TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY
Julkaisu 763 • Publication 763

Salvador Esqué

A New Approach for Numerical Simulation of Fluid Power Circuits Using Rosenbrock Methods

Tampere 2008

Tampereen teknillinen yliopisto. Julkaisu 763
Tampere University of Technology. Publication 763

Salvador Esqué

A New Approach for Numerical Simulation of Fluid Power Circuits Using Rosenbrock Methods

Thesis for the degree of Doctor of Technology to be presented with due permission for public examination and criticism in Konetalo Building, Auditorium K1702, at Tampere University of Technology, on the 28th of November 2008, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2008

ISBN 978-952-15-2055-6 (printed)
ISBN 978-952-15-2101-0 (PDF)
ISSN 1459-2045

Esqué Solé, Salvador "**A New Approach for Numerical Simulation of Fluid Power Circuits Using Rosenbrock Methods**"

Tampere University of Technology, Finland, 2008

Keywords: numerical integration, Rosenbrock methods, fluid power, simulation, ordinary differential equations

Abstract

The mathematical formulation of the dynamics observed in fluid power systems involves the numerical solution of differential equations. Because of the intrinsic characteristics and physics of fluid power circuits, the numerical integrators employed to solve such system of equations must retain certain properties in order to guarantee the accuracy, stability and efficiency of the numerical solution. In this thesis, different classes of numerical integration methods used for stiff systems have been analyzed and tested in order to quantitatively and qualitatively assess their performance against the numerical stiffness, high nonlinearities and discontinuities typically shown in the differential equations arisen in fluid power circuits.

Numerical integration methods of the Rosenbrock class – although rarely employed in the simulation of fluid power circuits – have shown excellent numerical stability properties and also above-the-average efficiency (solution accuracy to number of integration steps ratio) when compared to other popular single and multiple-step integration formulas. At the same time, the formulation of Rosenbrock methods involves a reduced number of linear algebra operations, which makes them computationally inexpensive. The main drawback of employing a Rosenbrock formula is the fact that an accurate Jacobian evaluation of the ODE system needs to be provided at each integration step in order to maintain the accuracy and stability of the formula. In order to solve this disadvantage, a method is presented in this thesis for the systematic modelling of fluid power components and systems as ODEs, following an object-oriented and modular approach. By following this methodology, the analytical form of the Jacobian matrix can be automatically generated and fed to the integration formula for any given fluid power system. This has the advantage that the Jacobian

evaluation is done with a fraction of computational cost and also more accurately than a Jacobian obtained with numerical techniques.

The tests conducted in this thesis have confirmed that Rosenbrock formulas are good candidates for being used in real-time simulations (fixed integration step size) and in offline simulations (variable integration step size) of fluid power circuits. Their easy implementation, good stability, high efficiency and low computational costs make them, in most of the cases tested, superior to other popular codes.

Preface

The decision, on endeavouring oneself into a Doctor Degree program, is not easy to make. Strong motivation and a great deal of commitment are both required in order to succeed. As to motivation, I am very grateful to Prof. Asko Ellman: first for accepting me as an exchange student, and then for integrating me in his research group, where the interest and passion for this research topic arose. I also want to thank Prof. Robert Piché from the Department of Mathematics for his studies and papers dealing with Rosenbrock methods and also for his collaboration in some of my publications.

I would like to thank also the financial and academic support that I have received from the Graduate School Concurrent Mechanical Engineering, led by Prof. Erno Keskinen. I also want to transmit my gratefulness to the Head of the Department of Intelligent Hydraulics and Automation (IHA) Prof. Matti Vilenius for encouraging me in the early moments and for giving me support and confidence through all this time. I express the same gratitude to Dr Jouni Mattila for giving me the required time and flexibility, within working hours, needed to complete the writing of this thesis. I am grateful as well with all the personnel and colleagues of IHA for their help and for forming such a good and pleasant working environment, and also to Prof. Jose LM Lastra for the hassle-free and friendly discussions we have had during lunch and coffee breaks.

Concerning the commitment, all the dedication I have put in this work would not have been possible without the support, understanding and sacrifices of my closest relatives and companions. Therefore I express my warmest gratitude to Maarit, Salvador, Olga and Natalia, especially during those moments which I could not share my time with them.

