0; j < timeDivide; j++)
    {
        f1 = f1 + adj_pPV[i][j] * pn * 0.000708;
    }

}

//wind 维护成本
f2 = 0;
for (int i = 0; i < amountWindM; i++)
{
    for (int j = 0; j < timeDivide; j++)            {
        f2 = f2 + adj_pWindM[i][j] * pn * 0.005992;
    }
}

//与外部电网交换电量
f3 = 0;
for (int j = 0; j < timeDivide; j++)
{
    if (j <= 7)
    {
        if (adj_p9[numBus - 1][j] > 0)
        {
            f3 = f3 + adj_p9[numBus - 1][j] * pn * 0.17;
        }
        else
```

```
            {
                f3 = f3 + adj_p9[numBus - 1][j] * pn * 0.13;
            }
        }
        else if ((j >= 11 && j <= 15) || (j >= 19 && j <= 21))
        {
            if (adj_p9[numBus - 1][j] > 0)
            {
                f3 = f3 + adj_p9[numBus - 1][j] * pn * 0.83;
            }
            else
            {                       f3 = f3 + adj_p9[numBus - 1][j] * pn * 0.65;
            }
        }
        else
        {
            if (adj_p9[numBus - 1][j] > 0)
            {
                f3 = f3 + adj_p9[numBus - 1][j] * pn * 0.49;
            }
            else
            {
                f3 = f3 + adj_p9[numBus - 1][j] * pn * 0.38;
            }
        }

    }
```

```
            fx[m] = f1 + f2 + f3; //目标函数维护成本
            for (int i = 0; i < numBus; i++)
            {
                for (int j = 0; j < timeDivide; j++)
                {
                    if (adj_vm[i][j] < 0.9 || adj_vm[i][j] > 1.1)
                    {
                        fx[m] = fx[m] + 100;
                    }
                }
            }
            fpBest[m] = fx[m];
        }

        //初始化 fgBest
        fgBest = fpBest[0];

        //总集合初始化
        total_gBest = new double**[_numProcs];
        for (int i = 0; i < _numProcs; i++)
        {
            total_gBest[i] = new double*[(amountBattery + amountPV + amountWindM + 1)];
        }
        total_gBest[0][0] = new double[(amountBattery + amountPV + amountWindM + 1) *
timeDivide * total_g];
        memset(total_gBest[0][0], 0, sizeof(double) * _numProcs * (amountBattery + amountPV +
amountWindM + 1) * timeDivide);
        for (int i = 0; i < _numProcs - 1; i++)
```

```
    {
        total_gBest[i + 1][0] = total_gBest[i][0] + (amountBattery + amountPV + amountWindM +
1) * timeDivide;
    }
    for (int i = 0; i < _numProcs; i++)
    {
        for (int j = 0; j < (amountBattery + amountPV + amountWindM + 1) - 1; j++)
        {
            total_gBest[i][j + 1] = total_gBest[i][j] + timeDivide;
        }    }

    total_gBestMin = new double**[_numProcs];
    for (int i = 0; i < _numProcs; i++)
    {
        total_gBestMin[i] = new double*[(amountBattery + amountPV + amountWindM + 1)];
    }
    total_gBestMin[0][0] = new double[(amountBattery + amountPV + amountWindM + 1) *
timeDivide * total_g];
    memset(total_gBestMin[0][0], 0, sizeof(double) * _numProcs * (amountBattery + amountPV +
amountWindM + 1) * timeDivide);
    for (int i = 0; i < _numProcs - 1; i++)
    {
        total_gBestMin[i + 1][0] = total_gBestMin[i][0] + (amountBattery + amountPV +
amountWindM + 1) * timeDivide;
    }
    for (int i = 0; i < _numProcs; i++)
    {
        for (int j = 0; j < (amountBattery + amountPV + amountWindM + 1) - 1; j++)
```

