Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Corso di studi in Ingegneria Energetica

Object-Oriented Modeling and Dynamic Optimization of Energy Systems with Application to Combined-Cycle Power Plant Start-Up

Author:
Pietro Parini Matr. 801337

Advisor:
Prof. Francesco Casella

Academic year 2014 - 2015

You can never know everything, and part of what you know is always wrong.
Perhaps even the most important part.
A portion of wisdom lies in knowing that.
A portion of courage lies in going on anyway.

Robert Jordan

Ringraziamenti

Vorrei in primo luogo ringraziare il professor Casella che mi ha seguito in questa tesi e mi ha aiutato a scoprire un argomento di cui sapevo poco o niente.

Un grande grazie a mio fratello Alessandro senza la cui consulenza informatica questo lavoro sarebbe stato decisamente più difficile, e il resto della mia famiglia che mi ha sostenuto per tutto il percorso universitario.

Un ringraziamento particolare a Andrea "Pavel" Manfreda che mi ha accolto al mio arrivo a Milano e mi ha insegnato a vivere oltre che sopravvivere un appartamento di sette studenti universitari fuori sede, e ai compagni di Appa che ho avuto in questi anni: sia per avermi lasciato stare quando mi chiudevo in un angolo a leggere, sia per avermi chiamato quando ci stavo troppo tempo.

Mille grazie ai compagni di studio con cui ho preparato degli esami che mi hanno aiutato negli esami che mi risultavano più difficili. In particolare Davide Mariani e Pietro Gheorghiu.

E ultimo ma non meno importante, grazie agli amici di Genova che mi accolto tutte le volte che riuscivo a fuggire da Milano e tornare alla Superba.

Abstract

Due to the rising concerns of sustainable development, in recent years the renewable share of the total electrical energy produced worldwide has increased considerably and is expected to continue to do so. Since most of the new renewable power plants are intrinsically non-programmable, flexibility of the output has become a critical characteristic of thermal power plants. One way of increasing the flexibility of existing power plants is to devise advanced control techniques that determine the optimal way to shift the operating point of the plant. In order to apply an automatic control to any system, it is necessary to provide: a mathematical model of the system itself, formulate the optimization problem, solve the problem. Each step of this process has to be repeated many times before a reasonable solution can be obtained. This has never been tried before with problem of practical interest for the energy industry. In this work we do so using high level. We developed a model of a combined cycle power plant using the modeling language Modelica. Then we used said model to test three optimization methods based on the Direct Collocation implemented using the software JModelica and OpenModelica. The process we studied was the optimization of a warm start-up of the plant with the goal of reaching the full load without violating a constraint on the structural integrity of the steam turbine. All the methods reached an acceptable solution of the problem, so their performance was compared and commented to find out their strengths and weaknesses.

Keywords: Object-Oriented modeling, Optimal Control, Modelica, Optimization, Combined Cycle, Direct Collocation

Estratto in lingua italiana

Il nuovo scenario energetico

Negli ultimi anni lo scenario energetico è notevolmente cambiato. Le nuove preoccupazioni di sviluppo sostenibile hanno portato a una notevole crescita della quota di energia elettrica mondiale prodotta da fonti rinnovabili, in particolare sono stati costruiti molti impianti solari ed eolici. Queste fonti di energia sono intrinsecamente non programmabili, il che costringe le centrali tradizionali a modulare la loro produzione non solo in risposta alla domanda della rete, ma anche all'offerta variabile delle nuove rinnovabili.

Alla luce di questo fatto diventa interessante la possibilità di applicare tecniche di controllo ottimo alle centrali tradizionali per ottimizzare i loro transitori e aumentare la flessibilità della loro produzione. Queste tecniche richiedono un modello matematico della centrale da ottimizzare, a partire da questo modello viene poi scritto il problema di ottimizzazione, che può quindi essere sottoposto a un software di calcolo numeri per ottenere una soluzione.

Il problema è che è raro che questo processo fornisca una soluzione ragionevolmente corretta alla prima iterazione. Più spesso insorgeranno problemi in una di queste fasi e di conseguenza sarà necessario ripetere tutte e tre, ad esempio:

- il modello si rivela troppo complicato per il software, o viceversa si scopre che una dinamica che era stata trascurata è in realtà molto rilevante, e allora bisognerà semplificare o ampliare il modello;
- il problema di ottimizzazione è mal posto a causa della mancanza di uno o più vincoli, oppure la funzione obiettivo non porta alla soluzione desiderata e va modificata;
- il metodo di soluzione non è adatto al problema o impiega troppo tempo e va sostituito con un altro.

Questo tipo di approccio non è mai stato applicato a problemi di interesse reale nel settore energetico, quindi lo scopo di questo lavoro è provare a utilizzare strumenti e software innovativi e di alto livello e capire se sono abbastanza maturi per essere utilizzati in applicazioni pratiche

Per fare ciò abbiamo sviluppato un modello di un ciclo combinato usando il linguaggio di programmazione Modelica e lo abbiamo testato con con diversi metodi implementati tramite i software JModelica e OpenModelica, entrambi open-source.

Modelica

Modelica è un linguaggio di modellazione orientato agli oggetti. Questo linguaggio si presta molto bene al nostro scopo in quanto permette di modellare facilmente sistemi molto complessi a partire dai loro componenti.

L'approccio di fondo è di iniziare creando modelli dei componenti elementari tramite equazioni di bilancio e munirli di appropriati connettori che hanno la funzione di interfacciare i componenti tra di loro, queste connessioni generano poi equazioni algebriche quando il modello viene compilato. Successivamente i componenti elementari possono essere collegati tra di loro attraverso i connettori creando modelli più complessi. Questa operazione viene ripetuta finché non viene raggiunto il livello del sistema complessivo, a questo punto il modello di insieme conterrà tutte le equazioni e le variabili del sistema complessivo.

Questo approccio modulare permette di suddividere il problema in parti più semplici e gestibili che possono essere controllate per errori indipendentemente l'una dall'altra, e anche di sostituire o modificare con facilità un componente invece di dover riscrivere da capo il modello del sistema in caso le specifiche cambiassero. Questa eventualità è tutt'altro che rara nel campo dell'ottimizzazione, ambito nel quale è necessario sviluppare parallelamente il modello da ottimizzare e il metodo di ottimizzazione ottenere una soluzione.

La fase di connessione dei componenti viene resa particolarmente facile e intuitiva dal tool grafico OMEdit.

Ottimizzazione

Il problema che vogliamo risolvere è un problema di ottimizzazione dinamica. Una volta compilato dal software, il modello della nostra centrale è rappresentato da un

sistema di equazioni algebrico-differenziali (DAE) del tipo

$$F(\dot{x}(t),\, x(t),\, w(t),\, u(t)) = 0$$

e vincoli tra le variabili espressi da disequazioni nella forma

$$H(x(t),\, w(t),\, u(t)) \geq 0$$

dove $x(t)$ sono le variabili le cui derivate compaiono esplicitamente, dette variabili di stato; $w(t)$ sono le variabili algebriche; e $u(t)$ sono gli ingressi, le variabili controllate dall'utente.

Il nostro scopo è determinare le funzioni $u(t)$ che minimizzano una certa funzione di costo solitamente espressa nella forma

$$C(x,\, w,\, u) = \int_0^T L(x(t),\, w(t),\, u(t))dt$$

Dove T è l'intervallo di tempo considerato per l'ottimizzazione.

In questa tesi abbiamo deciso di risolvere il problema di ottimizzazione con il metodo della Direct Collocation. Questo metodo si basa sul dividere l'intervallo di ottimizzare in n sotto intervalli e approssimare ingressi e variabili come funzioni polinomiali a tratti, trasformando il problema dal determinare le le funzioni degli ingressi nel tempo (problema infinito dimensionale), a determinare un numero finito di coefficienti dei polinomi (problema finito dimensionale). Il problema da risolvere rimane comunque un problema non lineare (NLP), ma sono disponibili molte tecniche collaudate per risolvere questo genere di problemi.

Questo metodo è stato implementato tramite i software JModelica e OpenModelica, entrambi basati sul linguaggio Modelica e la sua estensione Optimica che aggiunge i costrutti necessari a definire problemi di ottimizzazione. La Direct Collocation è stata implementata in tre modi diversi:

- Direct Collocation sul sistema in forma DAE con JModelica;
- Direct Collocation sul sistema in forma DAE dopo aver semplificato il problema e ridotto il numero di variabili algebriche con JModelica;
- Direct Collocation sul sistema convertito in forma ODE con OpenModelica.

Un caso studio

Il caso di studio scelto per questo lavoro è un ciclo combinato ad un livello di pressione. Il processo studiato è uno start-up a caldo in cui la turbina a gas passa dal 15% al 100% del carico massimo. L'obiettivo dell'ottimizzazione è di portare la centrale al carico massimo senza violare i vincoli del problema. Le variabili libere sono il carico della turbina a gas e la portata di attemperamento del ciclo a vapore.

Il vincolo principale è lo stress termo-meccanico sull'albero della turbina a vapore, che non deve superare un valore di sicurezza nè essere soggetto a oscillazioni. Questo stress è causato dal gradiente di temperatura radiale che si forma sull'albero durante i transitori a causa della variazione della temperatura del vapore che entra in turbina.

Nello sviluppare il modello è stato necessario fare alcune ipotesi semplificative, in particolare riguardo alle proprietà di acqua e vapore, ma i risultati delle simulazioni fatte su di esso risultano in linea col comportamento di una centrale reale.

Per aumentare il numero di dati raccolti abbiamo preparato due varianti del modello:

- un modello di riferimento che non utilizza l'attemperamento durante lo start-up;
- un modello completo che usa entrambe le variabili di ottimizzazione.

Campagna di test

La campagna di test è stata condotta applicando ciascun metodo a entrambe le versioni del modello e variando il numero di passi temporali in cui è stato diviso l'intervallo di ottimizzazione.

I dati monitorati per confrontare le prestazioni degli algoritmi sono stati:

- numero di variabili del NLP;
- numero di iterazioni necessarie a raggiungere convergenza;
- tempo impiegato a risolvere il NLP, escluso il tempo necessario a calcolare le funzioni necessarie;
- tempo impiegato nel calcolare le funzioni.

Analizzando i dati è emerso che i due metodi di JModelica hanno un comportamento molto simile tra di loro, la versione con eliminazione delle variabili si è dimostrata più

efficiente della sua controparte, ma non in maniera troppo significativa. Il metodo di OpenModelica è risultato molto più lento degli altri due, ma anche più stabile. Mentre le prestazioni dei metodi di JModelica variano enormemente al variare della discretizzazione, arrivando a non convergere per certi valori del passo temporale, OpenModelica ha mostrato un andamento più regolare e affidabile.

La causa della variabilità delle prestazioni di JModelica è stata identificata in una difficoltà nell'approssimare le non linearità del modello che emerge o meno a seconda della discretizzazione dell'intervallo temporale.

Conclusioni

Il nostro caso studio ha rivelato che attualmente esistono strumenti di modellazione e ottimizzazione capaci di risolvere problemi di interesse in campo energetico.

Inoltre gli strumenti di ottimizzazione da noi utilizzati, JModelica e OpenModelica, sono ancora in fase di sviluppo. Attualmente stanno venendo testati miglioramenti ai metodi esistenti e nuovi metodi più avanzati. Il modello prodotto da questo lavoro, potrebbe essere facilmente utilizzato come caso studio anche per testare queste nuove funzionalità quando verranno implementate.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The worldwide demand for electrical energy has been steadily increasing for many years, especially many third-world countries are beginning to develop and their energy requirements are growing. Because of concerns of sustainable development and environmental preservation, the share of electrical energy produced from renewable energy sources has also grown as it can be seen from figure 1.1.

In particular, in order to deal with the problems of global climate change and atmospheric CO_2, it is expected that the policies and restrictions on power generations and carbon emissions will grow even stricter for the next years. For example figure 1.2 shows how the CO_2 emissions should be reduced in the future according to the International Energy Agency (IEA) in order to reach the 450 Scenario, which aims to limit the global increase in temperature to 2^oC by limiting concentration of greenhouse gases in the atmosphere to around 450 parts per million of CO_2.

A significant part of these renewable power plants are based on solar and wind energy, which are intrinsically non-programmable and mostly non-predictable, and lend themselves to distributed generation. These characteristics make it so that the power they supply to the electrical grid is highly irregular, forcing the traditional thermal power plants to modulate their power output not only in response to the varying demand of the users, but also to the non-predictable power supplied by the renewable plants. In light of this the estimation of figure 1.2 can be misleading, since with the right weather conditions, the renewable power plants can account for the main share of the electrical energy demand of a region.

Unfortunately the traditional thermal power plants are complex and relatively delicate systems, they typically have significant thermal or mechanical inertia and their thick-walled components are subject to heavy mechanical stress, so regulating their output quickly without compromising their service life can be very difficult. In order

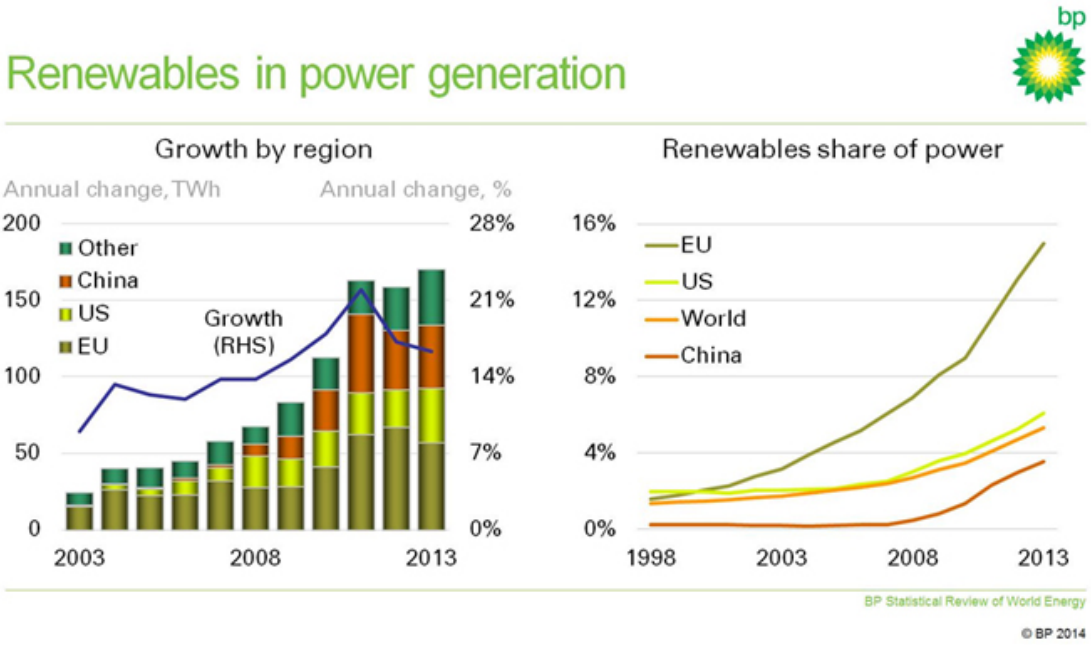

Figure 1.1: Share of renewable electrical energy in recent years[23]

to improve their flexibility it is necessary to quickly determine the best way to shift the operating point of the plant, so the idea of applying advanced control techniques to the plants is very attractive. Particularly because it could be easily applied to already existing plants that were designed when the need for flexibility was not as pronounced but are still operational due to their very long life-cycles.