Salvador Esqué

Tampere, October 25th 2008

TABLE OF CONTENTS

NOMENCLATURE

Symbol	Description	Units
a_B	constant of the bulk modulus equation model	[Pa]
a_{ij}	real coefficients of Runge-Kutta formulas	-
A	orifice/pipe cross-section area	[m^2]
$\boldsymbol{A}$	matrix of Runge-Kutta coefficients a_{ij}	-
A_A	cross-section area of cylinder chamber A	[m^2]
A_B	cross-section area of cylinder chamber B	[m^2]
$ATOL$	absolute error tolerance	-
b	viscous friction coefficient	[N s m^{-1}]
$\boldsymbol{b}$	vector of Runge-Kutta coefficients b_i	
b_B	constant of the bulk modulus equation model	[Pa]
b_i	coefficients of Runge-Kutta formulas	-
B	bulk modulus	[Pa]
B_c	bulk modulus of container	[Pa]
B_{eff}	effective bulk modulus of fluid	[Pa]
B_g	bulk modulus of gas	[Pa]
B_l	bulk modulus of liquid	[Pa]
c_1,c_2,c_3,c_4	empirical constants defining the analytical model of a pressure relief valve	-
C_q	flow discharge coefficient	-
d	orifice diameter	[m]
D	pipeline diameter	[m]
$\boldsymbol{e}$	vector of errors of the numerical solution components	-
F	force	[N]
F_C	Coulomb friction force	[N]
F_{ext}	external forces acting on a cylinder	[N]
F_{hyd}	hydraulic piston force	[N]
F_S	static friction force	[N]
F_μ	cylinder seal friction force	[N]
$\boldsymbol{G}$	vector of gravitational forces	-
h	step size of the integration	[s]
$\boldsymbol{H}$	stiffness matrix of a mechanism	-

i	control current	[A]
$\boldsymbol{I}$	identity matrix	-
$\boldsymbol{J}$	Jacobian matrix	-
k	number of previous steps employed by a multi-step integration formula	-
$\boldsymbol{k}_i$	intermediate stage i of the integration formula	-
K	variable flow coefficient of pressure relief valve	[$m^4\ N^{-1/2}\ s^{-1}$]
K_L	coefficient for laminar pressure losses	[$N\ s\ m^{-5}$]
K_T	coefficient for turbulent pressure losses	[$N\ s^2\ m^{-8}$]
L	longitudinal length of a pipeline	[m]
$L_k^{(m)}$	m-th derivative of the k-degree Laguerre polynomial	-
m	mass connected to cylinder	[kg]
$\boldsymbol{M}$	inertia matrix of a mechanism	
n_d	motor shaft speed	[rad s^{-1}]
N	dimension of an ODE system	-
p	- pressure - order of accuracy of an integration formula	[Pa] -
p_1	upstream pressure in a conduit	[Pa]
p_2	downstream pressure in a conduit	[Pa]
p_A	pressure in cylinder chamber A or at port A	[Pa]
p_B	pressure in cylinder chamber B or at port B	[Pa]
p_i	pressure at volume element i	[Pa]
p_{in}	pressure at the inlet of a pressure relief valve	[Pa]
p_j	pressure at volume element j	[Pa]
p_{loss}	total pressure loss across a short pipeline	[Pa]
p_{ref}	setting pressure of a pressure relief valve	[Pa]
$P(z)$	numerator polynomial of stability function $R(z)$	-
q	joint coordinates	-
Q	volumetric flow rate	[$m^3\ s^{-1}$]
$Q(z)$	denominator polynomial of stability function $R(z)$	-
Q_1	- upstream volumetric flow rate in a conduit - flow rate from valve characteristic curve (3.9)	[$m^3\ s^{-1}$]
Q_2	- downstream volumetric flow rate in a conduit - flow rate from valve characteristic curve (3.9)	[$m^3\ s^{-1}$]
Q_A	volumetric flow entering port A or chamber A of a cylinder	[$m^3\ s^{-1}$]
Q_B	volumetric flow leaving port B or entering chamber B of a cylinder	[$m^3\ s^{-1}$]
Q_i	volumetric flow rate entering a short pipeline	[$m^3\ s^{-1}$]
Q_{in}	incoming volumetric flow rate	[$m^3\ s^{-1}$]
Q_j	volumetric flow rate leaving a short pipeline	[$m^3\ s^{-1}$]

Q_{ij}	volumetric flow rate between internal volume elements of a short pipeline	[m^3 s^{-1}]
R	stability function of a numerical integration method	-
Re_{tr}	Reynolds number in the transition between laminar and turbulent flows	-
$RTOL$	relative error tolerance	-
s	number of stages of an integration formula	-
t	time	[s]
tol	error tolerance	-
V	volume of fluid container	[m^3]
V_g	volume of entrapped gas	[m^3]
V_p	volumetric pump displacement	[m^3 rad^{-1}]
V_t	volume of liquid and entrapped gas	[m^3]
x	actuator piston position	[m]
x_{max}	maximum position displacement of a cylinder piston	[m]
$\dot{x}_s$	transient velocity from static to Coulomb friction regimes	[m s^{-1}]
$\boldsymbol{y}(x_n)$	exact solution of function y evaluated at point x_n	-
$\boldsymbol{y}_n$	numerical solution y after n integration steps	-
z	- state variable of friction model	[m]
	- $z = h\lambda$	-
Z	characteristic impedance of a conduit	[kg m^{-4} s^{-1}]