```
        {
            total_gBestMin[i][j + 1] = total_gBestMin[i][j] + timeDivide;
        }
    }

    compare_fpBestT = new double[_numProcs];
    memset(compare_fpBestT, 0, sizeof(double) * _numProcs);

    total_fpBestT = new double*[_numProcs]; total_fpBestT[0] = new double[_numProcs * iter];
    memset(total_fpBestT[0], 0, sizeof(double) * _numProcs * iter);

    for (int i = 0; i < _numProcs - 1; i++)
    {
        total_fpBestT[i + 1] = total_fpBestT[i] + iter;
    }

    final_fpBestT = new double[iter];
    memset(final_fpBestT, 0, sizeof(double) * iter);

    scatterProc = 0;
}
```

```
void PSO::Cal_Adjustment(double *** _x, int _deltat)
{
    for (int m = 0; m < g; m++)
    {
        for (int i = 0; i < amountBattery + amountPV + amountWindM + 1; i++)
        {
            for (int j = 0; j < timeDivide; j++)
            {
                adj_p1[i][j] = _x[m][i][j];
            }
        }

        //将_x 值赋给 adjust_p
        ad->Adjustment_p(adj_p1);
        adj_p9 = ad->pReturn;
        adj_vm = ad->adj_vm;
        adj_pBat = ad->pBat;
        adj_pPV = ad->pPV;
        adj_pWindM = ad->pWindM;
        adj_SOC = ad->SOC;

        for (int i = 0; i < amountBattery + amountPV + amountWindM + 1; i++)
        {
            for (int j = 0; j < timeDivide; j++)
            {
                if (i < amountBattery)
                {
                    _x[m][i][j] = adj_pBat[i][j];
```

```
            }
            else if (i < amountBattery + amountPV)
            {
                _x[m][i][j] = adj_pPV[i - amountBattery][j];
            }
            else if (i < amountBattery + amountPV + amountWindM)
            {
                _x[m][i][j] = adj_pWindM[i - amountBattery - amountPV][j];
            }
            else
            {
                _x[m][i][j] = adj_p9[numBus - 1][j];
            }
        }
    }

    //pv 维护成本
    f1 = 0;
    for (int i = 0; i < amountPV; i++)
    {
        for (int j = 0; j < timeDivide; j++)
        {
            f1 = f1 + adj_pPV[i][j] * pn * 0.000708;
        }

    }

    //wind 维护成本
```

```
f2 = 0;
for (int i = 0; i < amountWindM; i++)
{
    for (int j = 0; j < timeDivide; j++)
    {
        f2 = f2 + adj_pWindM[i][j] * pn * 0.005992;
    }
}

//与外部电网交换电量
f3 = 0;
for (int j = 0; j < timeDivide; j++)
{
    if (j <= 7)
    {
        if (adj_p9[numBus - 1][j] > 0)
        {
            f3 = f3 + adj_p9[numBus - 1][j] * pn * 0.17;
        }
        else
        {
            f3 = f3 + adj_p9[numBus - 1][j] * pn * 0.13;
        }
    }
    else if ((j >= 11 && j <= 15) || (j >= 19 && j <= 21))
    {
        if (adj_p9[numBus - 1][j] > 0)
        {
```

```
                f3 = f3 + adj_p9[numBus - 1][j] * pn * 0.83;
            }
            else
            {
                f3 = f3 + adj_p9[numBus - 1][j] * pn * 0.65;
            }
        }
        else
        {
            if (adj_p9[numBus - 1][j] > 0)
            {
                f3 = f3 + adj_p9[numBus - 1][j] * pn * 0.49;
            }
            else
            {
                f3 = f3 + adj_p9[numBus - 1][j] * pn * 0.38;
            }
        }

    }

    fx[m] = f1 + f2 + f3;
    for (int i = 0; i < numBus; i++)
    {
        for (int j = 0; j < timeDivide; j++)
        {
            if (adj_vm[i][j] < 0.9 || adj_vm[i][j] > 1.1)
            {
```

