Hege Bruvik Kvandal

Toolbox for Specialized Power System Analysis

Master's thesis in Energy and Environmental Engineering
Supervisor: Olav Bjarte Fosso
June 2019

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electric Power Engineering

Hege Bruvik Kvandal

Toolbox for Specialized Power System Analysis

Master's thesis in Energy and Environmental Engineering
Supervisor: Olav Bjarte Fosso
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electric Power Engineering

Preface

This master thesis is the completion of my master studies in Energy and Environmental Engineering at the Department of Electric Power Engineering at the Norwegian University of Science and Technology (NTNU). It is a collaboration project with Statnett SF with the goal of modernizing a toolbox of applications for power system analysis. I chose this project because I find power system analysis to be a fascinating topic, especially when talking about solutions to challenges related to the adaption of the grid to renewable and green technologies. The preliminary specialization project introduced me to new programming languages and techniques for interfacing them that have been essential in the master thesis. The project has presented a steep learning curve, especially in the specialization project since it has been very programming intensive, but it has been equally rewarding since I now am much more comfortable with the topic. The course ELK 14 with Professor Olav Bjarte Fosso has been a useful resource to understand the load flow theory and algorithms. The work done on modernizing the toolbox before the start of the thesis, including translation of codes to C and Ctypes wrapper codes, by Leif Warland at Statnett has been very helpful. Finally, I would like to thank Professor Olav Bjarte Fosso for guidance in this thesis and my fellow students for support and motivation.

Trondheim, 11.06.2019

Hege Bruvik Kvandal

Abstract

Power system analysis is an integral part of the operation and planning of power systems. As the power system evolves with emerging green technologies and distributed generation, the tools that perform the analyses must adapt. Python is an object-oriented programming language with attributes that can be useful for the purpose of modernizing the tools. Currently, the tools for power system analysis considered in this thesis are written in Fortran, and the goal is to interface them with Python to take advantage of the functionalities the language provides. In this thesis, the DC optimal load flow analysis will be updated. The modernization is achieved by first translating the Fortran codes to C and then interfacing them with Python using Ctypes. In the translation process, the differences between the languages must be accounted for, and replacements for the Fortran optimization routines must be chosen. The linear programming solver Lpsolve is suggested as the optimization tool for the C codes. After the initial interfacing and testing, it can be concluded that even though the new version of the DC optimal load flow is not yet complete, and many tools remain to be interfaced, the information provided in this thesis can be used to continue the project and eventually lead to a toolbox designed for the future.

Sammendrag

Kraftsystemanalyse er en viktig del av både planlegging, drift og vedlikehold av kraftsystemet. Når kraftsystemet nå utvikles med nye, grønne teknologier og distribuert energiproduksjon, må verktøyene som utfører analysene tilpasses disse forandringene. Python er et objektorientert programmeringsspråk med egenskaper som vil være nyttige for å modernisere analyseverktøyene. Foreløpig er verktøyene for kraftsystemanalyse skrevet i Fortran, og målet er å lage et nytt grensesnitt i Python for å utnytte funksjonalitetene dette språket gir. I denne masteroppgaven vil analyseprogrammet for DC optimal lastflyt blir modernisert. Moderniseringen oppnås ved å først oversette Fortran-kodene til C, og deretter lage et grensesnitt til Python ved hjelp av Ctypes. I oversettelsesprosessen må det tas hensyn til forskjellene mellom de ulike programmeringsspråkene, og det må velges nye optimaliseringsrutiner som kan brukes i C. Lpsolve, et program som løser lineære programmeringsproblemer, er foreslått som et alternativ til Fortran-rutinene. Etter et grensesnitt er presentert og testet, kan det konkluderes med at selv om den nye versjonen av analyseprogrammet for DC optimal lastflyt ikke er fullført og mange analyseprogram gjenstår, kan teknikkene og informasjon presentert i denne oppgaven brukes til å fortsette moderniseringen og til slutt føre til et komplett analyseprogram designet for fremtiden.

Contents

List of Figures

List of Tables

Nomenclature

Δf^{kl}_{max}	Maximum allowed power flow change on line k-l
Δf^{kl}_{min}	Minimum allowed power flow change on line k-l
ΔP^{d}_{i}	Change in active power on bus i, down-regulation
ΔP^{u}_{i}	Change in active power on bus i, up-regulation
Ω	Ohm
$\mathbf{B}$	B matrix
$\mathbf{J}$	Jacobi matrix
$\mathbf{Y}_{bus}$	Y-bus matrix
θ	Voltage angle
$a_{ij,n}$	Power transfer distribution factor for line i-j with respect to bus n
B	Susceptance
b_{ii}	Diagonal B-matrix element
b_{ij}	Off-diagonal B-matrix element
c^{d}_{i}	Cost of decreased generation on bus i, down-regulation
c^{u}_{i}	Cost of increased generation on bus i, up-regulation
F	Objective function
G	Conductance
I	Current
P	Active power
Q	Reactive power
R	Resistance
S	Apparent power
V	Voltage

W	Watt
X	Reactance
Y	Admittance
Y_{ii}	Diagonal Y-bus matrix element
Y_{ij}	Off-diagonal Y-bus matrix element
Z	Impedance
AC	Alternating Current
API	Application Programming Interface
C	Programming language
DC	Direct Current
DCLF	Direct Current Load Flow
FDLF	Fast Decoupled Load Flow
GUI	Graphical User Interface
IDE	Integrated Development Environment
NRLF	Newton-Raphson Load Flow
PQ	Load bus
PTDF	Power Transfer Distribution Factor
PV	Voltage controlled bus

1 Introduction

1.1 Background

Power system analysis is essential in the operation and planning of power systems. Through a broad specter of analyses, information such as voltage magnitudes, voltage angles, and power flows can be obtained and used to monitor and improve existing systems and plan for new power grids. Further, it can be used to predict the behavior of the system in case of a planned or unexpected outage of a line, generator, or other unit connected to the grid.

Reliable analysis of power systems will only become more important in the future as the grid changes and evolves and the composition of elements changes from large and flexible central generators to distributed and inflexible renewable energy in order to reduce greenhouse gas emissions. The way consumers interact with the power grid is also changing with the introduction of electric vehicles, demand response, and prosumers, creating a two-way power flow.

Many algorithms for power system analysis exists, including, but not limited to regular load flow, DC optimal load flow, contingency analysis, and continuation power flow, all with their different uses, advantages, and disadvantages. Initially, these tools are written in Fortran, a programming language that is efficient, but not widely used anymore. To cope with the new challenges, however, these tools should be transferred and interfaced with Python, a new object-oriented environment with attributes such as interaction with code written in different languages and graphical user interfaces, GUIs, to scientific codes.

1.2 Specialization project

The focus in the specialization project leading up to this thesis was how to call functions written in the programming language C in Python. C and Fortran share many similarities, and there are many ways in which C can be interfaced with Python, therefore the transition between Fortran and Python will involve that some of the code is translated to C. Important aspects of the three relevant programming languages were discussed as well as some programming basics that are important when interfacing different languages.
Four different techniques for interfacing C with Python were explored, Ctypes, Cython, SWIG and Python-C-API, where the two latter were explained in detail and SWIG was determined to be the best choice for the task. SWIG, the Simplified Wrapper and Interface Generator, creates the wrapper code automatically, which saves time when there is much

code to be interfaced, but the wrapper code is nearly unreadable, which is a disadvantage [1].

1.3 The toolbox

The focus in this thesis is the collection of power system analysis tools written in Fortran, the toolbox for specialized power system analysis. It was created in the 90s by NTNU professor Olav Bjarte Fosso, to be used as a research tool to test methods. SINTEF has used it in some projects, but other than that it has not been used for much else, which was not the intent, either. However, the application system does work for large grids, which is why Statnett is interested in modernizing the toolbox so it can be used in their analyses. The advantage with such program systems compared to commercial programs is that they can run different cases without reading the case descriptions more than once. It makes it possible to read a case and then do customized studies based on scripts.

1.4 Problem formulation and scope

The process of modernizing the toolbox was already begun by Statnett at the start of this thesis, and Ctypes was chosen as the best technique for interfacing C with Python. Ctypes will be explained in depth and used for the interfacing. The work done by Statnett is made available through GitHub, an open-source software development platform. The goal for the thesis is to become acquainted with the application system and begin the transition from Fortran to Python and C. It is a large application system that will require more work than one master thesis, but the foundation and techniques for further work will be presented here.

In the rest of this thesis, the theory behind the interfacing process will be explained, as well as the theory behind the load flow. In order to limit the scope of the project, only the DC optimal load flow will be translated and interfaced, but the process will be explained in detail and should apply to the other load flows as well. Codes written in Fortran will be explained; therefore the theory section will include Fortran programming basics that are relevant for the DC optimal load flow in the application system. An essential part of the DC optimal load flow program is optimization. Consequently, the fundamentals of linear programming will be explained. The method section explains the DC optimal load flow application and explores the process of translating the Fortran codes to C. A new optimization algorithm will be presented along with examples on how to implement it in a load flow program. Finally, the choices made during the modernization process and the result will be discussed.

2 Theory

Sections 2.1 through 2.4 are based on the theory section in the specialization project "Toolbox for Specialized Power System Analysis" from 2018 by the author [2]. They are included to give an understanding of the theory that, though still important, is of lesser significance in the thesis than it was in the specialization project.

2.1 The programming languages

In this thesis, as in the specialization project, the three programming languages Fortran, C, and Python are central to the task of modernizing the power system analysis tools. The tools are originally written in Fortran, which is one of the oldest programming languages, created by IBM in the 1950s. Fortran is designed for fast numerical calculation and scientific computing, which makes it well suited for programs that require fast and efficient handling of large amounts of data, like power system analysis [3].

C is a general-purpose language that, like Fortran, can be used to write highly efficient code. It is one of the most popular programming languages, which makes it more advantageous in open-source projects compared to Fortran, which is lesser known today [4].

Python was released in 1991, making it the newest programming language of the three. It is also a general purpose language, but unlike C and Fortran, it is designed to be easy to use, with very readable syntax. Python has many qualities which makes it suitable for scientific computing, like fast prototyping and the possibility of interfacing different languages. The computationally demanding code can be written in C while the surrounding program and interface are written in Python to combine the advantages of both languages [5].

2.2 Data types

Data types are important to consider when it comes to interfacing different programming languages. It is especially significant when interfacing two languages, like C and Python, which uses different forms of *typing*. Typing determines the way variables are introduced in the code. In *static typed* languages, like C, all variables must be declared in the code before they are used, and once declared as a certain data type that variable cannot change. In *dynamically typed* languages, like Python, variables do not need any declaration before they are used and they can change data type throughout the code. This has to do with how Python assigns values to variables. It will not be further elaborated but is explained in more detail in [2].

In Fortran, there are five basic data types: integer, real, complex, logical, and character [6]. The primary data types in C are char, int, float, and double [7]. Int or integer is used for integers, real, float and double are used for floating point numbers, and char and character are for letters and strings. The Fortran type complex stores a number with a real and an imaginary part, and the logical data type has only the two values true and false. Python has the data type numbers, which includes int, long, float and complex, in addition to the types string, list, tuple, and dictionary. Python lists are mutable, and tuples are not. Dictionaries contain elements with a key and an associated value [8]. During the interfacing, arguments and variables are converted back and forth between the data types, and it is a crucial step in the process.

2.3 Array indexing and array order

Programs for power system analysis, and most other analyses that involve handling of data, includes arrays. In Fortran, array indexing starts at 1, meaning that the first element of an array can be accessed like "array[1]". In Python and C, the indexing starts at 0, and the first element of an array will therefore be "array[0]". The order in which the array elements are stored is also different between Fortran and C, the former stores the array elements column-wise, while the latter stores the elements row-wise. Python, like C, stores its array elements row-wise [9].

2.4 NumPy

NumPy is the most common module for array-handling in Python. It is an extension module with an n-dimensional array object and functions that can perform mathematical operations with such objects [10]. The following code snippet shows how to import NumPy, create two arrays, and add the arrays together.

```
import numpy as np

array1=np.array([1,2,3])
array2=np.array([-1, -2, -3])

array3=np.add(array1, array2)
```

2.5 Linked lists

A *linked list* is a list of contiguous elements, in which each element contains a variable and a pointer to the next variable in the list. An illustration of the structure of a linked list is given in Figure 2.1. The pointer to the first element in a linked list is called the "head," and the pointer of the last element is a null pointer to indicate the end of the list. The advantages of using a linked list compared to a regular list is that linked lists are dynamic, meaning elements can be added and removed anywhere in the list by changing to which element the pointer points [11].

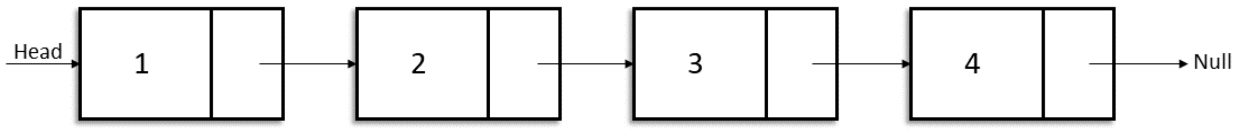


Figure 2.1: Structure of a linked list.

2.6 Fortran

One of the objectives in this thesis is the translation of the toolbox for power system analysis from Fortran to C. This requires a more thorough understanding of the language than what has been explained so far, in both the specialization project and in the thesis. This section will explain the concepts of loops, decisions, arrays, goto, common blocks, and basic syntax. More details can be found from the source at tutorialspoint.com [12].

2.6.1 Basic syntax

Fortran programs are built like an assemblage of modules, subroutines, functions, and the main program. In Fortran, a *function* takes a number of arguments and returns an output value. A *subroutine* takes a number of arguments on which it can perform one or more operations which may or may not modify them. It does not return any output [13]. Neither Python nor C makes this distinction between a function and a subroutine. The DC load flow program does not include any Fortran functions, only subroutines; therefore the expressions routine and function will be used interchangeably. The expression subroutine will be used about routines that are used inside another routine; therefore the same function can be described as both a routine and a subroutine due to the vertical hierarchy of the application system.

2.6.2 Decisions

A *decision* in programming tests one or more given conditions and conditional statements are executed based on the result [12]. The conditions are logical expressions which return either true or false. A decision in Fortran can look like the following example:

```
IF (expression) THEN
    conditional statement
ELSE
    conditional statement
```

The conditional statement can be anything. The logical expression must use one or more of the relational operators in Table 2.1.

Operator	Equivalent	Description
.EQ.	==	Equal to
.NE.	!=	Not equal to
.GT.	>	Greater than
.LT.	<	Less than
.GE.	>=	Greater than or equal to
.LE.	<=	Less than or equal to

Table 2.1: Relational operators in Fortran.

2.6.3 Loops

A loop is used when a block of code should be executed more than once. The number of times a loop should be executed can be determined by a predefined number, like in a for-loop, or it can rely on a logical expression like in a while-loop. In Fortran, the loops are called do-loops. The do-loop and the do-while-loop can look like the following example:

```
DO 100 I = 1, 20
    code
END DO

DO 200 WHILE (expression)
    code
END DO
```

The do-loop loop executes the block of code 20 times, and the do-while-loop continues until the logical expression is false. The number after "do" is a *label*. The number itself is not significant, but the loops must have different labels. Labels can be used to reference different parts of the code.

2.6.4 Goto

The goto-statement in Fortran allows for jumps back and forth in the code. If the statement "goto 100" is encountered, the program finds the label 100 and starts the execution from there. That can mean skipping code, or it can mean repeating code that has already been executed. It is often used after an if-statement. Using goto-statements is generally not considered good programming practice as it makes it hard to follow the flow in the code, and therefore makes the code harder to read.

2.6.5 Arrays

Arrays store variables of the same data type, and Fortran allows for arrays of up to seven dimensions. The most used are one- and two-dimensional arrays, like vectors and matrices [12].

2.6.6 Common blocks

A common block is a characteristic of Fortran 77 since this version of Fortran does not have *global variables*. Global variables are variables that are shared between more than one subroutine [14]. In order for several subroutines to use the same variables, they must be included in the input parameter list or included in a common block. Including the common variables in the input parameter list can be tedious and impractical if the number of variables is large, and therefore, common blocks were used. This is a source of errors since the variables can be changed by accident in other routines, and it can be hard to locate where the variables were changed.

2.7 GitHub

GitHub is a software development platform used to collaborate on open-source projects. A project can be public, meaning that anyone can contribute to the project, or it can be private, and then only users who are invited to collaborate on the project can contribute [15]. There are four central concepts in GitHub: *repositories*, *branches*, *commits*, and *Pull Requests*. A repository is where the project is organized. It contains all the files and information about the project, which can be almost anything, for example, images, videos, spreadsheets, and source code [16]. Branches are used to create different versions of a project that can be worked on simultaneously. The branches can be edited and experimented on, and then those changes can be committed to the master. A commit is made when changes to the project are saved. With each commit, there should be a commit message with an explanation of the proposed changes. Earlier commits can be read by the other project contributors so that everyone working on a project can follow the progress of the project. Pull Requests displays the difference between branches, and asks that the branches are merged. For example, if a branch contains proposed improvements to the master branch, which is the main branch in each repository, a Pull Request can be opened by the contributor who made the improvements. These changes can then be discussed by all contributors before the Pull Request is either approved and merged with the master branch, or denied [16]. GitHub is also a platform for version control, meaning that changes can be tracked and earlier versions of the project can be brought back if necessary.

2.8 SCons

SCons is a tool for building software. It uses Python scripts to build software for a range of languages, including C and Fortran [17]. In this thesis, it is used to create shared libraries needed in the interfacing process. A shared library is a collection of code in the form of a library file, ".dll" for Windows, that is loaded at runtime by any program that uses it. The installation and use of SCons will be explained in this section.

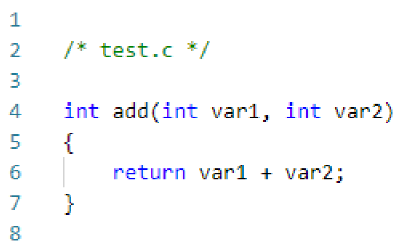
2.8.1 Installation

SCons can be downloaded from [17]. A prerequisite before installing SCons is to have Python installed since SCons is written in Python. After download, the following command will install SCons [18]:

```
python -m pip install scons
```

2.8.2 Building a shared library

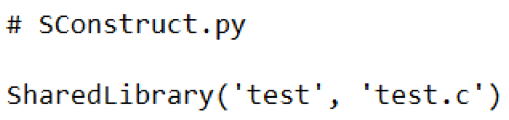
A small example will be used to illustrate how to build a shared library with SCons. The file "test.c" contains a function "add" which adds the two integers given to the function, see Figure 2.2.



```c

/* test.c */

int add(int var1, int var2)
{
    return var1 + var2;
}

```

Figure 2.2: File "test.c".

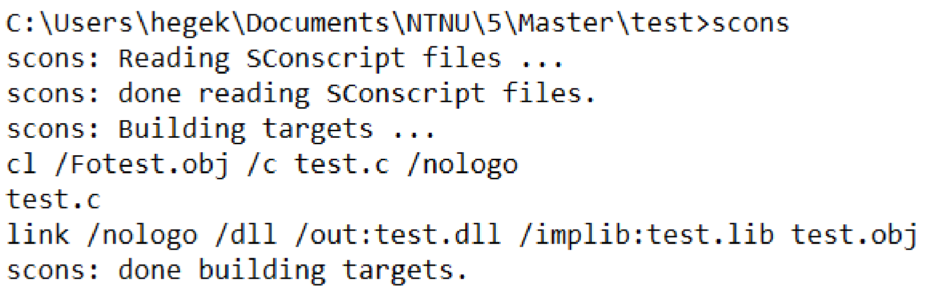
In order to build the library, an SConstruct file must be created. It is written in Python and contains information on how SCons should build the library. For the test file above the SConstruct file can be as in Figure 2.3.



```python
# SConstruct.py

SharedLibrary('test', 'test.c')
```

Figure 2.3: File "SConstruct.py".

The last step is to run the SConstruct file. The SConstruct file should be saved in the same folder as the source file(s), and then the command "scons" in the command prompt at that directory will create the shared library. The output from the example in Figure 2.3 will look like in Figure 2.4.



```
C:\Users\hegek\Documents\NTNU\5\Master\test>scons
scons: Reading SConscript files ...
scons: done reading SConscript files.
scons: Building targets ...
cl /Fotest.obj /c test.c /nologo
test.c
link /nologo /dll /out:test.dll /implib:test.lib test.obj
scons: done building targets.
```

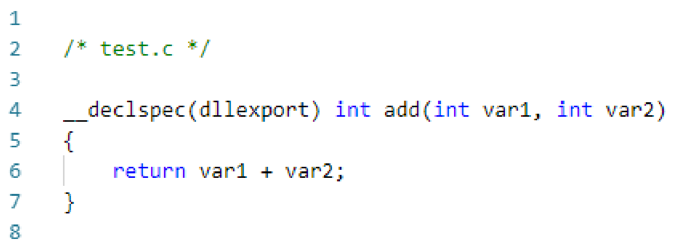
Figure 2.4: SCons output of library build.

It is important that the Python file is called "SConstruct.py", because that is the file that SCons looks for when building the library. The files are created in the same directory as the C source file and the SConstruct file. More information can be obtained from the Scons user guide [18].

2.9 Ctypes

Ctypes is a foreign function library for Python; a tool for interfacing C and Python. It uses C compatible data types to write wrapper code for shared libraries in pure Python [19]. The complete documentation for Ctypes can be read at [19], only the functions relevant for its use in the thesis are included here.

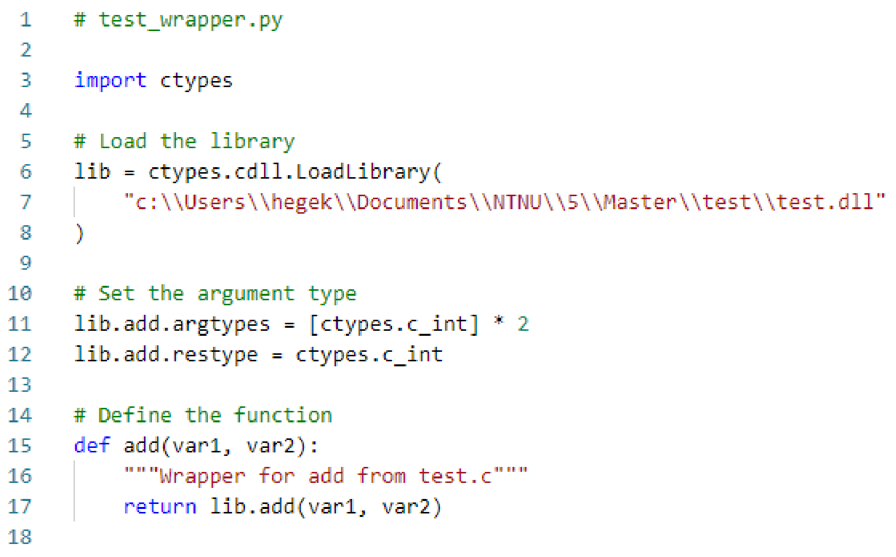
In order to use Ctypes the C code must be available in the form of a shared library. Such a library can be made with SCons, as explained in the previous section. If the source code to be interfaced includes functions that should be available from Python the keyword "__declspec(dllexport)" should be inserted at the start of each function declaration, as it is done below with the test example from Section 2.8.2 [20].



```c

/* test.c */

__declspec(dllexport) int add(int var1, int var2)
{
    return var1 + var2;
}

```

Figure 2.5: File "test.c", updated.

When the C source files are complete, use SCons (or another program) to build the shared library. Once the ".dll" file has been made, the wrapper can be written. The wrapper file for "test.c" and the function "add" is given in Figure 2.6 below.



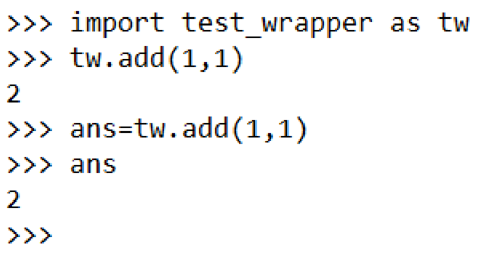
```python
# test_wrapper.py

import ctypes

# Load the library
lib = ctypes.cdll.LoadLibrary(
    "c:\\Users\\hegek\\Documents\\NTNU\\5\\Master\\test\\test.dll"
)

# Set the argument type
lib.add.argtypes = [ctypes.c_int] * 2
lib.add.restype = ctypes.c_int

# Define the function
def add(var1, var2):
    """Wrapper for add from test.c"""
    return lib.add(var1, var2)

```

Figure 2.6: File "test_wrapper.py".

First, Ctypes must be imported with the import statement. Next, the library must be loaded using the "LoadLibrary" function. *Cdll* is a Ctypes object used to load dynamic link libraries. It is used for libraries with functions using the standard cdecl calling convention. For functions using the stdcall calling convention, *windll* or *oledll* can be used. The MS standard C library uses the cdecl calling convention [19]. The name of the loaded library can be anything, "lib" was chosen for simplicity.

The next step is to specify the argument data types, which is done with the "argtypes" and "restypes" functions. Since both arguments in the add function are integers, int, the Ctypes data type c_int is used. All function arguments must be specified with the right type in the right order, which is the order in which they appear in the function parameter list. The return type is also an int, and the same applies.

The final step is to define the function(s). Use the Python return statement if the function returns a value, if not omit the return statement. The C function add is now available from Python in the same way as any other Python module and can be imported with the import statement. The name of the module is the same as the name of the wrapper file, in this example that is "test_wrapper". The example in Figure 2.7 illustrates the use of the function "add" after it has been interfaced with Python.



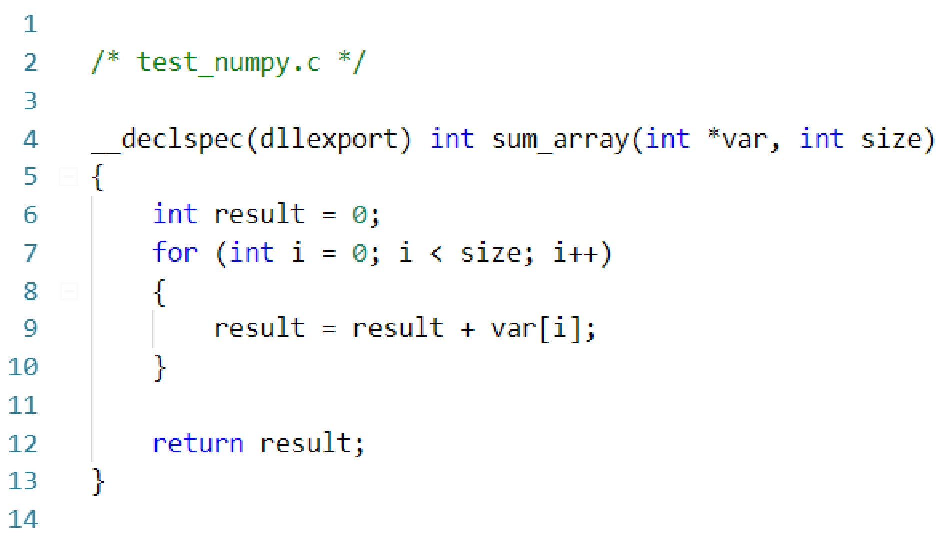
```
>>> import test_wrapper as tw
>>> tw.add(1,1)
2
>>> ans=tw.add(1,1)
>>> ans
2
>>>
```

Figure 2.7: Example of using the test_wrapper module.

It is important to add the path in which the module files are stored to PYTHONPATH. Doing this ensures that the module can be used from any Python interpreter or any program that uses Python. For the same reason, the whole library path should be specified in the "LoadLibrary" function.

2.9.1 NumPy support

NumPy has a foreign function interface for Ctypes called Ctypeslib, which can be used to specify array arguments in the wrapper code [21]. In addition, NumPy supports the use of C data types such as int, double, and bool. In Figure 2.8 the function "add" has been modified to a function "sum_array" that takes an array and the size of that array and returns the sum of all of the elements in the array.

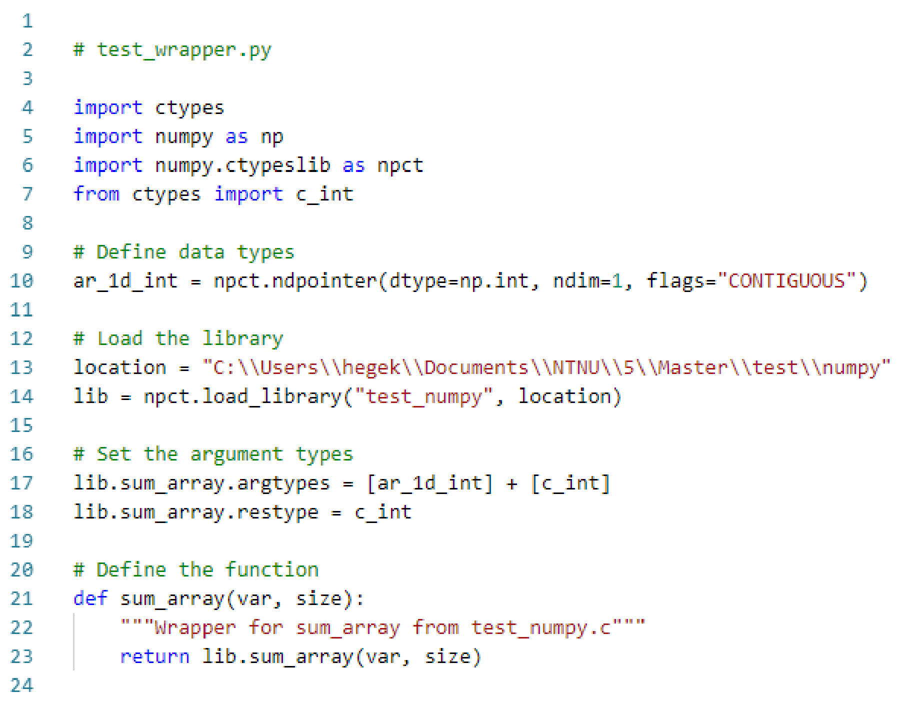


```c
/* test_numpy.c */

__declspec(dllexport) int sum_array(int *var, int size)
{
    int result = 0;
    for (int i = 0; i < size; i++)
    {
        result = result + var[i];
    }

    return result;
}
```

Figure 2.8: File "test_numpy.c".

Figure 2.9 shows the wrapper code for the "sum_array" function. Lines 4-7 imports the necessary modules to write the wrapper, which in addition to Ctypes includes NumPy and NumPy Ctypeslib. Ctypeslib includes a function for loading libraries, "load_library", which takes two strings as arguments: the name of the library and the location of the dll file.

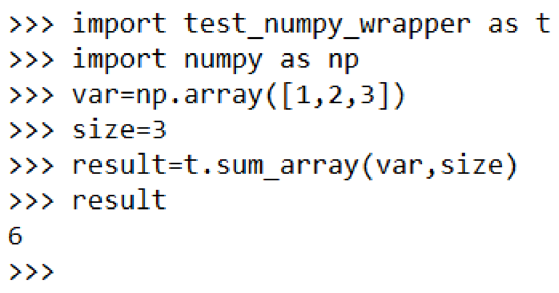
What is new in the wrapper code compared to Figure 2.6 is the definition of the pointer "ar_1d_int" of the "ndpointer" type from the Ctypeslib module. It is defined as an array of integers, thereof one dimension. The flag "CONTIGUOUS" specifies that the array elements are stored together in sequence [21]. Since the input parameters are a pointer to an array and an int, "argtypes" is set to ar_1d_int and c_int. The output argument is the sum of the elements in the array and is also an int. The "SConstruct.py" file must be changed with the updated filename.



```python

# test_wrapper.py

import ctypes
import numpy as np
import numpy.ctypeslib as npct
from ctypes import c_int

# Define data types
ar_1d_int = npct.ndpointer(dtype=np.int, ndim=1, flags="CONTIGUOUS")

# Load the library
location = "C:\\Users\\hegek\\Documents\\NTNU\\5\\Master\\test\\numpy"
lib = npct.load_library("test_numpy", location)

# Set the argument types
lib.sum_array.argtypes = [ar_1d_int] + [c_int]
lib.sum_array.restype = c_int

# Define the function
def sum_array(var, size):
    """Wrapper for sum_array from test_numpy.c"""
    return lib.sum_array(var, size)

```

Figure 2.9: File "test_numpy_wrapper.py".

Figure 2.10 shows how to import and use the function with a numpy array.



```
>>> import test_numpy_wrapper as t
>>> import numpy as np
>>> var=np.array([1,2,3])
>>> size=3
>>> result=t.sum_array(var,size)
>>> result
6
>>>
```

Figure 2.10: Example of how to use the function sum_array from the test_numpy_wrapper module.

2.10 Linear programming

Linear programming is a technique for optimizing a linear function based on linear criteria [22]. In an underdetermined system, which is a system with several variables, there exists not only one solution but rather a solution space. The goal of the optimization is to find the best solution within this solution space. Optimizing in this context means to maximize or minimize the function subject to the criteria. The function to be optimized is called the *objective function*, and the criteria are called the *constraints*. For the problem to be linear, the objective function and the constraints must be first order equations. The mathematical formulation of a standard linear optimization problem is given in Equations 1 and 2 [22].

The objective function F is on the form as given in Equation 1, where x represents a variable and a represents the variable coefficient.

$$\boldsymbol{min/max} \quad F = a_1x_1 + a_2x_2 + ... + a_nx_n \tag{1}$$

The constraints can be equalities or inequalities, and are generally on the form as given in Equation 2, where x represents a variable, b represents the variable coefficient and c is a constant.

$$\begin{aligned} b_{11}x_1 + b_{12}x_2 + ... + b_{1n}x_n &\leq c_1 \\ b_{21}x_1 + b_{22}x_2 + ... + b_{2n}x_n &\leq c_2 \\ &\vdots \\ b_{n1}x_1 + b_{n2}x_2 + ... + b_{nn}x_n &\leq c_n \end{aligned} \tag{2}$$

2.11 Load flow

There are four general requirements for a power system to operate successfully under balanced three-phase steady-state conditions:

1. Generation supplies the demand (load) plus losses.
2. Bus voltage magnitudes remain close to rated values.
3. Generators operate within specified real and reactive power limits.
4. Transmission lines and transformers are not overloaded. ([23], p. 309)

One of the most important uses of the load flow (or power flow) is to check that these requirements are not violated. It is done by calculating the bus voltage angles and magnitudes at each bus in the power system, and from this, the real and reactive power flow at each line and transformer can be calculated. A load flow problem is set up as a set of nonlinear algebraic equations, and there are many methods for solving these, some direct and some iterative, and they have different advantages and disadvantages. The following subsections will explain the concepts of the Newton-Raphson load flow (NRLF), the fast decoupled load flow (FDLF), and the direct current load flow (DCLF). The sections are based on the book Power System Analysis & Design 6th edition by Glover, Overbye and Sarma [23], and on lecture notes by Professor Olav Bjarte Fosso for the course ELK 14 at NTNU: [24], [25] and [26].