α_i	free coefficients of a numerical integration formula	-
β_i	free coefficients of a numerical integration formula	-
β_{ij}	coefficient grouping $a_{ij} + \gamma_{ij}$	-
γ	value of the diagonal elements of matrix $\boldsymbol{A}$ in singly diagonally implicit Runge-Kutta formulas	-
γ_{ij}	coefficients of the linear terms $\boldsymbol{Jk}_j$	-
Γ	- torque	[N m]
	- propagation function (2.3)	-
Δp	pressure drop (difference)	[Pa]
Δp_1 , Δp_2	pressures from valve characteristic curve	[Pa]
Δp_{ij}	pressure difference between port volumes i and j of a short pipeline	[Pa]
Δp_L	pressure loss due to laminar flow	[Pa]
Δp_T	pressure loss due to turbulent flow	[Pa]
Δp_{tr}	pressure drop in the transition between laminar and turbulent flows	[Pa]

$\delta_h(t_n)$	local error after integration step *n* of the numerical solution	-
ε_n	global error at integration step *n* of the numerical solution	-
λ	- eigenvalue of Jacobian matrix - scalar of the test equation	-
μ	constant in Van der Pol's equation	-
ξ	resistance coefficient	-
ρ	fluid density	[kg m^{-3}]
σ_0	stiffness parameter of friction force model	[N m^{-1}]
σ_1	damping parameter of friction force model	[N s m^{-1}]
ν	kinematic viscosity	[m^2 s^{-1}]
$\varnothing$	diameter	[m]
ϕ	non-linear system of equations	-
ω	angular position	[rad]
$\dot{\omega}$	angular velocity	[rad s^{-1}]

LIST OF ACRONYMS

BDF	Backward Differentiation Formula
CPU	Central Processing Unit
DAE	Differential and Algebraic Equation
DIRK	Diagonally Implicit Runge-Kutta
DOF	Degree of Freedom
FE	Function Evaluations
FP	Fluid Power
LMF	Linear Multi-stage Formula
ODE	Ordinary Differential Equation
RKF	Runge-Kutta-Formula
RMS	Root Mean Square
SDIRK	Singly-Diagonally Implicit Runge-Kutta
SIRK	Semi-implicit Runge-Kutta

1 INTRODUCTION

This introductory chapter starts with a historical overview concerning the evolution of differential equations and the problems they arise from. The methods employed to solve these equations are also illustrated. Problem definition and justification of the research is then followed. Finally, a description of the structure of this thesis with a summary of its chapter is given.

1.1 Background and brief historical overview

Differential equations are often used to describe physical systems. The solution of such equations provides information on how the system evolves and what the effect of parameters is. A very brief description of the origin of differential equations and the evolution of numerical methods for solving ordinary differential equations is followed.

1.1.1 Differential equations

A first order differential equation is an equation of the form

$$y'(x) = f\left(x, y(x)\right), \tag{1.1}$$

where $f(x, y)$ is a given function and $y(x)$ is the solution of the equation. The solution contains also a free parameter y_0 which is called the initial value problem and it is defined as

$$y(x_0) = y_0 . \tag{1.2}$$

Differential equations of order n have the form

$$y^{(n)} = f\left(x, y, y', y'', ..., y^{(n-1)}\right) \tag{1.3}$$

and they can be rewritten as first order system of differential equations for obtaining their solution.

Another type of problem arises when the conditions determining the particular solution of a differential equation are not given at the same point x_0 as in (1.2). Instead, the initial conditions are replaced by conditions of the type $y(x_0) = a$, $y(x_1) = b$. These types of problems are called boundary value problems, and their solution is more complex to obtain.

Differential equations appeared in scientific literature at the same time as differential calculus. In 1671, Isaac Newton discussed a solution of a first order differential equation by series expansion, whose terms were obtained recursively. The main origin of differential equations was due to geometrical problems, such as the inverse tangent problems considered by Gottfried Leibniz and Jakob Bernoulli during the same century.

In the 1750s the Euler-Lagrange equation was developed. This was one of the fundamental equations of the calculus of variations published later by Leonhard Euler. The Euler-Lagrange equation

$$\frac{\partial F}{\partial y} - \frac{d}{dx}\frac{\partial F}{\partial y'} = 0 \tag{1.4}$$

is used to solve functions of the type $F(x, y, y')$ which minimize or maximize the functional $S = \int_{x_0}^{x_1} F(x, y, y')\,dx$. It is generally used to solve optimization problems.

The mathematical formulation of physics involved the use of differential equations. In his “Dynamique” (1743), Jean le Rond d'Alembert introduced second order differential equations to compute mechanical motion. Brook Taylor and Johann Bernoulli formulated the problem of the vibrating string as a system of *n* linear differential equations, whose solutions determined the position of discretised mass points. From the previous system d’Alembert derived the famous partial differential equation for the vibrating string. The propagation of sound was also formulated similarly by Joseph-Louis Lagrange, who considered the medium to be a sequence of mass points. Lagrange introduced the terms *eigenvalue* and *eigenvector* to solve a second order linear equations with constant coefficients.