```
                fx[m] = fx[m] + 100;
            }
        }
    }

    if (fx[m] < fpBest[m])
    {
        for (int i = 0; i < amountBattery + amountPV + amountWindM + 1; i++)
        {
            for (int j = 0; j < timeDivide; j++)
            {
                pBest[m][i][j] = _x[m][i][j]; //pBest 更新
            }
        }
        best_SOC = adj_SOC;

        fpBest[m] = fx[m]; //fpBest 更新
    }

    if (fpBest[m] < fgBest)
    {
        for (int i = 0; i < amountBattery + amountPV + amountWindM + 1; i++)
        {
            for (int j = 0; j < timeDivide; j++)
            {
                gBest[i][j] = pBest[m][i][j]; //gBest 更新
            }
        }
```

```
                fgBest = fpBest[m]; //fgBest 更新
            }

        }

        fpBestT[_deltat] = fgBest;
}


void PSO::Cal_PSO()
{
        //mpi 计算
        rank = 0; //进程号
        root = 0; //根进程
        total_g = 0; //总粒子数
        numProcs = 0; //总进程数
        int nameLength = 0; //机器名长度
        startTime = 0;
        endTime = 0;
        costTime = 0;
        char processorName[MPI_MAX_PROCESSOR_NAME];

        MPI_Comm_rank(MPI_COMM_WORLD, &rank); //得到当前进程号
        MPI_Comm_size(MPI_COMM_WORLD, &numProcs); //得到总的进程数
        MPI_Get_processor_name(processorName, &nameLength); //得到机器名
        srand((unsigned)time(NULL) + rank); //取当前机器时间为随机种子，加上进程号是为了各个进程取得随机数不同
```

```
    //把数据进行发散
    MPI_Reduce(&g, &total_g, 1, MPI_INT, MPI_SUM, root, MPI_COMM_WORLD); //计算总粒子数
    MPI_Bcast(&total_g, 1, MPI_INT, root, MPI_COMM_WORLD);

    numProcs = total_g / g;

    if (rank == root)
    {
        startTime = MPI_Wtime();
    }

    //粒子群初始化
    Initial_PSO(numProcs);

    //更新位置
    for (int deltat = 0; deltat < iter; deltat++)
    {
        w = 0.9 - 0.5 * deltat / iter;
        for (int m = 0; m < g; m++)
        {
            for (int i = 0; i < amountBattery + amountPV + amountWindM + 1; i++)
            {
                for (int j = 0; j < timeDivide; j++)
                {
                    v[m][i][j] = w * v[m][i][j] + c1 * Random01() * (pBest[m][i][j] - x[m][i][j]) + c2 * Random01() * (gBest[i][j] - x[m][i][j]);
                    if (v[m][i][j] > vMax[i][j])
                    {
                        v[m][i][j] = vMax[i][j];
```

```
                    }
                    if (v[m][i][j] < -vMax[i][j])
                    {
                        v[m][i][j] = -vMax[i][j];
                    }
                    x[m][i][j] = x[m][i][j] + v[m][i][j];
                }
            }
        }

        //计算各粒子适应值
        Cal_Adjustment(x, deltat);

        //集合操作
        if (deltat != 0 && (deltat % (iter / gathernum) == 0 || deltat == iter - 1))
        {
            if (rank == root)
            {
                std::cout << "Gather begin..." << std::endl;
            }

            MPI_Barrier(MPI_COMM_WORLD);
            MPI_Gather(gBest[0], (amountBattery + amountPV + amountWindM + 1) *
timeDivide, MPI_DOUBLE, total_gBest[0][0], (amountBattery + amountPV + amountWindM + 1)
* timeDivide, MPI_DOUBLE, root, MPI_COMM_WORLD);
            MPI_Gather(&fpBestT[deltat], 1, MPI_DOUBLE, compare_fpBestT, 1,
MPI_DOUBLE, root, MPI_COMM_WORLD);

            if (rank == root)
            {
```

```
            scatterProc = MinRank(compare_fpBestT, numProcs);

            for (int i = 0; i < numProcs; i++)
            {
                for (int j = 0; j < (amountBattery + amountPV + amountWindM + 1); j++)
                {
                    for (int m = 0; m < timeDivide; m++)
                    {
                        total_gBestMin[i][j][m] = total_gBest[scatterProc][j][m];
                    }
                }
            }