2.11.1 The load flow problem

The input for a load flow problem is the system topology, often a single line diagram, the line impedances, the load of each load bus and the generator rating for every connected generator. The output from this will be the actual output of each generator, the voltage and magnitude at each bus, and the power flow on each line. Of the four variables identified at each bus, two are known and used as input and two are unknown and to be computed by the load flow. These variables are voltage angle θ, voltage magnitude V, and net real P and reactive power Q to the bus. Which variables that are known at each bus is determined by what type of bus each bus is. Buses are categorized into three types: slack bus, also known as swing bus, PQ bus, also known as load bus, and PV bus, also known as a voltage controlled or generator bus. As a general rule, most buses in a power system are PQ buses. PV buses are buses where generators, shunts, or other equipment are connected. There can only be one slack bus in a power system because the slack bus is the reference bus, often with voltage angle 0 degrees and voltage magnitude 1.0 per unit, from which the other buses are calculated [23].

The input data for the power system transmission lines are the series impedance Z, the shut admittance Y, the maximum power rating in MVA, and the bus numbers for the two buses where the line is connected. This representation of the line comes from the π-equivalent model, illustrated in Figure 2.11 [23].

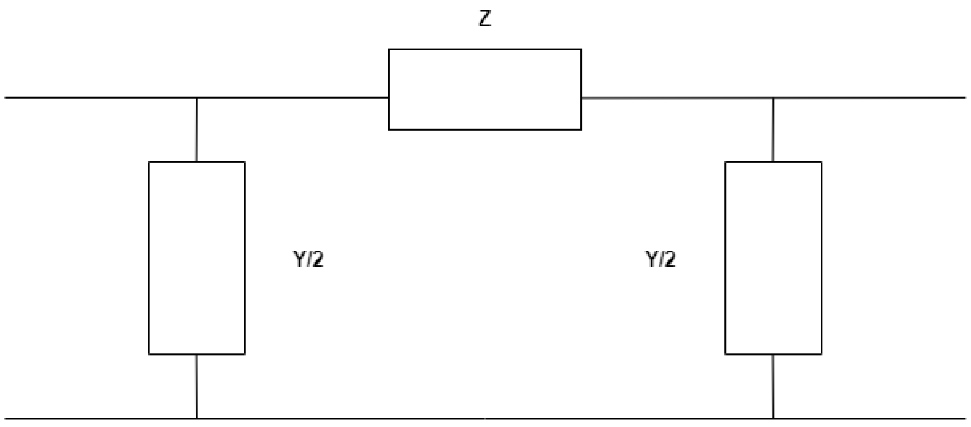


Figure 2.11: π-equivalent model of a transmission line.

For transformers the input data are the impedance, admittance, maximum power rating, and the buses between which the windings are connected when the transformers are represented by their equivalent circuits. This information is used to build the admittance matrix, also known as the Y_{bus} matrix. The diagonal elements in the Y_{bus} matrix, Y_{ii}, are the sum of all of the admittances connected to bus i. The off-diagonal elements, Y_{ij}, are the negative of the sum of admittances connected between bus i and j. For a three-bus system as in Figure 2.12, Y_{bus} is a 3x3 matrix as given in Equation 3 [23].

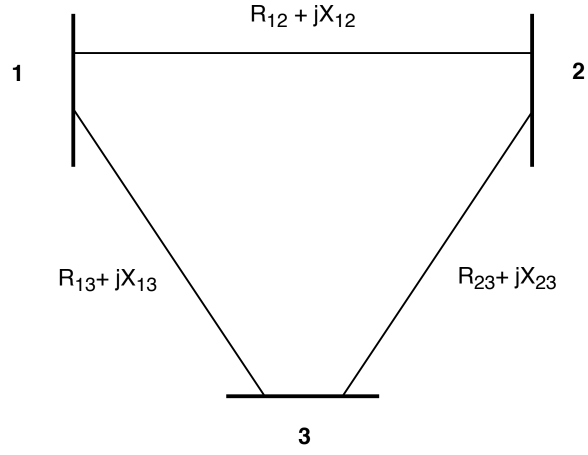


Figure 2.12: Single line diagram of a 3-bus system.

$$\mathbf{Y}_{bus} = \begin{bmatrix} Y_{11} & Y_{12} & Y_{13} \\ Y_{21} & Y_{22} & Y_{23} \\ Y_{31} & Y_{32} & Y_{33} \end{bmatrix} \tag{3}$$

In this case none of the elements in the matrix will be zero since all the buses are connected to each other. However, in a large, realistic system, most of the elements will be zero and the matrix will as a result be a sparse matrix.

The load flow equations are formed using Ohm's law with Y_{bus}. Ohm's law is given by Equation 4, where I is the current, V is the voltage, and Z is the impedance.

$$\mathbf{I} = \frac{\mathbf{V}}{\mathbf{Z}} \tag{4}$$

Combined with Equation 5, which gives the relationship between impedance and admittance, it results in Equation 6, which is the nodal equation for a power system [23].

$$\mathbf{Y} = \frac{1}{\mathbf{Z}} \tag{5}$$

$$\mathbf{I} = \mathbf{Y}_{bus}\mathbf{V} \tag{6}$$

The complex power S consists of the real and reactive power, P and Q respectively, and for a bus i it can be written as in Equation 7 [23]. V_i is the complex voltage at bus i, and I_i^* is the conjugate of the complex current at bus i.

$$\mathbf{S}_i = P_i + jQ_i = \mathbf{V}_i\mathbf{I}_i^* \tag{7}$$

Writing the complex voltage as in Equation 8 and the Y_{bus} elements as in Equation 9, the load flow equations can be written as in Equations 10 and 11, which are the equations the NRLF and FDLF solutions are based on [24]. Here G_{ij} is the conductance and B_{ij} is the susceptance between buses i and j, and i and j range from 1 to the number of buses in the system, N.

$$V_i = V_i e^{j\theta_i} \tag{8}$$

$$Y_{ij} = Y_{ij}e^{j\theta_i} = G_{ij} + jB_{ij} \tag{9}$$

$$P_i = V_i \sum_{j=1}^{N} V_j[G_{ij}cos(\theta_i - \theta_j) + B_{ij}sin(\theta_i - \theta_j)] \tag{10}$$

$$Q_i = V_i \sum_{j=1}^{N} V_j[G_{ij}sin(\theta_i - \theta_j) - B_{ij}cos(\theta_i - \theta_j)] \tag{11}$$

2.11.2 Newton-Raphson load flow

In the NRLF the load flow problem is solved as a set of equations on the form $\mathbf{y} = \mathbf{f(x)}$, where $\mathbf{y}$ is a vector with the real and reactive power, $\mathbf{x}$ is a vector with the variables V_i and θ_i, and $\mathbf{f(x)}$ is the real and reactive power flow equations as given in Equations 10 and 11. It is solved with the Newton-Raphson method, see Equation 12, which is an iterative solution method based on a Taylor series expansion of $\mathbf{f(x)}$. $\mathbf{J}$ is the Jacobi matrix, a matrix of the partial derivatives of $\mathbf{f(x)}$ [23].

$$\mathbf{x}(i+1) = \mathbf{x}(i) + \mathbf{J}^{-1}(i)[\mathbf{y} - \mathbf{f}(\mathbf{x}(i))] \tag{12}$$

Equation 13 shows the Newton-Raphson method translated into a Newton-Raphson load flow problem. Here the elements in the first matrix are the partial derivatives of the load flow equations, $\Delta\boldsymbol{\theta}$ and $\Delta\mathbf{V}$ are the changes in voltage angle magnitudes, and $\Delta\mathbf{P}(x)$ and $\Delta\mathbf{Q}(x)$ are the changes in the injected power.

$$-\begin{bmatrix} \frac{\delta\mathbf{P}}{\delta\boldsymbol{\theta}} & \frac{\delta\mathbf{P}}{\delta\mathbf{V}} \\ \frac{\delta\mathbf{Q}}{\delta\boldsymbol{\theta}} & \frac{\delta\mathbf{Q}}{\delta\mathbf{V}} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{\theta} \\ \Delta\mathbf{V} \end{bmatrix} = \begin{bmatrix} \Delta\mathbf{P}(\mathbf{x}) \\ \Delta\mathbf{Q}(\mathbf{x}) \end{bmatrix} = \mathbf{f(x)} \tag{13}$$

2.11.3 Fast decoupled load flow

The FDLF decouples the real and reactive power equations. The off-diagonal matrices, $\frac{\delta \mathbf{P}}{\delta \mathbf{V}}$ and $\frac{\delta \mathbf{Q}}{\delta \boldsymbol{\theta}}$, are usually neglectable and can therefore be set to zero. This is because the resistance is generally smaller than the reactance, meaning that $|G_{ij}| << |B_{ij}|$, and the angle is generally small, meaning that $sin(\theta_{ij}) \approx 0$. Equation 13 will then be changed to Equation 14 [24].

$$-\begin{bmatrix} \frac{\delta \mathbf{P}}{\delta \boldsymbol{\theta}} & 0 \\ 0 & \frac{\delta \mathbf{Q}}{\delta \mathbf{V}} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{\theta} \\ \Delta \mathbf{V} \end{bmatrix} = \begin{bmatrix} \Delta \mathbf{P}(\mathbf{x}) \\ \Delta \mathbf{Q}(\mathbf{x}) \end{bmatrix} = \mathbf{f}(\mathbf{x}) \tag{14}$$

The problem is now decoupled and can be written as two independent equations, see Equations 15 and 16.

$$\Delta \boldsymbol{\theta} = -\left[\frac{\delta \mathbf{P}}{\delta \boldsymbol{\theta}}\right]^{-1} \Delta \mathbf{P}(\mathbf{x}) \tag{15}$$

$$\Delta \mathbf{V} = -\left[\frac{\delta \mathbf{Q}}{\delta \mathbf{V}}\right]^{-1} \Delta \mathbf{Q}(\mathbf{x}) \tag{16}$$

These approximations are called Jacobian approximations, and they can be extended in the FDLF so that the Jacobian becomes independent of the voltage magnitudes and angles, and will therefore only need to be built once. The additional approximations are that $|G_{ij}| = 0$, $|V_i| = 1$, $sin(\theta_{ij}) = 0$ and $cos(\theta_{ij}) = 1$ [24]. The NRLF and FDLF should have the same solution to the load flow problem since they use the same power flow equations, Equations 10 and 11. The problem is now reduced to Equations 17 and 18, and can be solved iteratively until convergence. In some situations it can also be used to give an approximate, but fast solution to a load flow problem with only one iteration [23].

$$\Delta \boldsymbol{\theta} = \mathbf{B}^{-1} \frac{\Delta \mathbf{P}(\mathbf{x})}{\boldsymbol{\theta}} \tag{17}$$

$$\Delta \mathbf{V} = \mathbf{B}^{-1} \frac{\Delta \mathbf{Q}(\mathbf{x})}{\mathbf{V}} \tag{18}$$

Here $\mathbf{B}$ is the imaginary part of $\mathbf{Y}_{bus}$, slack bus not included.

2.11.4 DC load flow

The DC load flow builds on the fast decoupled power flow and includes further simplifications by removing the reactive power equations completely. This means that only the active power will be modeled. The voltage magnitudes are assumed to be 1.0 per unit, and the resulting active power flow equation is given in Equation 19 [26]. P_{ij} is the power flow on the line between buses i and j, θ_i and θ_j are the voltage angles at buses i and j respectively, and X_{ij} is the reactance between the buses.

$$P_{ij} = \frac{\theta_i - \theta_j}{X_{ij}} \tag{19}$$

Equation 19 describes is a linear problem that can be written as in Equation 20 for the whole system. It is solved with only one "iteration" and therefore gives a fast, though only approximate solution. Once the voltage angles are calculated, Equation 19 can be used to calculate the line flows [26].

$$-\mathbf{B}\boldsymbol{\theta} = \mathbf{P} \tag{20}$$

The assumptions made in the DC load flow will give a more accurate result when the voltage profile is flat, and the X/R-ratio is high. With increasing voltage, the significance of the resistance decreases, therefore the DC flow can often be applied to high voltage networks [27]. The assumptions make the DC load flow less accurate than the AC power flow, but useful when a fast solution is required.

2.11.5 Power transfer distribution factors

Power transfer distribution factors, PTDFs, indicate the change in power flow on a line due to changes in the power injections at the nodes. The PTDFs can be expressed as in Equation 21 [26], where P_{ij} is the active power flow between buses i and j, X_{ij} is the reactance between buses i and j, P_n is the active power at bus n, and $a_{ij,n}$ is the distribution factor for line i-j for a changed injection at bus n.

$$\begin{aligned} P_{ij} &= \frac{(x_{i1} - x_{j1})}{X_{ij}} P_1 + \frac{(x_{i2} - x_{j2})}{X_{ij}} P_2 + ... + \frac{(x_{in} - x_{jn})}{X_{ij}} P_n \\ P_{ij} &= a_{ij,1} P_1 + a_{ij,2} P_2 + ... + a_{ij,n} P_n \end{aligned} \tag{21}$$

Calculating the inverse of a large matrix is time-consuming, but there are faster options when only some of the inverse elements of the matrix are required. In a system $Ax + b$, a row of the inverse matrix A^{-1} can be found by replacing b with a vector of zeroes and a 1 in the desired row. The difference between two rows can be found in the same way, by replacing b with a vector of zeroes and a 1 and -1 in the specified rows. A fast calculation can then be performed by a forward and a backward substitution [26]. The equation system is presented in Equation 22. Here b represents the elements of the $\mathbf{B}$ matrix and

a the distribution factors.

$$\begin{bmatrix} b_{11} & b_{12} & b_{13} & ... & b_{1n} \\ b_{21} & b_{22} & & & \\ b_{31} & & \ddots & & \\ \vdots & & & & \\ b_{n1} & & & & b_{nn} \end{bmatrix} \begin{bmatrix} a_1 \\ a_i \\ \vdots \\ a_j \\ a_n \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{1}{X_{ij}} \\ \vdots \\ \frac{-1}{X_{ij}} \\ 0 \end{bmatrix} \tag{22}$$

Power transfer distribution factors can be used in load flow optimization problems.

2.11.6 Load flow optimization

DC optimal power flow is based on the DC solution to the power flow problem, which is the active power flow, and the economic dispatch of the system. The economic dispatch for a power system is when the generated power matches the required load, the operating costs are minimized, and the generated power and power flow are subject to the operational and transmission constraints. [23].

An objective function is formed to find the solution, see Equation 1. In the DC optimal load flow the objective is to minimize the cost. The objective function is subject to one or more constraints, which are the maximum transmission line power flows and the sum of regulation. The whole problem formulation is, therefore: how to find the combination of change injections to remove overloads in the system at minimum cost.

The mathematical formulation of the optimization problem is given in Equations 23 to 25. Equation 23 shows the objective function, F, that should be minimized. c_i^u is the cost for increased generation, and c_i^d is the cost for decreased generation. ΔP_i^u and ΔP_i^d are the needed changes in generation for the load flow solution to satisfy the constraints, up- and down-regulation respectively [26].

$$Min \quad F = \sum_i c_i^u \Delta P_i^u + \sum_i c_i^d \Delta P_i^d \tag{23}$$

Equation 24 is the balance equation, which says that the total sum of regulation must be zero. ΔP_i^u and ΔP_i^d represents the up- and down-regulation, respectively.

$$\sum_i \Delta P_i^u - \sum_i \Delta P_i^d = 0 \tag{24}$$

Equation 25 represents the transmission constraint(s) for the line(s) between buses k and l. a_i^{kl} are the PTDFs for the line. Δf_{min}^{kl} and Δf_{max}^{kl} are the maximum and minimum allowed power flow change on the line, based on the current base flow. There are two ways of deciding Δf^{kl}_{min} and Δf^{kl}_{max}: with zero as the reference and with the current flow as the reference. The latter is a faster method. Each line with a power flow that exceeds the maximum allowed power flow on that line will add a constraint to the optimization problem.

$$\Delta f^{kl}_{min} \leq \sum_i a^{kl}_i (\Delta P^u_i - \Delta P^d_i) \leq f^{kl}_{max} \tag{25}$$

After the initial DC load flow and optimization, the load flow is solved again, the angles are updated, and the line flows calculated, and if there are any more violations, new constraints are added to the problem. This is repeated until there are no more violations. The load flow optimization is, therefore, an iterative problem [26].

The economic dispatch represents a balanced system with minimal operating costs. When possible contingencies and other operational limitations are taken into account by adding preventative constraints to the problem, it is called *security constrained economic dispatch*. In other words, the cost must be minimized according to initial constraints and post contingencies while maintaining a balanced system [26].

The load flow optimization problem described in this section is a linear problem, as described in Section 2.10, that can be solved using a linear programming solver.

2.11.7 Per-unit system

The per-unit system is often used on quantities like power, current, voltage, and impedance. It defines these quantities as dimensionless ratios based on a base value, see Equation 26.

$$\textit{Per-unit value}[p.u.] = \frac{\textit{Actual value}[V, A, W, \Omega]}{\textit{Base value}[V, A, W, \Omega]} \tag{26}$$

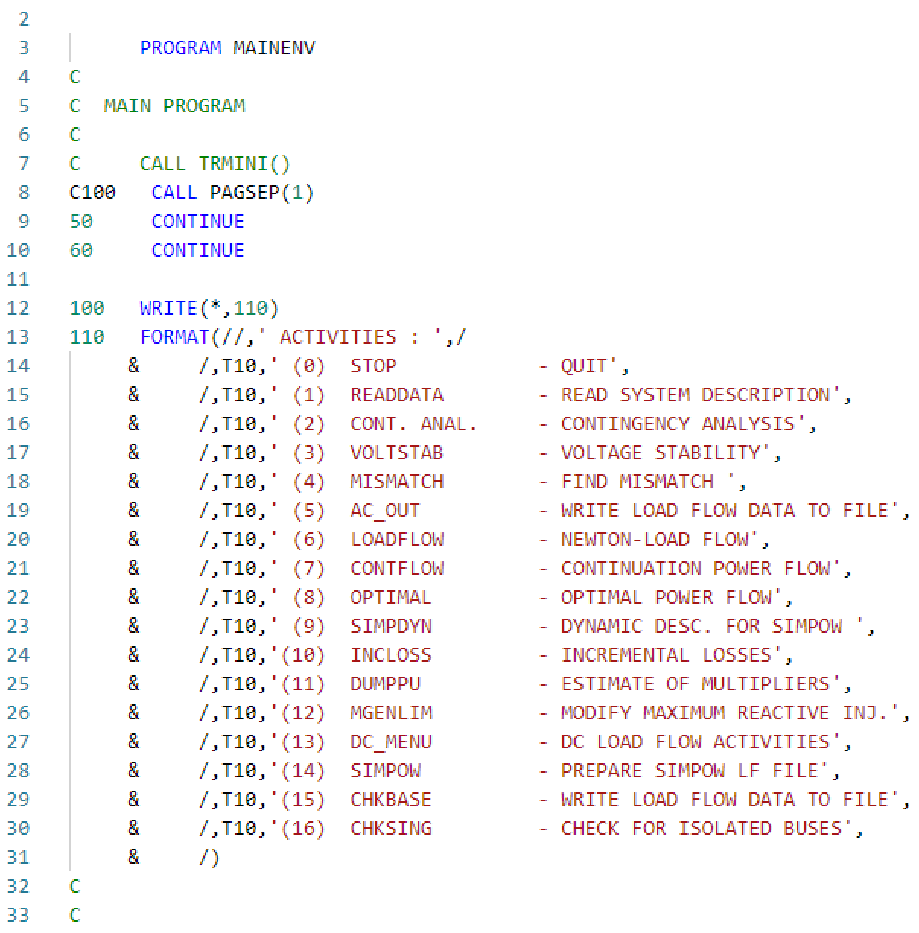
There are two main advantages of using per-unit values. Firstly, similar electrical equipment often have parameters within the same range of values when using per-unit values with the equipment ratings as the base values. Secondly, the transformer ratio can be eliminated when the base values are chosen correctly, which makes calculations easier [23].

3 Translation from Fortran to C

The translation process from Fortran to C is explained here, together with excerpts from the source codes. The complete files, both Fortran and C, are all given in the Appendix.

3.1 The Fortran program

This section will explain how the Fortran program system is built up. The main application is called mainenv, and it is a menu of the different functions and power flow tools the program offers, see Figure 3.1. The tools include, but are not limited to, contingency analysis, Newton-Raphson load flow, continuation power flow, optimal power flow, and DC load flow activities. The user chooses which analysis to use by entering the number corresponding to the desired tool, and then the program calls that routine.



```

      PROGRAM MAINENV
C
C  MAIN PROGRAM
C
C     CALL TRMINI()
C100   CALL PAGSEP(1)
50     CONTINUE
60     CONTINUE

100   WRITE(*,110)
110   FORMAT(//,' ACTIVITIES : ',/
     &     /,T10,' (0)  STOP         - QUIT',
     &     /,T10,' (1)  READDATA     - READ SYSTEM DESCRIPTION',
     &     /,T10,' (2)  CONT. ANAL.  - CONTINGENCY ANALYSIS',
     &     /,T10,' (3)  VOLTSTAB     - VOLTAGE STABILITY',
     &     /,T10,' (4)  MISMATCH     - FIND MISMATCH ',
     &     /,T10,' (5)  AC_OUT       - WRITE LOAD FLOW DATA TO FILE',
     &     /,T10,' (6)  LOADFLOW     - NEWTON-LOAD FLOW',
     &     /,T10,' (7)  CONTFLOW     - CONTINUATION POWER FLOW',
     &     /,T10,' (8)  OPTIMAL      - OPTIMAL POWER FLOW',
     &     /,T10,' (9)  SIMPDYN      - DYNAMIC DESC. FOR SIMPOW ',
     &     /,T10,'(10)  INCLOSS      - INCREMENTAL LOSSES',
     &     /,T10,'(11)  DUMPPU       - ESTIMATE OF MULTIPLIERS',
     &     /,T10,'(12)  MGENLIM      - MODIFY MAXIMUM REACTIVE INJ.',
     &     /,T10,'(13)  DC_MENU      - DC LOAD FLOW ACTIVITIES',
     &     /,T10,'(14)  SIMPOW       - PREPARE SIMPOW LF FILE',
     &     /,T10,'(15)  CHKBASE      - WRITE LOAD FLOW DATA TO FILE',
     &     /,T10,'(16)  CHKSING      - CHECK FOR ISOLATED BUSES',
     &     /)
C
C
```

Figure 3.1: Part of file "main_env.f" showing the menu.

The simplified structure of mainenv is given in Figure 3.2. Only some of the subroutines in dc_menu are included since they are in focus in the thesis, and they are highlighted in the figure. Subroutines that are not further discussed are omitted from the chart to increase readability. The other load flow tools under mainenv will also have structures similar to dc_menu.

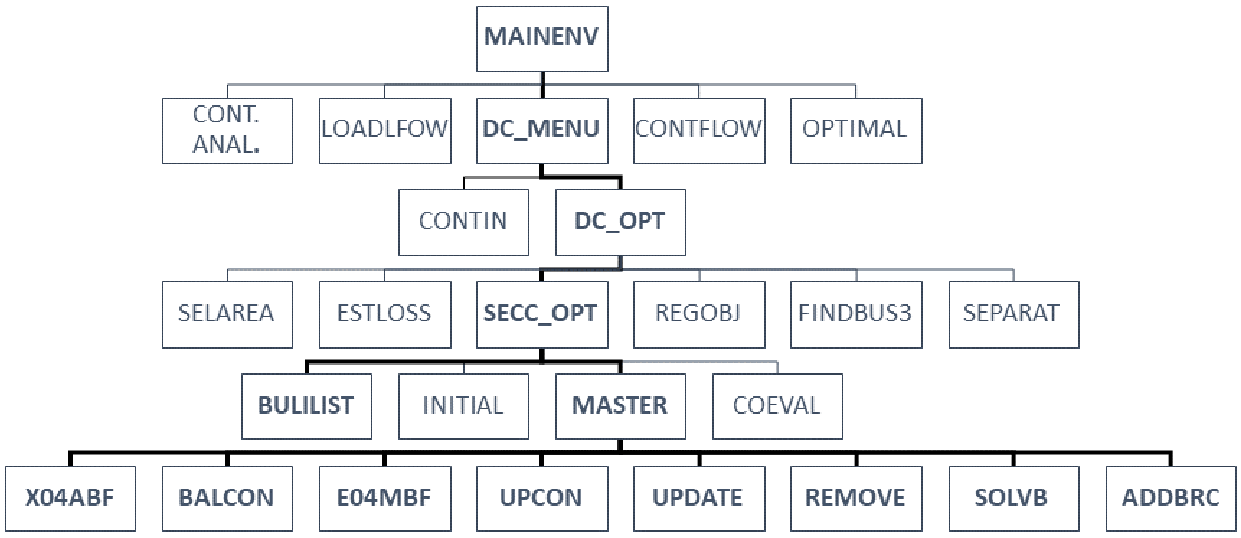


Figure 3.2: Simplified structure of mainenv with focus on dc_menu.

Dc_menu has two power flow options: contingency analysis, contin, and security-constrained optimal dispatch, secc_opt, in addition to routines for branch sensitivities and file handling not included in the figure. The routines that are focused on are secc_opt and master because these can be viewed as independent tools for analysis that can stand apart from the main program. Dc_opt sets up the problem for analysis and defines the parameters. Secc_opt performs the security constrained optimal dispatch using the subroutines bulilist, initial, master and coeval, all with a set of other subroutines. Bulilist creates linked lists of the lists given as arguments. Initial initializes the system description and factorizes the inverse of the B-matrix, $\mathbf{B}^{-1}$. Master sets up the optimization problem with cost minimization and sets up restrictions and constraints using distribution factors (PTDFs). Lastly, coeval performs an analysis of single line outages. The subroutines in the master routine perform tasks such as setting up the balance constraint and updating and removing restrictions. They will be further explained.

The idea for this toolbox is for Python to be the administrative language in the program. This means that when the application is finished, the only language the user will interact with is Python. The main objective is to rewrite the codes higher up in the program "hierarchy" in Figure 3.2 in Python, and translate the codes lower down to C and interface them with the new Python codes. The reason for this is that the "lower" codes perform the numerical calculations and data handling, actions which require speed and efficiency, a trait associated with Fortran and C. One of the advantages with Python, as mentioned, is that it is easy to extend it with modules written in other programming languages. In addition to this, Python has extensive support libraries and is easy to read and use [5], which makes it a suitable choice for administrative language. Python is also a free and open-source language available to everyone, which makes it ideal for use in open-source projects, which this project might be later. Mainenv and dc_menu are both "menus" not performing any calculations, but calling other functions that do and they should, therefore, be rewritten to Python. Dc_opt will also be rewritten to Python because it sets up the problem and defines the parameters. Those three routines transferred to Python will make up the framework for the DC optimization tool. The routines that perform the analyses should have high performance and will be translated from Fortran to C.

3.2 Visual Studio Code

Visual Studio Code is a useful tool when working with codes in more than one programming language. For this project, Visual Studio Code was downloaded from [28], and packages for Python, C and Fortran were installed. The advantages of using Visual Studio Code rather than a simple text editor include IntelliSense and continuous debugging of the code [28]. IntelliSense provides syntax highlighting and smart autocompletion of code based on variable types, function definitions and imported modules, which makes the programming faster and easier, and the continuous debugging and correction of code prevent many run-time errors. However, most importantly, it allows for side by side writing and editing of code in different programming languages.

Visual Studio Code can also automatically format code. Before the start of this thesis, Statnett had already begun the work of modernizing and translating the toolbox and used the standard formatting "black" [29]. Black can be installed and applied automatically every time the code is saved. Using standard formatting on all code makes it easier to read and work with, especially when working on open-source projects where there can be several people working with the same code.

3.3 Translating code from Fortran to C

The process of modernizing the toolbox is a process that will require many rounds of editing before it is complete. There are many things to consider when transferring a program from one language to another, from the overall function of the program down to the programming basics explained in Section 2. An important question is how much to change the original code when translating it from Fortran to C. The main idea is to transfer an already well-functioning program system to a new environment, and therefore the first round of translation and editing will be as close to the original as possible. The new program system should have a standard set of rules when it comes to naming functions, parameters, and variables like the original. A significant change in the program structure or adjustment of parameters in one part of the program will make similar changes necessary in other parts of the program.

First, the original version of the routines will be explained, followed by the new C version, which will highlight the modifications. Only when the two versions differ significantly from each other will the code be included in the section. The complete set of the codes is included in the Appendix.

3.4 Master

The description of the master routine given in "master.f" is that the routine solves the master problem in the security constrained dispatch problem. Briefly summarized it first sets up the balance restriction and optimizes the system accordingly. Next, the constraints are checked and updated before the DC load flow is run. If new constraints are necessary, these are added, and the process is repeated until an acceptable solution for the base condition has been found. The routine takes 29 input parameters as well as parameters from three different common blocks. The input parameters, along with a short description, is given in Table 3.1.

Table 3.1: Input parameters for the master routine. I = integer, R = real, C = character. () preceding the data type indicates an array of that type.

Name	**Type**	**Description**
Input data		
nbuses	I	Number of buses
ngens	I	Number of generator buses
igbus	I()	Pointers to generators
preffrst	I()	Pointer to first segment in pref. func.
prefnext	I()	Pointer to next segment
prefcost	R()	Cost vector for the segments
prefmw	R()	Maximum capacity for the segments

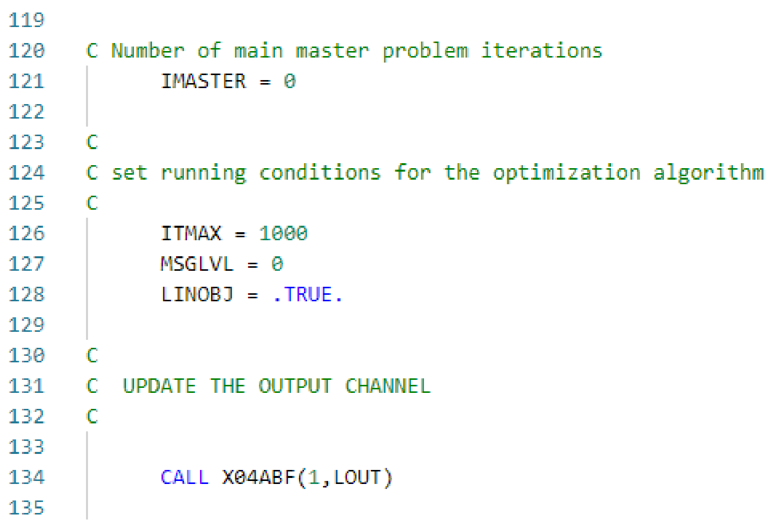
preftyp	I	Type of segment: sell = 1, buy = 2, gen = 3
pload	R()	Load vector
ifromb	I()	From buses for branches
itob	I()	To buses for branches
xinv	R()	Inverse for branch resistance
istat	I()	Status for branches
ratea	R()	Thermal rating for branch
Output data		
cincr	R()	Incremental cost for buses modelled in opt
pbuy	R()	Buying of power on bus
psell	R()	Selling of power on bus
pgen	R()	Generation of power on bus
teta0	R()	Voltage angles
basflow	R()	Base case flow for all branches
Other		
nlcold	I	Number of linear constrains before new are added
ibusno	I()	Bus number iterator
wconct	I()	Indicator if a branch is candidate for outage
wbus	I()	Array telling if a bus should be monitored
akoeff	R()	Array keeping sensitivities
ickt	I()	Branch identifier
sbase	R	Base value for apparent power S
busnam	C()	Bus names

In addition to the input variables, there are local variables and variables from common blocks. The local variables are given in Table 3.2 and the common block parameters are given in Table A.1 in Appendix A.1.

Table 3.2: Local parameters for the master routine. I = integer, R = real. () preceding the data type indicates an array of that type.

Name	Type	Description
ncnow	I	Number of constraints
ib	I	Bus number iterator
ig	I	Generator number iterator
il	I	Line number iterator
ifb	I	From bus iterator
itb	I	To bus iterator
iconp	I()	Constraint identifier
imaster	I	Number of master problem iterations
rside	R()	Right side of equation

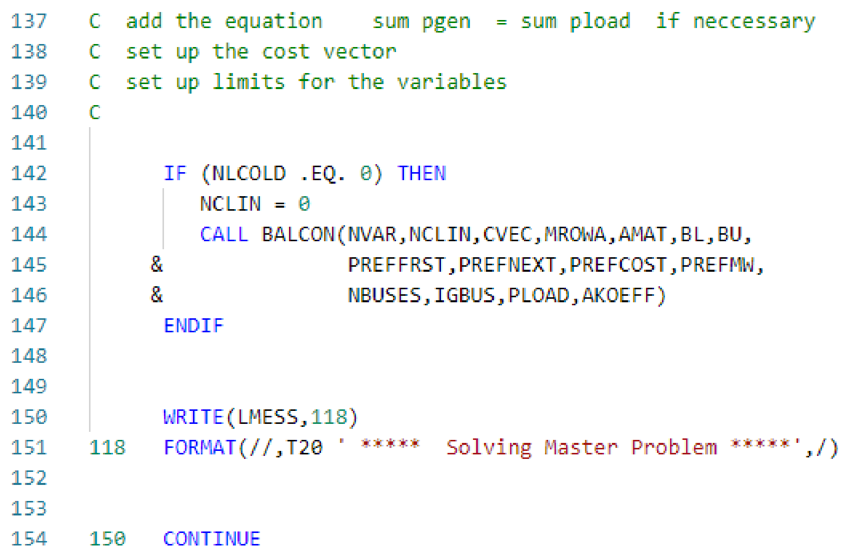
The master algorithm begins with the declaration of the variables, both the local parameters and the input parameters. It then sets the number of master iterations, imaster, equal to zero, and sets the running conditions for the optimization algorithm, itmax, msglvl, and linobj. In terms of the translation, those three variables are not crucial since another optimization algorithm will be implemented in C. Next, the output channel is updated with the routine X04ABF from the mathematical optimization tool NAG [30], which will be replaced in the translated version. Figure 3.3 shows these steps. The description of the routine and variable declarations are not included since the information is given above.



```

C Number of main master problem iterations
      IMASTER = 0

C
C set running conditions for the optimization algorithm
C
      ITMAX = 1000
      MSGLVL = 0
      LINOBJ = .TRUE.

C
C  UPDATE THE OUTPUT CHANNEL
C

      CALL X04ABF(1,LOUT)

```

Figure 3.3: File "master.f" part 1/5.