Specifically, the so called Optimal Control techniques focus on finding the best transient from one state to another for a given system while taking into consideration eventual constraints on the variables. These kind of techniques necessitate a mathematical model of the system to be controlled, which has to be both complete in all its relevant dynamics and simple enough for the problem to be solved in short enough time. Also, the Optimal Control ultimately consist of solving a type of mathematical problem, the optimization problem, whose solution is definitely non-trivial.

1.1 Aim of the thesis

Given this context, in this work we will explore the possibility of using a modern modeling language to create models of power plants and then use them in conjunction with appropriate software to determine the best way to shift their operating point, thus increasing their flexibility. The models created need to be both complete in all the relevant dynamics, and simple enough for the software to find a solution in good

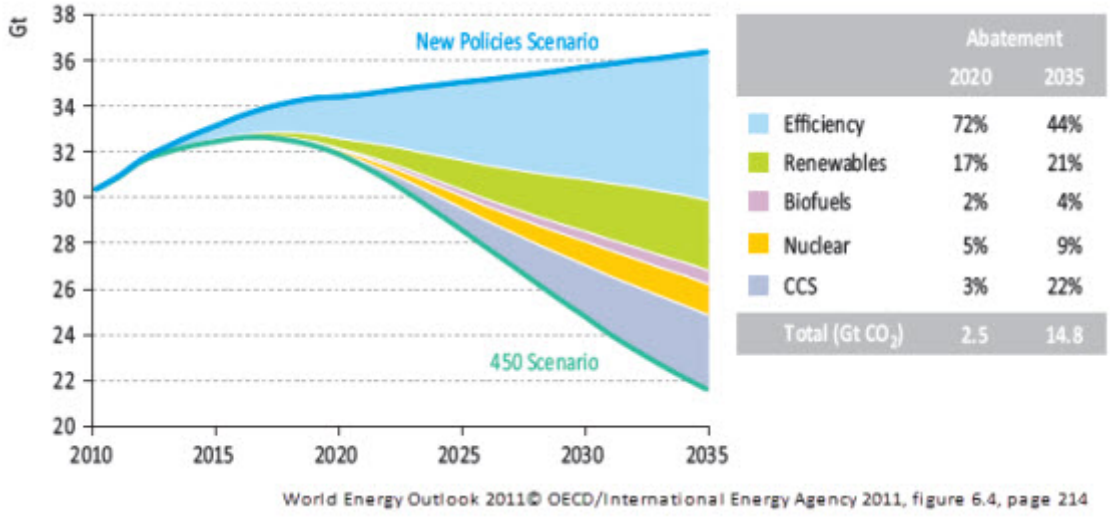

Figure 1.2: Difference in CO_2emission in different scenarios[12]

time.

This will be done, by developing a model of a combined cycle power plant as a case study and then using it to test multiple methods of optimizations applied to a practical situation. This will give information on whether or not these types of problem are feasible with these methods.

The tools used in this thesis are the modeling language Modelica, the OpenModelica v1.9.2 compiler and optimizer, the JModelica.org v1.15 compiler and optimizer, and the graphical editor OMEdit. The model will be created using OpenModelica, while the optimization will be done with both OpenModelica and JModelica.

1.2 Thesis structure

After this first chapter of introduction, chapter two describes the modeling language Modelica used in this work. The third chapter explores the theory of optimization and some of the possible methods of solution from a mathematical point of view, while chapter four explains the software that are used in this work to apply them to the case study which is explored in detail in chapter five. Finally in chapter six the data of the performance of the software are reported and compared, and chapter seven contains the conclusions and possibilities of future work.

Chapter 2

Modelica, an Object-oriented modeling language

Modelica is a tool-independent modeling language developed by the non-profit Modelica Association. It is widely used to simulate complex systems, is two defining characteristics of this language are being Object-oriented and equation based.

Object-oriented means that it focuses on creating objects independent of their boundary conditions that describe components and parts of a complex system, these objects can then be connected to form more complex objects until the whole system is described via physical connections.

Equation based on the other hand means that the governing equations of each objects are not definitions of a variable (i.e. $x := y + z + 4$), but declarative statements of equality constraints between two expressions (i.e. $x - y = z + 4$). Referring to the examples: the first permits only to calculate x as a function of y and z, the relation is causal; in the second, any of the variables can be calculated once the other two are known, the relation is a-causal. This property also extends to the connections between objects.

2.1 A simple example

This simple model of a Van der Pol oscillator shows what Modelica code looks like:

```
model VDP
  Real u "Control signal";
  Real x1(start=0, fixed=true) "First state, velocity";
  Real x2(start=1, fixed=true) "Second state, position";
equation
  der(x1) = (1 - x2^2) * x1 - x2 + u;
  der(x2) = x1;
  u = sin(time);
end VDP;
```

The first word of the object defines what class of object it is, in this case the "model" class, which is the most generic and used. Other classes have built in behaviors and are used for specific purposes. After that the name of the object is declared and the model itself can be defined.

The first part is the declaration of the variables of the model, each one preceded by an indication of its type (Real, Integer, Boolean, etc). It is also possible to declare another model as a variable, this will add all of that model variables and equations to new model. In the example all of the variables are real numbers: two state variables ($x1$, $x2$) and a control signal (u). Variables can also have attributes that add information about the variable, in this case both state variables have a specific start value and the start value is to be used for initialization.

After that the "equation" section contains the three equations that fully describe the behavior of the variables of the system, in this case two differential equations and an algebraic one. This is a stand-alone model, it can be directly simulated and does not admit connections with other models.

2.2 Connections

In order to model more complex systems, we need to separate it in its components and model each of them separately and the connect each component model to obtain the model of the whole system. But, in order to define the relations between components, it is necessary to first define a Connector class. Given the a-causal nature of the connections it is necessary to distinguish between two types of variables: potential (or across) variables, and flow (or through) variables. Differences in the values of across variables across a component are what trigger components to react, while flow variables

normally represent the flow of some conserved quantity like mass, momentum, energy, charge, etc.

Declaring an object as a connector implies that it contains at least one flow variable and one potential variable, else it is not a valid object. Also, it makes the object a viable target for the "connect" instruction that serves to connect different components. To understand how this instruction works it is easier to refer to this example of an electrical pin that is used to model electric circuits:

```
connector electrical_pin
  Real v ‘‘Electric Potential’’;
  flow Real i ’’Current’’;
end electrical_pin;
```

The across variable is the Electric Potential and the flow variable is the current. When two or more components with this type of connector are connected it will result in the following equations (the flow variables are considered positive if entering the component):

$$\begin{aligned} \sum i_j &= 0 \\ v_1 &= v_2 \\ v_1 &= v_3 \\ &\vdots \\ v_1 &= v_n \end{aligned} \tag{2.1}$$

These equations guarantee that at each node the charge is conserved, while the potential is the same in all the components connected to it.

If needed it is also possible to create causal connections by declaring variables as the built-in types "input" and "output".

2.3 Components

The model of a component will contain all the needed variables and equations plus one or more connectors. For example this is the model of a resistor:

```
model Resistor
  electrical_pin positive_pin ‘‘positive pin of the resistor’’;
  electrical_pin negative_pin ‘‘negative pin of the resistor’’;
  Real i ‘‘current flowing through the resistor’’;
  Real delta_v ‘‘difference of potential across the resistor’’;
  parameter Real R = 1 ‘‘resistance of the resistor’’;
equation
  delta_v = positive_pin.v - negative_pin.v;
  positive_pin.i = i;
  positive_pin.i + negative_pin.i = 0;
  delta_v = R * i;
end Resistor;
```

The model contains two instances of the connector described before, each containing two variables, and there are two more variables declared in the model itself. The resistance is declared as a "parameter" i.e. a constant that does not change throughout the simulation. There are also four equations, which would leave the system underdetermined since there would be more variables then equations. But when a connector is not connected to anything an equation stating that the flow variable is zero is automatically added, and when the connector is connected to something the equation of the connection discussed earlier are added to the total. This is very important because a model can be simulated only if the number of equations and the number of variables are the same, which is a necessary condition for the existence and uniqueness of the solution.

2.4 General form of a physical model

The model of a physical system can always be written as a system of Differential and Algebraic Equations (DAE), which is a generalization of a typical Ordinary Differential Equations system (ODE) where the derivatives of some variable do not appear explicitly in any equation. Because of this the convention is to distinguish between the state variables (x) whose derivatives appear explicitly in the system, and the algebraic variables (w) whose derivatives do not appear.

While the difference may seem trivial, the method used to solve DAE systems are different from the ones used to solve ODE, and are significantly more complicated.

To get an idea of the difference between DAE and ODE systems, consider the very simple example

$$\begin{cases} \dot{x} - w = 0 \\ x - u = 0 \end{cases} \tag{2.2}$$

where a sufficiently smooth function u is given. Clearly, the only solution is $x = u(t)$, $w = \dot{u}(t)$, and no initial conditions are needed. That is, if an arbitrary initial condition is imposed, it may well be inconsistent with the DAE. Furthermore, it can be seen that the solution depends on the derivative of the input which cannot happen in the ODE case. Another difference is that even if consistent initial values are given, the existence and uniqueness theory is more complicated and involves additional technical assumptions besides just sufficient smoothness as in the ODE case. The fact that in the first equation of this example, one needs to differentiate x , which implies differentiation of the input function u , in order to find w , makes a key difference. For a standard-form ODE, the solution is always more continuous than the input. In other words, a DAE may involve both integrations and differentiations[8].

It should also be noted that any differential equation of order n can be transformed in a system of n equations of the first order:

$$a\ddot{x}_1 + b\dot{x}_1 + cx_1 = g(x_1, t) \Longleftrightarrow \begin{cases} a\dot{x}_2 + bx_2 + cx_1 = g(x_1, t) \\ x_2 = \dot{x}_1 \end{cases} \tag{2.3}$$

Given these premises the general form of a model is a DAE system of n equations in the form $f_i(\underline{\dot{x}}, \underline{x}, \underline{w}, \underline{u}) = 0$ where $\underline{x}$ and $\underline{w}$ are the vectors of, respectively, the state and algebraic variables of the system, whose total number has to be n in order for the system to admit a solution; $\underline{\dot{x}}$ the vector of the first derivatives of the state variables, and $\underline{u}$ the vector of the inputs.

2.5 The potential of Modelica

The model of a complex system such as a power plant can easily have hundreds of variables and just as many equations, manually writing each one of these equations to create the model in one go would be a long and error-prone process with little possibility of checking the correctness of each part independently. With Modelica on the other hand it is possible to decompose the system in elementary components, write

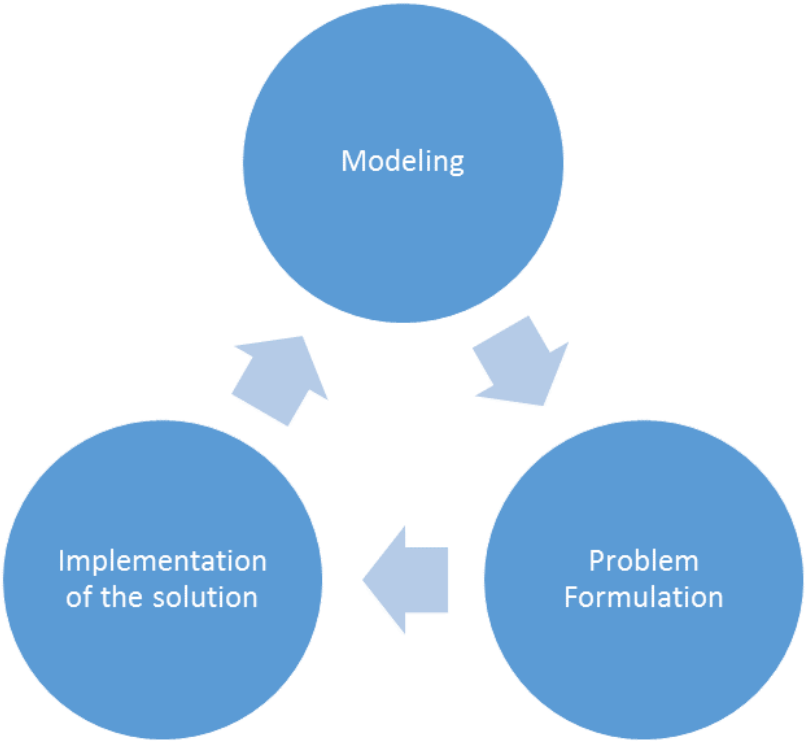

Figure 2.1: Solving an Optimization Problem

the code for each one, check it for errors, and then compose the system by adding the connections between each component.

This also means that the model of each system is highly modular: in order to modify a model the user can simply modify the involved components, or write new version of them, and then substitute them in the model of the system. For the same reasons, a component written for an old model could be re-used in a different model of the same domain.

This is of particular importance since the topic of this work is optimization. At its core the solution of an optimization problem consists of:

- Developing a model of the system;
- Formulating the optimization problem;
- Implementing a method of solution.

But most of the time these steps have to be iterated more than once in order to obtain a reasonable and significant solution. Any one of these steps may need to be repeated and modified, and that in turns force the user to do the same for the other two. Some examples of the possible problems:

- the model may turn out to be too complicated for the software to handle, or vice-versa a crucial dynamics may have been mistakenly neglected;
- the formulation of the problem may need to be changed since one or more constraints may be missing, or it might turn out that a different cost function is needed in order to obtain satisfactory results;
- the method of solution may not be suitable for the problem and may need to be substituted for another one.

This whole process becomes significantly easier if the high-level tools are used, and as it has been elaborated earlier Modelica model are very easy to modify.

2.6 Software for model development

There are many commercial and open source software that can be used to write and simulate Modelica code, the one used for this work is OpenModelica, supported by the non-profit organization Open Source Modelica Consortium [11].

OpenModelica contains various tools to write Modelica code with, from the low level Open Modelica Shell which interacts directly with the Modelica compiler, to the higher level Open Modelica Connection Editor (OMEdit) used in this work. While it supports the plain writing of Modelica code, the main feature of OMEdit is a user-friendly graphical interface that facilitates composing complex models from their parts. The tool adds a diagram and an icon view of the code beside the standard text view which are generated annotations added to the code of the model, the diagram view shows the graphical representation of the objects declared in the model, while the icon view shows how the current model will look when used in other models.

The important part is that the program translates changes to one view to the others, for example dragging and dropping the icon of a component to the diagram of the current model adds the declaration of the component to its code, and drawing a line between the connectors of two components adds the connect instruction to the code.

Chapter 3

Direct methods for optimal control problems

In mathematical terms, an optimization problem consist in minimizing or maximizing a function, called the objective function, while the independent variables are subject to some constraint. Since maximizing $f(x)$ is identical to minimizing $-f(x)$, the convention is to only deal with minimizing functions, and to call the function to be minimized "cost function".

In this work the focus will be on dynamic optimization, i.e. problems where the independent variables are the input or control variables of a physical system over an interval of time, and the cost function (C) is the time integral of a function (L) of the states and inputs. A simple example is finding the force (input) to be applied to a pendulum (system) order to stop it as quickly as possible, i.e. bringing the angular distance from the vertical θ and the velocity v to zero.