The problem of heat conduction led to the earliest first order systems. Joseph Fourier, in 1807, assumed that the energy that a particle passes to its neighbours is proportional to the difference of their temperatures. This can be expressed as a first order system with constant coefficients. Later, Fourier transformed the first order system to his well-known heat equation (a partial differential equation), which would be the origin of his Fourier series theory.

Lagrange formulated his Lagrangian mechanics (1788) combining the dynamics established by d'Alembert with the Lagrange-Euler equation of variation calculus and with the principle of least action. Lagrange mechanics is still widely used nowadays as a tool to obtain the equations of motion of complex mechanical systems. The trajectory of an object is derived by finding the path which minimizes the integral of the Lagrangian, which is the difference between the kinetic and the potential energy of the system.

1.1.2 Solution of differential equations

In general, it is extremely difficult to obtain an analytic solution to a given differential equation. Some of the most elemental differential equations can be solved explicitly. Euler begun to compile all possible differential equations which could be integrated by analytical methods. The results are collected in 800 pages in the Euler's opera Omnia. The book of [Kamke 1942] compiles a list with more than 1500 differential equations with their solutions. Numerical methods applied to problems of differential equations are needed to obtain an approximation of the solution when differential equations cannot be solved analytically.

The Euler method (1768) can be considered as the most basic numerical integration formula to solve first order differential equations with a given initial value. In order to simplify the illustration of the method, the autonomous form of a first order differential equation:

$$y'(x) = f(y(x)) \tag{1.5}$$

is considered instead of the non-autonomous form given in (1.1). Integrating the equation through the interval $[x_n, x_{n+1}]$ and approximating the integral of function f by a rectangular quadrature, the Euler method is obtained:

$$y(x_n + h) \approx y_{n+1} = y_n + hf(y_n), \tag{1.6}$$

where $h = x_{n+1} - x_n$ is the integration step size, and y_{n+1} and y_n are defined as approximations to $y(x_{n+1})$ and $y(x_n)$ respectively. The Euler method has an order of accuracy of one. A method is said to have a numerical accuracy of order p if the local integration error $y_{n+1} - y(x_n + h)$ is of the order of $O(h^{p+1})$.The low accuracy of this method led the mathematicians to look for higher order methods. To achieve formulas with higher order of accuracy, there are two main approaches:

- To use not only the previous calculated solution y_n to compute the next solution y_{n+1} but to make the solution y_{n+1} to depend on more previously calculated solutions. This approach leads to the so-called *multi-step* methods.
- To use more function evaluations f in the interval of integration $[x_n, x_{n+1}]$ to compute the solution at the point x_{n+1}. This procedure leads to the family of methods called *multi-stage*.

The first multi-step methods were published by Adams and Bashforth in 1883. The Adams-Bashforth methods are a special case of the methods known currently as linear multi-step methods, which have the form

$$\sum_{i=0}^{k} \alpha_i y_{n+1-i} = h \sum_{i=0}^{k} \beta_i f\left(y_{n+1-i}\right), \tag{1.7}$$

where α_i and β_i are free coefficients. Formula (1.7) is known as a k-step linear multi-step method since information of the last k steps is required to compute the solution y_{n+1}. k function evaluations f are also needed at previous calculated solutions.

Multi-stage methods appeared when Carle Runge described in 1895 an integration formula which had its origin in the midpoint rule equation (a Gaussian quadrature)

$$y\left(x_n + h\right) \approx y_{n+1} = y_n + hf\left(y\left(x_n + \frac{h}{2}\right)\right). \tag{1.8}$$

The accuracy of the midpoint rule formula is of order 2, which makes this method faster in achieving a desired accuracy, compared to the Euler formula in (1.6). However, to advance the solution from x_n to x_{n+1} the solution y at point $(x_n + h/2)$ is required though it is unknown. Newton iteration schemes were used to solve these non-linear equations. Instead, Runge applied the Euler formula (1.6) with a step size of $h/2$ to determine the solution $y\left(x_n + h/2\right)$. As a result, Runge rewrote the problem (1.8) into this multi-stage formula:

$$\begin{aligned} k_1 &= f\left(y_n\right) \\ k_2 &= f\left(y_n + \frac{h}{2} k_1\right). \\ y_{n+1} &= y_n + hk_2 \end{aligned} \tag{1.9}$$

Although the formula uses a first order approximation to determine an intermediate solution, the method retains an accuracy of order 2. Martin Kutta (1901) formulated the general scheme of the well-known Runge-Kutta methods. The following method:

$$\begin{aligned} k_1 &= f\left(y_n\right) \\ k_2 &= f\left(y_n + ha_{21}k_1\right) \\ k_3 &= f\left(y_n + h\left(a_{31}k_1 + a_{32}k_2\right)\right) \\ &\cdots \\ k_s &= f\left(y_n + h\left(a_{s1}k_1 + \ldots + a_{s,s-1}k_{s-1}\right)\right) \\ y_{n+1} &= y_n + h\left(b_1k_1 + \ldots + b_sk_s\right) \end{aligned} \tag{1.10}$$

is the general form of the *s*-stage explicit Runge-Kutta method, where a_{ij} and b_i are real coefficients. It was after the 1950s when implicit Runge-Kutta methods become of interest, mainly due to the stiff equation problem and the availability of faster computing devices. Butcher [Butcher 1964a] and Kuntzmann [Ceschino 1963] derived order conditions for the free coefficients a_{ij} and b_i, stating that by employing *s*-stages, an implicit Runge-Kutta formula of order 2*s* could be obtained.