The next step is to set up the balance constraint, which says that the sum of the load must equal the sum of the generated power, as given in Equation 24. If nlcold equals zero, there are no constraints, so the number of linear constraints is set to zero, and the balcon subroutine is called. Balcon sets the balance constraint, the cost vector and the maximum capacity for the variables. See Figure 3.4 for the code. After these initializations are complete, the solving of the master problem starts. The continue statement has the label 150 and will be referred to later in the code.



```
C  add the equation   sum pgen  = sum pload  if neccessary
C  set up the cost vector
C  set up limits for the variables
C

      IF (NLCOLD .EQ. 0) THEN
         NCLIN = 0
         CALL BALCON(NVAR,NCLIN,CVEC,MROWA,AMAT,BL,BU,
     &               PREFFRST,PREFNEXT,PREFCOST,PREFMW,
     &               NBUSES,IGBUS,PLOAD,AKOEFF)
      ENDIF


      WRITE(LMESS,118)
118   FORMAT(//,T20 ' *****  Solving Master Problem *****',/)


150   CONTINUE
```

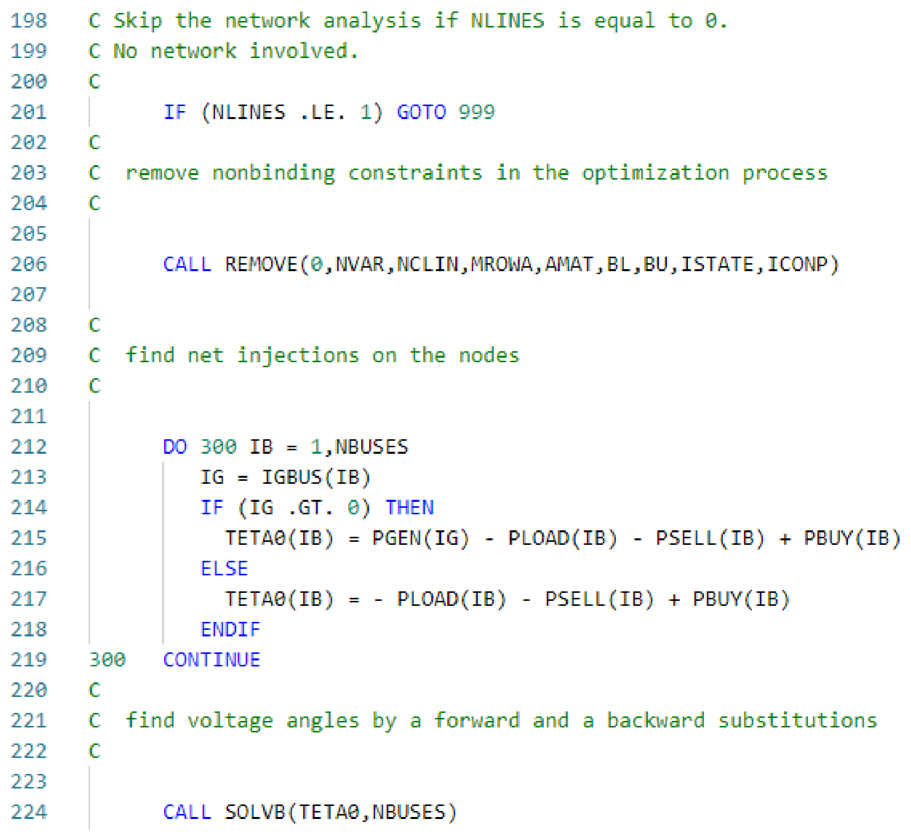
Figure 3.4: File "master.f" part 2/5.

The program moves on to calculate the optimum running condition with the given constraints. The master problem iterator is updated and the NAG optimization routine E04MBF, which maximizes or minimizes a function, is called. Documentation on E04MBF can be found at [31]. Figure 3.5 shows this section of the routine. If the optimization routine finds an acceptable solution, ifail will be set to zero, and the program will jump to label 250, which continues with updating the generation vector with the subroutine "update". If the optimization problem is infeasible, the upper and lower constraints must be modified, which is an option that is only used if the problem cannot be solved within the given limitations. The subroutine "upcon" updates the constraints.

```
C  find optimum with given constraints
C

      NCNOW = NCLIN
      IFAIL = 1

      NCTOTL = NVAR + NCLIN

      IMASTER = IMASTER + 1
C
      CALL E04MBF(ITMAX,MSGLVL,NVAR,NCLIN,NCTOTL,MROWA,
     &            AMAT,BL,BU,CVEC,LINOBJ,X,ISTATE,OBJLP,CLAMDA,
     &            IWORK,LIWORK,WORK,LWORK,IFAIL)
C
      WRITE(LMESS,123) IMASTER, OBJLP*SBASE
 123  FORMAT(/,1X,I2,'. Iteration - Power Rescheduling Cost : ',
     &               F15.3,/)

      IF (IFAIL .EQ. 0) GOTO 250
C
C  update upper and lower constraints  if the problem is infeasible
C  The option is only used when the constraints can not be satiesfies
C  The constraints are updated due to the close to optimal solution
C  returned from the nag-routine
C
      CALL UPCON(NVAR,NCLIN,MROWA,AMAT,BL,BU,X,ISTATE,TOLD)
      WRITE(LMESS,210)
 210  FORMAT(/,1X,'Infeasible case - Constraint Relaxation applied',
     &          ' in Master Problem',/)

 250  CONTINUE

C
C   update the generation vector
C

      CALL UPDATE(NBUSES,NGENS,CINCR,PBUY,PSELL,PGEN,IGBUS,
     &            PREFFRST,PREFNEXT,PREFTYP,NVAR,CLAMDA,CVEC,X)
```

Figure 3.5: File "master.f" part 3/5.

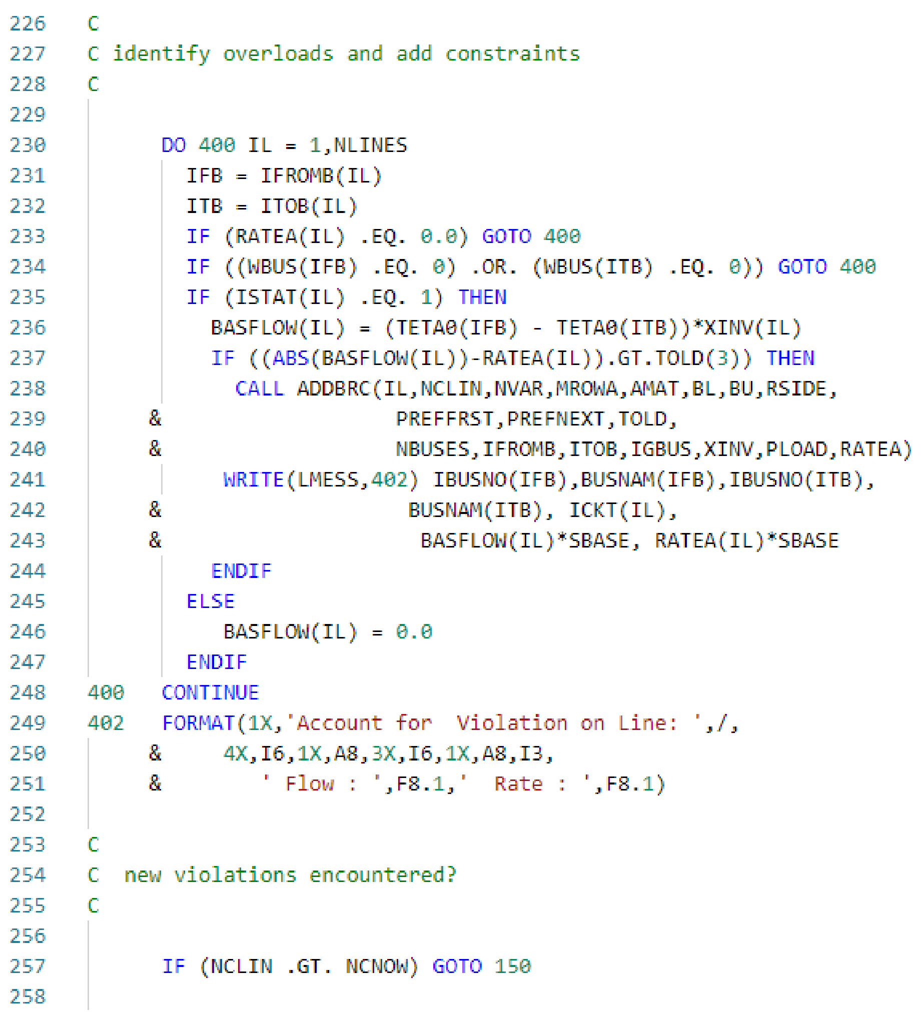
The next part of the code checks the nlines variable and jumps to label 999, which is almost at the end of the code, if the number of lines is less than or equal to one. That outcome means that there is no network to be analyzed, and therefore, the network analysis should be skipped. After this, the constraints that are no longer binding are removed with remove. Figure 3.6 shows the fourth part of the master routine. The net injection on all the buses is found with the do-loop starting on line 212. "Ib" iterates through all the buses, and "ig" determines whether the current bus is a generator or not, which in turn determines which calculation is performed. A DC load flow is then performed by "solvb" to find the voltage angles. As explained in Section 2.3, array indexing starts at 1 in Fortran and zero in C. In line 212 in Figure 3.6, ib is set to iterate from 1 to the number of buses, "nbuses". In the C code, ib must be set to iterate from zero to nbuses-1. This applies to all such instances.



```
C Skip the network analysis if NLINES is equal to 0.
C No network involved.
C
      IF (NLINES .LE. 1) GOTO 999
C
C  remove nonbinding constraints in the optimization process
C

      CALL REMOVE(0,NVAR,NCLIN,MROWA,AMAT,BL,BU,ISTATE,ICONP)

C
C  find net injections on the nodes
C

      DO 300 IB = 1,NBUSES
         IG = IGBUS(IB)
         IF (IG .GT. 0) THEN
           TETA0(IB) = PGEN(IG) - PLOAD(IB) - PSELL(IB) + PBUY(IB)
         ELSE
           TETA0(IB) = - PLOAD(IB) - PSELL(IB) + PBUY(IB)
         ENDIF
300   CONTINUE
C
C  find voltage angles by a forward and a backward substitutions
C

      CALL SOLVB(TETA0,NBUSES)
```

Figure 3.6: File "master.f" part 4/5.

The next step is to identify any overloads, and add corresponding constraints if there are any, see Figure 3.7. The loop iterates over all of the lines in the network and checks if any of them are overloaded. Two places in the loop the command "goto 400" appears after an if-statement: line 233 and line 234. The label 400 is the continue-statement at the end the loop, meaning that if the if-statement is true, the code skips to the next iteration of the loop, without performing any of the lines in the loop following the goto-statement. The line flow is calculated using Equation 19, and stored in the variable basflow. If the flow exceeds the corresponding value in ratea, the function addbrc is called, which then adds the constraint to the problem. If addbrc is called it updates the constraint counter, nclin, by one for every new constraint. After this loop, the if-statement in line 257 checks if there are more constraints now than before, and if that is true the code jumps to label 150, which is the start of the master problem. As a result, the code between label 150 and this if-statement runs in a loop until no further constraints are identified and the problem is considered solved.



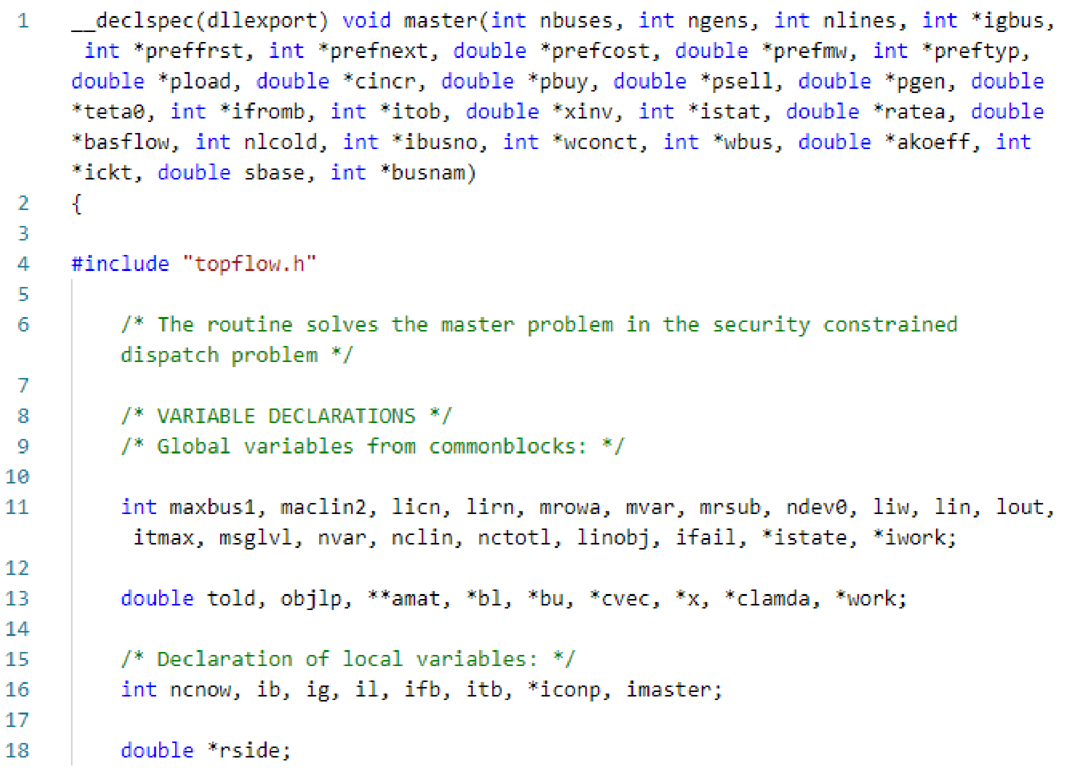
```
C
C identify overloads and add constraints
C

      DO 400 IL = 1,NLINES
        IFB = IFROMB(IL)
        ITB = ITOB(IL)
        IF (RATEA(IL) .EQ. 0.0) GOTO 400
        IF ((WBUS(IFB) .EQ. 0) .OR. (WBUS(ITB) .EQ. 0)) GOTO 400
        IF (ISTAT(IL) .EQ. 1) THEN
          BASFLOW(IL) = (TETA0(IFB) - TETA0(ITB))*XINV(IL)
          IF ((ABS(BASFLOW(IL))-RATEA(IL)).GT.TOLD(3)) THEN
            CALL ADDBRC(IL,NCLIN,NVAR,MROWA,AMAT,BL,BU,RSIDE,
     &                   PREFFRST,PREFNEXT,TOLD,
     &                   NBUSES,IFROMB,ITOB,IGBUS,XINV,PLOAD,RATEA)
           WRITE(LMESS,402) IBUSNO(IFB),BUSNAM(IFB),IBUSNO(ITB),
     &                    BUSNAM(ITB), ICKT(IL),
     &                     BASFLOW(IL)*SBASE, RATEA(IL)*SBASE
          ENDIF
        ELSE
           BASFLOW(IL) = 0.0
        ENDIF
400   CONTINUE
402   FORMAT(1X,'Account for  Violation on Line: ',/,
     &     4X,I6,1X,A8,3X,I6,1X,A8,I3,
     &        ' Flow : ',F8.1,'  Rate : ',F8.1)

C
C  new violations encountered?
C

      IF (NCLIN .GT. NCNOW) GOTO 150

```

Figure 3.7: File "master.f" part 5/5.

Lastly, the program prints the active transmission constraints after the current iteration and sets the violation counter, nclold, equal to the number of constraints nclin. This ends the routine. The entire code for the master routine is given in Appendix A.2.

The first step in translating this routine to C is to define the C data type for each of the input parameters since they must be known already in the function definition. The master routine is defined as a void function since the output data is included in the input parameters. Alternatively, it could have returned an int indicating if the function call was successful or not, but due to the size and complexity of the function, it was decided that master should be a void function, at least in the first translating attempts. Furthermore, the other routines called by the master function and the routine that calls the master routine do not have exception handling, and it is not a priority to include that at this stage in the modernization process.

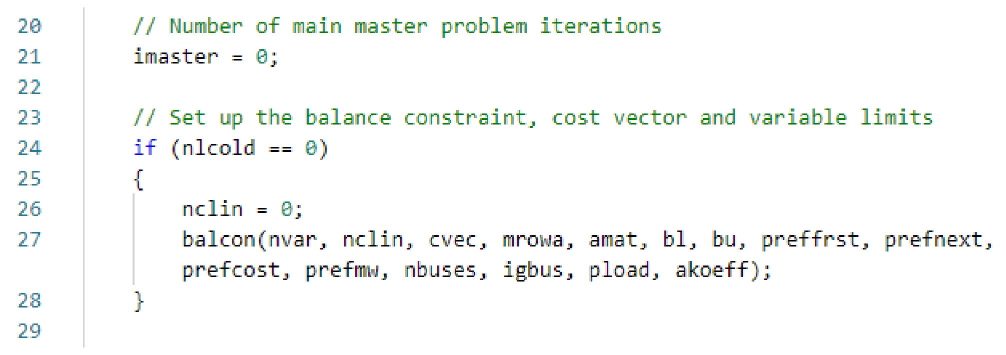
One of the codes created by Statnett is the header file "topflow.h", given in Appendix B.1. The function abs(x) returns the absolute value of the given value and will be used in master. In C, all variables must be declared before they are used; therefore, the declaration is written first. Since there are no common blocks in C, the common block variables must also be declared like any other variables, and if they have a value before the function call, they must be initialized. Some of the common block parameters should be initialized, but since the details of the toolbox are not yet specified, like the maximum number of buses, for example, they will only be declared for now. Further, many of the common block parameters are used in the optimization routines which will be replaced, and are only included for the sake of readability at this stage and will be changed later. The first part of the C code, which are the function definition and the variable declarations, are given in Figure 3.8.



```
__declspec(dllexport) void master(int nbuses, int ngens, int nlines, int *igbus,
 int *preffrst, int *prefnext, double *prefcost, double *prefmw, int *preftyp,
double *pload, double *cincr, double *pbuy, double *psell, double *pgen, double
*teta0, int *ifromb, int *itob, double *xinv, int *istat, double *ratea, double
*basflow, int nlcold, int *ibusno, int *wconct, int *wbus, double *akoeff, int
*ickt, double sbase, int *busnam)
{

#include "topflow.h"

    /* The routine solves the master problem in the security constrained
    dispatch problem */

    /* VARIABLE DECLARATIONS */
    /* Global variables from commonblocks: */

    int maxbus1, maclin2, licn, lirn, mrowa, mvar, mrsub, ndev0, liw, lin, lout,
     itmax, msglvl, nvar, nclin, nctotl, linobj, ifail, *istate, *iwork;

    double told, objlp, **amat, *bl, *bu, *cvec, *x, *clamda, *work;

    /* Declaration of local variables: */
    int ncnow, ib, ig, il, ifb, itb, *iconp, imaster;

    double *rside;
```

Figure 3.8: File "master.c" part 1/6.

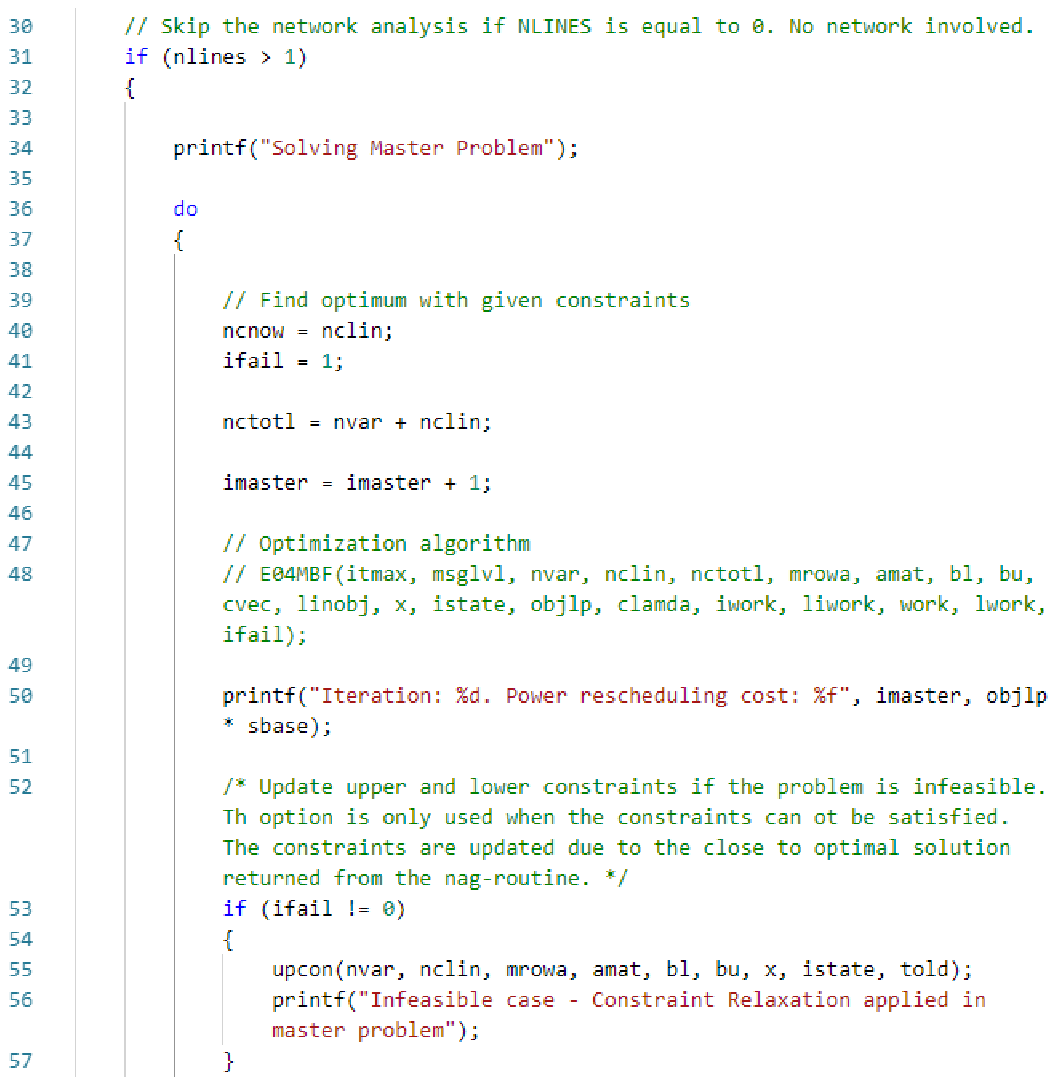
Next, as in the Fortran routine, the imaster variable is set to zero. Initialization of the variables for the optimization algorithm is not included since this routine will be replaced. If there are no constraints initially, nlcold = 0, nclin is set to zero, and the balcon function is called, see Figure 3.9.



```
// Number of main master problem iterations
imaster = 0;

// Set up the balance constraint, cost vector and variable limits
if (nlcold == 0)
{
    nclin = 0;
    balcon(nvar, nclin, cvec, mrowa, amat, bl, bu, preffrst, prefnext,
    prefcost, prefmw, nbuses, igbus, pload, akoeff);
}

```

Figure 3.9: File "master.c" part 2/6.

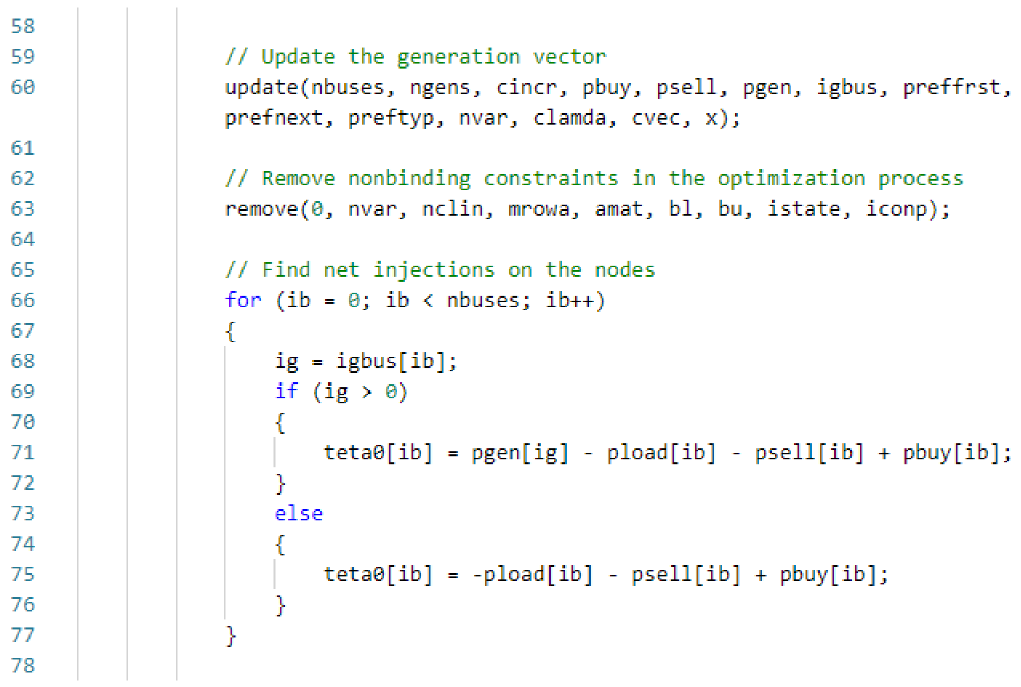
In the Fortran routine, the part of the code that solves the master problem starts at line 154 with the continue-statement with the label 150 and ends at line 257 with the "goto 150" statement. The goto-statement forms a loop that will run until no new constraints are added. The best way to implement a goto-statement in C is a do-while-loop, which runs at least once, and until the given condition is no longer valid. Line 201 in the Fortran routine is an if-sentence that skips the network analysis if there is no network, as explained above. If all that follows should be inside an if-statement, then the first half of the code would be inside the do-while-loop, and the other half would come after, which does not work. Therefore this statement is moved up to before the beginning of the do-while-loop and will include the master problem loop. The first part of the loop is given in Figure 3.10. The optimization algorithm is not yet implemented, but the old routine is included for readability. The ifail variable will have to be incorporated in the new optimization algorithm since it determines whether to update the constraints or not.



```c
// Skip the network analysis if NLINES is equal to 0. No network involved.
if (nlines > 1)
{

    printf("Solving Master Problem");

    do
    {

        // Find optimum with given constraints
        ncnow = nclin;
        ifail = 1;

        nctotl = nvar + nclin;

        imaster = imaster + 1;

        // Optimization algorithm
        // E04MBF(itmax, msglvl, nvar, nclin, nctotl, mrowa, amat, bl, bu, cvec, linobj, x, istate, objlp, clamda, iwork, liwork, work, lwork, ifail);

        printf("Iteration: %d. Power rescheduling cost: %f", imaster, objlp * sbase);

        /* Update upper and lower constraints if the problem is infeasible. Th option is only used when the constraints can ot be satisfied. The constraints are updated due to the close to optimal solution returned from the nag-routine. */
        if (ifail != 0)
        {
            upcon(nvar, nclin, mrowa, amat, bl, bu, x, istate, told);
            printf("Infeasible case - Constraint Relaxation applied in master problem");
        }
```

Figure 3.10: File "master.c" part 3/6.

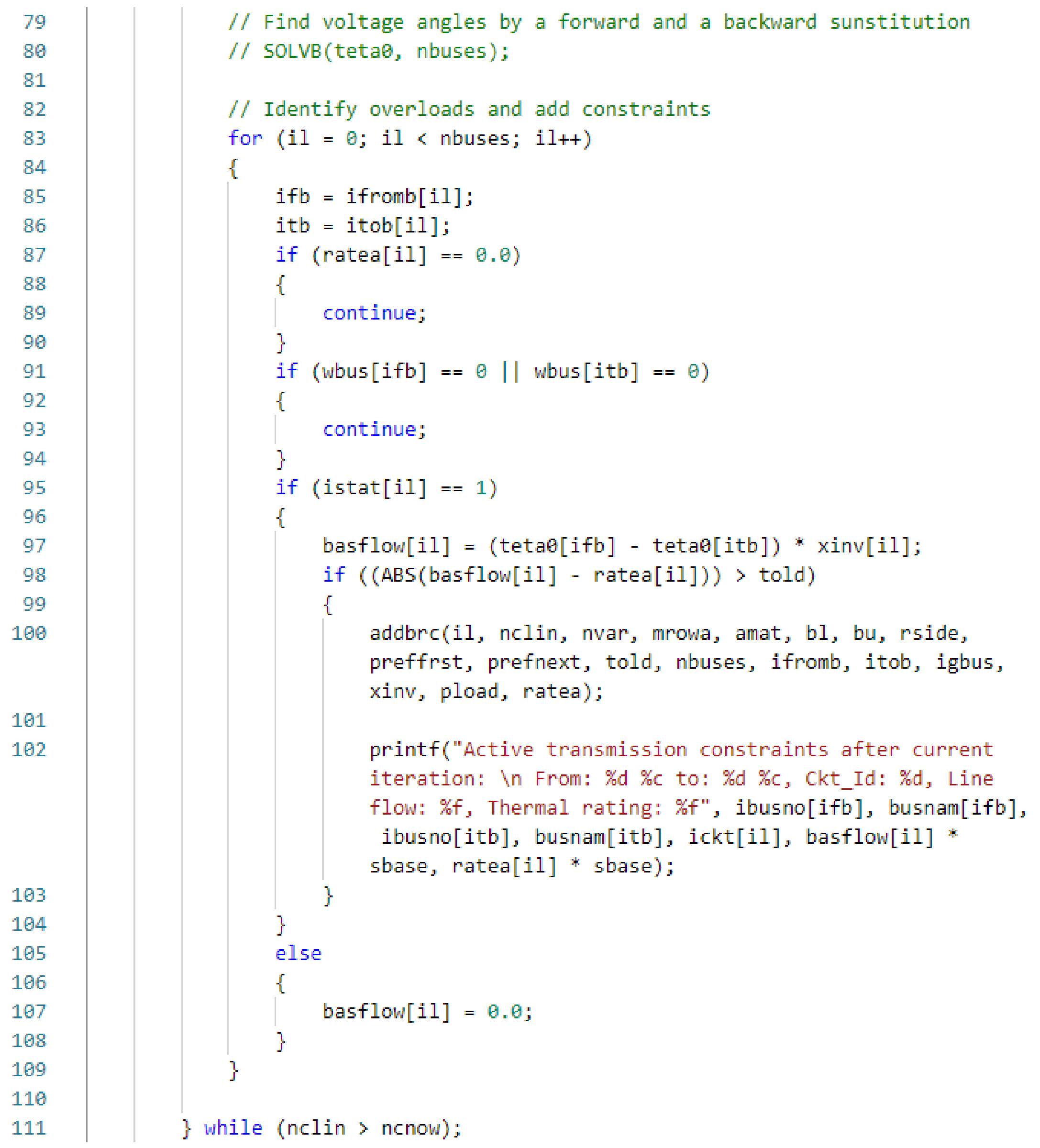
The next part of the loop, given in Figure 3.11 is identical to the Fortran routine. The functions update and remove are called to update the generation vector and remove nonbinding constraints, respectively, and the net injection on all buses is calculated. The for-loop is set to iterate from zero and until nbuses-1. Less than, "<", means that the loop will stop when ib=nbuses.



```c

// Update the generation vector
update(nbuses, ngens, cincr, pbuy, psell, pgen, igbus, preffrst,
prefnext, preftyp, nvar, clamda, cvec, x);

// Remove nonbinding constraints in the optimization process
remove(0, nvar, nclin, mrowa, amat, bl, bu, istate, iconp);

// Find net injections on the nodes
for (ib = 0; ib < nbuses; ib++)
{
    ig = igbus[ib];
    if (ig > 0)
    {
        teta0[ib] = pgen[ig] - pload[ib] - psell[ib] + pbuy[ib];
    }
    else
    {
        teta0[ib] = -pload[ib] - psell[ib] + pbuy[ib];
    }
}

```

Figure 3.11: File "master.c" part 4/6.

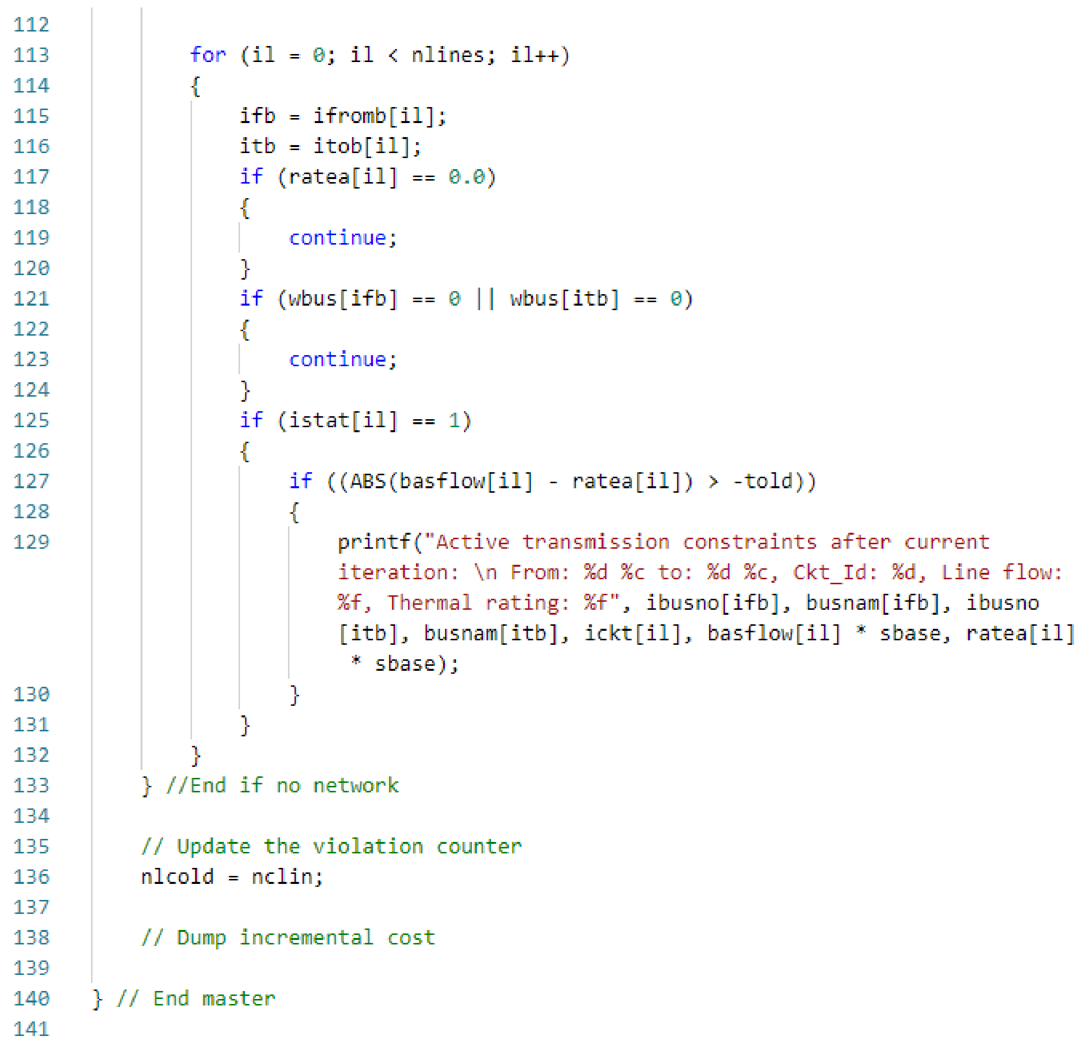
The next step in the master problem loop is to solve the DC load flow to find the voltage angles, and this part of the code is displayed in Figure 3.12. An appropriate algorithm for solving the DC load flow must also be implemented before the code is complete. The loop that calculates the line flows and adds new constraints needs a few modifications to perform the two goto jumps. Both goto-statements jump to the end of the loop, and therefore effectively begins the next iteration of the loop without executing the remainder of the loop statements. The C statement "continue" jumps out of the current iteration and immediately starts the next, and is therefore used to replace the jumps in the Fortran code. The condition in the do-while-loop comes at the end of the loop, meaning that the loop runs through one iteration before the condition is checked. If the logical expression is true, the next iteration is started.



```
            // Find voltage angles by a forward and a backward sunstitution
            // SOLVB(teta0, nbuses);

            // Identify overloads and add constraints
            for (il = 0; il < nbuses; il++)
            {
                ifb = ifromb[il];
                itb = itob[il];
                if (ratea[il] == 0.0)
                {
                    continue;
                }
                if (wbus[ifb] == 0 || wbus[itb] == 0)
                {
                    continue;
                }
                if (istat[il] == 1)
                {
                    basflow[il] = (teta0[ifb] - teta0[itb]) * xinv[il];
                    if ((ABS(basflow[il] - ratea[il])) > told)
                    {
                        addbrc(il, nclin, nvar, mrowa, amat, bl, bu, rside,
                        preffrst, prefnext, told, nbuses, ifromb, itob, igbus,
                        xinv, pload, ratea);

                        printf("Active transmission constraints after current
                        iteration: \n From: %d %c to: %d %c, Ckt_Id: %d, Line
                        flow: %f, Thermal rating: %f", ibusno[ifb], busnam[ifb],
                         ibusno[itb], busnam[itb], ickt[il], basflow[il] *
                        sbase, ratea[il] * sbase);
                    }
                }
                else
                {
                    basflow[il] = 0.0;
                }
            }

        } while (nclin > ncnow);
```

Figure 3.12: File "master.c" part 5/6.