        }

        MPI_Scatter(total_gBestMin[0][0], (amountBattery + amountPV + amountWindM +
1) * timeDivide, MPI_DOUBLE, gBest[0], (amountBattery + amountPV + amountWindM + 1) *
timeDivide, MPI_DOUBLE, root, MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);

        if (rank == root)
        {
            std::cout << "Scatter end..." << std::endl;
        }
    }

    //集合各进程，计算最终收敛曲线
    if (deltat == iter - 1)
    {
```

```
            MPI_Gather(fpBestT, iter, MPI_DOUBLE, total_fpBestT[0], iter, MPI_DOUBLE,
root, MPI_COMM_WORLD);

            for (int i = 0; i < iter; i++)
            {
                final_fpBestT[i] = total_fpBestT[0][i];
                for (int j = 1; j < numProcs; j++)
                {
                    if (final_fpBestT[i] > total_fpBestT[j][i])
                    {
                        final_fpBestT[i] = total_fpBestT[j][i];
                    }
                }
            }

            if (rank == root)
            {
                std::cout << deltat << "\t";
            }

        }

        if (rank == root)
        {
            endTime = MPI_Wtime();
            costTime = endTime - startTime;
        }
}
```

```
PSO::PSO()
{
    this->Initial();
    this->Cal_PSO();

    if (rank == root)
    {
        this->WriteData(); //根进程才写数据
    }
}

void PSO::WriteData()
{
    //写入数据库
    MYSQL* conn;
    conn = mysql_init(NULL);

    if (!(mysql_real_connect(conn, host, user, pass, dbName, port, unixSocket, flag)))
    {
        fprintf(stderr, "\nError:%s[%d]\n", mysql_error(conn), mysql_errno(conn));
        exit(1);
    }

    //写入新数据
    char strWriHistory[300];
    snprintf(strWriHistory, sizeof(strWriHistory), "insert into history(algorithm_id, cost_time, iter,
gather_num, g_max, process_num, date_time, result) values(%d, %f, %d, %d, %d, %d,
FROM_UNIXTIME(%d), %f)", extern_algorithmID, costTime, iter, gathernum, g, numProcs,
time(NULL), final_fpBestT[iter - 1]);
```

```
    mysql_query(conn, strWriHistory);

    int current_history_id = mysql_insert_id(conn);

    for (int i = 0; i < iter; i++)
    {       char strWriResults[300];
        snprintf(strWriResults, sizeof(strWriResults),
            "insert into results(history_id, algorithm_id, n, f) values(%d, %d, %d, %f)",
current_history_id, extern_algorithmID, i, final_fpBestT[i]);
        mysql_query(conn, strWriResults);
    }

    for (int i = 0; i < timeDivide; i++)
    {
        char strWriDistribution[300];
        snprintf(strWriDistribution, sizeof(strWriDistribution),
            "insert into power_distribution(history_id, algorithm_id, output_time,
output_P_Battery, output_P_PV, output_P_WindM, output_P_Exchange, output_P_Total, SOC)
values(%d, %d, %d, %f, %f, %f, %f, %f, %f)",
            current_history_id, extern_algorithmID, i, gBest[0][i], gBest[1][i], gBest[2][i],
gBest[3][i], gBest[0][i] + gBest[1][i] + gBest[2][i] + gBest[3][i], best_SOC[0][i]);
        mysql_query(conn, strWriDistribution);
    }

    mysql_close(conn);
}
```