1.2 Problem definition and justification for the research

The search of numerical methods with higher accuracy, while retaining the computational efficiency, is conditioned by numerical stability issues of the formulas. Curtis and Hirschfelder introduced the term *stiff equations* in the 1950s. They noticed that implicit numerical methods performed much better than explicit methods when solving stiff equations. Simply defined, it is said that stiff equations arise in a system of ordinary differential equations $y' = f\left(y\left(x\right)\right)$ when eigenvalues of its Jacobian matrix $\partial f / \partial y$ differ in orders of magnitude. Solutions to non-stiff equations are easy to obtain by simply using classical methods such as Adams or explicit Runge-Kutta formulas. Nevertheless, these methods become inefficient for solving stiff equations, since the step size is controlled to keep the formula stable rather than to fulfil the accuracy required. Methods for solving stiff equations need therefore new concepts of stability.

Nearly all available numerical codes for solving ordinary differential equations can be divided in two classes: those for solving stiff equations and those for solving non-stiff equations. Implicit methods are employed to solve stiff equations. Implicit methods, in op-

position to explicit methods, require more computational effort, since a set of non-linear algebraic equations must be solved at each integration step. The latest requires a modified Newton iteration scheme which makes use of an iteration matrix of the form $\left(I - ha_{ij}J\right)$, where I is the identity matrix, J is the Jacobian and ha_{ij} is a scalar. Every iteration of the scheme involves the following computational costs: *a*) evaluation of the Jacobian and formation of the Newton iteration matrix, *b*) factorization of the iteration matrix into *LU* form and *c*) forward and backward substitution to compute the correction.

Ordinary differential equations describing the dynamics of fluid power systems are of special interest. Numerical methods find these equations particularly difficult to solve due to the following characteristics listed below:

- Stiffness: It appears when different sizes (in orders of magnitude) of volumes are found in the system. Stiffness can also emerge due to the presence of large and small orifices in the circuit. Such orifices bring different levels of coupling between volumes.
- Strong non-linearities: Mainly are due to the pressure-dependency of the bulk modulus, the non-linear turbulent fluid flow equation and seal friction forces in actuators.
- Discontinuities: Might arise in the following situations: presence of on/off valves, and sudden opening and closing of flow paths.

Traditional numerical methods might fail to give an acceptable solution to the stiff fluid power equations unless excessive small time steps are taken. On the other hand, general implicit methods, although they might overcome stability issues, require much more computational effort than explicit methods.

Computational times have a major importance, especially in real-time applications, such as on-line conditioning monitoring, teleoperation, and hardware-in-the-loop. An example of such applications is showed in [Esqué 2003a], where a simulation model of a two-dof crane is driven man-in-the-loop within a virtual reality environment. In this real-time application, the employed numerical method required a relatively small integration step size in order to guarantee the numerical stability. As a consequence, the computational of the solution in real-time clock was compromised due to the excessive amount of operations.

Numerical methods capable of solving *efficiently* the dynamics involved in fluid power systems are required, particularly in the real-time simulation of relatively large sys-

tems, and in simulations with a long time intervals. The research presented in this thesis is therefore justified according to these requirements.

1.3 Research description

1.3.1 Objectives

The two main objectives of this research are stated below:

- The investigation on numerical integration formulas capable of dealing with the stiffness, non-linearities and discontinuities found in the formulation describing the dynamics of fluid power systems. Despite these properties found in fluid power circuits, the numerical integrator should provide acceptable solution accuracy, very good numerical stability and reduced computational costs. Integration formulas fulfilling these properties might also be good candidates for being used in real-time simulations involving fluid power circuits.
- Derivation of a systematic approach to formulate mathematical models of fluid power components following an object-oriented methodology. The dynamics of the resulting simulation models shall be formulated as system of ordinary differential equations in order to be solved by the above numerical integration formulas.

1.3.2 Contributions

The guidelines for the systematic mathematical modelling of the dynamics of fluid power components (as lumped-parameter models) have been presented. This has been developed following an object-oriented methodology, allowing the physical modelling of large interconnected systems of different physical domains. With this object-oriented methodology, modular subsystems and components can be constructed while retaining reusability and hierarchical properties.

A class of Rosenbrock formulas are introduced as numerical solvers of the system of ordinary differential equations originating in the formulation of fluid power systems. The performance of Rosenbrock formulas outstands, in most of the cases, the performance shown by numerical solvers commonly used in both real-time and offline simulations.