The last part of the master function is given in Figure 3.13. The constraints are printed to the screen as they are after the previous iteration. The print statement can be altered to be written to a file if that is desired in a later edit of the code. How the print statement is formatted is not significant at this stage of the translation, it should be made to fit the Python interface to the program. In the Fortran code, the incremental costs are written to a file "inccost.res", but as it is of little relevance at this stage, it was omitted in the C version of the master routine. The last action performed in master.c is to update the violation counter.



```c

        for (il = 0; il < nlines; il++)
        {
            ifb = ifromb[il];
            itb = itob[il];
            if (ratea[il] == 0.0)
            {
                continue;
            }
            if (wbus[ifb] == 0 || wbus[itb] == 0)
            {
                continue;
            }
            if (istat[il] == 1)
            {
                if ((ABS(basflow[il] - ratea[il]) > -told))
                {
                    printf("Active transmission constraints after current iteration: \n From: %d %c to: %d %c, Ckt_Id: %d, Line flow: %f, Thermal rating: %f", ibusno[ifb], busnam[ifb], ibusno[itb], busnam[itb], ickt[il], basflow[il] * sbase, ratea[il] * sbase);
                }
            }
        }
    } //End if no network

    // Update the violation counter
    nlcold = nclin;

    // Dump incremental cost

} // End master

```

Figure 3.13: File "master.c" part 6/6.

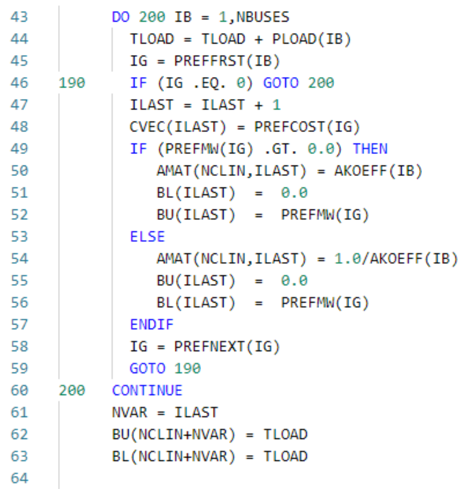
3.4.1 Balcon

Balcon is a function used in the master function. It does three things: it constructs the balance constraint between the load and the generation, see Equation 24, it prepares the optimization cost vector, and it sets up the variable limits. The balcon function itself does not use any common blocks, but some of the input variables are from common blocks in the master function. It takes 15 input parameters from the master function and defines four local variables which are listed in Table 3.3.

Table 3.3: Local variables in balcon.

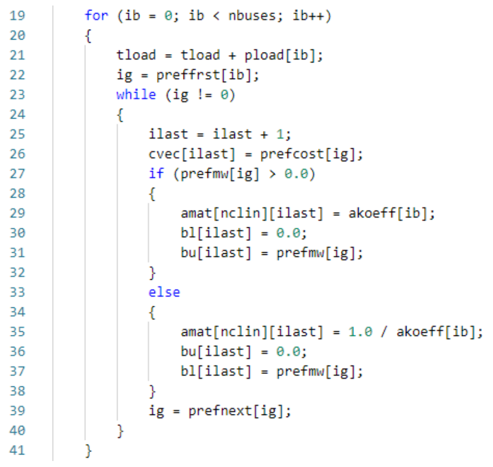
Name	Type	Description
Tload	R	Total load
Nseg	I	Number of segments
Ig	I	Generator number iterator
Ib	I	Bus number iterator

The Fortran loop that performs these operations contains two goto-statements, see Figure 3.14. The loop iterates through all of the buses. The result of the first if-statement is to jump directly to the start of the next iteration if ig is 0, meaning there is no generator connected to the current bus. If there are generators on the current bus, the cost vector and variable limits are updated. The loop ends in a goto-statement that jumps back to the if-statement determining of ig is zero or not, therefore this is written as a while-loop in the C code, running as long as ig is not equal to zero. See Figure 3.15 for the C loop in "balcon.c". The files are given in their entirety in the Appendix A.3 and B.4.



```
      DO 200 IB = 1,NBUSES
        TLOAD = TLOAD + PLOAD(IB)
        IG = PREFFRST(IB)
190     IF (IG .EQ. 0) GOTO 200
        ILAST = ILAST + 1
        CVEC(ILAST) = PREFCOST(IG)
        IF (PREFMW(IG) .GT. 0.0) THEN
           AMAT(NCLIN,ILAST) = AKOEFF(IB)
           BL(ILAST)  =  0.0
           BU(ILAST)  =  PREFMW(IG)
        ELSE
           AMAT(NCLIN,ILAST) = 1.0/AKOEFF(IB)
           BU(ILAST)  =  0.0
           BL(ILAST)  =  PREFMW(IG)
        ENDIF
        IG = PREFNEXT(IG)
        GOTO 190
200   CONTINUE
      NVAR = ILAST
      BU(NCLIN+NVAR) = TLOAD
      BL(NCLIN+NVAR) = TLOAD
```

Figure 3.14: Lines 43 to 64 in "balcon.f".



```
for (ib = 0; ib < nbuses; ib++)
{
    tload = tload + pload[ib];
    ig = preffrst[ib];
    while (ig != 0)
    {
        ilast = ilast + 1;
        cvec[ilast] = prefcost[ig];
        if (prefmw[ig] > 0.0)
        {
            amat[nclin][ilast] = akoeff[ib];
            bl[ilast] = 0.0;
            bu[ilast] = prefmw[ig];
        }
        else
        {
            amat[nclin][ilast] = 1.0 / akoeff[ib];
            bu[ilast] = 0.0;
            bl[ilast] = prefmw[ig];
        }
        ig = prefnext[ig];
    }
}
```

Figure 3.15: Lines 19 to 42 in "balcon.c".

3.4.2 Upcon

Upcon is used in the master function if the problem is infeasible. If the current constraints cannot be satisfied, the upper and lower constraints must be updated to get a solution. The function takes nine input parameters, and in the C version, four local variables are declared. The main difference between the Fortran version and the C version of the upcon function is the goto-statement in line 14 in the Fortran code that is changed to a continue-statement in line 13 in the C code. The complete codes are given in Appendix A.4 for the Fortran version and in Appendix B.5 for the C version.

3.4.3 Update

The function update is used in each loop of the master problem to update the generation vector after the optimization. The function takes 14 input parameters. It begins by setting the generation vectors to zero, and then it iterates through all the buses and updates the variables with the solution from the optimization algorithm. The goto-statements in line 41 and 55 in the Fortran function are changed to a while loop in the C function. The codes are given in Appendix A.5 for the Fortran file and in Appendix B.6 for the C file.

3.4.4 Remove

Remove is called in the master routine after the generation vector is updated, and it removes constraints that are no longer relevant in the optimization. The first of nine input parameters is ip, which is the status type to remove from the restrictions. In the master routine, the ip argument is zero. In remove, the status of the variables are checked against ip, and if the statuses are equal, the constraints are removed. All the constraints are checked. The source code for the function is given in the Appendix: A.6 for the Fortran code and B.7 for the C code.

3.4.5 Solvb

The function solvb calculates the voltage angles using a routine from the HSL Mathematical Software Library. As can be seen in Appendix A.7, the function declares the needed variables from two common blocks and calls the routine MA27CD from the HSL library [32]. This function must be rewritten once the new mathematical computation algorithms are in place.

3.4.6 Addbrc

The function addbrc adds a linear constraint to the optimization problem if the limits on a line are exceeded. After the DC load flow has been computed, the line flow is calculated on every line. The line flow is checked against the restrictions, and if there is a violation, addbrc is called on the violated branch, and a new linear constraint is added. The codes are given in Appendix A.8 for the Fortran file and in Appendix B.8 for the C file.

3.5 Bulilist

Bulilist is a subroutine in secc_opt, as shown in Figure 3.2. It is a function that is very central to the program because it sorts the variables needed to do the load flow analysis. The buses in a network are numbered, and bulilist takes in lists of different variables for the buses and creates linked lists where the variables are ordered according to the buses. Both the input and output parameters are given as input arguments to the function. It does not create a linked list precisely like it was illustrated in Figure 2.1, for that structs are needed. Instead, bulilist creates a list that simulates a linked list.

The input variables are listed in Table 3.4. After the variable declarations, preffrst, prefnext, and ipos are initialized by setting all elements to zero. The function then loops through all elements and creates a simulated linked list. Ibus "points" to which bus an element belongs. "Point" is quoted because it is not an actual pointer, only a simulated one. Preffrst "points" to the first element in a list, and prefnext "points" to the next element. Prefcost, prefmw and preftyp all "point" to the elements from the lists price, maxmw, and segtyp respectively. The Fortran source code for bulilist is given in Appendix A.9

Table 3.4: Input variables in bulilist.

Name	**Type**	**Description**
nbuses	I	Number of buses
ibus	I()	Connecting segment to busnumber
price	R()	Price per segment
maxmw	R()	Maximum capacity
segtyp	I()	Type of segment: sell = 1, buy = 2, gen = 3
nseg	I	Number of segments
preffrst	I()	Pointer to first segment in pref. func.
prefnext	I()	Pointer to next segment
prefcost	R()	Cost vector for the segments
prefmw	R()	Maximum capacity
preftyp	I()	Type of segment: sell = 1, buy = 2, gen = 3

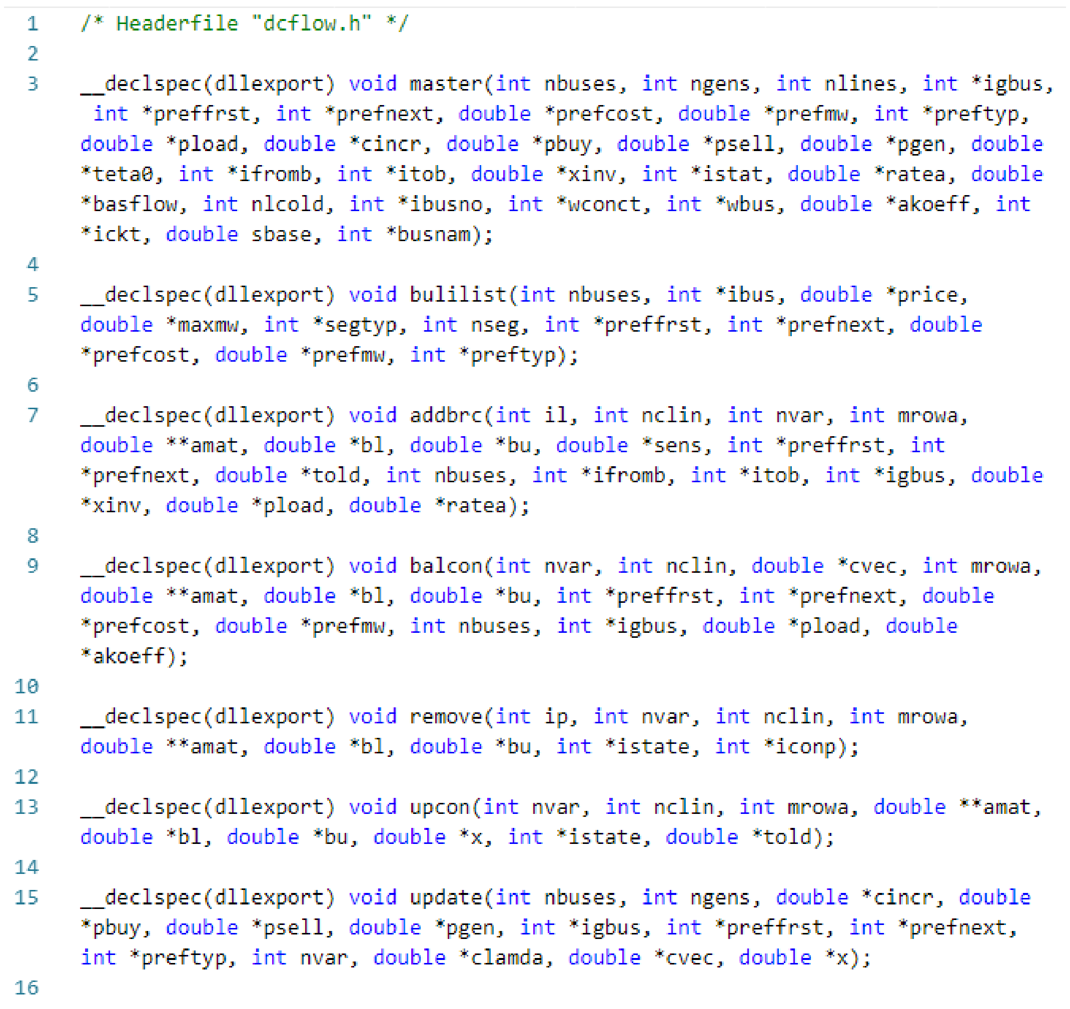
There are several ways to store and order variables for power system analysis, included, but not limited to, matrices, structs, and lists. Matrices are intuitive to use since the load flow problem is constructed using matrices, like the $\mathbf{Y}_{bus}$ in Equation 3 and the Jacobi matrix in the Newton-Raphson method in Equation 13. However, in most large networks, most of the elements will be zero since the buses are only connected to a handful of other buses, which results in a sparse matrix. For systems with thousands of buses, this will require a large amount of additional storage and capacity. An alternative that was considered when translating bulilist to C was to use structs to implement bulilist in a way that would create linked lists identical to the example in Figure 2.1. All variables associated with a bus could be stored in a single struct, and the pointer would point to the struct containing all the variables associated with the next bus. This would eliminate the problem of elements that are zero and still be an intuitive way to store the variables. However, changing from simulated linked lists to matrices or linked lists would have made changes to almost every routine and subroutine necessary. Therefore, in order to avoid the ripple effect that can come when implementing major changes in a routine, bulilist in C creates a simulated linked list in the same way it was written in Fortran. The code is given in Appendix B.9.

3.6 Initial and coeval

Initial initializes the system and factorizes the inverse B matrix, **B'**. The matrix is built using the subroutine buildb and is subsequently factorized by the routines MA27AD and MA27BD from the HSL Mathematical Software Library [32]. The coeval routine analyzes the effects on the system of all single branch outages. These routines have not been translated because they were not a priority. Therefore, in the following description of the translated secc_opt routine, they are left as comments to illustrate where they will be when they are finished and for the sake of readability. The Fortran files initial.f and coeval.f are given in Appendices A.10 and A.11 respectively.

3.7 Secc_opt

Secc_opt is a routine for analyzing the security constrained optimal dispatch. All of the functions described thus far are subroutines in secc_opt. Before they can be called in secc_opt, they must be defined, and the function definitions are provided in the header file "dcflow.h", see Figure 3.16.



```c
/* Headerfile "dcflow.h" */

__declspec(dllexport) void master(int nbuses, int ngens, int nlines, int *igbus,
 int *preffrst, int *prefnext, double *prefcost, double *prefmw, int *preftyp,
double *pload, double *cincr, double *pbuy, double *psell, double *pgen, double
*teta0, int *ifromb, int *itob, double *xinv, int *istat, double *ratea, double
*basflow, int nlcold, int *ibusno, int *wconct, int *wbus, double *akoeff, int
*ickt, double sbase, int *busnam);

__declspec(dllexport) void bulilist(int nbuses, int *ibus, double *price,
double *maxmw, int *segtyp, int nseg, int *preffrst, int *prefnext, double
*prefcost, double *prefmw, int *preftyp);

__declspec(dllexport) void addbrc(int il, int nclin, int nvar, int mrowa,
double **amat, double *bl, double *bu, double *sens, int *preffrst, int
*prefnext, double *told, int nbuses, int *ifromb, int *itob, int *igbus, double
*xinv, double *pload, double *ratea);

__declspec(dllexport) void balcon(int nvar, int nclin, double *cvec, int mrowa,
double **amat, double *bl, double *bu, int *preffrst, int *prefnext, double
*prefcost, double *prefmw, int nbuses, int *igbus, double *pload, double
*akoeff);

__declspec(dllexport) void remove(int ip, int nvar, int nclin, int mrowa,
double **amat, double *bl, double *bu, int *istate, int *iconp);

__declspec(dllexport) void upcon(int nvar, int nclin, int mrowa, double **amat,
double *bl, double *bu, double *x, int *istate, double *told);

__declspec(dllexport) void update(int nbuses, int ngens, double *cincr, double
*pbuy, double *psell, double *pgen, int *igbus, int *preffrst, int *prefnext,
int *preftyp, int nvar, double *clamda, double *cvec, double *x);

```

Figure 3.16: Header file "dcflow.h".

Secc_opt takes 51 input parameters and does not include any common blocks. Some of the variables are for the MA27 routines from the HSL Mathematical Software Library, as explained in the previous section. The routine starts by calling bulilist to build the linked list with the bus variables. The system is then built and factorized with the "initial" routine if two conditions are satisfied: the number of lines is more than one and the system is not already built. Next, master is called to solve the master problem. The master problem is the economic dispatch without any contingencies. The last step is to perform a contingency analysis based on single line outages with the coeval routine. This creates the security constrained dispatch. The Fortran code is given in Appendix A.12.

The translation of secc_opt is kept close to the original, see Appendix B.10. The two if-goto-statements are rewritten to a single if-statement with two conditions in the logical expression, to make the code easier. The function call to initial is included as a comment to avoid errors when testing the code. After the label "200 continue" in line 76 in the

Fortran version of secc_opt, the master and coeval functions are called. As with initial, the function call to coeval is a comment, for the same reasons. The "goto 200" statement after the call to secc_opt forms a loop at the 200 label, which in the C version is solved with a do-while-loop with the same condition as in the Fortran code. After the call to the master routine, if there is no network, the contingency analysis is skipped with a goto-statement that jumps to the end of the code. In C this is done with a break-statement, which immediately jumps out of the loop.

4 Building DCflow

The next sections describe how to build a shared library of the translated functions and import them to Python. The name of the library is "DCflow" as a way to refer to the translated DC load flow program system.

4.1 Building a shared library

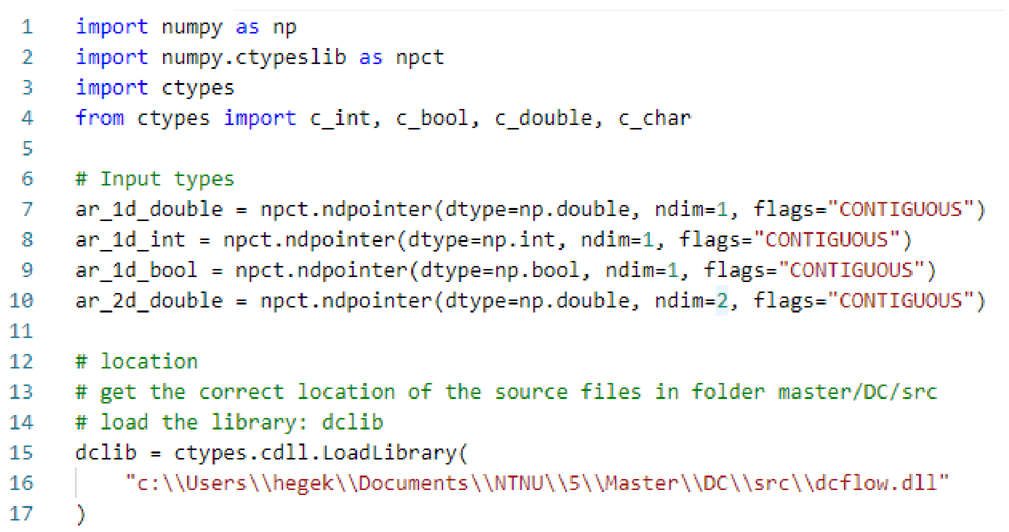
The SConstruct file for DCflow is given in Figure 4.1. The name of the library is defined first, followed by a list of the files that should be included. More files can be added later as they are translated from Fortran. Running the scons command as described in Section 2.8 creates the library which is then ready to be imported in Python.

```
SharedLibrary(
    "dcflow",
    [
        "secc_opt.c",
        "master.c",
        "bulilist.c",
        "balcon.c",
        "upcon.c",
        "update.c",
        "remove.c",
        "addbrc.c",
    ],
)
```

Figure 4.1: File "SConstruct.py".

4.2 Ctypes wrapper

Once the functions have been translated to C, the Ctypes wrapper can be written. Only one wrapper file is needed for the security constrained optimal dispatch (secc_opt). The first part of the wrapper file, "dcflow_wrapper.py", is given in Figure 4.2.

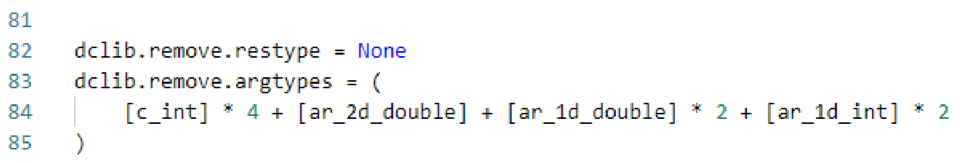


```python
import numpy as np
import numpy.ctypeslib as npct
import ctypes
from ctypes import c_int, c_bool, c_double, c_char

# Input types
ar_1d_double = npct.ndpointer(dtype=np.double, ndim=1, flags="CONTIGUOUS")
ar_1d_int = npct.ndpointer(dtype=np.int, ndim=1, flags="CONTIGUOUS")
ar_1d_bool = npct.ndpointer(dtype=np.bool, ndim=1, flags="CONTIGUOUS")
ar_2d_double = npct.ndpointer(dtype=np.double, ndim=2, flags="CONTIGUOUS")

# location
# get the correct location of the source files in folder master/DC/src
# load the library: dclib
dclib = ctypes.cdll.LoadLibrary(
    "c:\\Users\\hegek\\Documents\\NTNU\\5\\Master\\DC\\src\\dcflow.dll"
)
```

Figure 4.2: Ctypes wrapper file for secc_opt: "dcflow_wrapper.py".

The required packages are NumPy, Ctypeslib, and Ctypes, and these must be imported at the beginning of the script. The four C data types that are needed are int, double, bool, and char. When they are imported with name from Ctypes, it is not necessary to specify Ctypes each time they are used, which makes the code shorter and easier to read. The input types that are pointers to arrays must be defined before the functions can be wrapped. The function "ndpointer" from the Ctypeslib package defines an array for input and output parameter specifications [21]. Three limitations are specified: data type, dimension, and flags. The data type, dtype, is set to the specified data type. An array can only contain one data type. The dimension, dim, is set to 1 for a one-dimensional array, or 2 for a matrix. "CONTIGUOUS" is the only flag needed in this wrapper. The next step is to import the library created in Section 4.1. The library, which contains all of the files translated from Fortran to C, is named "dclib".

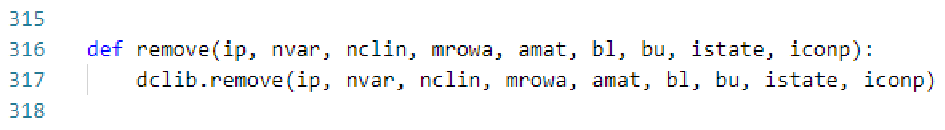
The data types for the input and output arguments must be specified for each function. All the Fortran routines have been translated into C void functions; therefore, restype is set to "None" for all functions. The argument definitions for the remove function is presented in Figure 4.3 as an example. The data types are listed in the same order as the parameters are listed in the function definition. This is done for all functions that are wrapped.



```

dclib.remove.restype = None
dclib.remove.argtypes = (
    [c_int] * 4 + [ar_2d_double] + [ar_1d_double] * 2 + [ar_1d_int] * 2
)
```

Figure 4.3: Argument data type definitions for remove in 9 "dcflow_wrapper.py".

The last step is to define the functions. The function definition for the remove function is given in Figure 4.4. The definition in line 316 is how the function will be called from Python, which in all cases in this wrapper code is the same as how the function was called from Fortran. The function can be modified to include more input parameters, and it can be modified to return one or more variables since that is possible in Python.



```
315
316   def remove(ip, nvar, nclin, mrowa, amat, bl, bu, istate, iconp):
317       dclib.remove(ip, nvar, nclin, mrowa, amat, bl, bu, istate, iconp)
318
```

Figure 4.4: Function definition for remove in "dcflow_wrapper.py".

The complete wrapper file, "dcflow_wrapper.py", is given in Appendix C. The DCflow module can now be imported to Python as follows:

```
import dcflow_wrapper as dcflow
```

4.3 Optimization algorithm

The focus of the master thesis has been on translating the Fortran codes in secc_opt to C, and the optimization algorithms and linear system solvers have been left out of the process thus far. At this point in the project, it seems reasonable to keep the new codes as close to the original as possible, as discussed earlier. The new optimization algorithms will require changes, but to keep these changes as small as possible and to keep the changes from creating a ripple-effect throughout the program system, the new optimization algorithms should be made to fit the program system and not the other way around. The best way to do this may be to create an interface for the algorithms that use the variables that already exist or copies data from them. The new optimization algorithms must replace the NAG routines used in the Fortran program. The algorithms for mathematical computation of systems of linear equations from the HSL library must also be replaced. An overview of these routines is given in Table 4.1.

Table 4.1: Optimization algorithms and linear system solvers used in the Fortran version of the DC load flow program secc_opt and subroutines.

Routine	**Used in**	**Description**	**Documentation**
X04ABF	master.f	Update the output channel	[30]
E04MBF	master.f	Find optimum with given constraints	[31]
MA27CD	solvb.f	Solve system $B' \cdot x = RHS$	[32]
MA27AD	initial.f	Symbolic factorization of matrix	[32]
MA27BD	initial.f	Numerical factorization of matrix	[32]

4.3.1 Lpsolve

Lpsolve is a free linear programming solver [33]. Lpsolve does not have a model size limit, which will be useful as the size of the networks to be analyzed with the toolbox is not yet known. It works well and is simple to use, which is why it was chosen as the optimization tool. Lpsolve can be used in different ways: from the LPSolve IDE [34], and using the Lpsolve API from a range of different programming languages, including Python and C. The Lpsolve API (Application Programming Interface) is a collection of functions that builds and solves a linear problem.

4.3.2 Optimization example

The example in Figure 4.5 is taken from the lecture notes by Professor Olav Bjarte Fosso in the course ELK 14 at NTNU [26]. The figure represents a three-bus system with same line reactances and power flow limits. Bus 1 and 2 are generator (PV) buses with a maximum capacity of 150 MW. Bus 3 is a 180 MW load (PQ) bus and is used as the slack bus. MC_1 and MC_2 are the marginal cost of the generation for generator 1 and 2 respectively; it represents the cost of change in generation given in euros per MWh. The maximum line flow on all three lines is 100 MW. Losses are ignored. The example will be used to illustrate how Lpsolve can be used in a load flow program, and by extension, DCflow, from Python and C.

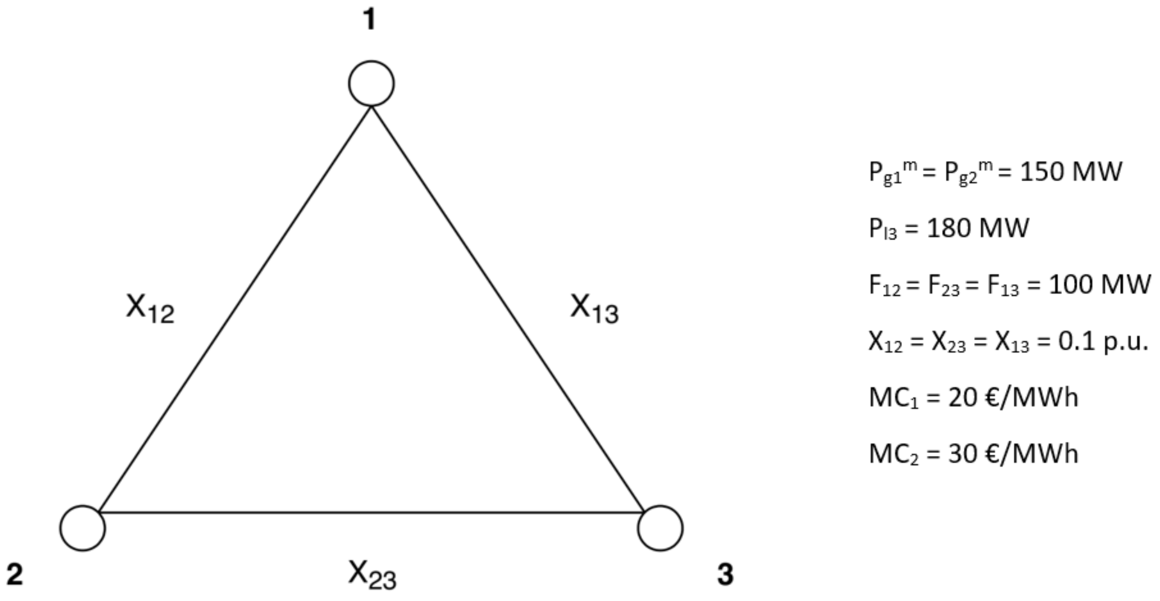


Figure 4.5: DC optimal power flow example from lecture notes in course ELK 14 at NTNU [26].

The economic dispatch without the transmission constraints gives overload on line 1-3. Since generator 1 is cheaper than generator 2, generator 1 will produce at maximum capacity, which causes overload. The solution before regulation is given in Table 4.2.

Table 4.2: Economic dispatch in example system without transmission constraints.

Line	**P [MW]**	**Transmission constraint [MW]**	**Bus**	**P [MW]**	**Max. capacity [MW]**
1-2	60	100	1	150	150
2-3	70	100	2	30	150
1-3	110	100	3	180	slack bus

The problem in its mathematical form is presented in Equations 27 to 29. The objective function is to minimize the cost of the changed generation necessary to satisfy all constraints, including transmission constraints, and it is given in Equation 27. Here x1 and x2 are dP1+ and dP1-, the increased and decreased generation, respectively, on bus 1, and x3 and x4 are dP2+ and dP2-, the increased and decreased generation, respectively, on bus 2.

$$min : 20x1 + 20x2 + 30x3 + 30x4 \tag{27}$$

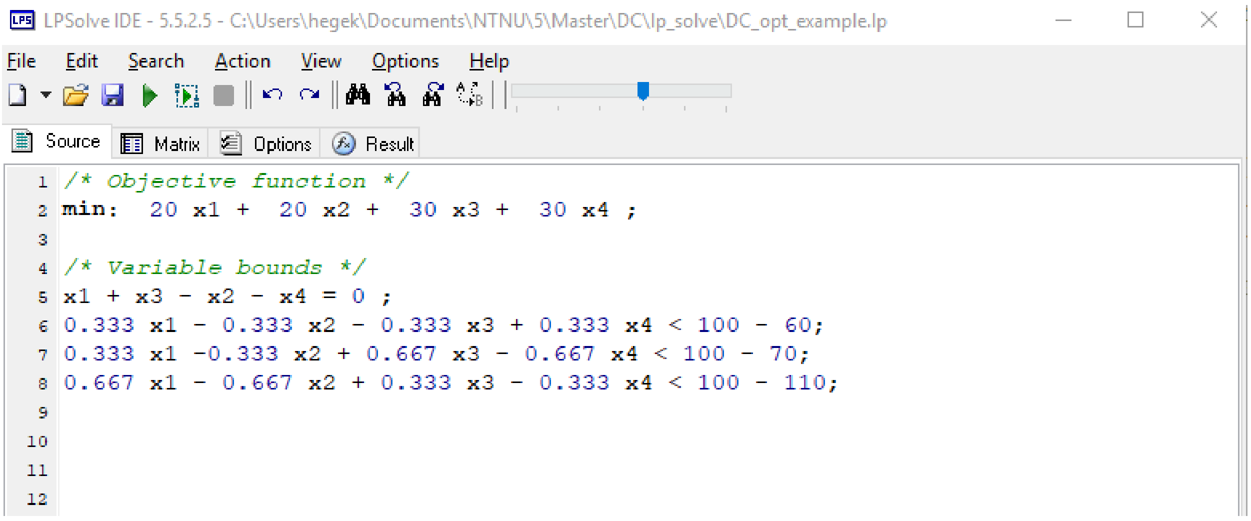
The balance constraint is given in Equation 28.

$$x1 - x2 + x3 - x4 = 0 \tag{28}$$

Lastly, the transmission constraints are given in Equation 29, on the form as in Equation 25.

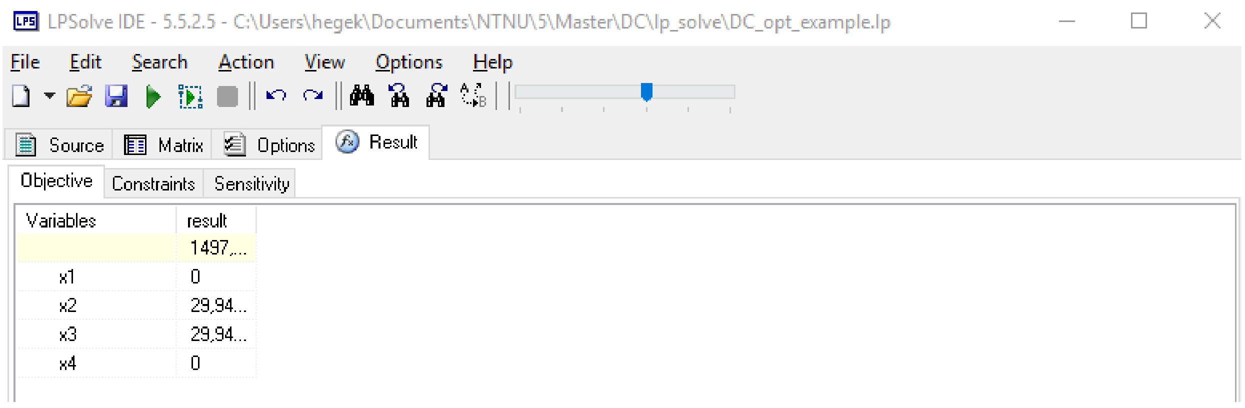
$$\begin{aligned} 0.333x1 - 0.333x2 - 0.333x3 + 0.333x4 &< 100 - 60; \\ 0.333x1 - 0.333x2 + 0.667x3 - 0.667x4 &< 100 - 70; \\ 0.667x1 - 0.667x2 + 0.333x3 - 0.333x4 &< 100 - 110; \end{aligned} \tag{29}$$

The example was first solved in the LPSolve IDE, see Figure 4.6. The objective function and the constraints are written in the IDE on the form they are presented in the equations above. The solution is obtained by pressing "Solve", which presents the results as shown in Figure 4.7.