This is the general form of the optimization problem:

$$
\begin{array}{lll}
minimize & & \\
C(x,\, w,\, u) = \int_0^T L(x(t),\, w(t),\, u(t))dt & & (cost\ function) \\
subject\ to & & \\
G(x(0),\, w(0)) = 0 & & (fixed\, initial\, conditions) \\
F(\dot{x}(t),\, x(t),\, w(t),\, u(t)) = 0 & \forall t \epsilon [0,T] & (DAE\, System) \\
H(x(t),\, w(t),\, u(t)) \geq 0 & \forall t \epsilon [0,T] & (path\, constraints)
\end{array}
\tag{3.1}
$$

For simplicity of notation the vector sign as been omitted from all the variables and F, G and H functions represent respectively the whole DAE system, the fixed initial

conditions, and all the constraint. Similarly, x_0 is the vector of all the initial values of the state variables.

3.1 Basic approach

In order minimize a function with only one variable it is only necessary to find where the first derivative is zero and then check the sign of the second derivative. In case of function with two or more variables it becomes necessary to determine the gradient and the hessian matrix which is significantly more computationally complex, but there are many efficient algorithms to solve this kind of problem numerically. The dynamic optimization on the other hand is a problem of even higher complexity, since instead of having to determine a finite number of variables it is necessary to determine a function over a continuous interval of time.

In this work we will focus on the so called direct methods, that are based on approximating the input function as a piecewise constant function, such a function is completely determined once the value of the function in each interval is determined. this transforms the problem from a continuous and infinite-dimensional one to a finite dimensional one. The problem is still going to be a Non-Linear Program (NLP), but there are many well tested iterative algorithm that can solve such problems.

3.2 Single Shooting

The single shooting method [14] is based on discretizing only the inputs u. The time interval is divided in n time steps with $t_i = \frac{T}{n} \cdot i$ for $i\epsilon[0, n]$ and the inputs are defined as

$$\begin{cases} u(t) = q_i \\ t_{i-1} \leq t \leq t_i \end{cases} \tag{3.2}$$

thus making the vector q of the values of each input variable in each time step the vector of the variables to be optimized. The optimization problem becomes:

$$
\begin{aligned}
&\textit{minimize} \\
&C(q) = \textstyle\int_0^T L(x(t,q),\, w(t,q),\, u(t,q))dt \\
&\textit{subject to} \\
&\begin{cases}
F(\dot{x}(t,q),\, x(t,q),\, w(t,q),\, u(t,q)) = 0 & \forall t \epsilon [0,T] \\
H(x(t,q),\, w(t),\, u(t,q)) \geq 0 & \forall t \epsilon [0,T] \\
G(x(0),\, w(0)) = 0
\end{cases}
\end{aligned}
\tag{3.3}
$$

When using this method in each iteration the system is simulated with the current iteration of u, obtaining new values for the state and algebraic variables in the whole interval. Then using these values the derivatives of the cost function are evaluated and the new iteration of u is calculated, so the simulation and optimization are sequential.

Pros, and Cons:

+ Can use state-of-the-art ODE/DAE solvers.

+ Few degrees of freedom even for large ODE/DAE systems.

+ Active set changes easily treated.

+ Need only initial guess for controls q.

- Cannot use knowledge of x in initialization.

- ODE solution$x(t,q)$ can depend very non-linearly on q.

- Unstable systems difficult to treat.

3.3 Multiple Shooting

The multiple shooting [14] method discretize the the input u in the same way as the single shooting, but it also discretize the state and algebraic variables by writing them as piecewise functions. For each time step two vector of parameters s_i and r_i are introduced that represents the initial values respectively of the state and algebraic variables in the i^{th} time step, i.e. $x(t_{i-1}) = s_i$, $w(t_{i-1}) = r_i$

The successive step is to solve the DAE problem in the time step

$$
\begin{cases}
F(\dot{x}_i(t,q_i,s_i,r_i),\, w_i(t,q_i,s_i,r_i),\, q_i) = 0 & \forall t \epsilon [t_{i-1}, t_i] \\
x_i(t_{i-1}) - s_i = 0 \\
w_i(t_{i-1}) - r_i = 0
\end{cases}
\tag{3.4}
$$

to determinate the states variables x_i in each time step.

Then the cost function for the time step is calculated as

$$C_i(s_i, q_i, r_i) \coloneqq \int_{t_{i-1}}^{t_i} L(x_i(t, q_i s_i, r_i),\, w_i(t, q_i s_i, r_i),\, q_i)dt \tag{3.5}$$

which gives the total cost function $C(s, q, r) = \sum_{i=1}^{n} C_i(s_i, q_i, r_i)$.

With these premises, it is possible to add the path constraints, the starting values of the state variables, and also the constraints for the continuity of the states variables.

$$\begin{aligned}
&\textit{minimize}\\
&C(s, q, r) = \textstyle\sum_{i=1}^{n} C_i(s_i, q_i, r_i).\\
&\textit{subject to}\\
&\begin{cases}
H(x_i(t, q_i s_i, r_i),\, w_i(t, q_i s_i, r_i),\, q_i) \geq 0 & \forall t \epsilon [t_{i-1}, t_i], \forall i \epsilon [1, n]\\
G(s_0,\, r_0) = 0 & \\
x_i(t_{i-1}) - s_i = 0 & \forall i \epsilon [2, n]\\
w_i(t_{i-1}) - r_i = 0 & \forall i \epsilon [2, n]
\end{cases}
\end{aligned} \tag{3.6}$$

Compared to the previous method, the multiple shooting obviously has many more variables, but the variables are less coupled with each another, so the matrix of the problem to be solved is going to be sparse. This characteristic can be exploited by using a solver specifically written for sparse problem to achieve a significantly easier calculation. Contrary to the single shooting method, the simulation and optimization of the solution are simultaneous.

Pros:

\+ uses adaptive ODE/DAE solvers

\+ but NLP has fixed dimensions

\+ can use knowledge of x in initialization.

\+ can treat unstable systems well.

\+ robust handling of path constraints.

\+ easy to parallelize.

\- more variables than single shooting and less sparse than collocation (see below).

3.4 Direct Collocation

When the Collocation method [19, 14] is used the time interval is discretized as in the previous methods, but there is an additional parameter to be considered: the number of collocation points (m). In this method, both the u and the x are approximated as piecewise polynomial functions of degree $m-1$, these functions are obtained by interpolating the values of the inputs and variables in m points of the time step.

This interpolation can be done by using the Lagrange polynomials:

$$
\begin{array}{l}
For\, a\, generic\, function\, y(t) \\
\\
y(t) = \sum_{j=1}^{m} y_j \cdot l_j(t) \\
y_j = y(t_j) \forall j \epsilon [1,m] \\
l_j(t) = \prod_{k \epsilon [1,m]/j} \frac{t-t_k}{t_j-t_k} \\
\dot{y}(t) = \sum_{j=1}^{m} y_j \cdot \dot{l_j}(t) \\
\dot{l_j}(t) = \frac{\partial l_j}{\partial t} = \sum_{h \epsilon [1,m]/j} (\frac{1}{t_j-t_h} \cdot \prod_{k \epsilon [1,m]/j,h} \frac{t-t_k}{t_j-t_k})
\end{array}
\tag{3.7}
$$

In order to apply this polynomials to the problem and simplify calculations an auxiliary variable $\tau \epsilon [0,1]$ is defined such that $t = t_{i-1} + \tau \cdot (t_i - t_{i-1})$, and the same collocation points τ_j are used in each time step. As a consequence of this, the polynomials $l_j(\tau)$ are the same for each time step and variable. The exact values of the τ_j depends of which version of the algorithm is used, but its necessary to place one of them at the start of each time step to ensure the continuity of the state variables. Also, let us define the values of the input, state and algebraic variables in each collocation point of each time step respectively as q, s and r, which means that each variable will contribute $n \cdot m$ values.

$$
\begin{array}{l}
t_{ij} = t_{i-1} + \tau_j \cdot (t_i - t_{i-1}) \\
\begin{array}{llllll}
u_i(\tau) = \sum_{j=1}^{m} q_{ij} \cdot l_j(\tau) & \forall i \epsilon [1,n] & with & u_i(t_{ij}) = q_{ij} & \forall i,j \epsilon [1,n] x [1,m] \\
x_i(\tau) = \sum_{j=1}^{m} s_{ij} \cdot l_j(\tau) & \forall i \epsilon [1,n] & with & x_i(t_{ij}) = s_{ij} & \forall i,j \epsilon [1,n] x [1,m] \\
w_i(\tau) = \sum_{j=1}^{m} r_{ij} \cdot l_j(\tau) & \forall i \epsilon [1,n] & with & w_i(t_{ij}) = r_{ij} & \forall i,j \epsilon [1,n] x [1,m]
\end{array}
\end{array}
\tag{3.8}
$$

Under these hypotheses, the derivatives of the state variables are easily calculated by taking the derivative of the polynomials x_i without forgetting the normalized time:

$$
\dot{x}_t(\tau) = \frac{dx_i(\tau)}{dt} = \frac{d\tau}{dt} \cdot \frac{dx_i(\tau)}{d\tau} = \frac{1}{t_i - t_{i-1}} \cdot \sum_{j=1}^{m} s_{ij} \cdot l'_j(\tau) \tag{3.9}
$$

Note that if $m = 1$ the Lagrange polynomials are not defined, but this just means that x, w and u are written as piecewise constant functions. In this case simply taking the derivative of x would net $\dot{x}_i(t) = 0$ for for each time step, so its also defined as

$$\dot{x}_t = \frac{s_{i+1} - s_i}{t_{i+1} - t_i}. \tag{3.10}$$

With this we have defined all the the relevant variables as a function of q,s and r, and we can write the final form of the optimization problem to be solved:

$$\begin{aligned}
&\textit{minimize} \\
&C(q,s,r) = \textstyle\int_0^T L(x(t,s),\, w(t,r),\, u(t,q))dt \\
&\textit{subject to} \\
&\begin{cases}
F(\dot{x}(t,s),\, x(t,s),\, w(t,r),\, u(t,q,)) = 0 & \forall t \epsilon [0,T] \\
H(x(t,s),\, w(t,r),\, u(t,q,)) \geq 0 & \forall t \epsilon [0,T] \\
G(s_0,\, r_0) = 0 & \\
x_{i-1}(t_{i-1}, s_{i-1}) = s_{i,1} & \forall i \epsilon [2,n] \\
\dot{x}_t(\tau) = \frac{1}{t_i - t_{i-1}} \cdot \sum_{j=1}^{m} s_{ij} * l'_j(\tau) & \forall i,j \epsilon [1,n] x [1,m]
\end{cases}
\end{aligned} \tag{3.11}$$

Like the multiple shooting method, the direct collocation introduces a very large number of variables, especially if a high number of collocation points is used. But like the multiple shooting the matrix of the problem is sparse, so the calculation are significantly easier. Note that in this approach only the continuity of the state variables is imposed.

The main drawback of this method is that, contrary to the other to methods, the discretization of the state variables depends on the grid, so the system of equations has to be solved with a fixed time step. If this was not the case it would be possible to use variable-step solvers that can increase the accuracy of the solutions.

\+ More sparse than multiple shooting

\+ Large scale, but very sparse NLP.

\+ Can use knowledge of x in initialization (important in online context).

\+ Can treat unstable systems well.

\+ robust handling of path and terminal constraints.

\- Adaptivity needs new grid, changes NLP dimensions.

- no guarantee on the accuracy of the solution of the differential equations.

Chapter 4

Optimization with Modelica

As it was previously discussed, Modelica was created as a modeling language, and by itself it can only used for simulation purpose. But in recent years, given the growing interest in solving dynamic optimization problems, the possibility of directly optimizing a model written with Modelica was taken into consideration. There are different ways this functionality can be added:

Technically, it could be possible to use already existing features of Modelica to modify a model in an optimization problem, but the resulting code would be convoluted, and the improper use of Modelica's constructs would significantly limit the modularity that is one of the strong points of Modelica.

On the other end of the spectrum, it would have been possible to develop tool-oriented solutions, for example Graphical User Interfaces (GUIs), within a simulation-based software tool (this is the approach implemented in the commercial software Dymola). The user would then set up the optimization problem by entering information in dedicated fields in the GUI. The main drawback of this approach is a lack of flexibility since the GUI can be used only with the software and method of optimization it was designed for, and the user can only change specific parameters for the optimization.

For this work it was decided to use the extension Optimica, created by Johan Åkesson and described in [2]. Optimica is completely reliant on Modelica for the definition of the models while it introduces new types and constructs necessary to define an optimization problem that replace or enhance already existing types and constructs of Modelica.

4.1 Structure of Optimica

The main feature introduced by Optimica is a new class, "optimization", which indicates that the object will be an optimization problem, and allows the use of the other features of Optimica. Again, an example will be used to illustrate how the code is structured:

```
optimization VDP_opt(
objectiveIntegrand=(x1 - 0)^2 + (x2 - 0)^2 + (u - 0)^2,
startTime=0, finalTime=10)
  input Real u(start=0, fixed=true, free=true) "Control signal";
  Real x1(start=0, fixed=true) "First state, velocity";
  Real x2(start=1, fixed=true) "Second state, position";
equation
  der(x1) = (1 - x2^2) * x1 - x2 + u;
  der(x2) = x1;
constraint
  u <= 0.75
end VDP_opt;
```

This code solves the problem of bringing a Van der Pol oscillator, described in section 2.1, from the starting condition to being at rest at position $x_2 = 0$ in the minimum time possible, without violating a limit on the value of the control signal. The two codes can be compared to highlight the features of Optimica.

As mentioned earlier, this time the object belongs to the "optimization" class. Its variables are the same as the simulation model, but this time u is declared as an input and possesses the new attribute "free" indicating that that is the variable of the optimization. Because of this there is no equation to prescribe the value of $u(t)$, if there was the system would be already determined leaving no room for optimization. There is an additional section in the code named "constraints" where all the path constraint can be added as inequalities. The last addition are the annotation to the object located after its name that include the extremes of the optimization interval and the objective function. In this case the objective function as a time integral where the integrand is the sum of squares of the difference of the value of each variable from the objective value.

In cases of practical interest the norm, according with Modelica philosophy of modularity, is to write the model of the system separately and declare it as a variable in

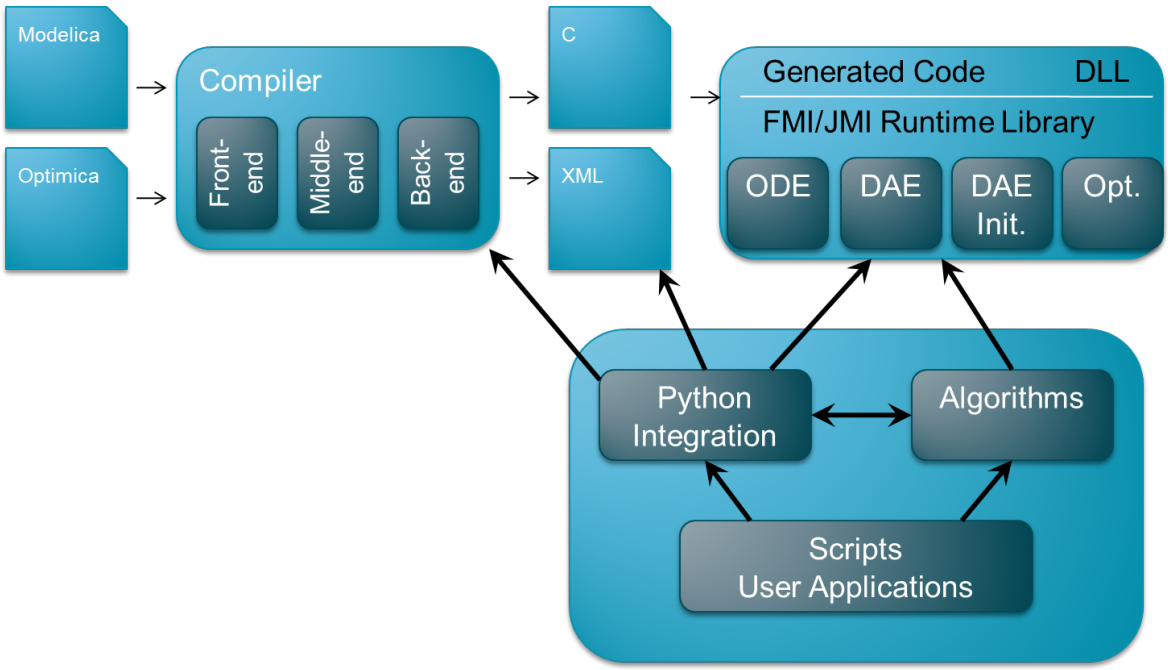

Figure 4.1: JModelica.org platform architecture.

the optimization object which will explicitly contain only the information about the optimization itself.