A systematic way to obtain the analytical expression of the Jacobian matrix of the system has been also presented. This task can be performed by an algorithm prior to starting the numerical integration. An analytical form of the Jacobian matrix is beneficiary for

the numerical integration solver, since it can evaluate the Jacobian more accurately and with less computational costs.

1.3.3 Methodology

Physical based lumped-parameter models of fluid power systems have been developed in the early stages of this research. Advantages of physical based models are that model parameters have a physical meaning in the real component and therefore they can be found in manufacturer's data sheets or determined empirically. The developed models have also been formulated taking into account a topology which confers modular and hierarchical properties. All simulation models have been compiled and organized in a software library from where they can be called as subroutines. In that way, a fluid power circuit is defined within an algorithm by simply calling the subroutines, each representing a fluid power component or subsystem. The algorithm then generates the system of ordinary differential equations and its analytical Jacobian matrix, which are fed to the numerical integration formula. The use of this algorithm has provided a straight-forward way to define and construction fluid power system models. Errors due to symbolic manipulation and composition of the equations are also avoided since all the algebraic formulation is automatically generated and in the adequate format in order to be used by the numerical integration formula.

During the past decades, there has been plenty of research on the construction of numerical integration formulas for solving stiff ordinary differential equations. A broad literature survey on the proposed formulas has been conducted. From this survey, many of the proposed numerical formulas for solving ODEs have been tested by means of solving test models of fluid power circuits constructed with the algorithm and library described above.

All numerical integration formulas, their driver algorithms, the library of fluid power components models and the algorithm used to define and construct the simulation models have been coded in FORTRAN language, under the Digital Visual Fortran (Digital Equipment Corporation) programming environment, running on a Windows XP computer platform.

1.3.4 Assumptions and Limitations of Scope

Both the mathematical modelling of fluid power systems and the numerical integration of differential equations are very broad fields, which can be approached in a number of different ways. This research focuses in the mathematical formulation of fluid power elements, as lumped-parameter models, realized with ordinary differential equations. Modelling representations derived from energy-balance methods, transport delay lines, and frequency domain are not within the scope of this research. Despite these assumptions and simplifications models can still replicate accurately the behaviour of a fluid power circuit at a system level [Ellman 1996a]. Empirical validation of mathematical models is not a target of the research conducted in this thesis and therefore it has been omitted.

During the numerical tests, the maximum dimension (i.e. number of state variables) of the modelled system has been 20. In the context of mobile fluid power applications, this dimension represents a relatively mid-large system.

1.4 Thesis structure

This thesis is divided in five chapters, briefly described below, followed by conclusions. The thesis is also supported by four peer-reviewed publications, not reprinted in this thesis, which are referenced and summarized below.

The first introductory chapter starts with a historical overview concerning the use of differential equations and their applications from the early days till the present. It is followed by the definition of the problem, the justification for the research and a description of the research, including objectives, contributions, methodology and limitations of scope.

Second chapter provides an overview of different formulation approaches employed to model the dynamics of fluid power systems. This chapter also discusses the state of the art on the numerical integration formulas and software packages employed to obtain the numerical solution of the simulation models.

Third chapter, entitled ‘Lumped-parameter models of fluid power components and systems’, introduces a systematic modelling method which ensures modular and hierarchical properties. The mathematical formulation of these models is presented for some of the most common fluid power elements.

Fourth chapter, ‘Numerical integration of ODEs arising in fluid power systems’, analyses the computational costs of implicit multi- and single-step integration formulas, as

well as their numerical stability properties. Based on theoretical analyses, *L*-stable Rosenbrock formulas are presented as a good candidate for the numerical integration of ODEs arising in FP systems, due to their good stability properties and computational efficiency. Finally, a systematic way to form the analytical Jacobian matrix of the system is shown. This ensures an accurate and computationally cheap evaluation of the Jacobian.

In the fifth chapter, 'Performance of Rosenbrock formulas', numerical integration tests are carried out employing different numerical integration formulas. Accuracy of the solution, numerical stability and computational efficiency are analyzed. These numerical tests confirm that the performance of Rosenbrock formulas, in most of the cases studied, surpass the performance of other popular ODE integrators. The chapter concludes highlighting the advantages (in terms of accuracy and efficiency) of employing analytical Jacobian matrices instead of numerically-evaluated Jacobians.

Refereed publications

Parts of this dissertation have also been published through the following peer-reviewed publications:

I. Esqué, S., Raneda, A., Ellman, A. (2003), "Techniques for studying a mobile hydraulic crane in virtual reality". International Journal of Fluid Power Vol 4 No 2 pp. 25-34.