LPSolve IDE - 5.5.2.5 - C:\Users\hegek\Documents\NTNU\5\Master\DC\lp_solve\DC_opt_example.lp
File Edit Search Action View Options Help
Source Matrix Options Result

```
/* Objective function */
min:  20 x1 +  20 x2 +  30 x3 +  30 x4 ;

/* Variable bounds */
x1 + x3 - x2 - x4 = 0 ;
0.333 x1 - 0.333 x2 - 0.333 x3 + 0.333 x4 < 100 - 60;
0.333 x1 -0.333 x2 + 0.667 x3 - 0.667 x4 < 100 - 70;
0.667 x1 - 0.667 x2 + 0.333 x3 - 0.333 x4 < 100 - 110;
```

Figure 4.6: Screenshot of the optimization example in LPSolve IDE.


LPSolve IDE - 5.5.2.5 - C:\Users\hegek\Documents\NTNU\5\Master\DC\lp_solve\DC_opt_example.lp
File
Edit
Search
Action
View
Options
Help
Source
Matrix
Options
Result
Objective
Constraints
Sensitivity

Variables	result
	1497,...
x1	0
x2	29,94...
x3	29,94...
x4	0

Figure 4.7: Screenshot of the solution to the optimization example in LPSolve IDE.

The results are presented in a table, and it shows the objective function and the variables. A summary of the results is given in Table 4.3. The results are rounded up for clarity.

Table 4.3: Results from the optimization example in LPSolve IDE.

Variable in IDE	**Variable in example**	**Value**
Objective	Objective	1500
x1	dP1+	0 MW
x2	dP1-	30 MW
x3	dP2+	30 MW
x4	dP2-	0 MW

The changes from the optimization results in the system state given in Table 4.4. This is the economic dispatch considering the transmission constraints.

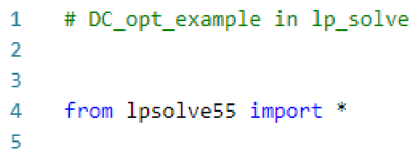
Table 4.4: Economic dispatch in example system considering transmission constraints.

Line	**P [MW]**	**Bus**	**P [MW]**
1-2	40	1	120
2-3	80	2	60
1-3	100	3	180

It can be seen from Table 4.4 in conjunction with Table 4.3 that the generation at bus 1 has been reduced with 30 MW, and that the corresponding increased generation has been made at bus 2. This results in a line flow on line 1-2 of 100 MW, which is at the limit. This is the cheapest solution satisfying all constraints and is thus the optimized solution. The cost of the regulation is 1500 euros.

4.3.3 Optimization example in Python

The Lpsolve API can be imported to Python as an extension module. For the extension module to be available from Python, the Lpsolve shared library, lpsolve55.dll, must be downloaded from [35]. The location of the library must then be added to Path and PYTHONPATH [36]. Lpsolve can be imported, as shown in Figure 4.8, which is the first part of the optimization example in Python. The example is the same three-bus example from the previous section.



```
# DC_opt_example in lp_solve


from lpsolve55 import *

```

Figure 4.8: DC optimization example solved in Python using Lpsolve part 1/3.

The standard way of calling Lpsolve functions is presented in line 7 in Figure 4.9. The return value "ret" will be zero if the function call succeeds. It is not required to use the return value, but it can be used to give error messages if a function call fails. The first step is to create a linear problem structure with the function "make_lp", which returns a pointer, lp, to a matrix structure with allocated rows and columns as specified by the function arguments. New rows and columns can be added later. Zero rows are specified because the constraints will be added separately, and four columns are specified corresponding to the four variables: x1, x2, x3, and x4. The first argument in the functions will generally be this matrix structure, lp. The "set_verbose" function specifies which messages should be reported to the user, and the flag "IMPORTANT" will return only warnings and errors. The objective function is set with the function "set_obj_fn" where the arguments are lp and a list of the relation between the variables in the objective function given in Equation 27. The last step in constructing the problem is to add the constraints, which will add rows to the matrix lp. The first constraint is the balance constraint, see Equation 28, and the next three are the transmission constraints from Equation 29.

```
 7  # Syntax: [ret1, ret2, ...] = lpsolve('functionname', arg1, arg2, ...)
 8
 9  # Create a linear problem lp
10  lp = lpsolve("make_lp", 0, 4)
11  lpsolve("set_verbose", lp, IMPORTANT)
12
13  # Set the objective function
14  ret = lpsolve("set_obj_fn", lp, [20, 20, 30, 30])
15
16  # Set the linear constraints
17  ret = lpsolve("add_constraint", lp, [1, -1, 1, -1], EQ, 0)
18  ret = lpsolve("add_constraint", lp, [0.333, -0.333, -0.333, 0.333],LE, 100 - 60)
19  ret = lpsolve("add_constraint", lp, [0.333, -0.333, 0.667, -0.667],LE, 100 - 70)
20  ret = lpsolve("add_constraint", lp, [0.6667, -0.667, 0.333, -0.333],LE, 100-110)
21
```

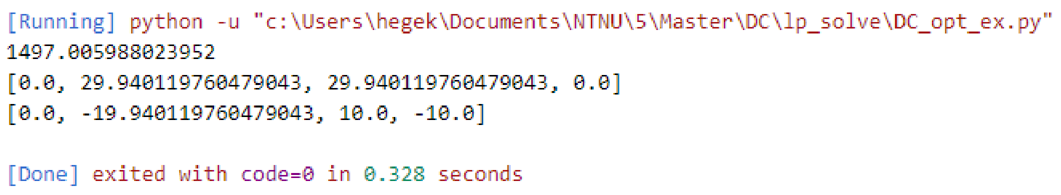
Figure 4.9: DC optimization example solved in Python using Lpsolve part 2/3.

Once the problem is set up, names can be added to the variables and constraints, and this is done in the last part of the code in Figure 4.10, with the functions "set_col_name" and "set_row_name". Lastly, the problem is solved with the "solve" function using the matrix lp as the argument, as can be seen from line 34 in Figure 4.10. The objective function, variables, and constraints are retrieved with their respective "get" functions which are then printed to the screen. The functions "get_variables" and "get_constraints" both return a status in addition to the vector with the results, and the [0] following the function call suppresses the output of the status. The last line in the code frees up the allocated memory by deleting lp. More detailed information about these functions and all the other functions included in the API can be found at [37].

```
22  # Set names
23  ret = lpsolve("set_col_name", lp, 1, "dP1+")
24  ret = lpsolve("set_col_name", lp, 2, "dP1-")
25  ret = lpsolve("set_col_name", lp, 3, "dP2+")
26  ret = lpsolve("set_col_name", lp, 4, "dP2-")
27  ret = lpsolve("set_row_name", lp, 1, "Balance")
28  ret = lpsolve("set_row_name", lp, 2, "F12")
29  ret = lpsolve("set_row_name", lp, 1, "F23")
30  ret = lpsolve("set_row_name", lp, 1, "F13")
31
32  # Solve the system and print results
33  ret = lpsolve("write_lp", lp, "a.lp")
34  lpsolve("solve", lp)
35  print(lpsolve("get_objective", lp))
36  print(lpsolve("get_variables", lp)[0])
37  print(lpsolve("get_constraints", lp)[0])
38  lpsolve("delete_lp", lp)
39
```

Figure 4.10: DC optimization example solved in Python using Lpsolve part 3/3.

Lpsolve supports the use of NumPy arrays and matrices instead of Python lists. When the code is run, it produces the result in Figure 4.11. The first number is the result of the objective function, the second line is the variables, and the final line is the constraints. It can be seen that the obtained result is the same as presented in Table 4.3. The Python example file is given in its entirety in Appendix D.1.



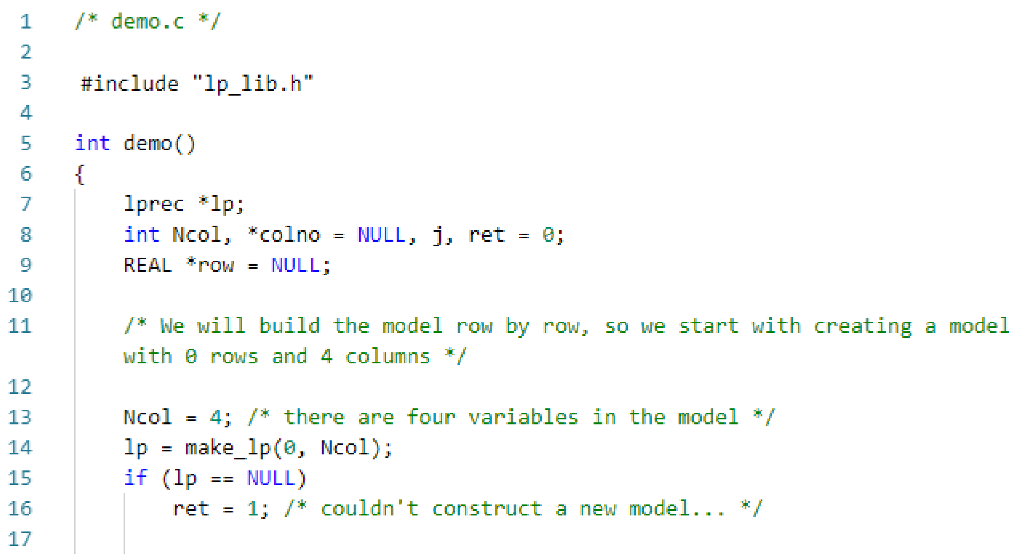
```
[Running] python -u "c:\Users\hegek\Documents\NTNU\5\Master\DC\lp_solve\DC_opt_ex.py"
1497.005988023952
[0.0, 29.940119760479043, 29.940119760479043, 0.0]
[0.0, -19.940119760479043, 10.0, -10.0]

[Done] exited with code=0 in 0.328 seconds
```

Figure 4.11: Result of Lpsolve example in Python.

4.3.4 Optimization example in C

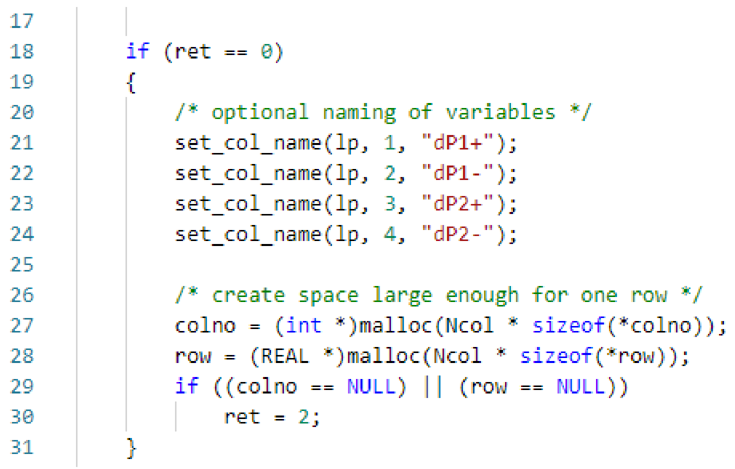
The Lpsolve API can be used from C with the header file "lp_lib.h", which can be downloaded from [33] together with the shared library files. The directory where these files are stored should be added to Path and additional dependencies so the files can be accessed. The example used to demonstrate the C API is the demo.c example from [38], using the values from the optimization example. The whole example file is given in Appendix D.2, and excerpts will be presented here.

The API functions used are mostly the same in Python and C, but because C is a compiled language all variables must be declared before use and memory must be allocated for arrays, therefore, extra steps are needed when using Lpsolve in C. The first part of the example is given in Figure 4.12. First, a pointer to an lprec structure is created, and the row and column variables are declared and initialized. Ncol, the number of columns, is set equal to the number of variables, which is four. The model is created with "make_lp", and as in the Python example, the number of rows is set to zero so the model can be built by adding the constraints row by row. The variables can be named as shown in Figure 4.13.



```c
/* demo.c */

#include "lp_lib.h"

int demo()
{
    lprec *lp;
    int Ncol, *colno = NULL, j, ret = 0;
    REAL *row = NULL;

    /* We will build the model row by row, so we start with creating a model
    with 0 rows and 4 columns */

    Ncol = 4; /* there are four variables in the model */
    lp = make_lp(0, Ncol);
    if (lp == NULL)
        ret = 1; /* couldn't construct a new model... */

```

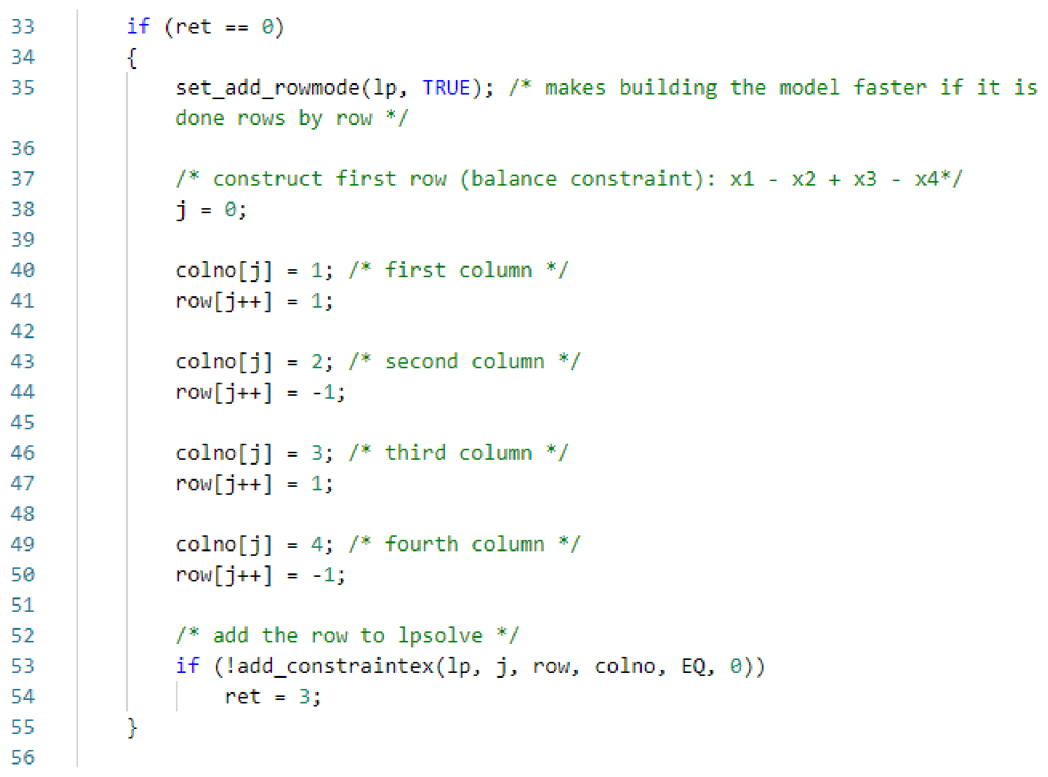
Figure 4.12: DC optimization example solved in C using Lpsolve part 1/6.



```c

    if (ret == 0)
    {
        /* optional naming of variables */
        set_col_name(lp, 1, "dP1+");
        set_col_name(lp, 2, "dP1-");
        set_col_name(lp, 3, "dP2+");
        set_col_name(lp, 4, "dP2-");

        /* create space large enough for one row */
        colno = (int *)malloc(Ncol * sizeof(*colno));
        row = (REAL *)malloc(Ncol * sizeof(*row));
        if ((colno == NULL) || (row == NULL))
            ret = 2;
    }
```

Figure 4.13: DC optimization example solved in C using Lpsolve part 2/6.

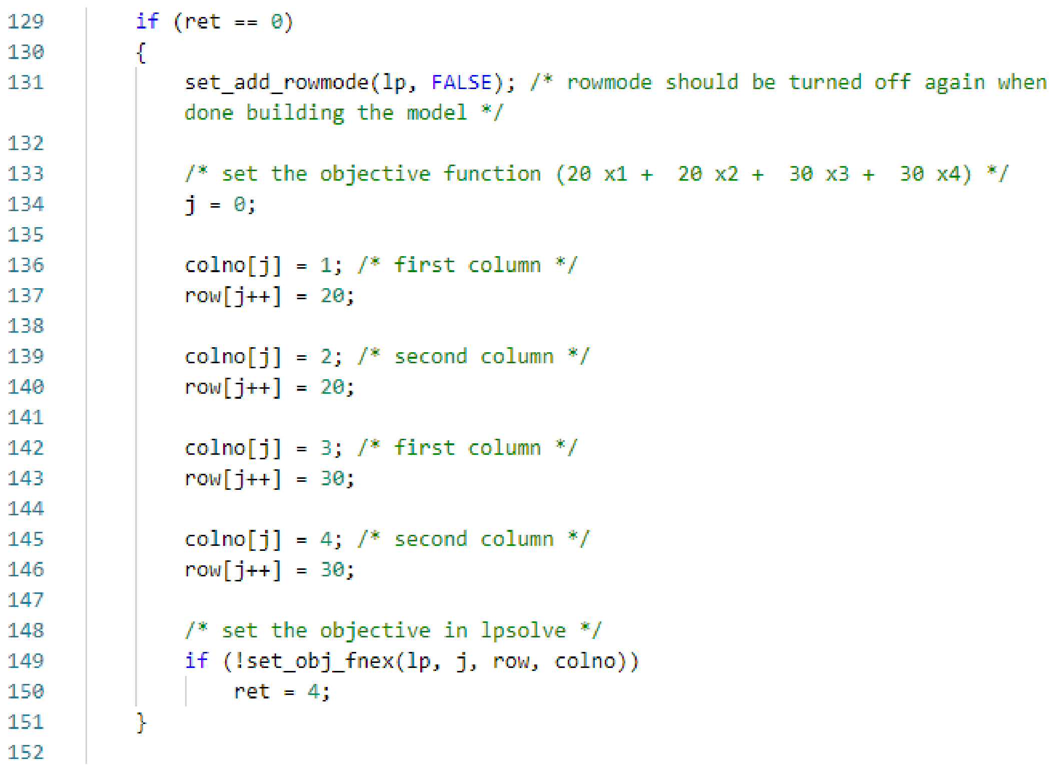
Figure 4.14 shows how to add constraints. The procedure is demonstrated with the balance constraint but is valid for all constraints on that form. The "set_add_rowmode" function improves the speed of the model building if the constraints are added row by row, which they are in this example. For large models, like power systems, the speed will be improved significantly. It should be turned on by setting it to "TRUE" before setting the constraints, and then turned off by setting it to "FALSE". The constraints are added using the "add_constraintex" function, and the transmission constraints are added in the same way as the balance constraint.



```c
    if (ret == 0)
    {
        set_add_rowmode(lp, TRUE); /* makes building the model faster if it is done rows by row */

        /* construct first row (balance constraint): x1 - x2 + x3 - x4*/
        j = 0;

        colno[j] = 1; /* first column */
        row[j++] = 1;

        colno[j] = 2; /* second column */
        row[j++] = -1;

        colno[j] = 3; /* third column */
        row[j++] = 1;

        colno[j] = 4; /* fourth column */
        row[j++] = -1;

        /* add the row to lpsolve */
        if (!add_constraintex(lp, j, row, colno, EQ, 0))
            ret = 3;
    }

```

Figure 4.14: DC optimization example solved in C using Lpsolve part 3/6.

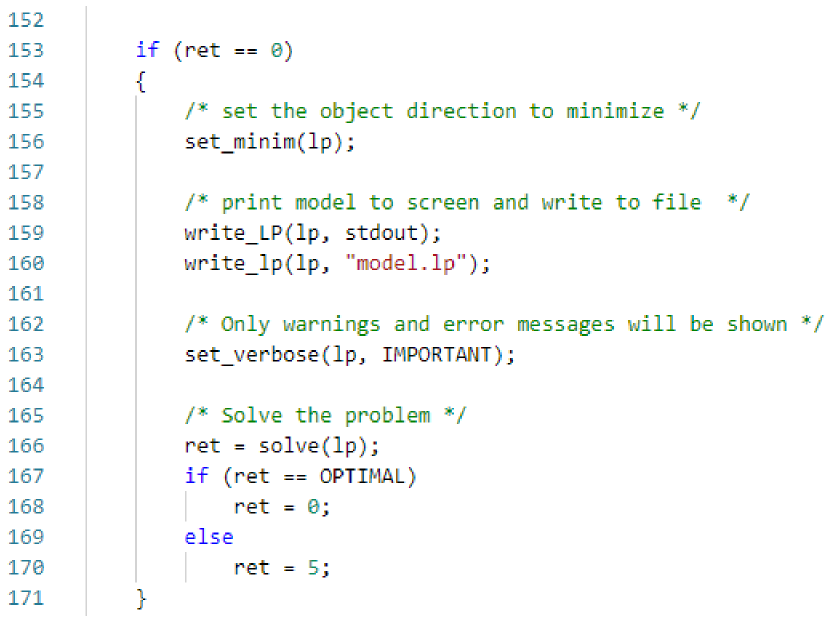
The objective function is set with the "set_obj_fnex" function as in Figure 4.15, after the row entry mode has been set to false.



```
    if (ret == 0)
    {
        set_add_rowmode(lp, FALSE); /* rowmode should be turned off again when done building the model */

        /* set the objective function (20 x1 +  20 x2 +  30 x3 +  30 x4) */
        j = 0;

        colno[j] = 1; /* first column */
        row[j++] = 20;

        colno[j] = 2; /* second column */
        row[j++] = 20;

        colno[j] = 3; /* first column */
        row[j++] = 30;

        colno[j] = 4; /* second column */
        row[j++] = 30;

        /* set the objective in lpsolve */
        if (!set_obj_fnex(lp, j, row, colno))
            ret = 4;
    }

```

Figure 4.15: DC optimization example solved in C using Lpsolve part 4/6.

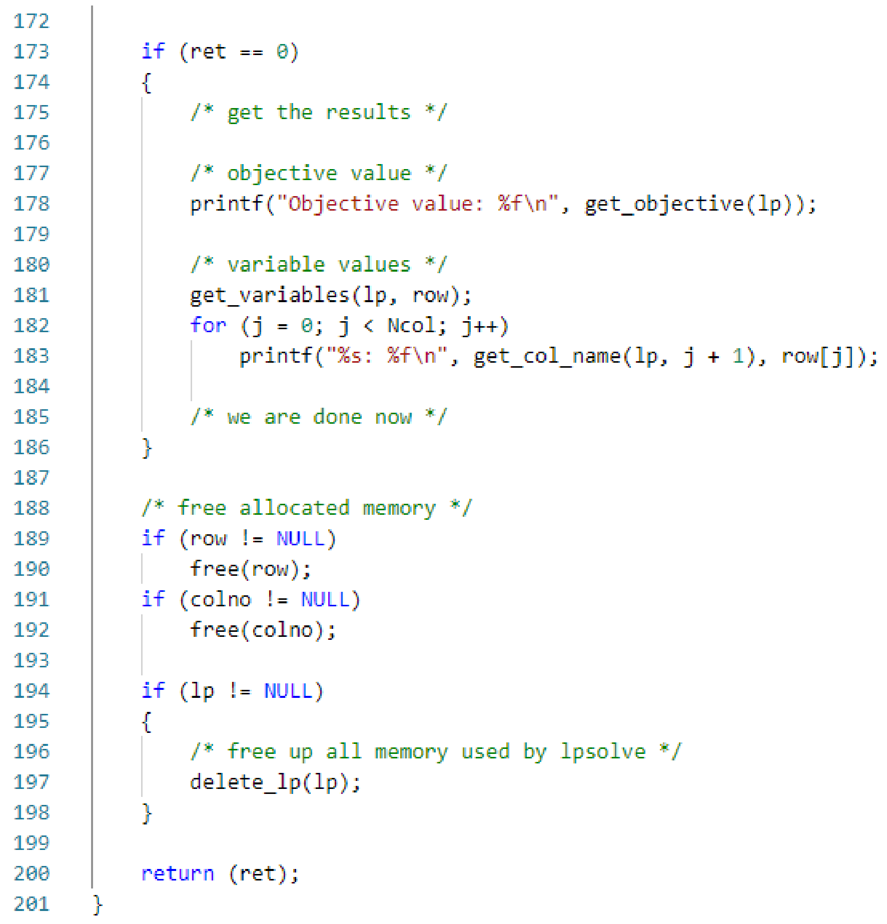
Next, the object direction is set to minimize using "set_minim", see Figure 4.16. The model can be printed to screen, and it can be written to a file, with the functions "write_LP" (stream) and "write_lp" (file). As in the Python example, the problem is solved with "solve".



```

    if (ret == 0)
    {
        /* set the object direction to minimize */
        set_minim(lp);

        /* print model to screen and write to file  */
        write_LP(lp, stdout);
        write_lp(lp, "model.lp");

        /* Only warnings and error messages will be shown */
        set_verbose(lp, IMPORTANT);

        /* Solve the problem */
        ret = solve(lp);
        if (ret == OPTIMAL)
            ret = 0;
        else
            ret = 5;
    }
```

Figure 4.16: DC optimization example solved in C using Lpsolve part 5/6.

The final step is to print the result and free the allocated memory, as shown in Figure 4.17. The code should yield the same result as when solved using the IDE and Python.



```
    if (ret == 0)
    {
        /* get the results */

        /* objective value */
        printf("Objective value: %f\n", get_objective(lp));

        /* variable values */
        get_variables(lp, row);
        for (j = 0; j < Ncol; j++)
            printf("%s: %f\n", get_col_name(lp, j + 1), row[j]);

        /* we are done now */
    }

    /* free allocated memory */
    if (row != NULL)
        free(row);
    if (colno != NULL)
        free(colno);

    if (lp != NULL)
    {
        /* free up all memory used by lpsolve */
        delete_lp(lp);
    }

    return (ret);
}
```

Figure 4.17: DC optimization example solved in C using Lpsolve part 6/6.

4.4 Implementation of the optimization algorithm in DCflow

The new optimization algorithm should be implemented in the master routine where the old function E04MBF was used, either written directly in the master routine or written as a stand-alone function. E04MBF finds the optimal operating point given constraints, which is what Lpsolve was used for in the example in Section 4.3.4. The structure of the example: make a model, add constraints, add objective function and solve, will be the same in the C master function, only for a much larger system. It must take into account that the systems to be analyzed by DCflow can be of variable sizes and configurations, not a fixed size as in the example. The algorithm should, therefore, be implemented as a loop or several loops in order for it to work on any system.

4.5 Tests

In order to assess the functionality of the translated codes and the Ctypes interface, tests were written in Python. The functions are tested separately for it to be easier to locate and analyze any errors. The setup for the test of the balcon function will be explained, and the tests for the other functions will be identical. The Python test script "dcflow_test_balcon.py" is given in Appendix E.1. The test initializes all variables needed for the dcflow system and calls the balcon function. The test scripts for the other functions are identical except for the function call, and they are therefore not included.

The optimization algorithms and linear system solvers, the replacements for the NAG-routines and HSL-routines have not been implemented in the codes; therefore, functionality and accuracy of the load flow algorithms cannot yet be tested. Preparing a test case in Python with realistic data from a power system and initializing all of the required variables for the DCflow functions would be time-consuming, and since the results would not be valid without all the necessary functions in place, a test where most of the variables are zero was tested instead. Such a test does not test the performance of the load flow itself, but it tests the functions for major errors and checks that they can run through the code. In addition, the test checks that the interface between C and Python in the Ctypes wrapper code works, that there are no type errors in the input parameters and that the functions have been defined correctly.

For the codes to be tested without checking the variables, a statement is added to the end of the source code to each function, in order to check that the function call runs all the way through the code, since the codes have been written with limited error messages and does not return a value that can be checked. The statement is a printf statement that will print the message "***END OF <NAME>***" to the screen, <name> being the name of the respective function. Many of the functions contain for-loops that will not run if variables such as nbuses, ngens, and nlines are zero; therefore, these are given a non-zero value, see Table 4.5. Both the number of variables and the maximum number of variables are set to an arbitrarily high value.

Table 4.5: Non-zero variables in "dcflow_test.py".

Variable	**Value**
nbuses	3
ngens	2
nlines	3
nvar	100
mvar	100

The system tested is, with the values in Table 4.5, a three-bus system with two generators and three lines. The initialization of the variables are sorted by parameter type: int, int*, double*, double**, double and char*. The busnam variable is an array of busnames, char*, in C, but since a suitable datatype has not yet been found in Python, it is set as an array of ints, int*, for the testing. The array data types are initialized as NumPy arrays of zeroes with the appropriate NumPy data type, intc or double, and size according to one of the variables in Table 4.5. "Intc" is identical to data type "int" in C. After the variable initialization, the function is called. The results are presented in Section 5.

4.6 GitHub

The last step in the process of modernizing the toolbox was to upload the work to the repository "topflow" on GitHub. Topflow is the name of the project started by Leif Warland in Statnett, who has interfaced the Newton-Raphson load flow in the Fortran program. All the relevant files: C source files, header files, and wrapper codes were uploaded to the folder "DCflow" so that the work can be collaborated on and continued by others.

5 Results and discussion

5.1 Translation

The translated versions of the codes presented in this thesis are most likely not the final versions that will be used when the toolbox is finished. They do, however, describe tools and techniques, like SCons, Ctypes, and do-while-loops, that can be applied to other parts of the toolbox and that can be used to further improve DCflow. The decision to keep the translated routines close to the original Fortran routines gave more time to focus on the interfacing part of the problem, which is a very critical and central part of the thesis, and one that requires a lot of trial and error. That leaves the assumption that the Fortran codes perform well and do not require significant improvements. Bulilist is such a function where it was decided that the already implemented solution is the best one. The alternatives that were considered, C structs and matrices, are both applicable solutions but were not chosen for two reasons. First of all, they would require more memory and would make the code slower. Matrices in load flows are often sparse, which is not efficient. Simulated linked lists are probably the most straightforward and most efficient alternative of the three. Second, a change in how the variables are stored would require structural changes in all or most of the other codes in DCflow, in addition to codes in the other load flow programs. When the overall design for the application system has not yet been decided, such a change does not seem necessary. The ripple effect has been mentioned several times, and it has been a conscious decision to avoid it. The efficiency of the Fortran codes should be kept while utilizing the advantages that Python gives, which is the overall goal.

A disadvantage with working with the codes this way is that once the new optimization algorithms and system solvers are implemented, the functions that use these will need changes in both the function source code and the wrapper code. Variables associated with the old algorithms will no longer be needed and must be removed and replaced with the new variables. If these variables come as input parameters to the functions, this will require the function to be defined again in the wrapper code. Other adjustments not included here may also require such changes. The translated codes can therefore not be viewed as a finished product, but rather a foundation to build future versions on.

5.2 Ctypes

In the preliminary specialization project, four methods for interfacing C with Python were looked at: Python-C-API, Ctypes, Cython and SWIG. The focus was on SWIG as the most suitable interfacing tool, see Appendix F for details. Before the start of the thesis, Ctypes had already been chosen as the method for interfacing and used to interface part of the Newton-Raphson load flow in the application system. The main reason for using Ctypes over SWIG was therefore that using only one interfacing method is more orderly and simpler than using two. Nonetheless, one of the advantages of SWIG is that it auto-generates the wrapping code, and that makes it useful on codes that require maintenance and frequent updating, which the DCflow codes will require until they are finished. Using SWIG instead of Ctypes may have done the work of updating the wrapper codes faster. The downside, however, is that the auto-generated code is almost unreadable and non-editable, which makes debugging difficult. The Ctypes wrapper code is written in pure Python, which is an advantage since Python is the preferred language in the project, and the wrapper code is therefore completely readable and controllable. To conclude, Ctypes is a reasonable choice, but it is important to note that it is not the only option.

5.3 Lpsolve

The Lpsolve example in C presented in Section 4.3.4 shows how Lpsolve can solve a simple load flow optimization problem. The API is made available through the header file "lp_lib.h". However, the code failed to run and gave the error message: "cannot open source file 'dlfcn.h'" from the header file. A google search on the error message revealed that there are relatively few forum discussions on the subject and none that provided a working solution. A post on GitHub suggests that the issue is related to dynamic loading in Windows [39]. According to the post, the header "dlfcn.h" is not used in Windows and the functions referenced in the file, dlopen, dlsym and dlclose, have Windows equivalents that should be used instead. These functions are how the operating system Linux handles dynamically loaded libraries when working with C, and they are included with the "dlfcn.h" header file [40]. However, the "dlfcn.h" header file is used in the "lp_lib.h" header file that is necessary for the Lpsolve API to be available in C. For that reason, the problem seems to be related to the operating system in use. The operating system used in this thesis is Windows 10, but the error message seems to stem from that the expected operating system is Linux.

There are other options for optimization software that can be considered instead of Lpsolve, like Clp and Gurobi. Clp (Coin-or linear programming) is a linear programming solver that, similar to Lpsolve, is open-source and can be used as a callable library [41]. Gurobi is a commercial mathematical programming solver that comes with both a C interface and an interactive shell in Python [42]. Of the three, Lpsolve is the easiest to use, but other factors that have not been considered here, like robustness and performance, may make another option more suitable.

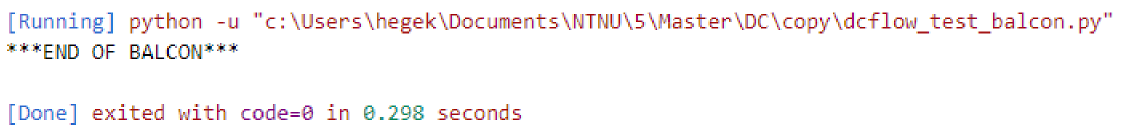
It also might be worth considering using Linux for further development of the toolbox. Linux is a free and open-source operation system popular among programmers and developers and may be easier to use and less error-prone than Windows. Further, Linux has many active support communities that can provide solutions to many problems that can occur.

5.4 Tests

The results of the tests described in Section 4.5 are presented here.

5.4.1 Master subroutines

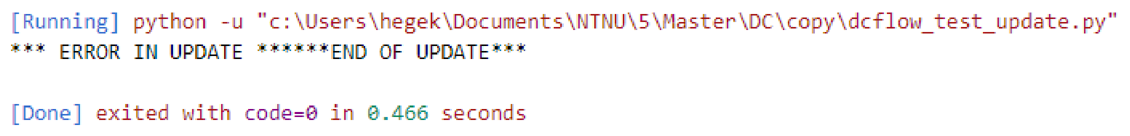
The first code to be tested was balcon, with the script "dcflow_test_balcon.py" given in Appendix E.1. The result of the test is presented in Figure 5.1, and it shows that the function was called without errors. "Code=0" means the code was run successfully. This does not necessarily mean that the function will work as expected during a load flow analysis, but it means that the function interface with Python is, as far as can be told from this test, free from errors. NumPy arrays are successfully used in the C function. There is, however, no reason to suspect that the function will not work properly during a load flow since the C version is kept close to the original Fortran function.



```
[Running] python -u "c:\Users\hegek\Documents\NTNU\5\Master\DC\copy\dcflow_test_balcon.py"
***END OF BALCON***

[Done] exited with code=0 in 0.298 seconds
```

Figure 5.1: Result of testing balcon.