4.2 JModelica.org

JModelica.org [3] is an extensible Modelica-based open source platform for optimization, simulation and analysis of complex dynamic systems. It is a result of research at the Department of Automatic Control, Lund University, and is now maintained and developed by Modelon AB in collaboration with academia.

Figure 4.1 represents the architecture of JModelica, the most important aspects are:

- The compiler front-ends (one for Modelica and one for Modelica/Optimica) transforms Modelica and Optimica code into a flat model representation. The compilers also check the correctness of model descriptions and reports errors.
- The compiler back-ends generates C code and XML code for Modelica and Optimica. The C code contains the model equations, cost functions and constraints whereas the XML code contains model meta data such as variable names and parameter values.
- JModelica.org uses Python for scripting. For this purpose, JModelica.org provides a number of different Python packages that provide: integration with state

of the art DAE and ODE solvers, interactions with the JModelica.org compilers, and drivers for the optimization algorithms.

- JModelica also contains CasADi [4], a symbolic framework for algorithmic (a.k.a. automatic) differentiation and numeric optimization; and Ipopt [24], a tool for solving non-linear problems.

4.2.1 The standard solution algorithm: Direct Collocation with JModelica

The method implemented with JModelica is based on the Direct Collocation described in section 3.4. The tool start by transforming the Modelica/Optimica code of the problem into the explicit form of the DAE system, this is then transferred to CasADi which applies the collocation algorithm obtaining a problem in the form of equation 3.11 and determines the exact formulation of the first and second derivatives necessary to construct the the gradient and hessian matrix of the objective function via symbolic manipulation. Finally, CasADi interfaces with Ipopt to solve the resulting NLP where the equations of the DAE system now act as equality constraints.

4.2.2 A modified version of the solution algorithm: variable elimination

The modular approach of Modelica makes it so that the model of a system contains a great number of algebraic variables which are actually doubles of other algebraic or state variables, or could otherwise be easily eliminated by substituting their corresponding equations. For example most double variables are created by the connection of components, just by looking at the example of the electrical connections in section 2.2 it is clear that the variables v_2 to v_n could be substituted by the variable v_1 in every equation.

The method described in the previous section solves the DAE system as it is, so it has to manage all of its variables resulting in an increased computation time. Because of this an alternative version of the previous method has been experimentally implemented: in this version, while the model is transferred to CasADi the equation are subjected to the Block Lower Triangular (BLT) transformation described in [9] which reorders the equations of the system to obtain a form that can be more easily solved and eliminates a part of the algebraic variables as described above.[18]

4.3 OpenModelica

While OpenModelica was created with the aim of writing and simulating models, an optimization function is currently being developed. This feature is very much under-development at the moment, but a fully operational method of optimization is available.

4.3.1 Direct Collocation with OpenModelica

Just like with JModelica, the method currently available in OpenModelica is base on the Direct Collocation of section 3.4, though OpenModelica approaches the problem in a subtly different way.

First the the Modelica and Optimica code is read by the compiler and transformed into the relative DAE system, then the equations are manipulated with the symbolic manipulation techniques described in [9] to transform the model in a semi-ODE system

$$F(\dot{x}, x, w, u, t) \Longrightarrow \begin{cases} \dot{x} = f(x, u, t) \\ w = g(x, u, t) \end{cases} \tag{4.1}$$

This operation separates the solution of the differential equations from the algebraic ones and eliminates most of those, highly simplifying the problem at the cost of making the determination of the relevant derivatives harder.

Afterwards, the collocation algorithm is applied and the first derivatives are determined exactly via symbolic differentiation if possible, or by numerical finite differences otherwise. The second derivatives on the other hand are approximated numerically either by Modelica itself with a finite differences method from the first derivatives, or after the problem is transferred to Ipopt as part of the resolution with a limited memory Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm. Contrary to JModelica, with this method it is necessary to evaluate the derivatives anew at each iteration resulting in increased computational time.[22]

4.4 New developments

While JModelica is still being developed and expanded, it is a fairly viable tool to be used for optimization using the method of section 4.2.1. There are also other solution methods that are not yet available in the stable release of the software but have already

been successfully tested, such as the Multiple Shooting[21] described in in section 3.3, and a new algorithm that combine the Multiple Shooting and the Collocation[17].

The optimization with OpenModelica on the other hand is not as mature, but because of this it is also full of possibilities. Currently, there many experimental flags available that modify how the Collocation algorithm is implemented that could hopefully be used to improve its performance in specific cases[22]. It also being studied the possibility of integrating external tools for the calculation of the derivatives, a critical point in the optimization, such as described in [7].

Chapter 5

Case Study

The case study used to test the potential of Modelica in modeling and optimizing power plants is the start-up of a one pressure combined cycle. The process modeled and optimized is a supposed warm start-up of the plant, i.e. bringing the gas turbine from 15% to 100% of the full load, starting with a warm steam turbine shaft. The initial phase of the transient (i.e., the gas turbine start-up and the initial steam generator start-up) are not considered for simplicity. The objective is to bring the plant to full load as quickly as possible, the free variables for the optimization are the current load of the gas turbine and the de-superheating flow rate. The main constraint of the process is the preservation of the steam turbine: as the temperature of the steam rises, so does the temperature of the rotor of the turbine. But because of the significant thermal inertia, a temperature gradient forms across the rotor resulting in non-uniform dilatation and thermo-mechanical stress. Generally speaking this phenomenon also interests other thick-walled components, such as the drum of the evaporator and the superheater header, but in this work the rotor of the turbine is considered the critical component.

We will study two configuration of the plant: the complete model that uses both the load of the gas turbine and the attemperation to control the start-up, and a baseline model that uses only the load.

The model in [10] was used as a starting point to write the model used in this thesis. That model was a very simplified version of a combined cycle, among other issues the steam is approximated as a perfect gas, but it provided some components for this work.

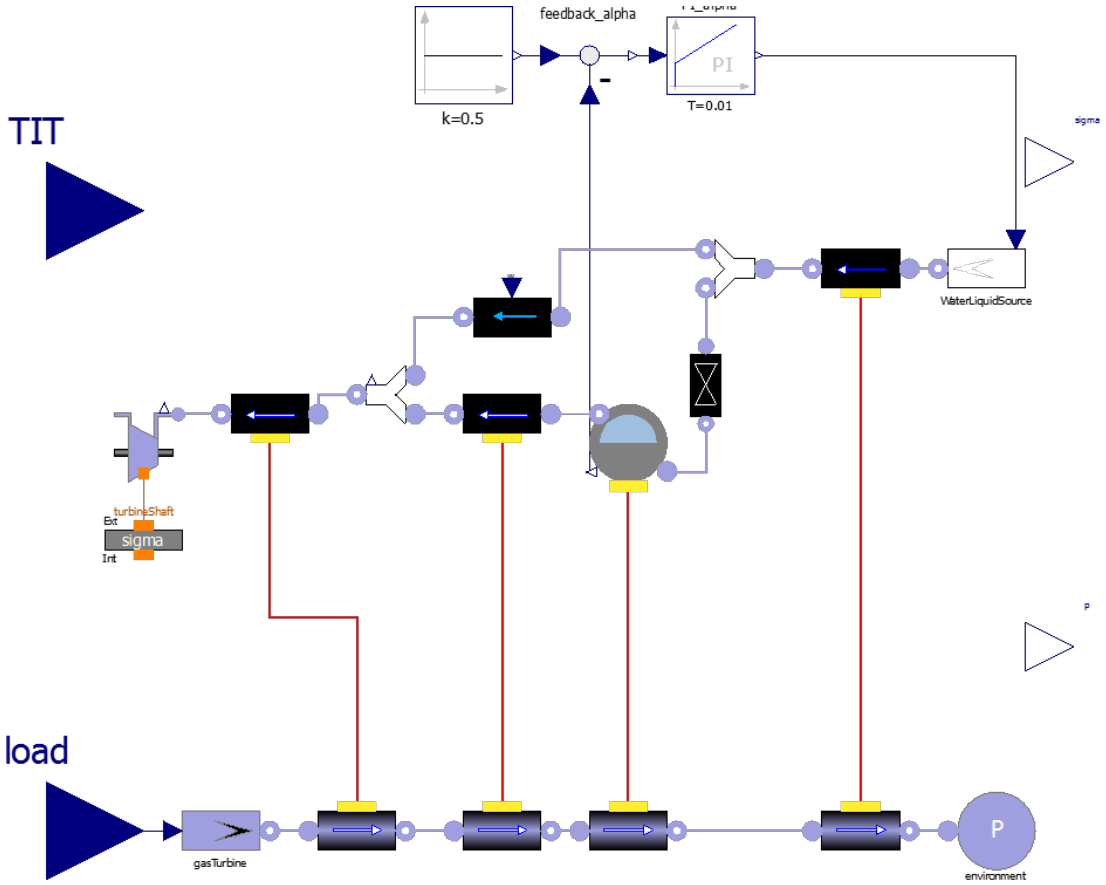

Figure 5.1: Outline of the model of the plant.

5.1 Details of the model

The quantitative data for the modeling of the combined cycle have been taken from a plant studied in the course "Sistemi Energetici Avanzati" [1]. The main parameters of the cycle at full load are reported here:

Gas Turbine	
Power[MW]	68.095
Turbine Outlet Temperature (TOT)[K]	815
Exhaust gas mass flow[kg/s]	197.6
Steam Turbine	
Power[MW]	29.161
Evaporator Pressure[MPa]	3.240

The outline of the model, as it appears in OMEdit, is reported in figure 5.1. The following sections will cover the main components of the plant and modeling assumption made for each of them.

5.1.1 Fluids models

While the exhaust gas of the gas turbine can be reasonably approximated as an ideal gas with constant specific heat, the water and steam are not so easily approximated.

The main resource for a model of water and steam, is the IF95 formulation by the The International Association for the Properties of Water and Steam (IAPWS)[20], but the correlation implemented in that model are implicit and fairly complicated, making them not suitable for simulation purpose, let alone optimization. Modelica contains the IF97 formulation that uses explicit correlation that are less complicated than the IF95, but they still impose an heavy computational load and present occasional discontinuities that make it unsuitable for our purpose.

So, the approach used was to consider the three states of the water that come into play: subcooled liquid, saturation, and superheated vapor, and find independent functions for the variation of the properties of interest in each region and at a later stage ensure their continuity.

Saturation First the saturation region is considered. In this state the fluid has only one degree of freedom, so all the properties are written as a function of pressure in the form of polynomials whose coefficients obtained by means of linear regression of the values of said properties taken from IF97 tables.

Subcooled liquid Second, in the subcooled region, the properties of interest are the specific volume (v) and the specific entalpy (h). By approximating the liquid water as an incompressible fluid, and considering the boundary condition with the saturation region, it is possible to write the following equations:

$$\begin{cases} dv(T,p) = 0 \\ v(T, p_{sat}(T)) = v_{sat}(T) \end{cases} \tag{5.1}$$

$$\begin{cases} dh = c_p \cdot dT + v * dp \\ h(T, p_{sat}(T)) = h_{sat}(T) \end{cases} \tag{5.2}$$

The dependence of the specific volume on the pressure is neglected, and the correlation of the saturation region is re-used:

$$v(T,p) = v_{sat}(T) \tag{5.3}$$

Since we have already determined an expression for the specific volume, and its variation is less pronounced than the specific heat's, the specific entalpy is evaluated as:

$$h(T,p) = h_{sat}(T) + v_{sat}(T) \cdot (p - p_{sat}(T)) \tag{5.4}$$

Superheated vapor Finally, the superheated steam is modeled using a cubic equation of state based upon a truncated virial expansion as described in [15]:

$$p_r = \frac{T_r}{v_r} \cdot \left[1 + \left(C_1 + \frac{C_2}{T_r^{\alpha}} + \frac{C_3}{T_r^{\beta}} + \frac{C_4}{T_r}\right) \cdot \frac{1}{v_r} + \left(C_5 + \frac{C_6}{T_r^{\gamma}} + \frac{C_7}{T_r^{\delta}} + \frac{C_8}{T_r}\right) \cdot \frac{1}{v_r^2}\right] \tag{5.5}$$

The subscript "r" indicates that all properties are written as dimensionless numbers using the critical temperature and pressure, the gas constant R and the molar mass of water.

The twelve constants of the equation are determined by fitting the specific heat at constant volumes and the derivatives of pressure in respect to specific volumes and temperature with the data obtained from the steam tables of the National Institute of Standards and Technology reported in [16].