   The article addresses the problem of real-time simulation of a mobile hydraulic crane. A mathematical model of a hydraulic system controlling a multi-body linked mechanism by using Lagrange's equations of motion is presented. The article describes the hardware and software implementation of the virtual interface, as well as the computational performance of the simulation in terms of data transmission between computers, visualization refresh rate, and numerical integration rate. It is concluded that the bottleneck for achieving real-time simulation is located in the numerical integration of the mathematical model. Due to the stiffness of the system, the integration time step had to be reduced excessively in order to avoid numerical oscillations in the solution given by an *A*-stable formula

II. Esqué, S., Ellman, A. (2002), "Pressure Build-Up in Volumes". Bath Workshop on Power Transmission and Motion Control, PTMC 2002, Bath, UK. Published in the

book Power Transmission and Motion Control, edited by C.R. Burrows and K.A. Edge, Professional Engineering Publishing Limited. London, UK, pp. 25-38.

In this paper, the pressure generation equation is presented and used for modelling three basic components widely present in fluid power systems: constant volume element, cylinder actuator, and a pipeline. Flow variables are determined by a modified orifice flow equation. The mathematical models of the presented components are written as sets of ordinary differential equations. The paper also derives the Jacobian matrices of the above elements. The modular approach of the formulation allows building a volume-network, where volume-elements can be interconnected by orifice-model interfaces (such as valves, pumps…) and therefore, a complete fluid power system can be assembled.

III. Esqué, S., Ellman, A., Piché, R. (2002), "Numerical integration of pressure build-up volumes using an *L*-stable Rosenbrock method". Proceedings of the 2002 ASME International Mechanical Engineering Congress and Exposition, November 17-22, 2002, New Orleans, Louisiana, USA.

The paper begins by reviewing the most popular single-step formulas used in solving stiff ordinary differential equations. The author proposes a simple and efficient integration method for solving the ordinary differential equations arisen from fluid power systems: a two-stage Rosenbrock formula derived from a general one-step semi-implicit Runge-Kutta method. The formula has *L*-stability properties and its numerical accuracy is of second order. The integration algorithm also implements an embedded estimation of the error and step size selection. The numerical method is tested in a dynamic simulation model consisting of two fluid power component models. The numerical method showed excellent numerical stability, even in stiff conditions and in regions of discontinuity. The Rosenbrock formula also showed a remarkably good computational efficiency.

IV. Esqué, S., Ellman, A. (2005), "An efficient numerical method for solving the dynamic equations of complex fluid power systems". Bath Workshop on Power Transmission and Motion Control, PTMC 2005, Bath, UK. Published in the book Power Transmis-

sion and Motion Control, edited by C.R. Burrows and K.A. Edge, Professional Engineering Publishing Limited. London, UK, pp. 179-191

The paper is a clear continuation of the work presented in Paper III. The article focuses mainly in the computational efficiency of the formula presented in the previous paper and also extends the simulation tests to fluid power systems composed of a number of fluid power elements. It is shown that the efficiency of the numerical integrator is further improved when the code is able to derive the full Jacobian matrix analytically as a function of the system state variables. The advantages of this approach are quantified when compared to the conventional computation of the Jacobian matrix by means of numerical differentiation. The results show that the use of an analytical Jacobian matrix of the system reduces significantly the computational time to advance in the integration. A second advantage is seen in the improvement of the numerical stability of the integration formula.

2 STATE OF THE ART

A general overview on the mathematical modelling of the dynamics of fluid power systems is presented. A broader description focused on the formulation of fluid power components, as lumped-parameter models, is given in Chapter 3 and in [Esqué 2002a]. A survey and review of the most popular numerical integration methods for solving numerically stiff systems of ODEs is then followed. More detailed analysis of these numerical methods is given in Chapter 4. Chapter 3 ends with a brief survey of simulation software packages used in the modelling and simulation of fluid power applications.

2.1 Modelling approaches for the dynamics of fluid power systems

Fluids are characterized by their continuous deformation and compressibility. One of the main equations describing the state of a fluid is the pressure generation equation. Such equation is derived from the continuity equation (conservation of mass law) and the density equation of the fluid (a function of pressure and temperature), which is defined by the term *bulk modulus*. The generally used pressure generation equation states that the pressure p generated in a confined volume V is determined by

$$\frac{dp}{dt} = \frac{B_{eff}}{V}\left(Q - \frac{dV}{dt}\right) \tag{2.1}$$

where Q is the net incoming volumetric flow to the control volume and B_{eff} is the effective bulk modulus. A second equation, but not less important, is the fluid flow equation through orifices. The turbulent orifice flow formula in (2.2) is the general accepted equation describing the volumetric fluid flow Q through a sudden short restriction for high Reynolds numbers. C_q is called the discharge coefficient which depends on the contraction geometry

of the orifice, A is the cross-sectional area of the orifice, ρ is the fluid density and Δp is the pressure drop across the orifice.

$$Q = C_q A \sqrt{\frac{2}{\rho} \Delta p} \tag{2.2}$$

The lumped-parameter modelling of fluid power circuits as systems of ordinary differential equations is extensively used and motivated due to the existence of robust ODE solution techniques, such as numerical methods and simulation software packages. In lumped-parameter model approach, fluid power components such as actuators, accumulators, and control valves can be formulated as a combination of control volumes (pressure generation (2.1)) and orifices (flow equation (2.2)). A detailed description of the lumped-parameter modelling of fluid power elements and components is presented in Chapter 3 and in [Esqué 2002a].