The result of the test of the update function is given in Figure 5.2. It prints the error message "*** ERROR IN UPDATE ***" and the end-of-code message that signals that no errors were discovered during the code run. The error message was expected due to the last if-statement in "update.c", see Appendix B.6 lines 50-53, which says that if the variable ilast is not equal to the variable nvar, an error has occurred and the error message should be printed. In the test script, nvar is set to 100 and ilast is initialized to zero in the update function, and not updated since all of the arrays are zero. It can, therefore, be concluded that the interface for update works.



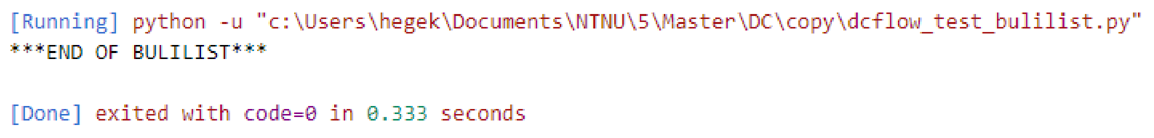
```
[Running] python -u "c:\Users\hegek\Documents\NTNU\5\Master\DC\copy\dcflow_test_update.py"
*** ERROR IN UPDATE ******END OF UPDATE***

[Done] exited with code=0 in 0.466 seconds
```

Figure 5.2: Result of testing update.

The remaining master subroutines: upcon, remove and addbrc produced similar results when tested, and are therefore not included here. Consequently, the function interfaces are expected to work during a load flow case test.

5.4.2 Bulilist

The result of the bulilist function does, as the tests of the master subroutines, not reveal any errors in the Ctypes interface to Python, as shown in Figure 5.3.

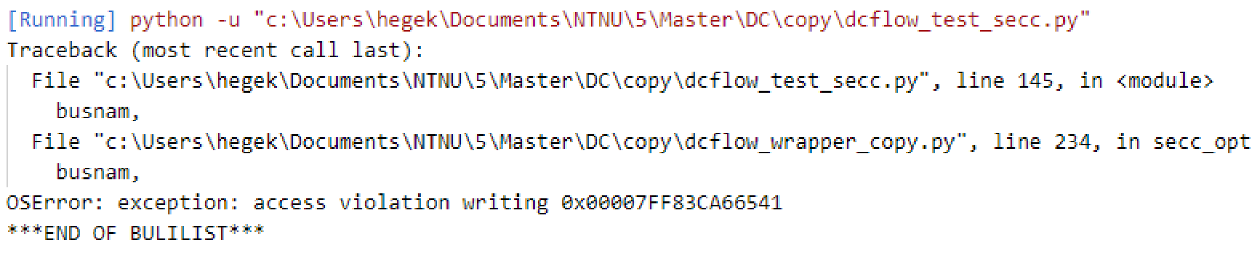


```
[Running] python -u "c:\Users\hegek\Documents\NTNU\5\Master\DC\copy\dcflow_test_bulilist.py"
***END OF BULILIST***

[Done] exited with code=0 in 0.333 seconds
```

Figure 5.3: Result of testing bulilist.

5.4.3 Secc_opt

The test of secc_opt resulted in the outcome in Figure 5.4. The function call to bulilist is successful as the end-of-bulilist message shows, but the end of secc_opt is not reached since the end-of-secc_opt message is not printed. Instead, there is an OSError with the message "exception: access violation writing 0x00007FF83CA66541", which can mean that the function is trying to access an address it does not have access to. The error can occur when trying to access and write to an array at a place that exceeds the size of the array. Secc_opt is wrapped with the same wrapper code as the previously tested functions and does not use any variable types not used in these functions. Given the type of error, it might be that the test itself causes the error. If all variables had been initialized with the correct size and value for a load flow, the error might have been avoided. However, such a test would be time-consuming to prepare and would not produce any meaningful results concerning the DC load flow as the optimization algorithms and system solvers are not yet implemented. The fault may also come from the master function since the bulilist function is called and prints the end-of-bulilist message, which suggests that the error is not caused by bulilist. Moreover, the bulilist function did not produce any errors when it was tested, as already described.

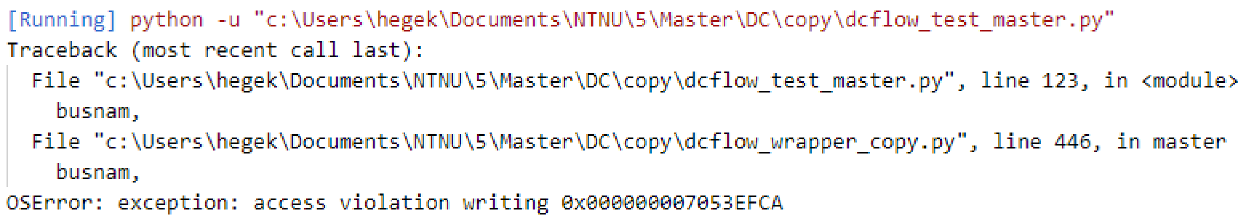


```
[Running] python -u "c:\Users\hegek\Documents\NTNU\5\Master\DC\copy\dcflow_test_secc.py"
Traceback (most recent call last):
  File "c:\Users\hegek\Documents\NTNU\5\Master\DC\copy\dcflow_test_secc.py", line 145, in <module>
    busnam,
  File "c:\Users\hegek\Documents\NTNU\5\Master\DC\copy\dcflow_wrapper_copy.py", line 234, in secc_opt
    busnam,
OSError: exception: access violation writing 0x00007FF83CA66541
***END OF BULILIST***
```

Figure 5.4: Result of testing secc_opt.

5.4.4 Master

The output from the test of the master function is given in Figure 5.5, and shows that it throws the same error as secc_opt. It might suggest that the error lies within the master function. The error can come from the C source code of the master function, the Ctypes interface, or the Python test script. A google search for solutions to the error gave many results, but none that were directly applicable to or could solve the problem. Running new tests with load flow data after the optimization algorithm has been implemented may lead to a solution to the problems encountered here.



```
[Running] python -u "c:\Users\hegek\Documents\NTNU\5\Master\DC\copy\dcflow_test_master.py"
Traceback (most recent call last):
  File "c:\Users\hegek\Documents\NTNU\5\Master\DC\copy\dcflow_test_master.py", line 123, in <module>
    busnam,
  File "c:\Users\hegek\Documents\NTNU\5\Master\DC\copy\dcflow_wrapper_copy.py", line 446, in master
    busnam,
OSError: exception: access violation writing 0x000000007053EFCA
```

Figure 5.5: Result of testing master.

5.5 Open-source

The work done on the toolbox, both from Statnett and in this thesis, have been uploaded to the GitHub repository topflow created by Leif Warland. Per now, the project is private, but it could benefit from the peer-review that comes with being completely open-source in the future.

6 Conclusion

This thesis has worked on modernizing a Fortran program for DC optimal power flow, modernizing meaning translating the Fortran codes to C and creating a new interface for the functions in Python. Python is a language that is easy to use and comes with attributes that are useful when creating tools that can work with new and developing technologies for green and distributed energy solutions. The translated codes have been kept as close to the original Fortran codes as possible, where feasible, to keep the efficiency and functionality of the original codes and prevent additional changes to the program system. The C functions have been interfaced with Python using Ctypes and tested with a Python script that revealed that though two of the tests resulted in errors that were not resolved, most of the functions have been successfully interfaced. The optimization algorithm Lpsolve has been suggested as a replacement for the routines used in the Fortran program. Though not all of the tests were successful, the thesis has put down a foundation on which to build further modernization. The tools and techniques necessary to interface the translated codes to Python have been presented, but there is room to consider other options in the future.

The process of creating DCflow has required a lot of trial and error, and not all issues were resolved. Nonetheless, such a process has resulted in a steep learning curve on a topic that is both important and interesting. How to create libraries in different programming languages and make them available to Python as extension modules is something that, given the qualities of the object-oriented language Python, can be very useful in the future. Working closely with the load flow codes has provided a deeper understanding of how the algorithms work and supplements the theoretical knowledge of power system analyses. Hopefully, the knowledge presented here can be used to complete the toolbox for power system analysis with tools that are designed to handle the future challenges of a green energy system.

7 Further work

This thesis has laid the groundwork for modernizing the toolbox for power system analysis. To complete the DC optimal load flow, the algorithms for optimization and mathematical programming replacing the NAG and HSL routines must be implemented. The optimization solver Lpsolve has been suggested, but other tools might be more suitable. What is important is that the optimization algorithm should be adapted to fit the load flow, not the other way around. This can be achieved by creating an interface for the algorithm that is incorporated into the functions where it is needed. Such a design may be less efficient, but in return it results in a program that is flexible and can be modified with new routines as required. A flexible code should be a goal in a program designed for a changing energy system. It also might be worth to consider a Fortran-C interface that can be interfaced with Python, to avoid the need to translate large program systems. An interface between two languages is a weak spot in the code that can be the source of errors, however translating large program systems can also be troublesome and time-consuming.

More testing of the functions that have been interfaced should be performed, with realistic power system data. The Fortran routine that runs the DC optimal power flow, dc_opt, should be written in Python to define all variables and set up the system needed for the analysis. The remaining subroutines in dc_opt should be interfaced with Python in the same manner as secc_opt. The steps described here will hopefully lead to a finished and functioning DCflow, as a tool in the modernized toolbox for power system analysis.

8 References

[1] D. Beazley, "Automated scientific software programming with swig," *Future Generation Computer Systems*, vol. 19, pp. 599–609, 5 2003.

[2] H. B. Kvandal, "Toolbox for specialized power system analysis," 2018.

[3] Tutorials Point. (2018). Fortran - overview, [Online]. Available: `https://www.tutorialspoint.com/fortran/fortran_overview.htm`. [Accessed: 29.11.2018].

[4] ——, (2018). C language - overview, [Online]. Available: `https://www.tutorialspoint.com/cprogramming/c_overview.htm`. [Accessed: 01.12.2018].

[5] H. P. Langtangen, *Python Scripting for Computational Science*, 3rd ed. Springer-Verlag Berlin Heidelberg, 2008.

[6] Tutorials Point. (2018). Fortran - data types, [Online]. Available: `https://www.tutorialspoint.com/fortran/fortran_data_types.htm`. [Accessed: 27.01.2019].

[7] ——, (2019). C - data types, [Online]. Available: `https://www.tutorialspoint.com/cprogramming/c_data_types.htm`. [Accessed: 02.05.2019].

[8] ——, (2018). Python - variable types, [Online]. Available: `https://www.tutorialspoint.com/python/python_variable_types.htm`. [Accessed: 27.01.2019].

[9] Oracle. (2010). Array indexing and order, [Online]. Available: `https://docs.oracle.com/cd/E19957-01/805-4940/z400091044d0/`. [Accessed: 11.12.2018].

[10] NumPy. (2019). Numpy, [Online]. Available: `http://www.numpy.org/`. [Accessed: 03.03.2019].

[11] W. Savitch, *Absolute C++*, 5th ed. Pearson Education Limited, 2013.

[12] Tutorials Point. (2019). Fortran - quick guide, [Online]. Available: `https://www.tutorialspoint.com/fortran/fortran_quick_guide.htm`. [Accessed: 05.02.2019].

[13] Stanford University. (1996). Fortran procedures and functions, [Online]. Available: `https://web.stanford.edu/class/me200c/tutorial_90/08_subprograms.html`. [Accessed: 05.02.2019].

[14] ——, (1995). Common blocks, [Online]. Available: `https://web.stanford.edu/class/me200c/tutorial_77/13_common.html`. [Accessed: 11.02.2019].

[15] GitHub. (2019). Github.com, [Online]. Available: `https://github.com/`. [Accessed: 30.01.2019].

[16] GitHub Guides. (2016). Hello world, [Online]. Available: `https://guides.github.com/activities/hello-world/`. [Accessed: 30.01.2019].

[17] SCons. (2019). Scons: A software construction tool, [Online]. Available: https://scons.org/. [Accessed: 06.02.2019].

[18] S. Knight. (2019). Scons 3.0.5 user guide, [Online]. Available: https://scons.org/doc/production/PDF/scons-user.pdf. [Accessed: 06.02.2019].

[19] Python Software Foundation. (2019). Ctypes - a foreign function library for python, [Online]. Available: https://docs.python.org/3/library/ctypes.html. [Accessed: 19.03.2019].

[20] C. Robertson. (2019). Exporting from a dll using __declspec(dllexport), [Online]. Available: https://docs.microsoft.com/en-us/cpp/build/exporting-from-a-dll-using-declspec-dllexport?view=vs-2019. [Accessed: 24.04.2019].

[21] The SciPy Community. (2019). C-types foreign function interface (numpy.ctypeslib), [Online]. Available: https://docs.scipy.org/doc/numpy/reference/routines.ctypeslib.html. [Accessed: 04.02.2019].

[22] T. S. Ferguson. (2019). Linear programming: A concise introduction, [Online]. Available: https://www.math.ucla.edu/~tom/LP.pdf. [Accessed: 15.05.2019].

[23] J. D. Glover, *Power system analysis & design*, eng, Boston, Mass, 2017.

[24] O. B. Fosso, "Fast decoupled power flow," 2019.

[25] ——, "Load flow examples," 2019.

[26] ——, "Dc optimal power flow with examples," 2019.

[27] D. Van Hertem, J. Verboomen, K. Purchala, R. Belmans, and W. L. Kling, "Usefulness of dc power flow for active power flow analysis with flow controlling devices," in *The 8th IEE International Conference on AC and DC Power Transmission*, Mar. 2006, pp. 58–62. DOI: 10.1049/cp:20060013.

[28] Microsoft. (2019). Visual studio code, [Online]. Available: https://code.visualstudio.com/. [Accessed: 01.03.2019].

[29] L. Langa. (2019). Black, [Online]. Available: https://github.com/python/black. [Accessed: 07.02.2019].

[30] The Numerical Algorithms Group Ltd. (2015). Nag library routine document x04abf, [Online]. Available: https://www.nag.com/numeric/fl/nagdoc_fl25/html/x04/x04abf.html. [Accessed: 03.04.2019].

[31] ——, (2012). E04 - minimizing or maximizing a function, [Online]. Available: https://www.nag.co.uk/numeric/fl/nagdoc_fl24/html/E04/e04conts.html. [Accessed: 03.04.2019].

[32] Computational Mathematics Group at the STFC Rutherford Appleton Laboratory. (2011). Ma27, [Online]. Available: `http://www.hsl.rl.ac.uk/archive/specs/ma27.pdf`. [Accessed: 03.04.2019].

[33] lpsolve. (2019). Introduction to lp_solve 5.5.2.5, [Online]. Available: `http://lpsolve.sourceforge.net/5.5/`. [Accessed: 04.05.2019].

[34] ——, (2019). Lpsolve ide, [Online]. Available: `http://lpsolve.sourceforge.net/5.5/IDE.htm`. [Accessed: 03.05.2019].

[35] Slashdot Media. (2019). Lpsolve download, [Online]. Available: `https://sourceforge.net/projects/lpsolve/`. [Accessed: 03.05.2019].

[36] lpsolve. (2019). Using lpsolve from python, [Online]. Available: `http://lpsolve.sourceforge.net/5.5/Python.htm`. [Accessed: 03.05.2019].

[37] ——, (2019). Lpsolve api reference, [Online]. Available: `http://lpsolve.sourceforge.net/5.5/lp_solveAPIreference.htm`. [Accessed: 03.05.2019].

[38] ——, (2019). Formulation of an lp problem in lpsolve, [Online]. Available: `http://lpsolve.sourceforge.net/5.5/`. [Accessed: 03.05.2019].

[39] S. Caron. (2011). Windows support: Dynamic loading on windows, [Online]. Available: `https://github.com/imageworks/Field3D/issues/18`. [Accessed: 15.05.2019].

[40] D. A. Wheeler. (). Dynamically loaded (dl) libraries, [Online]. Available: `https://dwheeler.com/program-library/Program-Library-HOWTO/x172.html`. [Accessed: 16.05.2019].

[41] GitHub. (2019). Clp, [Online]. Available: `https://github.com/coin-or/Clp`. [Accessed: 16.05.2019].

[42] Gurobi. (2019). Gurobi.com, [Online]. Available: `http://www.gurobi.com/`. [Accessed: 17.05.2019].

[43] swig.org. (2018). Swig and python, [Online]. Available: `http://www.swig.org/Doc1.3/Python.html#Python_nn2`. [Accessed: 01.11.2018].

[44] T. L. Cottom, "Using swig to bind c++ to python," *Computing in Science & Engineering*, vol. 5, pp. 88–97, 2 2003.

[45] The SciPy Community. (Oct. 5, 2018). Numpy.i: A swig interface file for numpy, [Online]. Available: `https://docs.scipy.org/doc/numpy/reference/swig.interface-file.html`.

A Fortran routines

This section contains the relevant Fortran codes belonging to the DC optimal power flow program in mainenv, and a table with the common block variables in "master.f".

A.1 Common block parameters in master.f

Table A.1: Common block parameters for the master routine. I = integer, R = real. () or ()() preceding the data type indicates an array or matrix of that type.

Name	**Type**	**Description**
param.cmn		
maxbus1	I	Maximum number of buses
maxlin1	I	Maximum number of lines
Parameters for factorization and solution		
licn	I	3(maxbus1+2*maxlin1)
lirn	I	maxbus1+2*maxlin1
Parameters for optimization		
mrowa	I	Maximum number of linear constraints
mvar	I	Maximum number of variables
mrsub	I	Maximum size of subproblem
mctotl	I	mrowa+mvar
liwork	I	8*maxbus1
lwork	I	2*(mrowa+1)^2+4*mrowa+6*mvar+mrowa
Parameters for MA27		
ndev0	I	3*(maxbus1+maxlin1)
liw	I	2*ndev0+3*maxbus1+300
Variables used for output of information		
told	R	Tolerances used for classification
lin	I	Input
lout	I	Output
optim.cmn		
Variables for the master optimization algorithm		
itmax	I	Maximum number of iterations
msglvl	I	Code for output of data
nvar	I	Number of variables on the optimization

nclin	I	Number of linear constraints
nctotl	I	Total number of constraints (nvar + nclin)
mrowa	I	Maximum number of linear constraints
amat	R()()	Matrix containing the constraints
bl	R()	Lower constraints
bu	R()	Upper constraints
cvec	R()	Constraint vector
linobj	L	Logical variable which indicates an available objective
x	R()	Solution vector
istate	I()	Indicates the status of the constraints
objlp	R	Objective value
clamda	R()	Dual variables
scratch.cmn		
Variables for scratch arrays		
iwork	I()	Integer vector, size determined by factorization algorithm
work	R()	Real array, size determined by the master optimization algorithm

A.2 master.f

```
C-----------------------------------------------------------------------C
C LIBRARY: MASTERLIB   ! PROGRAM RESPONSIBLE:           !   Date:      C
C PROG.SYS: SECCON     !          OLAV BJARTE FOSSO     !   10.05.89   C
C-----------------------------------------------------------------------C

      SUBROUTINE MASTER(NBUSES,NGENS,NLINES,IGBUS,PREFFRST,PREFNEXT,
     &                  PREFCOST,PREFMW,PREFTYP,PLOAD,CINCR,PBUY,PSELL,PGEN,
     &                  TETA0,IFROMB,ITOB,XINV,ISTAT,RATEA,BASFLOW,NLCOLD,
     &                  IBUSNO,LMESS,WCONCT,WBUS,AKOEFF,ICKT,SBASE,BUSNAM)

C-----------------------------------------------------------------------C
C DESCRIPTION:                                                          C
C      THE ROUTINE SOLVES THE MASTER PROBLEM IN THE SECURITY            C
C      CONSTRAINED DISPATCH PROBLEM.                                    C
C                                                                       C
C-----------------------------------------------------------------------C

C-----------------------------------------------------------------------C
C  COMMENTS:                                                            C
C                                                                       C
C=======================================================================C
C  NAME  !   TYPE  !  COMMENT                                           C
C-----------------------------------------------------------------------C
C GLOBAL VARIABLES:                                                     C
C                                                                       C
C                                                                       C
C COMMON BLOCKS                                                         C

      INCLUDE '../common/param.cmn'
      INCLUDE '../common/optim.cmn'
      INCLUDE '../common/scratch.cmn'


C-----------------------------------------------------------------------C
C  INPUTDATA:                                                           C
C-----------------------------------------------------------------------C
C  NAME  !   TYPE  !  COMMENT                                           C
C-----------------------------------------------------------------------C
C  NBUSES      I        NUMBER OF BUSES                                 C
C  NGENS       I        NUMBER OF GENERATOR BUSES                       C
C  IGBUS()     I        POINTERS TO GENERATORS                          C
C  PREFFRST()  I        POINTER TO FIRST SEGMENT IN PREF.FUNC.          C
```

```
C  PREFNEXT()  I        POINTER TO NEXT SEGMENT                        C
C  PREFCOST()  R        COST VECTOR FOR THE SEGMENTS                   C
C  PREFMW()    R        MAXIMUM CAPACITY FOR SEGMENTS                  C
C  PREFTYP     I        TYPE OF SEGMENT SELL = 1, BUY = 2, GEN = 3     C
C  PLOAD()     R        LOAD-VECTOR                                    C
C  IFROMB()    R        FROM BUSES FOR BRANCHES                        C
C  ITOB()      R        TO BUSES FOR BRANCHES                          C
C  XINV()      R        INVERSE OF BRANCH REACTANCE                    C
C  ISTAT()     I        STATUS FOR BRANCHES                            C
C  RATEA()     R        THERMAL RATING FOR BRANCH                      C
C                                                                      C
C----------------------------------------------------------------------C

C----------------------------------------------------------------------C
C  OUTPUTDATA:                                                         C
C----------------------------------------------------------------------C
C  NAME  !   TYPE  !  COMMENT                                          C
C----------------------------------------------------------------------C
C  CINCR()     R        INCREMENTAL COST FOR BUSES MODELLED IN OPT     C
C  PBUY()      R        BUYING OF POWER ON BUS                         C
C  PSELL()     R        SELLING                                        C
C  PGEN()      R        GENERATION                                     C
C  TETA0()     R        VOLTAGE ANGLES                                 C
C  BASFLOW()   R        BASE CASE FLOW FOR ALL BRANCHES                C
C                                                                      C
C----------------------------------------------------------------------C

C----------------------------------------------------------------------C
C  LOCAL VARIABLES:                                                    C
C----------------------------------------------------------------------C
C  NAME ! TYPE ! COMMENT                                               C
C----------------------------------------------------------------------C
C TETA0    R8       THE RIGHT HAND SIDE WHEN SOLVING THE AX = B        C
C NCNOW    I        VARIABLE USED FOR NUMBER OF LINEAR CONSTRAINTS     C
C IB       I        VARIABLE USED FOR INTERNAL BUSNUMBERS              C
C IG       I        VARIABLE USED FOR GENERATOR BUSES                  C
C IFB      I        A SPECIFIC FROM-BUS FOR A BRANCH                   C
C ITB      I        A SPECIFIC TO-BUS FOR A BRANCH                     C
C                                                                      C
C----------------------------------------------------------------------C

C----------------------------------------------------------------------C
C          DECLARATION OF MAIN VARIABLES                               C
C----------------------------------------------------------------------C
```

```

      INTEGER   NBUSES             , NGENS          , IGBUS(NBUSES)     ,
     &          PREFFRST(NBUSES)   , PREFNEXT(*)    , PREFTYP(*)        ,
     &          IFROMB(NLINES)     , ITOB(NLINES)   , ISTAT(NLINES)     ,
     &          IBUSNO(NBUSES)     , WCONCT(*)      , WBUS(*)           ,
     &          ICKT(NLINES)
C
      REAL      PREFCOST(*)        , PREFMW(*)      , PLOAD(NBUSES)     ,
     &          CINCR(NBUSES)      , PBUY(NBUSES)   , PSELL(NBUSES)     ,
     &          PGEN(NGENS)        ,                  XINV(NLINES)      ,
     &          RATEA(NLINES)      , BASFLOW(NLINES), AKOEFF(*)         ,
     &          SBASE

      REAL*8    TETA0(NBUSES)

      CHARACTER*8    BUSNAM(NBUSES)


C------------------------------------------------------------------C

C------------------------------------------------------------------C
C          DECLARATION OF LOCAL VARIABLES                          C
C------------------------------------------------------------------C

      INTEGER   NCNOW              , IB                  , IG        ,
     &          IL                 , IFB                 , ITB       ,
     &          ICONP(MROWA)       , IMASTER

      REAL*8    RSIDE(MAXBUS1)

C------------------------------------------------------------------C


C Number of main master problem iterations
      IMASTER = 0

C
C set running conditions for the optimization algorithm
C
      ITMAX = 1000
      MSGLVL = 0
      LINOBJ = .TRUE.

C
```

```
C  UPDATE THE OUTPUT CHANNEL
C

      CALL X04ABF(1,LOUT)

C
C  add the equation     sum pgen   = sum pload   if neccessary
C  set up the cost vector
C  set up limits for the variables
C

      IF (NLCOLD .EQ. 0) THEN
         NCLIN = 0
         CALL BALCON(NVAR,NCLIN,CVEC,MROWA,AMAT,BL,BU,
     &                    PREFFRST,PREFNEXT,PREFCOST,PREFMW,
     &                    NBUSES,IGBUS,PLOAD,AKOEFF)
      ENDIF


      WRITE(LMESS,118)
118   FORMAT(//,T20 ' *****  Solving Master Problem *****',/)


150   CONTINUE

C
C
C
C  find optimum with given constraints
C

      NCNOW = NCLIN
      IFAIL = 1

      NCTOTL = NVAR + NCLIN

      IMASTER = IMASTER + 1
C
      CALL E04MBF(ITMAX,MSGLVL,NVAR,NCLIN,NCTOTL,MROWA,
     &                 AMAT,BL,BU,CVEC,LINOBJ,X,ISTATE,OBJLP,CLAMDA,
     &                 IWORK,LIWORK,WORK,LWORK,IFAIL)
C
      WRITE(LMESS,123) IMASTER, OBJLP*SBASE
123   FORMAT(/,1X,I2,'. Iteration - Power Rescheduling Cost : ',
     &                  F15.3,/)

      IF (IFAIL .EQ. 0) GOTO 250
C
C  update upper and lower constraints  if the problem is infeasible
C  The option is only used when the constraints can not be satiesfies
C  The constraints are updated due to the close to optimal solution
C  returned from the nag-routine
C
      CALL UPCON(NVAR,NCLIN,MROWA,AMAT,BL,BU,X,ISTATE,TOLD)
      WRITE(LMESS,210)
210   FORMAT(/,1X,'Infeasible case - Constraint Relaxation applied',
     &          ' in Master Problem',/)

250   CONTINUE

C
C   update the generation vector
C

      CALL UPDATE(NBUSES,NGENS,CINCR,PBUY,PSELL,PGEN,IGBUS,
     &              PREFFRST,PREFNEXT,PREFTYP,NVAR,CLAMDA,CVEC,X)
C
C Skip the network analysis if NLINES is equal to 0.
C No network involved.
C
      IF (NLINES .LE. 1) GOTO 999
C
C  remove nonbinding constraints in the optimization process
C

      CALL REMOVE(0,NVAR,NCLIN,MROWA,AMAT,BL,BU,ISTATE,ICONP)

C
C  find net injections on the nodes
C

      DO 300 IB = 1,NBUSES
         IG = IGBUS(IB)
         IF (IG .GT. 0) THEN
           TETA0(IB) = PGEN(IG) - PLOAD(IB) - PSELL(IB) + PBUY(IB)
         ELSE
           TETA0(IB) = - PLOAD(IB) - PSELL(IB) + PBUY(IB)
         ENDIF
```

```
300    CONTINUE
C
C  find voltage angles by a forward and a backward substitutions
C

       CALL SOLVB(TETA0,NBUSES)

C
C identify overloads and add constraints
C

       DO 400 IL = 1,NLINES
          IFB = IFROMB(IL)
          ITB = ITOB(IL)
          IF (RATEA(IL) .EQ. 0.0) GOTO 400
          IF ((WBUS(IFB) .EQ. 0) .OR. (WBUS(ITB) .EQ. 0)) GOTO 400
          IF (ISTAT(IL) .EQ. 1) THEN
             BASFLOW(IL) = (TETA0(IFB) - TETA0(ITB))*XINV(IL)
             IF ((ABS(BASFLOW(IL))-RATEA(IL)).GT.TOLD(3)) THEN
                CALL ADDBRC(IL,NCLIN,NVAR,MROWA,AMAT,BL,BU,RSIDE,
     &                       PREFFRST,PREFNEXT,TOLD,
     &                       NBUSES,IFROMB,ITOB,IGBUS,XINV,PLOAD,RATEA)
                WRITE(LMESS,402) IBUSNO(IFB),BUSNAM(IFB),IBUSNO(ITB),
     &                        BUSNAM(ITB), ICKT(IL),
     &                         BASFLOW(IL)*SBASE, RATEA(IL)*SBASE
             ENDIF
          ELSE
             BASFLOW(IL) = 0.0
          ENDIF
400    CONTINUE
402    FORMAT(1X,'Account for  Violation on Line: ',/,
     &     4X,I6,1X,A8,3X,I6,1X,A8,I3,
     &          ' Flow : ',F8.1,'  Rate : ',F8.1)

C
C  new violations encountered?
C

       IF (NCLIN .GT. NCNOW) GOTO 150


451    FORMAT(//,1X,'Active transmission  ',
     &   'constraints after current iteration:',
     &   //,1x, 'From:',T20,'To:',T37,'Ckt_Id',
     &    T47,'Flow  [MW]',T60,'Rate  [MVA]')

      WRITE(LMESS,451)

      DO 450 IL = 1, NLINES
         IFB = IFROMB(IL)
         ITB = ITOB(IL)
         IF (RATEA(IL) .EQ. 0.0) GOTO 450
         IF ((WBUS(IFB) .EQ. 0) .OR. (WBUS(ITB) .EQ. 0)) GOTO 450
         IF (ISTAT(IL) .EQ. 1) THEN
           IF ((ABS(BASFLOW(IL))-RATEA(IL)).GT. -TOLD(3)) THEN
              WRITE(LMESS,452) IBUSNO(IFB),BUSNAM(IFB),IBUSNO(ITB),
     &                        BUSNAM(ITB), ICKT(IL),
     &                        BASFLOW(IL)*SBASE, RATEA(IL)*SBASE
           ENDIF
         ENDIF

450   CONTINUE

452   FORMAT(1X,I6,1X,A8,3X,I6,1X,A8,3X,I3,
     &        4X,F8.1,4X,F8.1)



999   CONTINUE
C
C  update the violation counter
C
      NLCOLD = NCLIN
C
C DUMP INCREMENTAL COST
C
      OPEN(29,FILE='inccost.res',STATUS='UNKNOWN')
      WRITE(29,*) ' OBJECTIVE FUNCTION :  ', OBJLP
      DO 850 IB = 1, NVAR
         WRITE(29,851) IB, CVEC(IB), CLAMDA(IB), X(IB)
850   CONTINUE
851   FORMAT(1X,I4,'  COST :',F8.3,'  LAMBDA :  ', F9.4, '   X :  ', F8.3)
      CLOSE(29)

C
C   END
C

```

```
      END
```

A.3 balcon.f

```
C
C
C  A ROUTINE DOES:
C     1. SETS UP THE BALANCE CONSTRAINT BETWEEN THE LOAD AND GENERATION
C     2. SETS UP THE COST VECTOR FOR THE OPTIMIZATION.
C     3. SETS UP THE VARIABLE LIMITS
C
C

      SUBROUTINE BALCON(NVAR,NCLIN,CVEC,MROWA,AMAT,BL,BU,
     &                   PREFFRST,PREFNEXT,PREFCOST,PREFMW,
     &                   NBUSES,IGBUS,PLOAD,AKOEFF)

C
C  DECLARATIONS
C

      INTEGER*4 NVAR            , NCLIN            , MROWA

      INTEGER   NBUSES          , IGBUS(*)         , PREFFRST(*)      ,
     &          PREFNEXT(*)

      REAL*8    CVEC(*)         , AMAT(MROWA,*)    , BL(*)            ,
     &          BU(*)

      REAL      PREFCOST(*)     , PREFMW(*)        ,
     &          PLOAD(*)        , AKOEFF(*)
C
C
C   LOCAL DECLARATIONS
C
      REAL      TLOAD
      INTEGER NSEG              , IG               , IB

C
C
C
      NVAR = 0
      NCLIN = 1
      TLOAD = 0.0
      ILAST = 0
C
```

```
      DO 200 IB = 1,NBUSES
        TLOAD = TLOAD + PLOAD(IB)
        IG = PREFFRST(IB)
190     IF (IG .EQ. 0) GOTO 200
        ILAST = ILAST + 1
        CVEC(ILAST) = PREFCOST(IG)
        IF (PREFMW(IG) .GT. 0.0) THEN
           AMAT(NCLIN,ILAST) = AKOEFF(IB)
           BL(ILAST)  =  0.0
           BU(ILAST)  =  PREFMW(IG)
        ELSE
           AMAT(NCLIN,ILAST) = 1.0/AKOEFF(IB)
           BU(ILAST)  =  0.0
           BL(ILAST)  =  PREFMW(IG)
        ENDIF
        IG = PREFNEXT(IG)
        GOTO 190
200   CONTINUE
      NVAR = ILAST
      BU(NCLIN+NVAR) = TLOAD
      BL(NCLIN+NVAR) = TLOAD

C
C
C    END
C

      END
```

A.4 upcon.f

```

      SUBROUTINE UPCON(NVAR,NCLIN,MROWA,AMAT,BL,BU,X,ISTATE,TOLD)

C
C

      INTEGER MROWA, NVAR, NCLIN, ISTATE(*)

      REAL*8  AMAT(MROWA,*), BL(*), BU(*), X(*)
      REAL TOLD(*)

      DO 500 IS = 1,NCLIN
        VALUE = 0.0
        IF (ISTATE(NVAR+IS) .GE. 0) GOTO 500
        DO 100 IB = 1,NVAR
          VALUE = VALUE + AMAT(IS,IB)*X(IB)
100     CONTINUE
        IF (ISTATE(NVAR+IS) .EQ. -1) THEN
           RESID = ABS(BU(NVAR+IS)) - ABS(VALUE)
           BU(NVAR+IS) = VALUE + TOLD(2)
        ELSEIF (ISTATE(NVAR+IS) .EQ. -2) THEN
           RESID = ABS(BL(NVAR+IS)) - ABS(VALUE)
           BL(NVAR+IS) = VALUE - TOLD(2)
        ENDIF
500   CONTINUE

C
C    END
C

      END
```