With this information we can find the specific volume as as a function of p and T using equation (5.5). As for the specific entalpy, we will proceed similarly to how we did for the subcooled liquid. From thermodynamics:

$$\left(\frac{\partial h_r}{\partial T_r}\right)_{v_r} = c_{v,r} + v_r \cdot \left(\frac{\partial p_r}{\partial T_r}\right)_{v_r} \tag{5.6}$$

With this, using the boundary condition with the saturation region, we obtain:

$$h_r(T_r, v_r) = h_{r,sat}(v_r) + \int_{T_{r,sat}}^{T_r} \left(\frac{\partial h_r}{\partial \theta}\right)_{v_r} d\theta \tag{5.7}$$

5.1.2 Gas Turbine

Because the dynamics of the gas turbine (GT) are much faster than those of the steam cycle, it is approximated by a generator of hot gas whose temperature and mass flow(w) are an algebraic function of the load. In accord with the methods of regulation of a GT with variable inlet guide vanes (VIGV), up to 50% of the full load the VIGV are

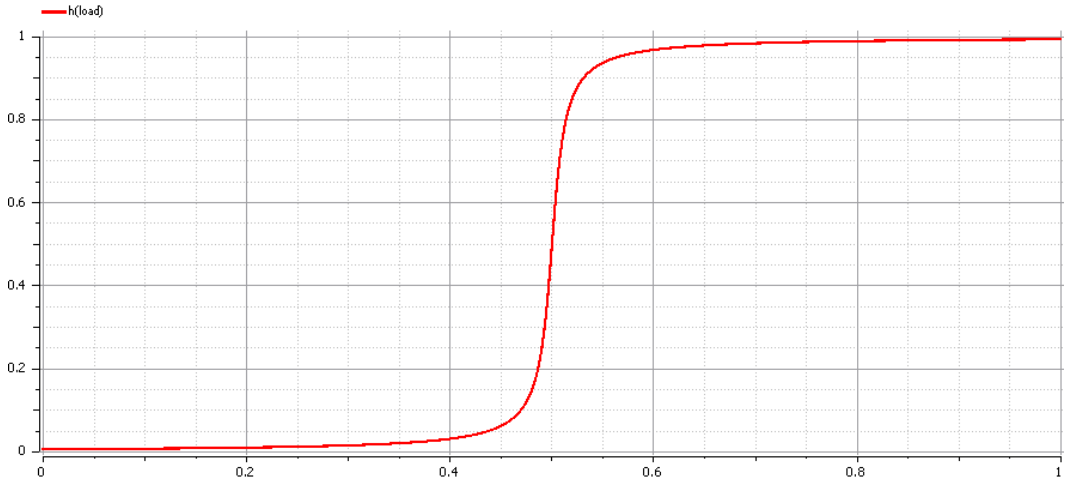

Figure 5.2: h(load)

supposed to be closed and the exhaust gas mass flow is constant while the Turbine Outlet Temperature (TOT), which is the temperature of the exhaust gasses at the outlet of the GT, grows linearly with load. Afterwards, up to full load, the VIGV are supposed to be progressively opened and the exhaust gas mass flow grows linearly with the load while the TOT is constant.

$$\begin{cases} TOT = TOT_{min} + \frac{TOT_{max} - TOT_{min}}{0.5 - 0.15} \cdot (load - 0.15) & load \leq 0.5 \\ TOT = TOT_{max} & load > 0.5 \end{cases} \tag{5.8}$$

$$\begin{cases} w = w_{min} & load \leq 0.5 \\ w = w_{max} + \frac{w_{max} - w_{min}}{1 - 0.5} \cdot (load - 1) & load > 0.5 \end{cases} \tag{5.9}$$

The behavior just described would result in the TOT as a function of the load and the exhaust gas mass flow as a function of the load being piecewise defined functions that are continuous but not smooth, as they would present a discontinuity of the first derivative when the load is 50% of the maximum.

This fact would cause problem during the optimization because the derivative are not defined in that point, so the functions are modified to into continuous functions:

$$TOT = \left[TOT_{min} + \frac{TOT_{max} - TOT_{min}}{0.5 - 0.15} \cdot (load - 0.15)\right] \cdot [1 - h(load)] + TOT_{max} \cdot h(load) \tag{5.10}$$

$$w = w_{min} \cdot [1 - h(load)] + \left[w_{max} + \frac{w_{max} - w_{min}}{1 - 0.5} \cdot (load - 1)\right] \cdot h(load) \tag{5.11}$$

where $h(load)$ is a continuous approximation of the Heaviside step function based on the arctangent function as it can be seen in figure 5.2.

5.1.3 Heat exchangers

The Heat Recovery Steam Generator (HRSG) is modeled as eight separate components as shown in figure 5.1. There are four banks of heat exchangers: the economizer, the evaporator, and two parts for the superheater; and each of them has two components, one for the gas side and one for the water/steam side. Aside from the evaporator on the steam side, all the banks are structured in the same way.

First a thermal "node" is defined that contains the mass and energy balance equations for a finite volume:

$$\frac{\partial M_i}{\partial t} = w_{in,i} + w_{out,i} = 0 \tag{5.12}$$

This mass balance states that there is no variation in the amount of fluid contained in the finite volume. Note that according to Modelica's conventions, all the flow variable are considered positive when entering the volume.

$$\frac{\partial E_i}{\partial t} = C_i \cdot \frac{\partial T_{out,i}}{\partial t} = w_{in,i} \cdot h_{in,i} + w_{out,i} \cdot h_{out,i} + \dot{Q}_i \tag{5.13}$$

In this energy balance C is the total heat capacity of the fluid contained in the volume and the metal walls, while the temperature used to estimate the energy is the temperature of the fluid at the outlet of the volume. The first two terms of the right side of the equation are the advective contributions of the mass flows entering and exiting the volume, while $\dot{Q}$ is the heat flow exchanged with the other side of the heat exchanger.

The are three thermal resistances in the heat exchange between the two sides of the HRSG: convective exchange between gas and metal wall, conductive exchange through the metal wall, and convective exchange between metal wall and water. Given the high conductivity of the metal and the relatively high convective exchange of steam, those thermal resistance are neglected and the metal wall on the gas side is considered to have the same temperature of the water.

The temperatures used to model the heat flow are the arithmetic means of the inlet and outlet temperature of the fluids.

$$T_{mean,i} = \frac{T_{in,i} + T_{out,i}}{2} \tag{5.14}$$

So the heat flow is calculated as:

$$\dot{Q}_i = G_i \cdot (T_{wall,gas,i} - T_{wall,water,i}) \tag{5.15}$$

The parameter G is the thermal conductance relative to the convective heat exchange on the gas side. The value of G_i has been modeled as a function of the mass flow of the exhaust gas ($w_{gas,i}$) as a power law:

$$G = G_{n,i} \cdot \left(\frac{w_{gas,i}}{w_{gas,i,n}} \right)^k \tag{5.16}$$

where G_n and w_n are the values of the thermal conductance and mass flow at full load.

Each bank of the HRSG on each side is modeled as a number of these thermal nodes connected in series, outlet of each one connected to the inlet of the next

$$w_{in,i+1} = -w_{out,i} \tag{5.17}$$

$$T_{in,i+1} = T_{out,i} \tag{5.18}$$

The thermal ports of each node of each component are connected to the thermal ports of the component representing the other side of the bank. The ports are connected in reverse order, first port of one side to the last port of the other, to model the counter-current heat exchange.

$$\dot{Q}_{i,water} = -\dot{Q}_{m-i+1,gas} \tag{5.19}$$

where m is the total number of nodes of the bank.

The number of thermal ports used for each banks determines the accuracy and complexity of the model: more nodes means a better approximation of the heat exchange at the cost of increased numerical complexity and vice-versa.

The distributed pressure drop on the pipes is neglected, but the presence of a valve between the economizer and the evaporator is acknowledged to prevent evaporation of the liquid in the economizer.

Evaporator The evaporator on water side is modeled differently from the other components of the HRSG since it has to account for the phase change. Note that

this component contains all parts of the evaporator: the drum, the risers and the downcomers.

This model is based on the one described in [5]. The same balances of mass and energy previously described are used, but since the water is in a two-phase state the temperature is dependent on the pressure. So the state variables used are the pressure of the drum and the volumetric fraction of vapor.

In order to prevent both over and underflow a controller is set up to regulate the mass flow supplied by the condensation pump so that the volumetric fraction of vapor in the evaporator is about 50%.

5.1.4 Steam turbine

The steam turbine is supposed to follow Stodola's ellipse law. Also, since the expansion ratio is high and the variation of the temperature at inlet is low compare to the inlet temperature itself, the law is simplified into:

$$w_{steam} = \frac{w_{steam,n}}{p_{in,n}} \cdot p_{in} \tag{5.20}$$

where w_{steam} is the mass flow of the steam in the turbine, p_{in} is the inlet pressure, and the subscript "n" indicates the properties at full load. So the steam elaborated by the turbine depends linearly on the inlet pressure.

5.1.5 Rotor of the steam turbine

The rotor is modeled as an hollow cylinder, the heat flow is considered one-dimensional in the radial direction. The temperature distribution is approximated using eight temperature nodes distributed linearly from the internal to the external radius. The internal wall is supposed to be adiabatic, while the external wall is supposed to be at the same temperature of the steam due to the relatively high heat transfer coefficient during the transient of interest.

The stress on the outer surface of the shaft is calculated following the theory of linear deformation as described in [13]:

$$\sigma = \frac{\alpha E}{1-\nu} \cdot (T_{ext} - T_{mean}) \tag{5.21}$$

where α is the coefficient of thermal dilatation, E is Young's modulus, ν is Poisson's ratio, T_{ext} is the temperature on the external surface of the shaft, and T_{mean} is the mean radial temperature of the shaft.

For simplicity sake, the initial condition chosen for the shaft is that all the temperature nodes are at the starting temperature of the steam. While this specific temperature profile is unlikely, it is a reasonable starting condition.

5.1.6 Attemperation

The attemperation consist in drawing some water from the economizer and inject it in the last part of the superheater in order to lower the temperature of the steam and protect the most thermally strained part of the steam cycle, i.e. the last banks of pipes of the superheater and the rotor of the first stage of the turbine.

The control system is set up as two nested control loops: the internal loop controls the temperature of the steam immediately after the attemperation using the value of the liquid mass flow, while the outer loop controls the temperature of the steam at the inlet of the turbine using the set point of the internal loop. Since the dynamics of the internal loop is significantly faster than the external loop, the internal loop is approximated as an ideal control where the controlled variable perfectly follows the set-point.

5.2 Set-up of the optimization

In order to better explain the final set up of the optimization problem the Optimica code used is reported here with a following explanation:

```
optimization StartupAtt
(objectiveIntegrand = (u1 - 1)^2 +
                      0.01 * (plant.w_att / w_att_max - 0)^2,
                      startTime = 0, finalTime = 5000)
  CombinedCycle.Optimization.Plants.CC_Att_WarmStartup plant;
  Real u1(start = 0.15, fixed = true, min = 0, max = 1);
  input Real du1;
  input Real u2;
  parameter Real sigma_max = 2.8e8;
  parameter Real w_att_max = 1.5;
equation
  u1 = plant.load;
  du1 = der(u1);
  u2 = plant.TIT;
constraint
  du1 >= 0;
  du1 <= 0.1/60;
  plant.sigma / sigma_max <= 1;
  plant.w_att >= 0;
  plant.w_att / w_att_max <= 1;
end StartupAtt;
```

Leaving the cost function aside for a moment, we can see that the code starts by importing the full model of the plant with all its equations and variables.

After that a variable u1 and an input du1 are introduced, in the equation section they are defined as equal to the load of the gas turbine and its time derivative, and together they represent the first input of the optimization problem. The derivative du1 is introduced in order to use it in the constraints.

Then the second input u2 is introduced, in the equation section it is defined as equal to the set point temperature of the attemperation.

At the end of the section two constant parameters are introduced, they are the maximum stress allowed on the rotor of the steam turbine, set at 280MPa; and the maximum flow of water drawn from the economizer for the attemperation, set at roughly 1% mass flow of water at full power.

The value of the maximum stress has been chosen in order to limit the life-time con-

sumption of the component, whose details are outside of the scope of this work, under the hypothesis that the stress is not subjected to cycles that would induce fatigue. This hypothesis is verified a posteriori from the results of the optimization.

Lastly, the constraints of the problem are enforced: apart from the already discussed limits on the stress and attemperation, the load is defined as non-decreasing and its increase is limited at 10% of the full load every minute, and the attemperation mass flow is defined as non-negative. Physically, it is impossible to have an inversion of the attemperation mass flow, but the controller would choose negative values of the mass flow if the measured temperature is lower than the set-point. In reality this is solved with a saturation of the control, but since in our model we have used an ideal controller to simplify the process we have to impose this constraint during the optimization.

Now we can interpret the cost function: it penalizes both having a load of the gas turbine lower then the full load and using the attemperation, but since the attemperation has a weighting coefficient of 0.01 its contribute is much smaller and becomes relevant only when the turbine is close to full load.

It also important to note that, regardless of the software used to solve the optimization problem, it is necessary to run a reference simulation to be used as the initial guess of the solution algorithm. Without a user provided initial guess the solver defaults to set the inputs to zero for the initial guess, but that is not a feasible solution for the current model. The actual inputs of this simulation are not critical, while choosing inputs closer to the optimal solution will lead to a faster convergence, a good software should be able to eventually reach convergence from most feasible starting trajectory. The reference simulation used in this work is a regular increase of the load up to full load in about 10000 s coupled with a weak attemperation across the whole transient.

5.3 Solution of the Optimization

The optimization problem that has been described admits only one solution. At the end of the optimization process all the software and methods converge to a more or less precise approximation of it. We will now discuss the solutions of both the baseline and the attemperation model with the aid of the graphs produced by JModelica of the most relevant variables, green lines for the baseline model and blue lines for the attemperation model.

In figure 5.3 we can see the load of the gas turbine. The trends of the two curves are similar, but it is clear that by using the attemperation its possible to reach the full load more than 1000 s earlier.

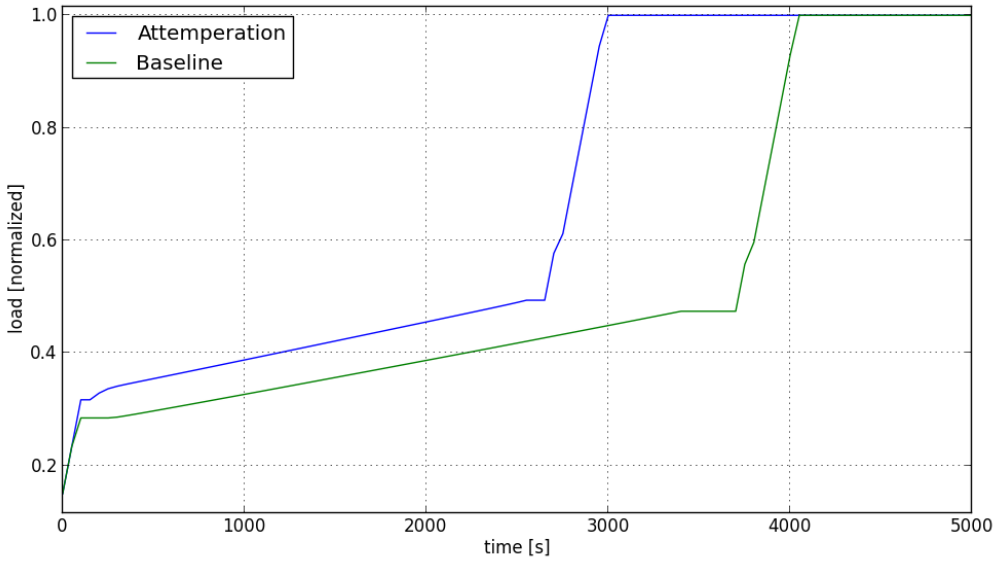

Figure 5.3: Solution of the optimization - Load.

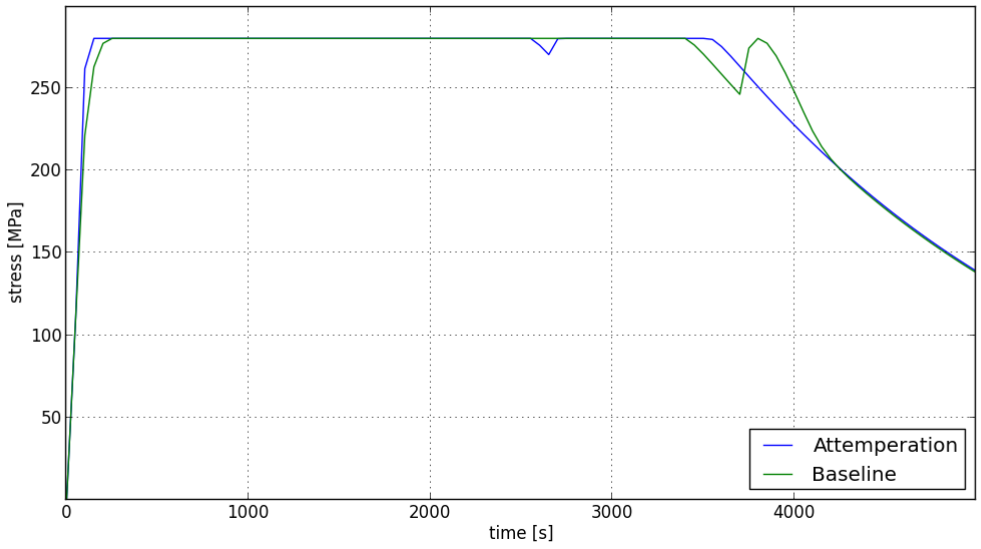

Figure 5.4: Solution of the optimization - Stress.

As for the trend itself, we can see that at first the load is increased as fast as possible, this causes a significant temperature gradient between the cold shaft of the turbine and the steam whose temperature grows rapidly since at this point increasing the load of the gas turbine means increasing the temperature of the exhaust gas. As it can be seen in figure 5.4, this continues until the stress reaches the maximum allowed value, at which point the rate of increase of the load is significantly reduced to match the rate of increase of the steam temperature to the thermal diffusion in the blades of the turbine. When the load reaches 50% of the full load, the regulation logic of the gas turbine changes, the temperature of the exhaust gas remains constant, while the mass flow increases, so the temperature of the steam starts increasing more slowly with the load. This permits to use an higher rate of increase and quickly reach full load.

The graph of the stress of both models present some irregularities around the time when the respective load reaches 50%. This is due to the strong non-linearity of the regulation logic of the gas turbine and it will be described in detail in section 6.3.

Figure 5.5 reports the temperature of the steam at the inlet of the turbine. The two lines are almost coincident in this graph, showing that the trajectory of the stress mainly depends on trajectory of the inlet temperature. The baseline model can obtain this profile only by modulating the load of the gas turbine, while the attemperation model can use the attemperation to reduce the temperature of the steam at a given load, or vice-versa admit an higher load for a given value of the temperature.

From the graph of the water mass flow used for the attemperation of figure 5.6 we find further confirmation of this analysis. The attemperation is used as much as possible until the full load has been reached. If the constraint on the maximum mass flow were relaxed, the attemperation would still be used as much as possible, and the trajectory of the load would still have the same trend, but it would be translated to the top and left.

Finally, figure 5.7 shows the evaporation pressure. The main information that can be gathered is that the evaporation pressure of the steam cycle closely follows the load of the gas turbine.

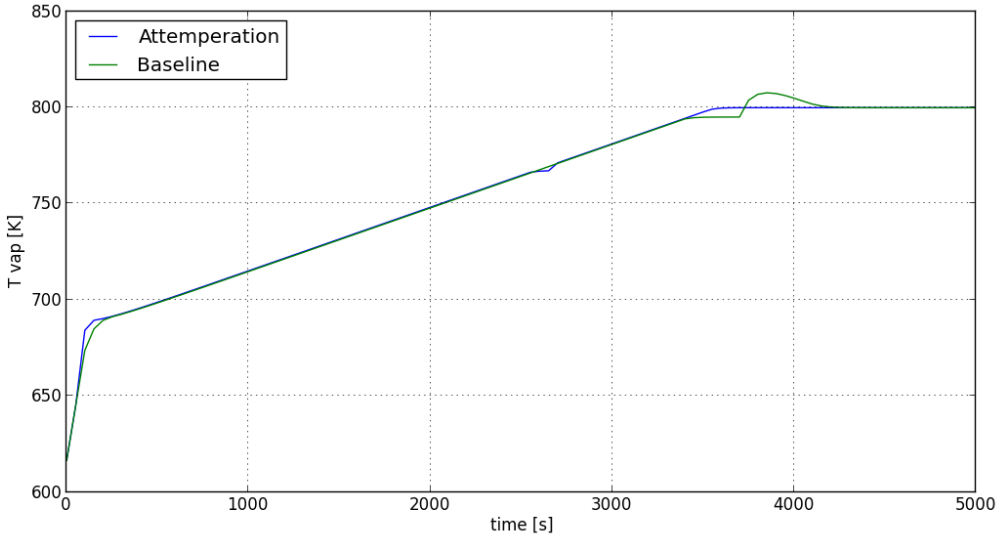

Figure 5.5: Solution of the optimization - Turbine Inlet Temperature.

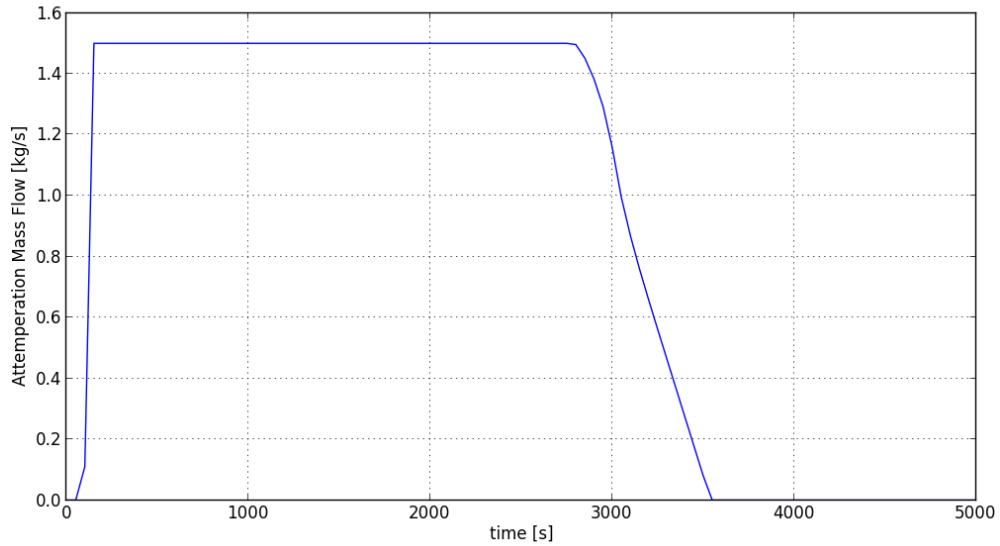

Figure 5.6: Mass flow drawn for the attemperation

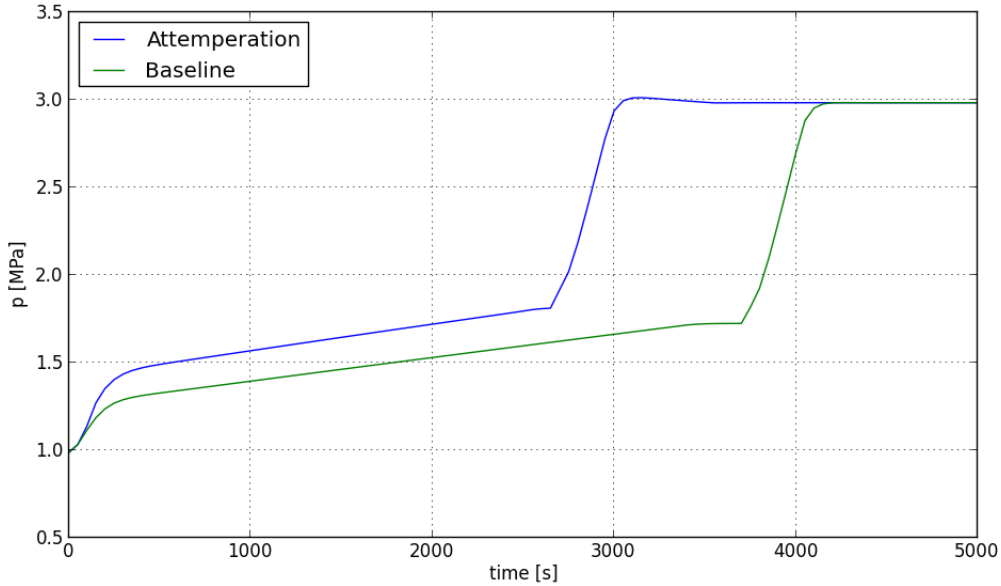

Figure 5.7: Solution of the optimization - Evaporator Pressure.

Chapter 6

Results of the test campaign

We will now summarize how the test campaign was conducted and the parameters of the performance analysis.

We have selected three optimization methods to be tested:

- standard Direct Collocation with JModelica (JM_{st});
- Direct Collocation with elimination of algebraic variables with JModelica (JM_{ve});
- Direct Collocation on ODE with OpenModelica (OM).

We have prepared two model to whom the methods can be applied:

- start-up of a Baseline Combined Cycle power plant (BASE);
- start-up of a Combined Cycle power plant with the addition of the attemperation (ATT).

Referring to the nomenclature of section 3.4, the methods were tested on both models several times with different discretizations of the time interval, the number of time steps used are $n = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]$, while the number of collocation points is one for all the tests, $m = 1$. We will also refer to the boundary of the time steps as evaluation points.

Since the model used is fairly simplified, the ideal trajectory of our model is only an approximation of the ideal trajectory of the real plant. Because of this, increasing the number of collocation points in order to achieve a better approximation of solution of the model would be useless since it would add numerical complexity without achieving a useful payoff.

Note: Because of problem of convergence of the baseline model the data for $n = [90, 100]$ for the JM_{st} method are not available, and JM_{ve} has been been tested on slightly different values of n, $n = [10, 20, 32, 40, 52, 58, 71, 80, 90, 100]$. The cause of this behavior will be discussed further on.

The quantities monitored to compare the performance of the algorithms, as given by the output of the solver, are:

- "Total number of variables" (N_{var}): the number of variables of the optimization problem after it has been discretized and converted to a NLP;
- "Number of Iterations" (N_{iter}): the number of iterations needed for the method to reach convergence;
- "Total CPU secs in IPOPT (w/o function evaluations)" (t_{NLP}): the time used for solving all the iteration of the NLP minus the time spent evaluating the relevant functions.
- "Total CPU secs in NLP function evaluations" (t_{eval}): the time spent evaluating the necessary function in each iteration, such as the objective function and the derivative necessary for the gradient and the hessian matrix.

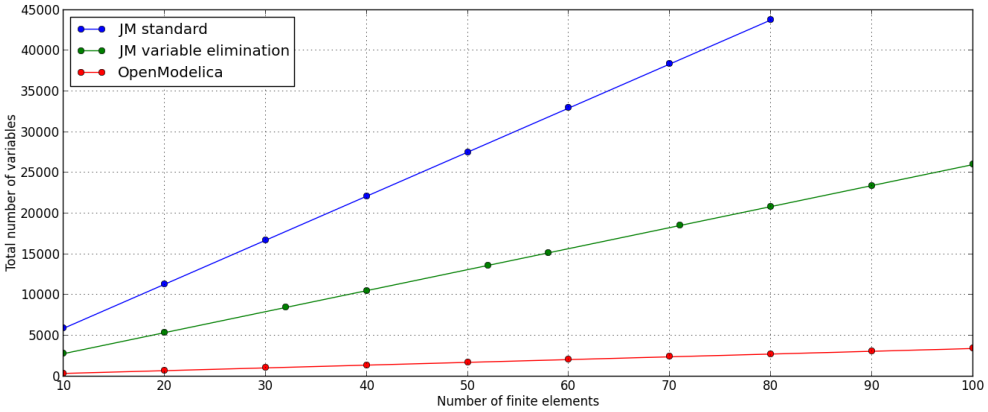

Figure 6.1: Baseline model - Number of Variables

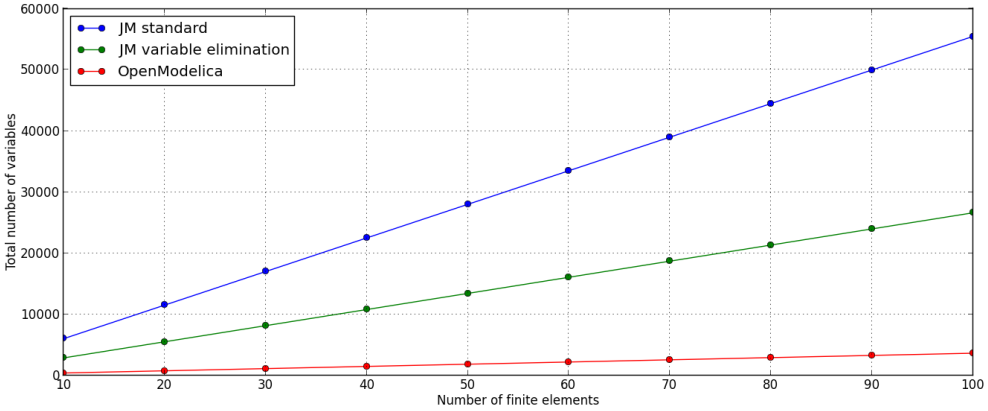

Figure 6.2: Attemperation model - Number of variables

6.1 Number of Variables

Figures 6.1 and 6.2 report N_{var} as a function of n for each method respectively for the Baseline and the Attemperation models. As it can be easily seen the trend is almost linear in every case. Also, while the Attemperation model has more variables than the Baseline one given the same method, the difference is not very significant.

By comparing the methods we can see that the variable elimination method of JModelica works as intended, it produces roughly half as many variables as the standard method for every value of n. OpenModelica is even better from this point of view: by transforming the problem to ODE form it further reduces the number of variables to less than one tenth of the standard JModelica algorithm.

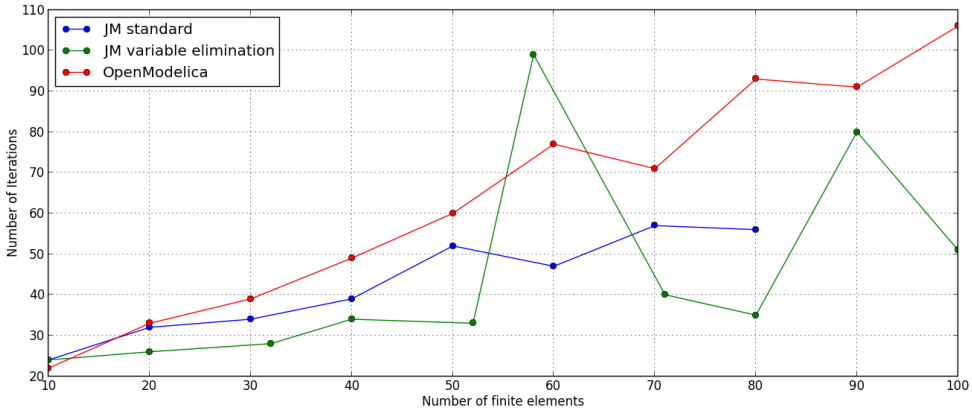

Figure 6.3: Baseline model - Number of Iterations

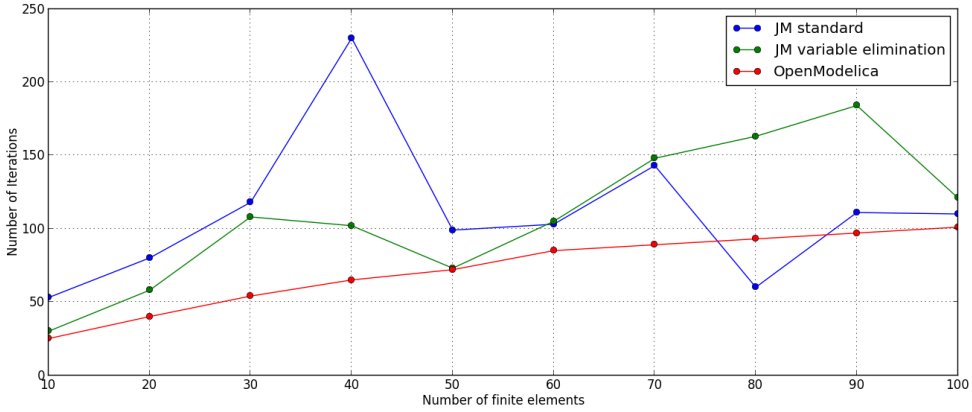

Figure 6.4: Attemperation model - Number of Iterations

6.2 Performance analysis

Compared to the number of variables, it is more difficult to individuate a clear trend in the number of iterations, figures 6.3 and 6.4 for Baseline and Attemperation models. OpenModelica shows a steady increase in the number of iteration, it even requires almost the same number of iterations for both model, but JModelica presents significant high spikes in the number of iterations with both methods. These spikes occur for specific values of n, and are caused by the non-linearity in the behavior of the gas turbine described in section 5.1.2.