A.5 update.f

```

      SUBROUTINE UPDATE(NBUSES,NGENS,CINCR,PBUY,PSELL,PGEN,IGBUS,
     &                    PREFFRST,PREFNEXT,PREFTYP,NVAR,CLAMDA,CVEC,X)
C
C   the algorithm concludes the result from the optimization
C   program
C
C
C  DEKLARATIONS
C
      INTEGER*4 NVAR
C
      REAL*8    CLAMDA(*)          , CVEC(*)          , X(*)
C
      INTEGER   NBUSES             , NGENS            , IGBUS(NBUSES)     ,
     &          PREFFRST(NBUSES)   , PREFNEXT(*)      , PREFTYP(*)
C
      REAL      CINCR(NBUSES)      , PBUY(NBUSES)     , PSELL(NBUSES)     ,
     &          PGEN(NGENS)

C
C
C  initialize
C
      DO 50 IB = 1,NBUSES
         CINCR(IB) = 0.0
         PBUY(IB) = 0.0
         PSELL(IB) = 0.0
50    CONTINUE
C
      DO 60 IG = 1,NGENS
         PGEN(IG) = 0.0
60    CONTINUE
C
C
C   update all variables
C
      ILAST = 0
      DO 200 IB = 1,NBUSES
         IS = PREFFRST(IB)
190      IF (IS .EQ. 0) GOTO 200
         ILAST = ILAST + 1
```

```
          IF (PREFTYP(IS) .EQ. 1) THEN
             PSELL(IB) = PSELL(IB) - X(ILAST)
          ELSEIF (PREFTYP(IS) .EQ. 2) THEN
             PBUY(IB) = PBUY(IB) + X(ILAST)
          ELSEIF (PREFTYP(IS) .EQ. 3) THEN
             IG = IGBUS(IB)
             PGEN(IG) = PGEN(IG) + X(ILAST)
          ELSE
             WRITE(*,*) ' ***** ERROR IN VECTOR - PREFTYP() ****'
          ENDIF
          CINCR(IB) = CLAMDA(ILAST) - CVEC(ILAST)
          IS = PREFNEXT(IS)
          GOTO 190
200    CONTINUE

       IF (ILAST .NE. NVAR) WRITE(*,*) ' **** ERROR IN UPDATE ****'
C
C   END
C

       END
```

A.6 remove.f

```
C-----------------------------------------------------------------------C
C LIBRARY: MASTERLIB   ! PROGRAM RESPONSIBLE:           !   Date:      C
C PROG.SYS: SECCON     !          OLAV BJARTE FOSSO     !  10.05.89    C
C-----------------------------------------------------------------------C

      SUBROUTINE REMOVE(IP,NVAR,NCLIN,MROWA,AMAT,BL,BU,ISTATE,ICONP)

C-----------------------------------------------------------------------C
C DESCRIPTION:                                                          C
C      THE ROUTINE REMOVES CONSTRAINTS WITH CERTAIN ATTRIBUTES          C
C      FROM THE OPTIMIZATION PROBLEM                                    C
C                                                                       C
C-----------------------------------------------------------------------C

C-----------------------------------------------------------------------C
C  COMMENTS:                                                            C
C                                                                       C
C=======================================================================C
C  NAME  !   TYPE  !  COMMENT                                           C
C-----------------------------------------------------------------------C
C GLOBAL VARIABLES:                                                     C
C                                                                       C
C                                                                       C
C COMMON BLOCKS                                                         C


C-----------------------------------------------------------------------C
C  INPUTDATA:                                                           C
C-----------------------------------------------------------------------C
C  NAME  !   TYPE  !  COMMENT                                           C
C-----------------------------------------------------------------------C
C  NVAR          I         NUMBER OF VARIABLES                          C
C  MROW          I         MAXIMUM NUMBER OF LINEAR CONSTRAINTS         C
C   IP           I         STATUS TYPE TO REMOVE                        C
C  ICONP()       I         CONSTRAINT IDENTIFIER                        C
C                                                                       C
C-----------------------------------------------------------------------C

C-----------------------------------------------------------------------C
C  OUTPUTDATA:    (ALTERED FROM THE INPUTVALUES)                        C
C-----------------------------------------------------------------------C
C  NAME  !   TYPE  !  COMMENT                                           C
```

```
C-----------------------------------------------------------------------C
C NCLIN           I          NUMBER OF LINEAR CONSTRAINTS               C
C ISTATE()        I          VECTOR OF VARIABLE STATUS IN OPT.          C
C AMAT(,)         R8         MATRIX INCLUDING LINEAR CONSTRAINTS        C
C BU()            R8         VECTOR CONTAINING UPPER LIMITS             C
C BL()            R8         VECTOR CONTAINING LOWER LIMITS             C
C ICONP()         I          CONSTRAINT IDENTIFIER                      C
C                                                                       C
C-----------------------------------------------------------------------C

C-----------------------------------------------------------------------C
C  LOCAL VARIABLES:                                                     C
C-----------------------------------------------------------------------C
C  NAME ! TYPE ! COMMENT                                                C
C-----------------------------------------------------------------------C
C IB          I          VARIABLE FOR AN INTERNAL BUSNUMBER             C
C IG          I          VARIABLE FOR A GENERATOR NUMBER                C
C INCLIN      I          VARIABLE INCLUDING THE INITIAL NCLIN           C
C                                                                       C
C-----------------------------------------------------------------------C

C-----------------------------------------------------------------------C
C           DECLARATION OF MAIN VARIABLES                               C
C-----------------------------------------------------------------------C

      INTEGER   NVAR               , NCLIN                , MVAR          ,
     &          ISTATE(*)          , IP                   , ICONP(*)

      REAL*8    AMAT(MROWA,*)      , BL(*)                , BU(*)

C-----------------------------------------------------------------------C

C-----------------------------------------------------------------------C
C           DECLARATION OF LOCAL VARIABLES                              C
C-----------------------------------------------------------------------C

      INTEGER   IB                 , IG                   , INCLIN

C-----------------------------------------------------------------------C

C
C     REMOVE CONSTRAINTS
C
      INCLIN = NCLIN

      DO 500 IB = INCLIN, 1, -1
         IF (ISTATE(IB + NVAR) .NE. IP) GOTO 500
         DO 300 IG = 1,NVAR
            AMAT(IB,IG) = AMAT(NCLIN,IG)
300      CONTINUE
         BU(IB + NVAR) = BU(NCLIN + NVAR)
         BL(IB + NVAR) = BL(NCLIN + NVAR)
         ISTATE(IB + NVAR) = ISTATE(NCLIN + NVAR)
         ICONP(IB) = ICONP(NCLIN)
         NCLIN = NCLIN - 1
500   CONTINUE

C
C
C     END
C

      END
```

A.7 solvb.f

```
C
C
C   ROUTINE FOR SOLVING THE SYSTEM   B*X = RHS
C
C

      SUBROUTINE SOLVB(RHS,NBUSES)

C
C   COMMON BLOCKS
C

      INCLUDE '../common/param.cmn'
      INCLUDE '../common/ma27dp.cmn'


      REAL*8 RHS(*)

C
C
C   SOLVING EQUATION
C

      CALL MA27CD(NBUSES,A,NDEV0,IW,LIW,W,MAXFRT,RHS,IW1,NSTEPS)


C
C   END
C

      END
```

A.8 addbrc.f

```
C---------------------------------------------------------------------C
C LIBRARY: MASTERLIB   ! PROGRAM RESPONSIBLE:          !  Date:      C
C PROG.SYS: SECCON     !          OLAV BJARTE FOSSO    !  10.05.89   C
C---------------------------------------------------------------------C

      SUBROUTINE ADDBRC(IL,NCLIN,NVAR,MROWA,AMAT,BL,BU,SENS,
     &                  PREFFRST,PREFNEXT,TOLD,
     &                  NBUSES,IFROMB,ITOB,IGBUS,XINV,PLOAD,RATEA)
C---------------------------------------------------------------------C
C DESCRIPTION:                                                        C
C      THE ROUTINE ADDS A LINEAR CONSTRAINT TO THE OPTIMIZATION       C
C      PROBLEM FOR A VIOLATED BRANCH.                                 C
C                                                                     C
C                                                                     C
C---------------------------------------------------------------------C

C---------------------------------------------------------------------C
C  COMMENTS:                                                          C
C                                                                     C
C=====================================================================C

C---------------------------------------------------------------------C
C          DECLARATION OF MAIN VARIABLES                              C
C---------------------------------------------------------------------C

      INTEGER*4 NCLIN             , MROWA             , NVAR

      INTEGER   PREFFRST(*)       , PREFNEXT(*)       , NBUSES          ,
     &          IFROMB(*)         , ITOB(*)           , IGBUS(*)        ,
     &          IL

      REAL*8    AMAT(MROWA,*)     , BL(*)             , BU(*)           ,
     &          SENS(*)

      REAL      XINV(*)           , RATEA(*)          , PLOAD(*)        ,
     &          TOLD(*)

C---------------------------------------------------------------------C

C---------------------------------------------------------------------C
C          DECLARATION OF LOCAL VARIABLES                             C
C---------------------------------------------------------------------C
```

```fortran


C-----------------------------------------------------------------------C


C
C
C  CALCULATE THE SENSITIVITY
C

      DO 100 IB = 1, NBUSES
         SENS(IB) = 0.0
100   CONTINUE

      IFR = IFROMB(IL)
      ITB = ITOB(IL)
      SENS(IFR) = XINV(IL)
      SENS(ITB) = - XINV(IL)
      CALL SOLVB(SENS,NBUSES)

C
C
      CONMOD = 0.0
      NCLIN = NCLIN + 1
      ILAST = 0
      DO 200 IB = 1,NBUSES
        CONMOD  = CONMOD + SENS(IB)*PLOAD(IB)
        IG = PREFFRST(IB)
190     IF (IG .EQ. 0) GOTO 200
        ILAST = ILAST + 1
        AMAT(NCLIN,ILAST) = SENS(IB)
        IG = PREFNEXT(IG)
        GOTO 190
200   CONTINUE
C
      IF (ILAST .NE. NVAR) WRITE(*,*) ' ***** ERROR IN ADDBRC *****'

C
C  ENTER UPPER AND LOWER CONSTRAINTS
C
      BL(NVAR + NCLIN) = CONMOD - RATEA(IL) + TOLD(2)
      BU(NVAR + NCLIN) = CONMOD + RATEA(IL) - TOLD(2)

C
```

```
C
C     END
C

      END
```

A.9 bulilist.f

```
C
      SUBROUTINE BULILIST(NBUSES,IBUS,PRICE,MAXMW,SEGTYP,NSEG,
     &                    PREFFRST,PREFNEXT,PREFCOST,PREFMW,PREFTYP)
C
C   THE ROUTINE BUILDS A LINKED LIST FROM INPUT LISTS
C
C
C
C   DECLARATIONS
C
      INTEGER  IBUS(*)          , SEGTYP(*)      , NSEG          ,
     &         PREFFRST(*)      , PREFNEXT(*)    , PREFTYP(*)
C
      REAL     PRICE(*)         , MAXMW(*)       ,
     &         PREFCOST(*)      , PREFMW(*)
C
C   INITIALIZE
C
      DO 50 IS = 1,NBUSES
         PREFFRST(IS) = 0
 50   CONTINUE
C
      DO 60 IS = 1,NSEG
         PREFNEXT(IS) = 0
 60   CONTINUE
C
      IPOS = 0
C
C    BUILD LIST
C
      DO 500 IS = 1,NSEG
         IPOS = IPOS + 1
         IB = IBUS(IS)
         INEXT = PREFFRST(IB)
         IF (INEXT .EQ. 0)   THEN
           PREFFRST(IB) = IPOS
         ELSE
 190     IF (INEXT .EQ. 0) GOTO 200
         IB = INEXT
         INEXT = PREFNEXT(IB)
         GOTO 190
 200     PREFNEXT(IB) = IPOS
```

```
            ENDIF
            PREFMW(IPOS)  = MAXMW(IS)
            PREFCOST(IPOS)  = PRICE(IS)
            PREFTYP(IPOS)   = SEGTYP(IS)
500     CONTINUE
C
C     END
C

        END
```

A.10 initial.f

```
C-----------------------------------------------------------------------C
C LIBRARY: START       ! PROGRAM RESPONSIBLE:         !  Date:      C
C PROG.SYS: SECCON     !        OLAV BJARTE FOSSO     !  14.12.89   C
C-----------------------------------------------------------------------C

      SUBROUTINE INITIAL(LOUT,A,IRN,ICN,NZ,NSTEPS,IKEEP,IFLAG,IW,
     &                   IW1,MAXFRT,W,LIW,LA,
     &                   NBUSES,ISTAT,IFROMB,ITOB,ICKT,XINV,NLINES)

C-----------------------------------------------------------------------C
C DESCRIPTION:                                                          C
C      THE ROUTINE INITIALIZES THE SYSTEM DESCRIPTION AND               C
C      FACTORIZES B                                                    C
C                                                                       C
C-----------------------------------------------------------------------C

C-----------------------------------------------------------------------C
C  COMMENTS:                                                            C
C                                                                       C
C=======================================================================C

C-----------------------------------------------------------------------C
C          DECLARATION OF MAIN VARIABLES                                C
C-----------------------------------------------------------------------C

      INTEGER*2  IRN(LA)      , ICN(LA)             , IW(LIW)    ,
     &           IKEEP(NBUSES*3)
C
      INTEGER    IW1(NBUSES*2)   , NSTEPS             , IFLAG         ,
     &           MAXFRT          , NZ                 , LIW
C
      REAL*8     A(LA)           , W(NBUSES)
C
      INTEGER  NBUSES            , NLINES          , ISTAT(NLINES)     ,
     &         IFROMB(NLINES)    , ITOB(NLINES)    , ICKT(NLINES)
C
      REAL     XINV(NLINES)
C
C-----------------------------------------------------------------------C

C-----------------------------------------------------------------------C
C          DECLARATION OF LOCAL VARIABLES                               C
```

```
C-----------------------------------------------------------------------C


C-----------------------------------------------------------------------C

C
C   d i a g n o s t i c s
C
        COMMON/MA27DD/ U, LP, MP, LDIAG

        REAL*8 U
        INTEGER  LP, MP, LDIAG

        LP = LOUT
        MP = LOUT
        LDIAG = 0

C
C
C   INITIALIZATIONS FOR NAG-FO1BRF
C
C

C
C   BUILD SYSTEM DESCRIPTION
C

        CALL BUILDB(A,IRN,ICN,NZ,NBUSES,
      &               ISTAT,IFROMB,ITOB,ICKT,XINV,NLINES)

C
C
C   SYMBOLIC FACTORIZATION OF THE MATRIX
C
        IFLAG = 0

        CALL MA27AD(NBUSES,NZ,IRN,ICN,IW,LIW,IKEEP,IW1,NSTEPS,IFLAG)

C
C   NUMERICAL FACTORIZATION OF THE MATRIX
C

        CALL MA27BD(NBUSES,NZ,IRN,ICN,A,LA,IW,LIW,IKEEP,NSTEPS,
      &               MAXFRT,IW1,IFLAG)
```

```


C
C     END
C

        END
```

A.11 coeval.f

```
C-----------------------------------------------------------------------C
C LIBRARY: MASTERLIB  ! PROGRAM RESPONSIBLE:          !   Date:     C
C PROG.SYS: SECCON     !          OLAV BJARTE FOSSO   !   09.03.91  C
C-----------------------------------------------------------------------C

      SUBROUTINE COEVAL(NBUSES,NLINES,PREFFRST,PREFNEXT,PINIT,
     &                  IFROMB,ITOB,XINV,ISTAT,BASFLOW,NLCOLD,
     &                  ICKT,IBUSNO,RATEA,LMESS,RMEXT,WCONCT,WBUS,SBASE,
     &                  BUSNAM)
C-----------------------------------------------------------------------C
C DESCRIPTION:                                                          C
C      THE ROUTINE PERFORMS A LINEAR SCREENING OF ALL SINGLE            C
C      BRANCH OUTAGES WITH AN EXEPTION OF THOSE CAUSING                 C
C      SEPARATION                                                       C
C                                                                       C
C-----------------------------------------------------------------------C

C-----------------------------------------------------------------------C
C  COMMENTS:                                                            C
C                                                                       C
C=======================================================================C
C  NAME  !   TYPE  !  COMMENT                                           C
C-----------------------------------------------------------------------C
C GLOBAL VARIABLES:                                                     C
C                                                                       C
C                                                                       C
C COMMON BLOCKS                                                         C

      INCLUDE '../common/param.cmn'
      INCLUDE '../common/contproc.cmn'
      INCLUDE '../common/cominfo.cmn'

      COMMON / REMED / IGCAND(20),  NMOD , ILC
      INTEGER  IGCAND, NMOD, ILC
C-----------------------------------------------------------------------C
C  INPUTDATA:                                                           C
C-----------------------------------------------------------------------C
C  NAME  !   TYPE  !  COMMENT                                           C
C-----------------------------------------------------------------------C
C                                                                       C
C                                                                       C
C                                                                       C
```

```
C                                                                          C
C                                                                          C
C--------------------------------------------------------------------------C

C--------------------------------------------------------------------------C
C   OUTPUTDATA:                                                            C
C--------------------------------------------------------------------------C
C   NAME    !     TYPE    !   COMMENT                                      C
C--------------------------------------------------------------------------C
C                                                                          C
C                                                                          C
C                                                                          C
C--------------------------------------------------------------------------C

C--------------------------------------------------------------------------C
C   LOCAL VARIABLES:                                                       C
C--------------------------------------------------------------------------C
C   NAME  !  TYPE  !  COMMENT                                              C
C--------------------------------------------------------------------------C
C                                                                          C
C                                                                          C
C                                                                          C
C                                                                          C
C--------------------------------------------------------------------------C

C--------------------------------------------------------------------------C
C             DECLARATION OF MAIN VARIABLES                                C
C--------------------------------------------------------------------------C
C
      INTEGER       NBUSES          , NLINES        , PREFFRST(NBUSES)          ,
     &              PREFNEXT(*)     , IFROMB(NLINES)    , ITOB(NLINES)          ,
     &              ISTAT(NLINES)   , ICKT(NLINES) , IBUSNO(NBUSES)             ,
     &              WCONCT(*)       , WBUS(*)
C
      REAL          XINV(NLINES)    , BASFLOW(NLINES)   , PINIT(NBUSES)   ,
     &              RATEA(NLINES)

      CHARACTER*8   BUSNAM(NBUSES)
C--------------------------------------------------------------------------C

C--------------------------------------------------------------------------C
C             DECLARATION OF LOCAL VARIABLES                               C
C--------------------------------------------------------------------------C

```

```
      INTEGER  IBOUT              , IB                  , IWORST      ,
     &         I                  , J                   , K           ,
     &         L                  , IVOVL(30)           , NOVL        ,
     &         ICSEP(MAXBUS1)      , IVRES(30)           , NRES        ,
     &         IDUM

      REAL*8   RHS(MAXBUS1)        , CVEC(MVAR)          , CLAMDA(MCTOTL)

      REAL     POSTFL(MAXLIN1)     , DINV                 , PWORST      ,
     &         DUMOBJ             , PGUSED(30)          , PRMIN(30)   ,
     &         PRMAX(30)          , PCMIN(30)           , PCMAX(30)

C-----------------------------------------------------------------C

      RMULT = RMEXT

C
C INITIALIZE SEPARATION COUNTER
C
      NSEP = 0
C
C DEFINE DUMMY CAPACITY
C
      DO 25 I = 1,ILC
         PRMIN(I) = 20.0
         PRMAX(I) = 20.0
25    CONTINUE
C
C
C  CHECK ALL CONTINGENCIES
C

      DO 500 IBOUT = 1,NLINES
        IF (WCONCT(IBOUT) .EQ. 0) GOTO 500
        IF (ISTAT(IBOUT) .NE. 1) GOTO 500
        IF (ABS(BASFLOW(IBOUT)) .LT. TOLD(2)) GOTO 500

        I = IFROMB(IBOUT)
        J = ITOB(IBOUT)

C
C
C  CALCULATE REQUIRED INVERSE FACTORS
C
```

```

      DO 100 IB = 1,NBUSES
         RHS(IB) = 0.0
100   CONTINUE

      RHS(I) = 1.0
      RHS(J) = - 1.0

      CALL SOLVB(RHS,NBUSES)

C
C   CALCULATE COMPENSATION
C

      DINV = (1.0 - XINV(IBOUT)*(RHS(I) - RHS(J)))/BASFLOW(IBOUT)

      IF (ABS(DINV) .LE. TOLD(1)) THEN
        NSEP = NSEP + 1
        ICSEP(NSEP) = IBOUT
        GOTO 500
      ENDIF
C
C  FIND FLOW
C
      IWORST = 0
      PWORST = 0.0
      POSTFL(IBOUT) = 0.0
      NOVL = 0

      DO 400 IB = 1,NLINES
        IF (ISTAT(IB) .NE. 1) GOTO 400
        IF (IB .EQ. IBOUT) GOTO 400
        K = IFROMB(IB)
        L = ITOB(IB)
        IF ((WBUS(K) .EQ. 0) .OR. (WBUS(L) .EQ. 0)) GOTO 400
C
        POSTFL(IB) = BASFLOW(IB) + XINV(IB)*(RHS(K) - RHS(L))/DINV
        DFLOW = ABS(POSTFL(IB)) - RMULT*RATEA(IB)
        IF (DFLOW .LT. TOLD(3)) GOTO 400
        IF (RATEA(IB) .EQ. 0.0) GOTO 400

C
      POVL = POSTFL(IB)

```

```

C
C
C TERMINATE THE CONTINGENCY ANALYSES IF TOO MANY CONSTRAINTS ARE ADDED
C
      IF (NLCOLD .LT. 50) THEN
         IFR = IFROMB(IBOUT)
         ITR = ITOB(IBOUT)
         WRITE(LMESS,222) IBUSNO(IFR), BUSNAM(IFR), IBUSNO(ITR),
     &  BUSNAM(ITR), ICKT(IBOUT),
     &     IBUSNO(K), BUSNAM(K),  IBUSNO(L), BUSNAM(L), ICKT(IB),
     &     POSTFL(IB)*SBASE, RATEA(IB)*SBASE, RMULT
222   FORMAT(1X,'OUTAGED BRANCH :  ',I6,1X,A8,3X,I6,1X,A8,3X,I3,/,
     & 8X, 'OVERLOAD ON   :  ', I6,1X,A8,3X,I6,1X,A8,I3, 2F8.1,F6.2,/)


         CALL PREVENTIV(IBOUT,IB,RHS,NBUSES,NLINES,IFROMB,ITOB,PINIT,
     &              XINV,POVL,RMULT,RATEA,PREFFRST,PREFNEXT,NLCOLD)
      ELSE
         ISTAT(IBOUT) = 1
         GOTO 999
      ENDIF
C
         NOVL = NOVL + 1
         IVOVL(NOVL) = IB
         IF (DFLOW .GT. PWORST) THEN
            PWORST = DFLOW
            IWORST = IB
         ENDIF
400   CONTINUE

C
C
C
      INS = 1
      IF (INS .EQ. 1) GOTO 500
C
C   SOLVE THE SUBPROBLEM IN ORDER TO FIND BENDERS CUT
C
C
       IF (IWORST .GT. 0) THEN
C           WRITE(LOUT,*) ' '
C           WRITE(LOUT,*) ' SOLVING SUBPROBLEM : '
C           WRITE(LOUT,490) IBUSNO(I), IBUSNO(J), ICKT(IBOUT)
```

```
 490        FORMAT(1X, 'OUTAGE :  ',2I6 ,I4 ,/)
CC
CC  UPDATE THE COMPENSATION INFORMATION
CC
CC
            ISTAT(IBOUT) = 0
            NUMOUT = 1
            MULTOUT(1) = IBOUT
            CIJ(NUMOUT) = 1.0/XINV(IBOUT) - (RHS(I) - RHS(J))
            DO 492 IB = 1,NBUSES
               SENSMAT(IB,NUMOUT) = RHS(IB)
 492        CONTINUE
C
CC
CC  DEFINE REMEDIAL ACTIONS IF PERMITTED
CC
C
C           IF (IREMED(IBOUT) .NE. 1) GOTO 495
C           IF (NMOD .LE. 1) GOTO 495
C           CALL REDREM(IWORST,POSTFL,NUMOUT,IVOVL,NOVL,IVRES,NRES,
C    &         IGCAND,NMOD,ILC,PRMIN,PRMAX,PCMIN,PCMAX,
C    &         ICKT,IBUSNO,XINV,IFROMB,ITOB,RATEA,ISTAT,NBUSES,NLINES)
CC
CC  OUTPUT OF RESCHEDULING UNITS
CC
C         WRITE(LOUT,*) ' RESCHEDULING UNITS'
C         WRITE(LOUT,*) (IBUSNO(IVRES(IS)) , IS = 1,NRES)
C
495         CONTINUE
C
CC
CC  SOLVE THE SUBPROBLEM
CC
C           CALL   SOLVSUB(IWORST,POSTFL,NUMOUT,NRES,IVRES,
C     &                     PGUSED,CVEC,CLAMDA,DUMOBJ,IDUM,
C     &                     PCMIN,PCMAX,PREFFRST,ICKT,IBUSNO,XINV,
C     &                     IFROMB,ITOB,RATEA,ISTAT,NBUSES,NLINES)
C
CC
CC UPDATE THE POST CONTINGENCY RESCHEDULING
C
CC
            IP = IFIRST(IBOUT) - 1
            DO 510 IS = 1,NRES
```

```
                 IRESCH(IP+IS) = IVRES(IS)
                 PFUSED(IP+IS) = PGUSED(IS)
  510          CONTINUE
               ILAST(IBOUT) = IP + NRES
C
CC
CC    FORM AND ADD BENDERS CUT
CC
C
           IF (IDUM .EQ. 1) THEN
              CALL ADDCUT(CVEC,CLAMDA,DUMOBJ,PREFFRST,PREFNEXT,NLCOLD,
     &                         NBUSES)
           ENDIF
C
           ENDIF
C
C
C
C RESTORE
C
               ISTAT(IBOUT) = 1

 500     CONTINUE

 999     CONTINUE


C
C   END
C

         END
```

A.12 secc_opt.f

```

      SUBROUTINE SECC_OPT(LOUT,NBUSES,NGENS,NLINES,IGBUS,PLOAD,PGEN,
     &           PSELL,PBUY,IFROMB,ITOB,ICKT,XINV,ISTAT,RATEA,IBUSNO,
     &           IBUS,PRICE,MAXMW,SEGTYP,NSEG,LMESS,RMEXT,
     &           PREFFRST,PREFNEXT,PREFCOST,PREFMW,PREFTYP,
     &           TETA0,BASFLOW,CINCR,ICONT,NICONT,IBUILD,
     &           A,IRN,ICN,NZ,NSTEPS,IKEEP,IFLAG,IW,IW1,MAXFRT,W,LIW,LA,
     &           WCONCT,WBUS,AKOEFF,SBASE,BUSNAM)

C   security constrained dispatch
C
C
C  DECLARATIONS
C
      INTEGER  LOUT              , NBUSES         , NGENS          ,
     &         NLINES            , IGBUS(NBUSES)  , IFROMB(NLINES) ,
     &         ITOB(NLINES)      , ICKT(NLINES)   , ISTAT(NLINES)  ,
     &         IBUS(NSEG)        , SEGTYP(NSEG)   , NSEG           ,
     &         PREFFRST(NBUSES)  , PREFNEXT(*)    , PREFTYP(*)     ,
     &         ICONT             , IBUILD         , IBUSNO(NBUSES) ,
     &         WCONCT(*)         , WBUS(*)
C
      REAL     PLOAD(NBUSES)     , PGEN(NGENS)    , PSELL(NBUSES)  ,
     &         PBUY(NBUSES)      , XINV(NLINES)   , RATEA(NLINES)  ,
     &         PRICE(NSEG)       , MAXMW(NSEG)    , PREFCOST(*)    ,
     &         PREFMW(*)         , BASFLOW(NLINES), CINCR(NBUSES)  ,
     &         AKOEFF(*)         , SBASE
C
      REAL*8   TETA0(NBUSES)

      CHARACTER*8    BUSNAM(NBUSES)
C
C  VARIABLES FOR THE HARWELL MA27 ROUTINE
C

      INTEGER*2  IRN(LA)         , ICN(LA)             , IW(LIW)        ,
     &           IKEEP(3*NBUSES)
      INTEGER    IW1(2*NBUSES)   , NSTEPS              , IFLAG           ,
     &           MAXFRT          , NZ                  , LIW             ,
     &           LA
      REAL*8     A(LA)           , W(NBUSES)
C
```

```


10      CONTINUE
C
C  build linked list of the variable description
C
C  All variables are ordered in a linked list to an appropriate
C  description for each bus.
C

        CALL BULILIST(NBUSES,IBUS,PRICE,MAXMW,SEGTYP,NSEG,
     &      PREFFRST,PREFNEXT,PREFCOST,PREFMW,PREFTYP)
C
C  build the system description  and factorize
C
C  If NLINES is less or equal to 1 no network is involved and no network
C  analysis is performed.
C  If IBUILD is 0 an adequate description is already made.
C
        IF (NLINES .LE. 1) GOTO 100
        IF (IBUILD .EQ. 0) GOTO 100
        CALL INITIAL(LOUT,A,IRN,ICN,NZ,NSTEPS,IKEEP,IFLAG,IW,
     &                     IW1,MAXFRT,W,LIW,LA,
     &                     NBUSES,ISTAT,IFROMB,ITOB,ICKT,XINV,NLINES)
C

100     CONTINUE
C
        IK = 0
        NLCOLD = 0

C
C
200     CONTINUE
C
C
C  Solve the main problem without any contingency analysis.
C
C  Base Case
C
        CALL MASTER(NBUSES,NGENS,NLINES,IGBUS,PREFFRST,PREFNEXT,PREFCOST,
     &                 PREFMW,PREFTYP,PLOAD,CINCR,PBUY,PSELL,PGEN,
     &                 TETA0,IFROMB,ITOB,XINV,ISTAT,RATEA,BASFLOW,NLCOLD,
     &                 IBUSNO,LMESS,WCONCT,WBUS,AKOEFF,ICKT,SBASE,BUSNAM)
```

```
C
C  skip the contingency analysis
C

      IF (ICONT .EQ. 0) GOTO 999
      IF (NLINES .LE. 1) GOTO 999
C
C  contingency evaluation and constraint generation
C
      IK = IK + 1

      WRITE(LMESS,300) IK
300   FORMAT(/,1X, 'CONTINGENCY EVALUATION NUMBER: ', I3,/)
      IF (IK .GT. NICONT) GOTO 999

      NCNOW = NLCOLD

      CALL COEVAL(NBUSES,NLINES,PREFFRST,PREFNEXT,PLOAD,
     &            IFROMB,ITOB,XINV,ISTAT,BASFLOW,NLCOLD,
     &            ICKT,IBUSNO,RATEA,LMESS,RMEXT,WCONCT,WBUS,SBASE,
     &            BUSNAM)

      IF (NLCOLD .GT. NCNOW) GOTO 200


999   CONTINUE
C
C   END
C

      END
```

B C routines

This section contains the relevant C codes belonging to the DC optimal power flow program DCflow, translated from the corresponding Fortran codes given in Appendix A.