Depending on the number of time steps used, the disposition of the evaluation points across the time interval changes. The non-linear behavior can only be captured if the evaluation points are dense enough in the region where the non-linearity manifest itself. This is also the reason why the variable elimination algorithm of JModelica applied to

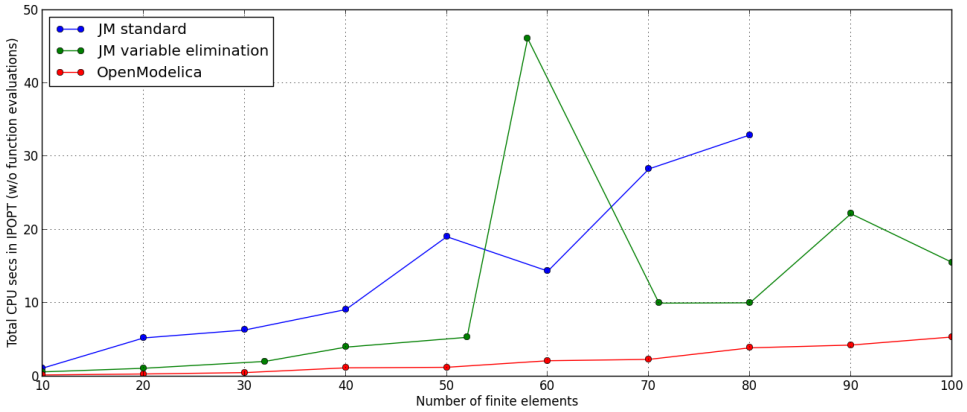

Figure 6.5: Baseline model - NLP solution time

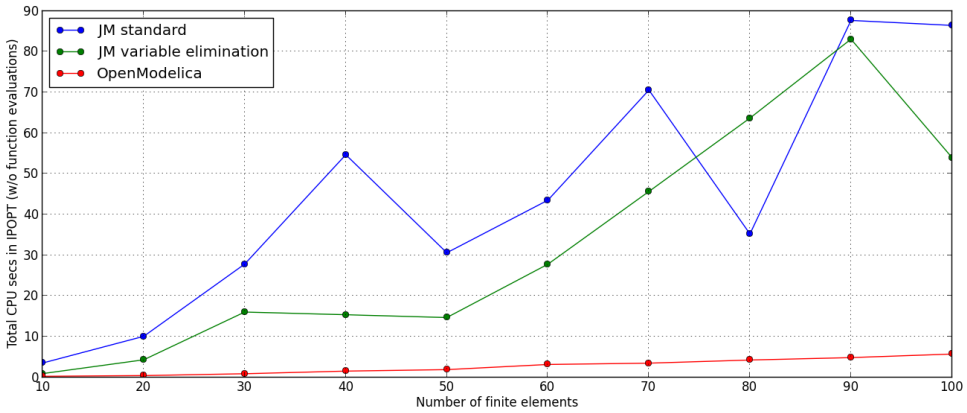

Figure 6.6: Attemperation model - NLP solution time

the Baseline model could not converge at all for certain values of n and, as mentioned earlier, in order to obtain data it was necessary to test it for different values of n. It is important to note that OpenModelica did not suffer this problem as much as JModelica.

As mentioned at the start of the chapter, the total time of the optimization has been divided in two parts t_{NLP} and t_{eval}. Figures 6.5 and 6.6 show t_{NLP} for the two models. It is immediately apparent that OpenModelica spends significantly less time than both of JModelica's algorithms in the solution of the NLP. Similarly, excluding the same spikes seen in the iteration graphs, the variable elimination algorithm spends about half the time spent by the standard algorithm. This was expected since OpenModelica transforms the problem in ODE form and eliminates as many algebraic variables as possible, making the NLP problem itself as simple as possible. The same happens with the variable elimination algorithm, albeit to a lesser extent.

By comparing the t_{eval}, the opposite pattern emerges. Figure 6.7 and 6.8 show the time spent in function evaluation only for the two JModelica algorithms, while the graph for OpenModelica for both models is in figure 6.9. The graphs were separated because the t_{eval} of OpenModelica is many times higher than the t_{eval} of JModelica. Again, this was expected, since OpenModelica needs to re-evaluate the derivatives on each iteration. The two JModelica algorithms continue to present the usual oscillations, the actual difference between the two algorithms on this parameter is negligible.

If one were to compare to total time of solution, OpenModelica turns out to always be the slowest software by far, but on the other hand it is also the most stable: it always converges to a solution and the total time required for the solution depends only on the number of variables and not on the discretization of time interval itself. While the solution time should be improved, this reliability is a very important and useful feature.

Regarding the two JModelica algorithms, they both suffer the same stability problem, but the variable elimination variant is definitely faster than the standard algorithm, so it can definitely be called an improvement.

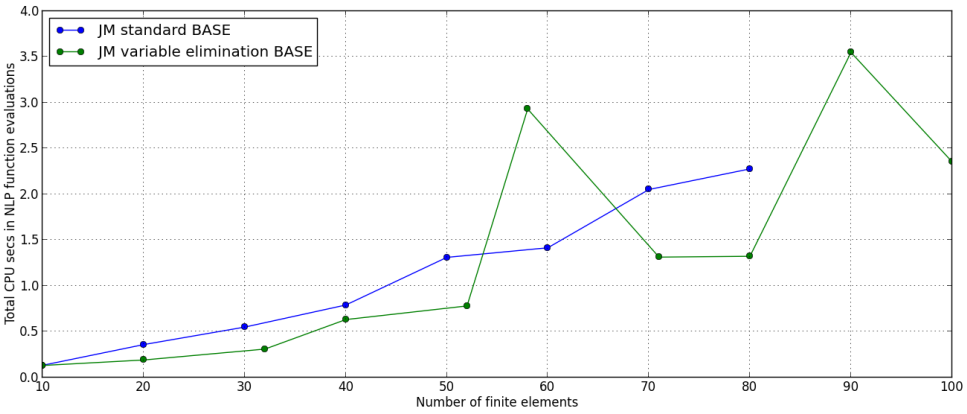

Figure 6.7: Baseline model - Function evaluation time

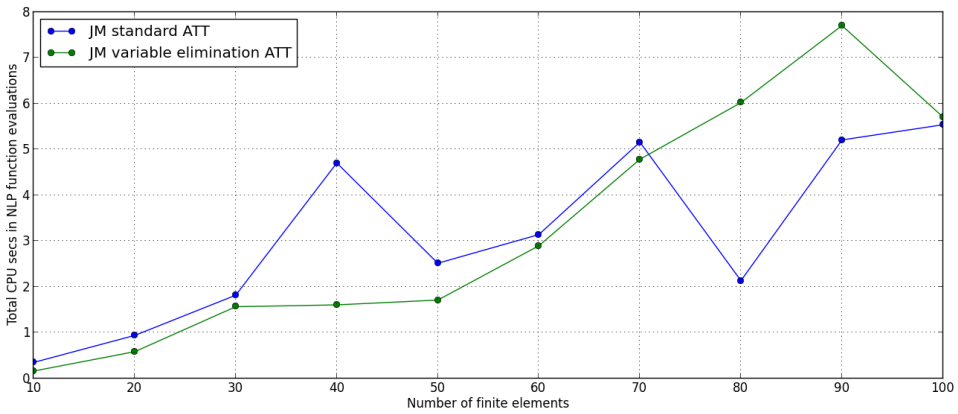

Figure 6.8: Attemperation model - Function evaluation time

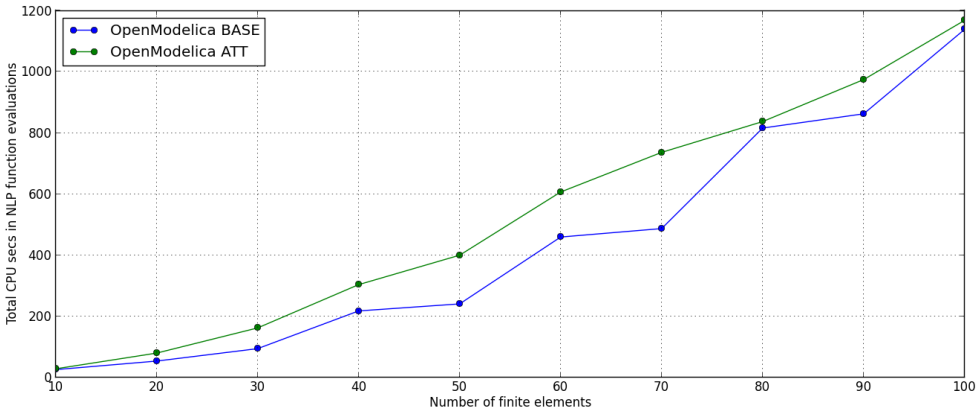

Figure 6.9: OpenModelica - Function evaluation time

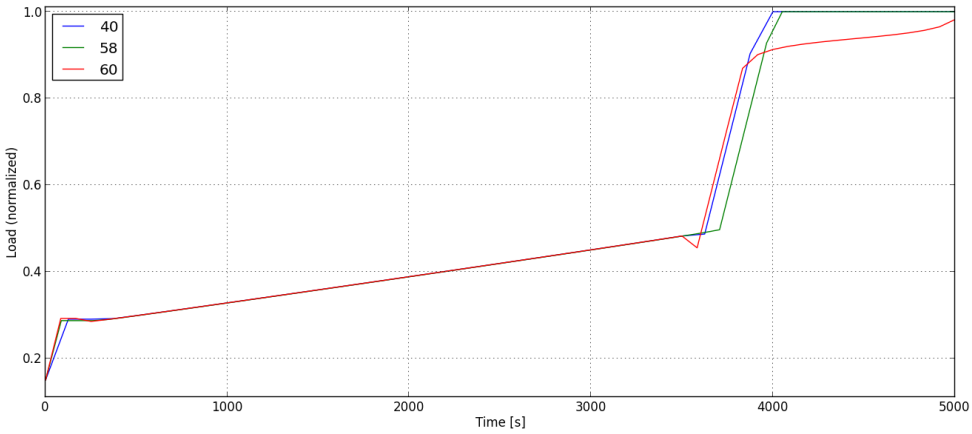

Figure 6.10: Baseline model - JM_{ve} - Load

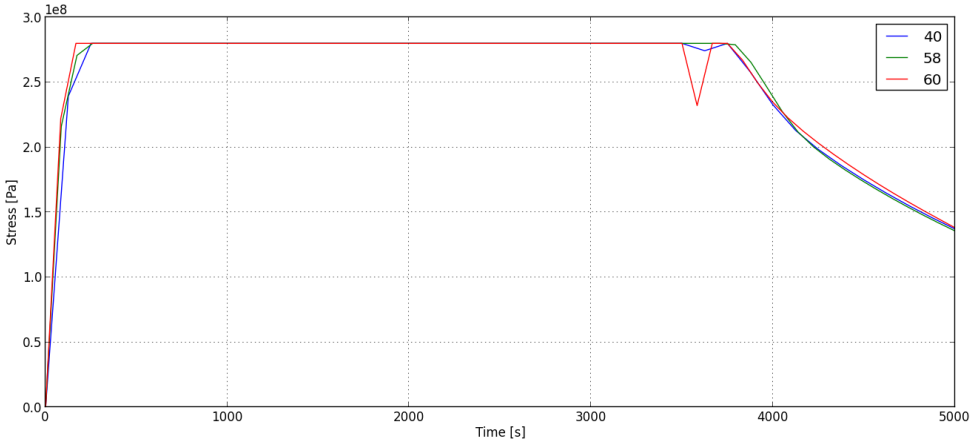

Figure 6.11: Baseline model - JM_{ve} - Stress

6.3 Discussion of the non-linearity

We will now analyze in detail the effects of the non-linear behavior of the gas turbine on the optimization. In order to do this we will compare the solutions found by the variable elimination algorithm of JModelica applied to the Baseline model with the numbers of time steps $n = [40, 58, 60]$. From the previous section we have seen that $n = 40$ converges fast, $n = 58$ converges very slowly, while $n = 60$ does not converge at all. In figures 6.10 and 6.11 the trajectories of the load and the stress for each solution are plotted. Note how the solutions start diverging significantly only only when the 50% of the full load is reached. That interval of time corresponds to where the $n = 40$ and $n = 60$ solutions present the irregularity in the trajectory of the stress. It is also worth noting how the $n = 60$ solution violates the constraint on the non-decreasing

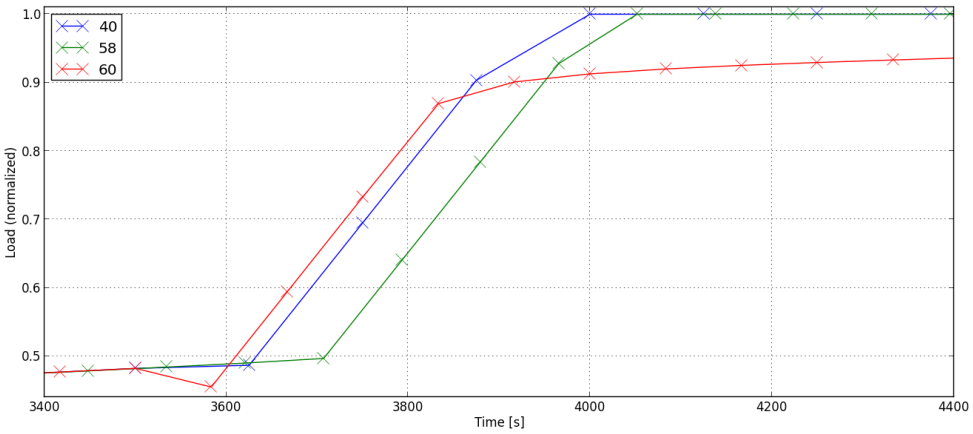

Figure 6.12: Baseline model - JM_{ve} - Load, detail

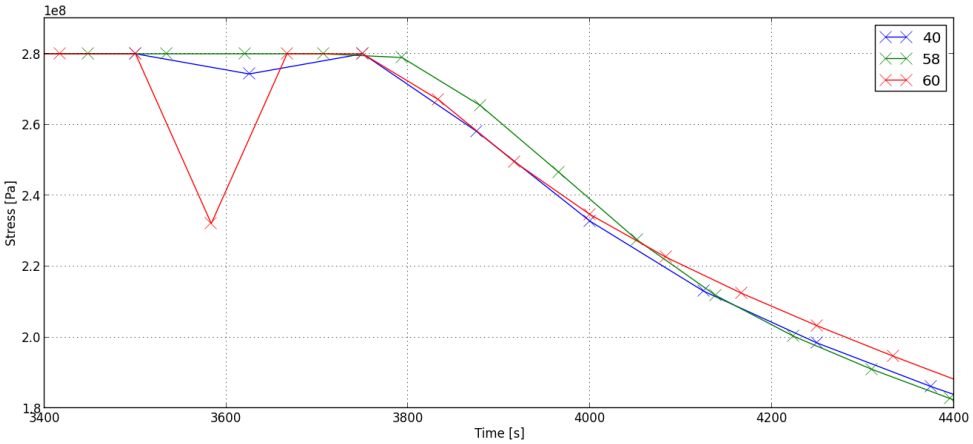

Figure 6.13: Baseline model - JM_{ve} - Stress, detail

load.

Figures 6.12 and 6.13 are the enlargements of the previous graphs in interval from $t = 3400s$ to $t = 4400s$ enhanced with x mark to indicate the evaluation points. From these figures it can be seen how heavily the discretization of the time interval influences how the non-linearity of the gas turbine is approximated. It turns out that discretization of the $n = 58$ solution is the best one, but it would have been impossible to deduce this a priori.

For comparison, figures 6.14 and 6.15 report the same graphs of the previous figures for OpenModelica. If we compare the graphs for the same number of evaluation points, we see that the trajectory are almost the same, but OpenModelica produces smoother trajectory in the non-linear region. Specifically the $n = 40$ solution is completely smooth, and for the $n = 60$ solution the downward spike is significantly reduced. This

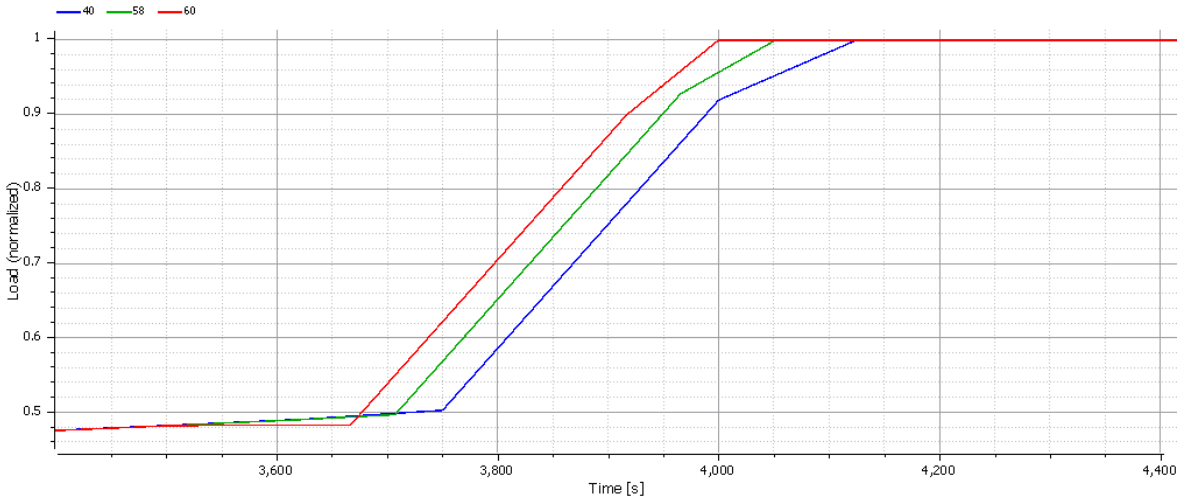

Figure 6.14: Baseline model - *OM* - Load, detail

reinforces our conclusion that OpenModelica is more stable than JModelica.

The simplest way to handle this problem would be to use a very large number of time steps to accurately approximate all non-linearities, but that would increase the computational cost without significant benefits since for a large part of the time interval an higher density of evaluation points would not improve the approximation of the solution. It is possible to use a non-uniform grid, but it is generally impossible to know in advance which part of the time interval needs a shorter time step. A viable solution would be to run a first optimization with a lower number of nodes, see where the non-linearity takes place, refine the grid in a suitable neighborhood and iterate until a smooth solution is found. Normally special solvers that can vary their time step during the process of solution would be used to make this process automatic, but as mentioned at the end of section 3.4 they are not compatible with the direct collocation method.

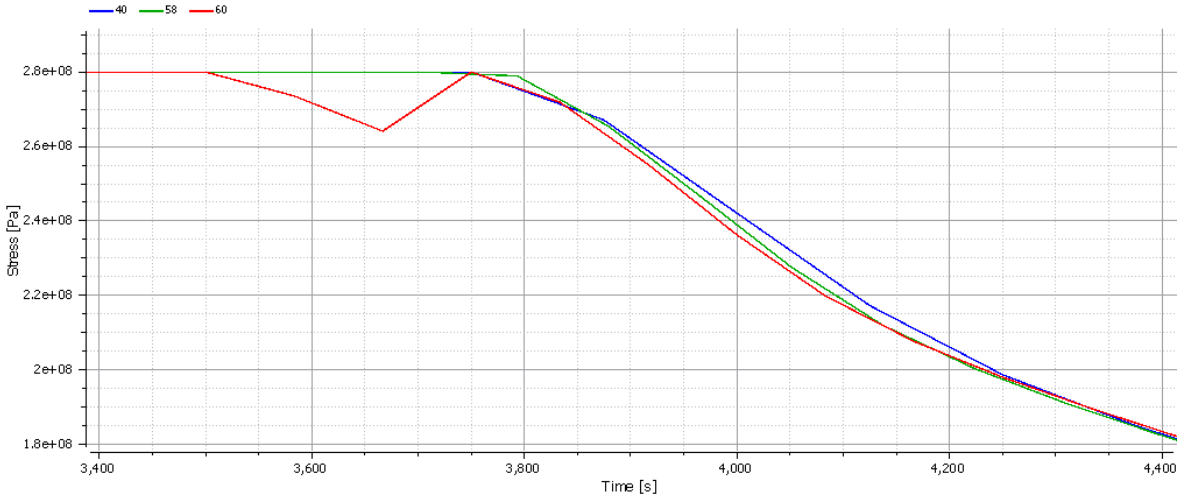

Figure 6.15: Baseline model - *OM* - Stress, detail

Chapter 7

Conclusions

The initial prompt of this work was to determine if Object-Oriented modeling could be used to implement Optimal Control techniques on thermal power plants in order to increase their flexibility and deal with the new demands of the energy market.

In order to verify that, we used the Modelica language to develop a model of a combined cycle power plant. This model, while simplified, well reproduces all the relevant characteristics of the real plant. Then we considered a process of warm start-up of the plant and we tested three different methods of optimization to find the optimal trajectory for all the variables of the plant during the transient.

Two methods were tried as implemented in the software JModelica, the third one was implemented with the software OpenModelica. All the implemented methods reached a good approximation of the optimal analytic trajectory, despite the fact that both software are still under development regarding the optimization functionality and they have not been validated yet.

Our test campaign revealed that the methods based on JModelica are significantly faster than OpenModelica. But on the other hand JModelica suffers from stability problems. In fact depending on the discretization of the time interval, the time required to solve the problem may increase a lot, or the algorithm may not converge at all. OpenModelica does not suffer from this problem, its algorithm always reaches convergence. This is due to the fact that OpenModelica uses two nested nonlinear solvers, one for the nonlinear algebraic equations in the model, and another one for the NLP, while JModelica solves both nonlinear problems simultaneously, so it is faster but less robust.

While this case study is only a simple process, it is still a proof of concept that this process can be used successfully and could be applied to different types of power plant to optimize their transitories and improve their flexibility.

7.1 Future work

The natural continuation of this work is to further validate this process by testing it on other types of power plants. The current candidates are:

- an Organic Rankine Cycle (ORC) fueled by the waste heat of an internal combustion engine;
- a closed Joule-Brayton cycle whose working fluid is supercritical CO_2;
- a once-through steam generator using molten salts as heat source.

From the point of view of the software, there is not a consolidated standard on how this type of problem should be handled. Even the software we used, JModelica and OpenModelica, are currently under development and their functionalities are liable to be modified and augmented with future releases. In particular there are several new mathematical methods that are about to be released. Given that this is a relatively new field, the mode developed for this work could be useful as a benchmark to test these future methods.

After interpreting the results of our test campaign, we can point out a specific feature that could considerably improve the performance of all the methods: that is the support for variable-step solvers. Section 6.3 has shown that the accuracy in handling the non-linearities of the model strongly depends on the discretization of the time interval. The best way to solve this problem would be to implement an automatic estimation of the error to detect the non-linearities and to allow the algorithm to reduce the size of the time step in their proximity.

Appendix A

Numerical Data of the test Campaign

Baseline model						JModelica standard algorithm				
n	10	20	30	40	50	60	70	80	90	100
N var	5884	11294	16704	22114	27524	32934	38344	43754		
N iter	24	32	34	39	52	47	57	56		
t NLP	1.099	5.223	6.301	9.095	19.036	14.342	28.284	32.925		
t eval	0.13	0.356	0.547	0.788	1.308	1.413	2.049	2.274		

Table A.1: Baseline Model - JModelica standard algorithm

Baseline model						JModelica variable elimination				
n	10	20	32	40	52	58	71	80	90	100
N var	2771	5351	8447	10511	13607	15155	18509	20831	23411	25991
N iter	24	26	28	34	33	99	40	35	80	51
t NLP	0.589	1.081	2.016	3.964	5.306	46.095	9.971	10.014	22.169	15.535
t eval	0.128	0.188	0.307	0.629	0.778	2.928	1.311	1.32	3.546	2.352

Table A.2: Baseline Model - JModelica variable elimination

Baseline model						OpenModelica				
n	10	20	30	40	50	60	70	80	90	100
N var	340	680	1020	1360	1700	2040	2380	2720	3060	3400
N iter	22	33	39	49	60	77	71	93	91	106
t NLP	0.166	0.304	0.489	1.142	1.215	2.11	2.293	3.869	4.247	5.352
t eval	24.77	52.82	93.52	217.0	240.1	459.05	486.8	816.0	862.5	1139

Table A.3: Baseline model - OpenModelica

Attemperation model						JModelica standard algorithm				
n	10	20	30	40	50	60	70	80	90	100
N var	5980	11480	16980	22480	27980	33480	38980	44480	49980	55480
N iter	53	80	118	230	99	103	143	60	111	110
t NLP	3.484	10.036	27.798	54.649	30.567	43.478	70.533	35.187	87.635	86.415
t eval	0.344	0.934	1.812	4.699	2.51	3.136	5.154	2.13	5.202	5.534

Table A.4: Attemperation model - JModelica standard algorithm

Attemperation model						JModelica variable elimination				
n	10	20	30	40	50	60	70	80	90	100
N var	2834	5474	8114	10754	13394	16034	18674	21314	23954	26594
N iter	30	58	108	102	73	105	148	163	184	121
t NLP	0.886	4.275	16.007	15.344	14.654	27.754	45.618	63.649	83.033	53.94
t eval	0.156	0.579	1.561	1.601	1.706	2.892	4.78	6.017	7.699	5.702

Table A.5: Attemperation model - JModelica variable elimination

Attemperation model						OpenModelica				
n	10	20	30	40	50	60	70	80	90	100
N var	360	720	1080	1440	1800	2160	2520	2880	3240	3600
N iter	25	40	54	65	72	85	89	93	97	101
t NLP	0.221	0.397	0.851	1.497	1.862	3.128	3.449	4.219	4.814	5.686
t eval	27.42	78.95	161.4	303.2	399.6	606.5	736.4	837.2	974.7	1169

Table A.6: Attemperation model - OpenModelica

Bibliography

[1] Esercitazione 1 - verifica delle prestazioni di un ciclo combinato. POLITECNICO DI MILANO - CORSO DI SISTEMI ENERGETICI AVANZATI.

[2] Johan Åkesson. Optimica - an extension of modelica supporting dynamic optimization. In *Proc. 6th International Modelica Conference 2008*, 2008.

[3] Johan Åkesson, K-E Årzén, Magnus Gäfvert, Tove Bergdahl, and Hubertus Tummescheit. Modeling and optimization with optimica and jmodelica. org-languages and tools for solving large-scale dynamic optimization problems. *Computers & Chemical Engineering*, 34(11):1737–1749, 2010.

[4] Joel Andersson. *A General-Purpose Software Framework for Dynamic Optimization*. PhD thesis, Arenberg Doctoral School, KU Leuven, Department of Electrical Engineering (ESAT/SCD) and Optimization in Engineering Center, Kasteelpark Arenberg 10, 3001-Heverlee, Belgium, October 2013.

[5] Karl Johan Åström and Rodney D Bell. Drum-boiler dynamics. *Automatica*, 36(3):363–378, 2000.

[6] Bernhard Bachmann, Peter Aronsson, and Peter Fritzson. Robust initialization of differential algebraic equations. In *EOOLT*, pages 151–163. Citeseer, 2007.

[7] Vitalij Ruge Willi Braun Bernhard Bachmann and Andrea Walther Kshitij Kulshreshtha. Efficient implementation of collocation methods for optimization using openmodelica and adol-c.

[8] S. L. Campbell, V. Hoang Linh, and L. R. Petzold. Differential-algebraic equations. 3(8):2849, 2008. revision 91199.

[9] Francesco Casella. From object-oriented models to simulation: Symbolic and numerical techniques.

[10] Francesco Casella, Filippo Donida, and Johan Åkesson. Object-oriented modeling and optimal control: A case study in power plant start-up. In *18th IFAC World Congress*, pages 9549–9554, 2011.

[11] Open Source Modelica Consortium. https://openmodelica.org/.

[12] Laura Cozzi. World energy outlook 2011. 2011.

[13] Leone Corradi Dell'Acqua. *Meccanica delle strutture*. McGraw-Hill, 1994.

[14] Moritz Diehl. Optimization for (nonlinear) model predictive control. In *Model Predictive Control Course Politecnico di Milano October 13, 2005*, October 13, 2005.

[15] JLM Fernandes. Fast evaluation of thermodynamic properties of superheated steam: a cubic equation of state. *Applied Thermal Engineering*, 16(1):71–79, 1996.

[16] Lester Haar. *NBS/NRC steam tables*. CRC Press, 1984.

[17] Evgeny Lazutkin, Abebe Geletu, Siegbert Hopfgarten, and Pu Li. Modified multiple shooting combined with collocation method in jmodelica. org with symbolic calculations. In *Proceedings of the 10th International Modelica Conference*, pages 999–1006.

[18] Fredrik Magnusson. Private communication.

[19] Fredrik Magnusson. Collocation methods in jmodelica. org. *ISSN 0280-5316*, 2012.

[20] Tamara Petrova and RB Dooley. The international association for the properties of water and steam - revised release on the iapws formulation 1995 for the thermodynamic properties of ordinary water substance for general and scientific use. 2014.

[21] Jens Rantil, Johan Åkesson, Claus Führer, and Magnus Gäfvert. Multiple-shooting optimization using the jmodelica. org platform. In *In 7th International Modelica Conference 2009*, 2009.

[22] Vitalij Ruge. Private communication.

[23] Christof Ruhl. World petroleum congress, moscow. In *Energy in 2013: Taking stock*, 2014.

[24] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.