B.1 topflow.h

```c
#ifndef HEADER_nettanalyse
#define HEADER_nettanalyse

#define PQ_BUS 1
#define PV_BUS 2
#define SWING_BUS   3
#define DISCONNECTED_BUS 4

#define ABS(x) (((x) < 0) ? (-(x)) : ((x)))
#define CONNECTED(i, con) (con[i] < 0 ? i : con[i])

#endif
```

B.2 dcflow.h

```c
/* Headerfile "dcflow.h" */

__declspec(dllexport) void master(int nbuses, int ngens, int nlines, int *
    igbus, int *preffrst, int *prefnext, double *prefcost, double *prefmw,
    int *preftyp, double *pload, double *cincr, double *pbuy, double *psell,
     double *pgen, double *teta0, int *ifromb, int *itob, double *xinv, int
    *istat, double *ratea, double *basflow, int nlcold, int *ibusno, int *
    wconct, int *wbus, double *akoeff, int *ickt, double sbase, int *busnam)
    ;

__declspec(dllexport) void bulilist(int nbuses, int *ibus, double *price,
    double *maxmw, int *segtyp, int nseg, int *preffrst, int *prefnext,
    double *prefcost, double *prefmw, int *preftyp);

__declspec(dllexport) void addbrc(int il, int nclin, int nvar, int mrowa,
    double **amat, double *bl, double *bu, double *sens, int *preffrst, int
    *prefnext, double *told, int nbuses, int *ifromb, int *itob, int *igbus,
     double *xinv, double *pload, double *ratea);

__declspec(dllexport) void balcon(int nvar, int nclin, double *cvec, int
    mrowa, double **amat, double *bl, double *bu, int *preffrst, int *
    prefnext, double *prefcost, double *prefmw, int nbuses, int *igbus,
    double *pload, double *akoeff);

__declspec(dllexport) void remove(int ip, int nvar, int nclin, int mrowa,
    double **amat, double *bl, double *bu, int *istate, int *iconp);

__declspec(dllexport) void upcon(int nvar, int nclin, int mrowa, double **
    amat, double *bl, double *bu, double *x, int *istate, double *told);

__declspec(dllexport) void update(int nbuses, int ngens, double *cincr,
    double *pbuy, double *psell, double *pgen, int *igbus, int *preffrst,
    int *prefnext, int *preftyp, int nvar, double *clamda, double *cvec,
    double *x);
```

B.3 master.c

```
__declspec(dllexport) void master(int nbuses, int ngens, int nlines, int *
    igbus, int *preffrst, int *prefnext, double *prefcost, double *prefmw,
    int *preftyp, double *pload, double *cincr, double *pbuy, double *psell,
     double *pgen, double *teta0, int *ifromb, int *itob, double *xinv, int
    *istat, double *ratea, double *basflow, int nlcold, int *ibusno, int *
    wconct, int *wbus, double *akoeff, int *ickt, double sbase, int *busnam)
{

#include "topflow.h"

     /* The routine solves the master problem in the security constrained
    dispatch problem */

     /* VARIABLE DECLARATIONS */
     /* Global variables from commonblocks: */

     int maxbus1, maclin2, licn, lirn, mrowa, mvar, mrsub, ndev0, liw, lin,
    lout, itmax, msglvl, nvar, nclin, nctotl, linobj, ifail, *istate, *iwork
    ;

     double told, objlp, **amat, *bl, *bu, *cvec, *x, *clamda, *work;

     /* Declaration of local variables: */
     int ncnow, ib, ig, il, ifb, itb, *iconp, imaster;

     double *rside;

     // Number of main master problem iterations
     imaster = 0;

     // Set up the balance constraint, cost vector and variable limits
     if (nlcold == 0)
     {
          nclin = 0;
          balcon(nvar, nclin, cvec, mrowa, amat, bl, bu, preffrst, prefnext,
    prefcost, prefmw, nbuses, igbus, pload, akoeff);
     }

     // Skip the network analysis if NLINES is equal to 0. No network
    involved.
     if (nlines > 1)
     {
```

```

    printf("Solving Master Problem");

    do
    {

        // Find optimum with given constraints
        ncnow = nclin;
        ifail = 1;

        nctotl = nvar + nclin;

        imaster = imaster + 1;

        // Optimization algorithm
        // E04MBF(itmax, msglvl, nvar, nclin, nctotl, mrowa, amat, bl,
 bu, cvec, linobj, x, istate, objlp, clamda, iwork, liwork, work, lwork,
 ifail);

        printf("Iteration: %d. Power rescheduling cost: %f", imaster,
 objlp * sbase);

        /* Update upper and lower constraints if the problem is
 infeasible. Th option is only used when the constraints cannot be
 satisfied. The constraints are updated due to the close to optimal
 solution returned from the nag-routine. */
        if (ifail != 0)
        {
            upcon(nvar, nclin, mrowa, amat, bl, bu, x, istate, told);
            printf("Infeasible case - Constraint Relaxation applied in
 master problem");
        }

        // Update the generation vector
        update(nbuses, ngens, cincr, pbuy, psell, pgen, igbus, preffrst
 , prefnext, preftyp, nvar, clamda, cvec, x);

        // Remove nonbinding constraints in the optimization process
        remove(0, nvar, nclin, mrowa, amat, bl, bu, istate, iconp);

        // Find net injections on the nodes
        for (ib = 0; ib < nbuses; ib++)
        {
            ig = igbus[ib];
```

```
                if (ig > 0)
                {
                    teta0[ib] = pgen[ig] - pload[ib] - psell[ib] + pbuy[ib
];
                }
                else
                {
                    teta0[ib] = -pload[ib] - psell[ib] + pbuy[ib];
                }
            }

            // Find voltage angles by a forward and a backward sunstitution
            // SOLVB(teta0, nbuses);

            // Identify overloads and add constraints
            for (il = 0; il < nbuses; il++)
            {
                ifb = ifromb[il];
                itb = itob[il];
                if (ratea[il] == 0.0)
                {
                    continue;
                }
                if (wbus[ifb] == 0 || wbus[itb] == 0)
                {
                    continue;
                }
                if (istat[il] == 1)
                {
                    basflow[il] = (teta0[ifb] - teta0[itb]) * xinv[il];
                    if ((ABS(basflow[il] - ratea[il])) > told)
                    {
                        addbrc(il, nclin, nvar, mrowa, amat, bl, bu, rside,
 preffrst, prefnext, told, nbuses, ifromb, itob, igbus, xinv, pload,
ratea);

                        printf("Active transmission constraints after
current iteration: \n From: %d %c to: %d %c, Ckt_Id: %d, Line flow: %f,
Thermal rating: %f", ibusno[ifb], busnam[ifb], ibusno[itb], busnam[itb],
 ickt[il], basflow[il] * sbase, ratea[il] * sbase);
                    }
                }
                else
                {
```

```c
                    basflow[il] = 0.0;
                }
            }

        } while (nclin > ncnow);

        for (il = 0; il < nlines; il++)
        {
            ifb = ifromb[il];
            itb = itob[il];
            if (ratea[il] == 0.0)
            {
                continue;
            }
            if (wbus[ifb] == 0 || wbus[itb] == 0)
            {
                continue;
            }
            if (istat[il] == 1)
            {
                if ((ABS(basflow[il] - ratea[il]) > -told))
                {
                    printf("Active transmission constraints after current iteration: \n From: %d %c to: %d %c, Ckt_Id: %d, Line flow: %f, Thermal rating: %f", ibusno[ifb], busnam[ifb], ibusno[itb], busnam[itb], ickt[il], basflow[il] * sbase, ratea[il] * sbase);
                }
            }
        }
    } //End if no network

    // Update the violation counter
    nlcold = nclin;

    // Dump incremental cost

} // End master
```

B.4 balcon.c

```
/* This routine:
    1. Sets up the balance constraint between the load and generation
    2. Sets up the cost vector for the optimization
    3. Sets up the variable limits
*/

__declspec(dllexport) void balcon(int nvar, int nclin, double *cvec, int
   mrowa, double **amat, double *bl, double *bu, int *preffrst, int *
   prefnext, double *prefcost, double *prefmw, int nbuses, int *igbus,
   double *pload, double *akoeff)
{

    /* Local variables */
    double tload;
    int nseg, ig, ib, ilast;

    nvar = 0;
    nclin = 1;
    tload = 0.0;
    ilast = 0;

    for (ib = 0; ib < nbuses; ib++)
    {
        tload = tload + pload[ib];
        ig = preffrst[ib];
        while (ig != 0)
        {
            ilast = ilast + 1;
            cvec[ilast] = prefcost[ig];
            if (prefmw[ig] > 0.0)
            {
                amat[nclin][ilast] = akoeff[ib];
                bl[ilast] = 0.0;
                bu[ilast] = prefmw[ig];
            }
            else
            {
                amat[nclin][ilast] = 1.0 / akoeff[ib];
                bu[ilast] = 0.0;
                bl[ilast] = prefmw[ig];
            }
            ig = prefnext[ig];
            }
        }

        nvar = ilast;
        bu[nclin + nvar] = tload;
        bl[nclin + nvar] = tload;

} // End balcon
```

B.5 upcon.c

```c
/* Update upper and lower constraints if the problem is infeasible. The
    option is only used when the constraints con not be satisfied. The
    constranits are updated due to the close to optimal solution returned
    from the nag-routine. */
#include "topflow.h"

__declspec(dllexport) void upcon(int nvar, int nclin, int mrowa, double **
    amat, double *bl, double *bu, double *x, int *istate, double *told)
{
     double value, resid;

     for (int is = 0; is < nclin; is++)
     {
          value = 0.0;
          if (istate[nvar + is] >= 0)
          {
               continue;
          }
          for (int ib = 0; ib < nvar; ib++)
          {
               value = value + amat[is][ib] * x[ib];
          }
          if (istate[nvar + is] == -1)
          {
               resid = ABS(bu[nvar + is]) - ABS(value);
               bu[nvar + is] = value + told[2];
          }
          else if (istate[nvar + is] == -2)
          {
               resid = ABS(bl[nvar + is]) - ABS(value);
               bl[nvar + is] = value - told[2];
          }
     }

} // End upcon
```

B.6 update.c

```c
__declspec(dllexport) void update(int nbuses, int ngens, double *cincr,
    double *pbuy, double *psell, double *pgen, int *igbus, int *preffrst,
    int *prefnext, int *preftyp, int nvar, double *clamda, double *cvec,
    double *x)
{

    /* The algorithm concludes the result from the optimization program */

    // Initialize generation vectors
    for (int ib = 0; ib < nbuses; ib++)
    {
        cincr[ib] = 0.0;
        pbuy[ib] = 0.0;
        psell[ib] = 0.0;
    }

    for (int ig = 0; ig < ngens; ig++)
    {
        pgen[ig] = 0.0;
    }

    // Update all variables
    int ilast = 0;
    for (int ib = 0; ib < nbuses; ib++)
    {
        int is = preffrst[ib];
        while (is != 0)
        {
            ilast = ilast + 1;
            if (preftyp[is] == 1)
            {
                psell[ib] = psell[ib] - x[ilast];
            }
            else if (preftyp[is] == 2)
            {
                pbuy[ib] = pbuy[ib] + x[ilast];
            }
            else if (preftyp[is] == 3)
            {
                ig = igbus[ib];
                pgen[ig] = pgen[ig] + x[ilast];
            }
            else
            {
                printf("Error in vector - preftyp");
            }

            cincr[ib] = clamda[ilast] - cvec[ilast];
            is = prefnext[is];
        }
    }

    if (ilast != nvar)
    {
        printf("*** ERROR IN UPDATE ***");
    }
}

// End update
```

B.7 remove.c

```c
/* The routine removes constraints with certaon attributes from
the optimization problem */

__declspec(dllexport) void remove(int ip, int nvar, int nclin, int mrowa,
    double **amat,
                                            double *bl, double *bu, int *istate, int
    *iconp)
{
    // Local variables
    int ib, ig, inclin;

    // Remove constraints
    inclin = nclin;

    for (ib = inclin - 1; ib = 0; ib--)
    {
        if (istate[ib + nvar] != ip)
        {
            continue;
        }

        for (ig = 1; ig < nvar; ig++)
        {
            amat[ib][ig] = amat[nclin][ig];
        }

        bu[ib + nvar] = bu[nclin + nvar];
        bl[ib + nvar] = bl[nclin + nvar];
        istate[ib + nvar] = istate[nclin + nvar];
        iconp[ib] = iconp[nclin];
        nclin = nclin - 1;
    }

} // End remove
```

B.8 addbrc.c

```c
/* The routine adds a linear constraint to the optimization problem for a
    violated branch */

__declspec(dllexport) void addbrc(int il, int nclin, int nvar, int mrowa,
    double **amat, double *bl, double *bu, double *sens, int *preffrst, int
    *prefnext, double *told, int nbuses, int *ifromb, int *itob, int *igbus,
     double *xinv, double *pload, double *ratea)
{

    // Local variables
    double conmod;
    int ifr, itb, ig, ilast;

    // Calculate the sensitivity
    for (int ib = 0; ib < nbuses; ib++)
    {
        sens[ib] = 0.0;
    }

    ifr = ifromb[il];
    itb = itob[il];
    sens[ifr] = xinv[il];
    sens[itb] = -xinv[il];

    //CALL SOLVB(sens, nbuses);

    conmod = 0.0;
    nclin = nclin + 1;
    ilast = 0;

    for (int ib = 0; ib < nbuses; ib++)
    {
        conmod = conmod + sens[ib] * pload[ib];
        ig = preffrst[ib];

        while (ig != 0)
        {
            ilast = ilast + 1;
            amat[nclin][ilast] = sens[ib];
            ig = prefnext[ig];
        }
    }

    if (ilast != nvar)
    {
        printf("***** ERROR IN ADDBRC *****");
    }

    // Enter upper and lower constraints
    bl[nvar + nclin] = conmod - ratea[il] + told[2];
    bu[nvar + nclin] = conmod + ratea[il] - told[2];

} // End addbrc
```

B.9 bulilist.c

```c
/* The routine builds a linked list from input lists */

__declspec(dllexport) void bulilist(int nbuses, int *ibus, double *price,
    double *maxmw, int *segtyp, int nseg, int *preffrst, int *prefnext,
    double *prefcost, double *prefmw, int *preftyp)
{
    // Local variables
    int is, ib;

    // Initialise
    for (is = 0; is < nbuses; is++)
    {
        preffrst[is] = 0;
    }

    for (is = 0; is < nseg; is++)
    {
        prefnext[is] = 0;
    }

    int ipos = -1, inext;

    // Build List
    for (is = 0; is < nseg; is++)
    {
        ipos = ipos + 1;
        ib = ibus[is];
        inext = preffrst[ib];
        if (inext == 0)
        {
            preffrst[ib] = ipos;
        }
        else
        {
            while (inext != 0)
            {
                ib = inext;
                inext = prefnext[ib];
            }
            prefnext[ib] = ipos;
        }
        prefmw[ipos] = maxmw[is];
```

```
            prefcost[ipos] = price[is];
            preftyp[ipos] = segtyp[is];
      }

} // End bullilist
```

B.10 secc_opt.c

```c

#include "topflow.h"
#include "dcflow.h"

__declspec(dllexport) void secc_opt(int lout, int nbuses, int ngens, int
    nlines, int *igbus, double *pload, double *pgen, double *psell, double *
    pbuy, int *ifromb, int *itob, int *ickt, double *xinv, int *istat,
    double *ratea, int *ibusno, int *ibus, double *price, double *maxmw, int
     *segtyp, int nseg, int lmess, int rmext, int *preffrst, int *prefnext,
    double *prefcost, double *prefmw, int *preftyp, double *teta0, double *
    basflow, double *cincr, int icont, int nicont, int ibuild, double *a,
    int *irn, int *icn, int nz, int nsteps, int *ikeep, int iflag, int *iw,
    int *iw1, int maxfrt, double *w, int la, int *wconct, int *wbus, double
    *akoeff, double sbase, int *busnam)
{
    /* Security constrained dispatch */

    // Build linked list of the variable description
    // All variables are ordered in a linked list to an approriate
    description for each bus

    bulilist(nbuses, ibus, price, maxmw, segtyp, nseg, preffrst, prefnext,
    prefcost, prefmw, preftyp);

    // Build the system description and factorize
    // If "nlines" is less or equal to 1 no network is involved and no
    network analysis is performed
    // If "ibuild" is 0 and adequate description is already made

    if (nlines > 1 && ibuild != 0)
    {
        //initial(lout, a, irn, icn, nz, nsteps, ikeep, iflag, iw, iw1,
    maxfrt, w, liw, la, nbuses, istat, ifromb, itob, ickt, xinv, nlines);
    }

    int ik = 0, ncnow;
    int nlcold = 0;

    // Security constrained dispatch
    do
    {
        // Solve the main problem without any contingency analysis
        master(nbuses, ngens, nlines, igbus, preffrst, prefnext, prefcost,
    prefmw, preftyp, pload, cincr, pbuy, psell, pgen, teta0, ifromb, itob,
    xinv, istat, ratea, basflow, nlcold, ibusno, wconct, wbus, akoeff, ickt,
     sbase, busnam);

        // Skip the analysis of no network involved:
        if (icont != 0 && nlines > 1)
        {
            break;
        }

        // Contingency evaluation and constraint generation
        ik = ik + 1;
        printf("Contingency evaluation number: ", ik);

        if ((icont != 0) && (nlines > 1) && (ik <= nicont))
        {
            ncnow = nlcold;

            // coeval(nbuses, nlines, preffrst, prefnext, pload, ifromb,
    itob, xinv, istat, basflow, nlcold, ickt, ibusno, ratea, lmess, rmext,
    wconct, wbus, sbase, busnam);

        } //if

    } while (nlcold > ncnow);

} // END secc_opt
```

C Ctypes wrapper

This section contains the Ctypes wrapper file for the C functions in DCflow.

C.1 dcflow_wrapper.py

```python
import numpy as np
import numpy.ctypeslib as npct
import ctypes
from ctypes import c_int, c_bool, c_double, c_char

# Input types
ar_1d_double = npct.ndpointer(dtype=np.double, ndim=1, flags="CONTIGUOUS")
ar_1d_int = npct.ndpointer(dtype=np.int, ndim=1, flags="CONTIGUOUS")
ar_1d_bool = npct.ndpointer(dtype=np.bool, ndim=1, flags="CONTIGUOUS")
ar_2d_double = npct.ndpointer(dtype=np.double, ndim=2, flags="CONTIGUOUS")

# location
# get the correct location of the source files in folder master/DC/src
# load the library: dclib
dclib = ctypes.cdll.LoadLibrary(
    "c:\\Users\\hegek\\Documents\\NTNU\\5\\Master\\DC\\src\\dcflow.dll"
)

# restype og argtypes
dclib.secc_opt.restype = None
dclib.secc_opt.argtypes = (
    [c_int] * 4
    + [ar_1d_int]
    + [ar_1d_double] * 4
    + [ar_1d_int] * 3
    + [ar_1d_double]
    + [ar_1d_int]
    + [ar_1d_double]
    + [ar_1d_int] * 2
    + [ar_1d_double] * 2
    + [ar_1d_int]
    + [c_int] * 3
    + [ar_1d_int] * 2
    + [ar_1d_double] * 2
    + [ar_1d_int]
    + [ar_1d_double] * 3
    + [c_int] * 3
    + [ar_1d_double]
    + [ar_1d_int] * 2
    + [c_int] * 2
    + [ar_1d_int]
    + [c_int]
    + [ar_1d_int] * 2
    + [c_int]
    + [ar_1d_double]
    + [c_int]
    + [ar_1d_int] * 2
    + [ar_1d_double]
    + [c_double]
    + [ar_1d_int]
)

dclib.addbrc.restype = None
dclib.addbrc.argtypes = (
    [c_int] * 4
    + [ar_2d_double]
    + [ar_1d_double] * 3
    + [ar_1d_int] * 2
    + [ar_1d_double]
    + [c_int]
    + [ar_1d_int] * 3
    + [ar_1d_double] * 3
)

dclib.balcon.restype = None
dclib.balcon.argtypes = (
    [c_int] * 2
    + [ar_1d_double]
    + [c_int] * 1
    + [ar_2d_double]
    + [ar_1d_double] * 2
    + [ar_1d_int] * 2
    + [ar_1d_double] * 2
    + [c_int]
    + [ar_1d_int]
    + [ar_1d_double] * 2
)

dclib.remove.restype = None
dclib.remove.argtypes = (
    [c_int] * 4 + [ar_2d_double] + [ar_1d_double] * 2 + [ar_1d_int] * 2
```

```
)

dclib.upcon.restype = None
dclib.upcon.argtypes = (
    [c_int] * 3 + [ar_2d_double] + [ar_1d_double] * 3 + [ar_1d_int] + [
    ar_1d_double]
)

dclib.update.restype = None
dclib.update.argtypes = (
    [c_int] * 2 + [ar_1d_double] * 4 + [ar_1d_int] * 4 + [c_int] + [
    ar_1d_double] * 3
)

dclib.bulilist.restype = None
dclib.bulilist.argtypes = (
    [c_int]
    + [ar_1d_int]
    + [ar_1d_double] * 2
    + [ar_1d_int]
    + [c_int]
    + [ar_1d_int] * 2
    + [ar_1d_double] * 2
    + [ar_1d_int]
)

dclib.master.restype = None
dclib.master.argtypes = (
    [c_int] * 3
    + [ar_1d_int] * 3
    + [ar_1d_double] * 2
    + [ar_1d_int]
    + [ar_1d_double] * 6
    + [ar_1d_int] * 2
    + [ar_1d_double]
    + [ar_1d_int]
    + [ar_1d_double] * 2
    + [c_int]
    + [ar_1d_int] * 3
    + [ar_1d_double]
    + [ar_1d_int]
    + [c_double]
    + [ar_1d_int]
)
```

```


# function declaration
def secc_opt(
    lout ,
    nbuses ,
    ngens ,
    nlines ,
    igbus ,
    pload ,
    pgen ,
    psell ,
    pbuy ,
    ifromb ,
    itob ,
    ickt ,
    xinv ,
    istat ,
    ratea ,
    ibusno ,
    ibus ,
    price ,
    maxmw,
    segtyp ,
    nseg ,
    lmess ,
    rmext ,
    preffrst ,
    prefnext ,
    prefcost ,
    prefmw ,
    preftyp ,
    teta0 ,
    basflow ,
    cincr ,
    icont ,
    nicont ,
    ibuild ,
    a ,
    irn ,
    icn ,
    nz ,
    nsteps ,
    ikeep ,
    iflag,
    iw,
    iw1,
    maxfrt,
    w,
    la,
    wconct,
    wbus,
    akoeff,
    sbase,
    busnam,
):
    dclib.secc_opt(
        lout,
        nbuses,
        ngens,
        nlines,
        igbus,
        pload,
        pgen,
        psell,
        pbuy,
        ifromb,
        itob,
        ickt,
        xinv,
        istat,
        ratea,
        ibusno,
        ibus,
        price,
        maxmw,
        segtyp,
        nseg,
        lmess,
        rmext,
        preffrst,
        prefnext,
        prefcost,
        prefmw,
        preftyp,
        teta0,
        basflow,
        cincr,
        icont ,
        nicont ,
        ibuild ,
        a ,
        irn ,
        icn ,
        nz ,
        nsteps ,
        ikeep ,
        iflag ,
        iw ,
        iw1 ,
        maxfrt ,
        w,
        la ,
        wconct ,
        wbus ,
        akoeff ,
        sbase ,
        busnam ,
    )


def addbrc(
    il ,
    nclin ,
    nvar ,
    mrowa ,
    amat ,
    bl ,
    bu ,
    sens ,
    preffrst ,
    prefnext ,
    told ,
    nbuses ,
    ifromb ,
    itob ,
    igbus ,
    xinv ,
    pload ,
    ratea ,
):
    dclib . addbrc (
        il,
        nclin,
        nvar,
        mrowa,
        amat,
        bl,
        bu,
        sens,
        preffrst,
        prefnext,
        told,
        nbuses,
        ifromb,
        itob,
        igbus,
        xinv,
        pload,
        ratea,
    )


def balcon(
    nvar,
    nclin,
    cvec,
    mrowa,
    amat,
    bl,
    bu,
    preffrst,
    prefnext,
    prefcost,
    prefmw,
    nbuses,
    igbus,
    pload,
    akoeff,
):
    dclib.balcon(
        nvar,
        nclin,
        cvec,
        mrowa,
        amat,
        bl,
        bu,
        preffrst,
        prefnext,
        prefcost,
        prefmw,
        nbuses,
        igbus,
        pload,
        akoeff,
    )


def remove(ip, nvar, nclin, mrowa, amat, bl, bu, istate, iconp):
    dclib.remove(ip, nvar, nclin, mrowa, amat, bl, bu, istate, iconp)


def upcon(nvar, nclin, mrowa, amat, bl, bu, x, istate, told):
    dclib.upcon(nvar, nclin, mrowa, amat, bl, bu, x, istate, told)


def update(
    nbuses,
    ngens,
    cincr,
    pbuy,
    psell,
    pgen,
    igbus,
    preffrst,
    prefnext,
    preftyp,
    nvar,
    clamda,
    cvec,
    x,
):
    dclib.update(
        nbuses,
        ngens,
        cincr,
        pbuy,
        psell,
        pgen,
        igbus,
        preffrst,
        prefnext,
        preftyp,
        nvar,
        clamda,
        cvec,
        x,
    )


def bulilist(
    nbuses,
    ibus,
    price,
    maxmw,
    segtyp,
    nseg,
    preffrst,
    prefnext,
    prefcost,
    prefmw,
    preftyp,
):
    dclib.bulilist(
        nbuses,
        ibus,
        price,
        maxmw,
        segtyp,
        nseg,
        preffrst,
        prefnext,
        prefcost,
        prefmw,
        preftyp,
    )


def master(
    nbuses,
    ngens,
    nlines,
    igbus,
    preffrst ,
    prefnext ,
    prefcost ,
    prefmw,
    preftyp ,
    pload ,
    cincr ,
    pbuy,
    psell ,
    pgen,
    teta0 ,
    ifromb ,
    itob ,
    xinv ,
    istat ,
    ratea ,
    basflow ,
    nlcold ,
    ibusno ,
    wconct ,
    wbus,
    akoeff ,
    ickt ,
    sbase ,
    busnam,
):
    dclib.master(
        nbuses ,
        ngens ,
        nlines ,
        igbus ,
        preffrst ,
        prefnext ,
        prefcost ,
        prefmw,
        preftyp ,
        pload ,
        cincr ,
        pbuy,
        psell ,
        pgen,
        teta0 ,
        ifromb ,
        itob ,
        xinv,
        istat,
        ratea,
        basflow,
        nlcold,
        ibusno,
        wconct,
        wbus,
        akoeff,
        ickt,
        sbase,
        busnam,
    )
```

D Lpsolve

This section contains Lpsolve load flow optimization examples in Python and C.

D.1 Lpsolve optimization example in Python

```python
# DC_opt_example in lp_solve


from lpsolve55 import *

# Syntax: [ret1, ret2, ...] = lpsolve('functionname', arg1, arg2, ...)

# Create a linear problem lp
lp = lpsolve("make_lp", 0, 4)
lpsolve("set_verbose", lp, IMPORTANT)

# Set the objective function
ret = lpsolve("set_obj_fn", lp, [20, 20, 30, 30])

# Set the linear constraints
ret = lpsolve("add_constraint", lp, [1, -1, 1, -1], EQ, 0)
ret = lpsolve("add_constraint", lp, [0.333, -0.333, -0.333, 0.333], LE, 100
    - 60)
ret = lpsolve("add_constraint", lp, [0.333, -0.333, 0.667, -0.667], LE, 100
    - 70)
ret = lpsolve("add_constraint", lp, [0.6667, -0.667, 0.333, -0.333], LE,
    100 - 110)

# Set names
ret = lpsolve("set_col_name", lp, 1, "dP1+")
ret = lpsolve("set_col_name", lp, 2, "dP1-")
ret = lpsolve("set_col_name", lp, 3, "dP2+")
ret = lpsolve("set_col_name", lp, 4, "dP2-")
ret = lpsolve("set_row_name", lp, 1, "Balance")
ret = lpsolve("set_row_name", lp, 2, "F12")
ret = lpsolve("set_row_name", lp, 1, "F23")
ret = lpsolve("set_row_name", lp, 1, "F13")

# Solve the system and print results
ret = lpsolve("write_lp", lp, "a.lp")
ret = lpsolve("solve", lp)
print(lpsolve("get_objective", lp))
print(lpsolve("get_variables", lp)[0])
print(lpsolve("get_constraints", lp)[0])
lpsolve("delete_lp", lp)
```

D.2 Lpsolve optimization example in c

```c
/* demo.c */

#include "lp_lib.h"

int demo()
{
    lprec *lp;
    int Ncol, *colno = NULL, j, ret = 0;
    REAL *row = NULL;

    /* We will build the model row by row, so we start with creating a
   model with 0 rows and 4 columns */

    Ncol = 4; /* there are four variables in the model */
    lp = make_lp(0, Ncol);
    if (lp == NULL)
        ret = 1; /* couldn't construct a new model... */

    if (ret == 0)
    {
        /* optional naming of variables */
        set_col_name(lp, 1, "dP1+");
        set_col_name(lp, 2, "dP1-");
        set_col_name(lp, 3, "dP2+");
        set_col_name(lp, 4, "dP2-");

        /* create space large enough for one row */
        colno = (int *)malloc(Ncol * sizeof(*colno));
        row = (REAL *)malloc(Ncol * sizeof(*row));
        if ((colno == NULL) || (row == NULL))
            ret = 2;
    }

    if (ret == 0)
    {
        set_add_rowmode(lp, TRUE); /* makes building the model faster if it
    is done rows by row */

        /* construct first row (balance constraint): x1 - x2 + x3 - x4*/
        j = 0;

        colno[j] = 1; /* first column */
        row[j++] = 1;

        colno[j] = 2; /* second column */
        row[j++] = -1;

        colno[j] = 3; /* third column */
        row[j++] = 1;

        colno[j] = 4; /* fourth column */
        row[j++] = -1;

        /* add the row to lpsolve */
        if (!add_constraintex(lp, j, row, colno, EQ, 0))
            ret = 3;
    }

    if (ret == 0)
    {
        /* construct second row (transmission constraint line 1-2):
        0.333x1 - 0.333x2 - 0.333x3 + 0.333x4 < 100 - 60;
        */
        j = 0;

        colno[j] = 1; /* first column */
        row[j++] = 0.333;

        colno[j] = 2; /* second column */
        row[j++] = -0.333;

        colno[j] = 3; /* third column */
        row[j++] = -0.333;

        colno[j] = 4; /* fourth column */
        row[j++] = 0.333;

        /* add the row to lpsolve */
        if (!add_constraintex(lp, j, row, colno, LE, 40))
            ret = 3;
    }

    if (ret == 0)
    {
        /* construct third row (transmission constraint line 2-3):
        0.333x1 -0.333x2 + 0.667x3 - 0.667x4 < 100 - 70;
        */
        j = 0;

        colno[j] = 1; /* first column */
        row[j++] = 0.333;

        colno[j] = 2; /* second column */
        row[j++] = -0.333;

        colno[j] = 3; /* third column */
        row[j++] = 0.667;

        colno[j] = 4; /* fourth column */
        row[j++] = -0.667;

        /* add the row to lpsolve */
        if (!add_constraintex(lp, j, row, colno, LE, 30))
            ret = 3;
    }

    if (ret == 0)
    {
        /* construct fourth row (transmission constraint line 1-3):
        0.667x1 - 0.667x2 + 0.333x3 - 0.333x4 < 100 - 110;
        */
        j = 0;

        colno[j] = 1; /* first column */
        row[j++] = 0.667;

        colno[j] = 2; /* second column */
        row[j++] = -0.667;

        colno[j] = 3; /* third column */
        row[j++] = 0.333;

        colno[j] = 4; /* fourth column */
        row[j++] = -0.333;

        /* add the row to lpsolve */
        if (!add_constraintex(lp, j, row, colno, LE, -10))
            ret = 3;
    }

    if (ret == 0)
    {
        set_add_rowmode(lp, FALSE); /* rowmode should be turned off again
when done building the model */

        /* set the objective function (20 x1 +  20 x2 +  30 x3 +  30 x4) */
        j = 0;

        colno[j] = 1; /* first column */
        row[j++] = 20;

        colno[j] = 2; /* second column */
        row[j++] = 20;

        colno[j] = 3; /* first column */
        row[j++] = 30;

        colno[j] = 4; /* second column */
        row[j++] = 30;

        /* set the objective in lpsolve */
        if (!set_obj_fnex(lp, j, row, colno))
            ret = 4;
    }

    if (ret == 0)
    {
        /* set the object direction to minimize */
        set_minim(lp);

        /* print model to screen and write to file  */
        write_LP(lp, stdout);
        write_lp(lp, "model.lp");

        /* Only warnings and error messages will be shown */
        set_verbose(lp, IMPORTANT);

        /* Solve the problem */
        ret = solve(lp);
        if (ret == OPTIMAL)
            ret = 0;
        else
            ret = 5;
    }
```

```c

    if (ret == 0)
    {
        /* get the results */

        /* objective value */
        printf("Objective value: %f\n", get_objective(lp));

        /* variable values */
        get_variables(lp, row);
        for (j = 0; j < Ncol; j++)
            printf("%s: %f\n", get_col_name(lp, j + 1), row[j]);

        /* we are done now */
    }

    /* free allocated memory */
    if (row != NULL)
        free(row);
    if (colno != NULL)
        free(colno);

    if (lp != NULL)
    {
        /* free up all memory used by lpsolve */
        delete_lp(lp);
    }

    return (ret);
}

int main()
{
    demo();
}
```

E Tests

This section contains the Python test script to test the DCflow function balcon.

E.1 dcflow_test_balcon.py

```
# Test of DCflow
# A test with zeros to test the interface

import dcflow_wrapper as dc
import numpy as np

# Create variables
# Number of buses: 3
# Number of generators: 2
# Number of lines: 3

# int
nbuses = 3
ngens = 2
nlines = 3
nseg = 0
lmess = 0
rmext = 0
icont = 0
nicont = 0
ibuild = 1
nz = 0
nsteps = 0
iflag = 0
maxfrt = 0
la = 0
nlcold = 0
il = 0
nvar = 100
nclin = 0
mrowa = 100
ip = 0
lout = 0

# int *
igbus = np.zeros((nbuses), dtype=np.intc)
ifromb = np.zeros((nbuses), dtype=np.intc)
```

```
itob = np.zeros((nbuses), dtype=np.intc)
ickt = np.zeros((nbuses), dtype=np.intc)
istat = np.zeros((nbuses), dtype=np.intc)
ibusno = np.zeros((nbuses), dtype=np.intc)
ibus = np.zeros((nbuses), dtype=np.intc)
segtyp = np.zeros((nbuses), dtype=np.intc)
preffrst = np.zeros((nbuses), dtype=np.intc)
prefnext = np.zeros((nbuses), dtype=np.intc)
preftyp = np.zeros((nbuses), dtype=np.intc)
irn = np.zeros((nbuses), dtype=np.intc)
icn = np.zeros((nbuses), dtype=np.intc)
ikeep = np.zeros((nbuses), dtype=np.intc)
iw = np.zeros((nbuses), dtype=np.intc)
iw1 = np.zeros((nbuses), dtype=np.intc)
wconct = np.zeros((nbuses), dtype=np.intc)
wbus = np.zeros((nbuses), dtype=np.intc)
istate = np.zeros((nbuses), dtype=np.intc)
iconp = np.zeros((nbuses), dtype=np.intc)

# double *
pload = np.zeros((nbuses), dtype=np.double)
pgen = np.zeros((nbuses), dtype=np.double)
psell = np.zeros((nbuses), dtype=np.double)
pbuy = np.zeros((nbuses), dtype=np.double)
xinv = np.zeros((nbuses), dtype=np.double)
ratea = np.zeros((nbuses), dtype=np.double)
price = np.zeros((nbuses), dtype=np.double)
maxmw = np.zeros((nbuses), dtype=np.double)
prefcost = np.zeros((nbuses), dtype=np.double)
prefmw = np.zeros((nbuses), dtype=np.double)
teta0 = np.zeros((nbuses), dtype=np.double)
basflow = np.zeros((nbuses), dtype=np.double)
cincr = np.zeros((nbuses), dtype=np.double)
a = np.zeros((nbuses), dtype=np.double)
w = np.zeros((nbuses), dtype=np.double)
akoeff = np.zeros((nbuses), dtype=np.double)
bl = np.zeros((nbuses), dtype=np.double)
bu = np.zeros((nbuses), dtype=np.double)
sens = np.zeros((nbuses), dtype=np.double)
told = np.zeros((nbuses), dtype=np.double)
cvec = np.zeros((nbuses), dtype=np.double)
x = np.zeros((nbuses), dtype=np.double)
clamda = np.zeros((nbuses), dtype=np.double)

```

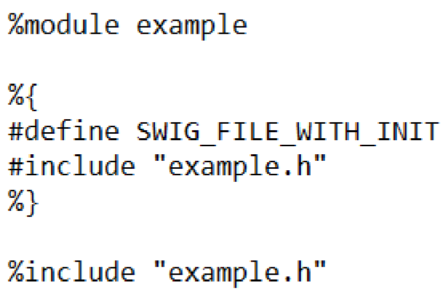
```
# double **
amat = np.zeros([2, 2], dtype=np.double)

# double
sbase = 100.0

# char *
busnam = np.zeros((nbuses), dtype=np.intc)

# Call function

dc.balcon(
    nvar,
    nclin,
    cvec,
    mrowa,
    amat,
    bl,
    bu,
    preffrst,
    prefnext,
    prefcost,
    prefmw,
    nbuses,
    igbus,
    pload,
    akoeff,
)
```

F SWIG (from specialization project)

This section contains information on SWIG, the Simplified Wrapper and Interface Generator, from the specialization project [2].

F.1 SWIG

SWIG is a tool for automatic generation of wrapper code [1]. In order to do this, SWIG requires information about the code to be wrapped, for example which functions to wrap and what type of input arguments and return arguments those functions should have. Given a header file (.h) and a C source file (.c), an interface file (.i) must be created with instructions to SWIG based on this information. An interface file can look like Figure F.1 [43], if the functions in the header file do not require any additional instructions. The %module statement determines the name of the module. Everything inside the %{%} block must be C code and it should contain the necessary #include statements. SWIG_FILE_WITH_INIT indicates that this is an extension module. After this block comes the special directives or rules to be applied to some or all of the functions to be wrapped. These can be specified using the %include directive as in Figure F.1, or the relevant functions can be listed instead [44].



```
%module example

%{
#define SWIG_FILE_WITH_INIT
#include "example.h"
%}

%include "example.h"
```

Figure F.1: Simple example of SWIG interface file "example.i"

For anything but very simple functions some additional directives might be needed in the interface file to make the module work like it is expected to. SWIG comes with instructions to convert input parameters to output parameters called *typemaps*. In order to wrap a function with output parameters given as input parameters from C to Python in such a way that the input parameters will be an output parameters in Python, the following typemap can be used in the interface file [5]:

```
%include "typemaps.i"
%apply double *OUTPUT {double *res }
```

This will be incorporated into the SWIG wrapper code, resulting in a function that can be called in the Pythonic way with an output parameter.

F.1.1 Support for NumPy

SWIG offers compatibility with NumPy with the interface file "numpy.i". This file includes typemaps for NumPy array conversions. It includes *input* arrays for arrays that are passed to the function, but are not modified inside the function or returned, *in-place* arrays for arrays that are altered by the function, and *argout* arrays that are originally input arrays and should be output arrays in Python. To include "numpy.i" the following code can be used in the interface file [45]:

```
%include "numpy.i"
%init %{
 import_array();
%}
```

Depending on how the array parameter is given in the C function, different signatures can be used. The following signatures apply to one-dimensional input arrays [45]:

- (DATA_TYPE IN_ARRAY1[ANY])
- (DATA_TYPE* IN_ARRAY1, int DIM1)
- (int DIM1, DATA_TYPE IN_ARRAY1)

These can be extended to more dimensions, but they can only be applied where the array dimension parameter comes right before or right after the array parameter, or if the array has hard coded dimensions.

Hege Bruvik Kvandal

Toolbox for Specialized Power System Analysis

NTNU
Norwegian University of
Science and Technology