The lumped-parameter model approach when applied to the problem of flow through conduits is commonly formulated with the Hagen-Poiseuille laminar flow equation (3.4)(a). In Section 3.2.3, a two-volume lumped model for a short pipe, which also accounts for losses and flow inertial effects, is presented. However, long transmission lines such as pipes and hoses have inertial, capacitive and resistive properties distributed along their length, and therefore distributed-parameter models are used. These models relate the fluid pressure and velocity as a function of position and time. Using Laplace transformed variables, the input-output behaviour of a straight transmission line with constant and circular cross section and filled with compressible Newtonian fluid is described as

$$\begin{aligned} p_1 &= p_2 \cosh \Gamma + Q_2 Z \sinh \Gamma \\ Q_1 &= \frac{p_2}{Z} \sinh \Gamma + Q_2 \cosh \Gamma \end{aligned} \tag{2.3}$$

where p and Q are Laplace transform of pressure and volumetric flow respectively, subscripts 1 and 2 are upstream and downstream locations in the transmission line, Z is the characteristic impedance of the conduit and Γ is the propagation operator (which relates the transformed variables at different location points). Since equation (2.3) is expressed in the frequency domain, it needs to be approximated by a finite number of states in order to reformulate it in time-domain and as set of ODEs. Stecki and Davis [Stecki 1986a] have identified and classified the existing transmission line models in the literature into 7 groups, according to their complexity. They conclude [Stecki 1986b] that the two-dimensional vis-

cous compressible flow model is the most suitable for long transmission lines. Piché and Ellman [Piché 1995] have derived a fluid transmission line model for one and two-dimensional pipe flows. By means of a modal approach, the transcendental transfer functions associated to the partial differential equations have been approximated to a lumped parameter model that can be realized as a system of first order differential equations.

In terms of numerical simulation, the lumped-parameter modelling of fluid power components may have a drawback; all the elements interconnected by a lumped volume element are strongly coupled between them. Very small volumes will therefore induce very fast transient and strong coupling. Due to this reason, partition of the lumped numerical analysis into different tasks for parallel computation is not trivial and might be also non viable. This implies that the simulation must be run in a centralized manner (rather than distributed) in order to take into account all the possible couplings between components.

An alternative that overcomes the coupling found in the lumped-parameter modelling is proposed in [Krus 1990]. Krus introduces a distributed-parameter modelling approach of fluid power systems based on the utilization of transport delay lines in the pipelines connecting components. In this approach the transmission of information is restricted to the speed of wave propagation. There is no immediate communication between components, and this allows the components to be decoupled. Distributed or partitioned numerical simulations become therefore possible. Limitations of this approach are: a) numerical integration advances with a constant integrator step size and b) no general ODE solver can be applied to integrate such formulation. Literature concerning the transmission line modelling method applied to fluid power applications is found in [Kitsios 1986], [Boucher 1986], [Burton 1994] and [Pollmeier 1996].

Another approach on the mathematical formulation of the dynamics of fluid power systems is found the *analytical system dynamics* [Layton 1998]. This method is based on the energy methods of Lagrange and Hamilton. This multidisciplinary modelling approach includes the constraints of the system in the equations of motion such that the model comprises a set of implicit differential equations and a set of algebraic constraint equations. This combination of equations is well known as differential algebraic equations (DAEs). Although there are numerical codes available for solving such problems, the numerical solution of initial value DAEs is still a current research topic.

2.2 Numerical methods and simulation packages

Ordinary differential equations are by far the most utilized formulation to model the physics arising in fluid power systems. One of the main reasons is the extensive research carried out during the last decades [Gupta 1985] on numerical integration methods for solving ODEs. In addition, popular general purpose simulation software packages loaded with ODE solvers have also contributed to the use of ODEs for describing the dynamics of systems.

Numerical integration formulas for the solution of ODEs can be classified in two different groups: explicit and implicit formulas. While explicit formulas are more suited for solving non-stiff systems, implicit formulas become more efficient when solving stiff systems of ODEs. Another classification among numerical integration formulas is made on the basis of their internal formulations: Single-step methods only make use of the previous calculated solution to determine the next one. In multi-step methods, the solution at one point is calculated as a function of several previously obtained solutions. Multi-step methods have therefore more complex formulation than single-step methods.

Runge-Kutta formulas – described in equations (1.9) and (1.10) – are the most popular single-step codes used in the integration of ODEs. Despite the numerically stiffness usually found in fluid power systems, explicit Runge-Kutta formulas are still used to integrate the differential equations arisen in these systems. Popular codes from the explicit Runge-Kutta family are the RK 4(5)[*] proposed by [Fehlberg 1969], and the DOPRI 5(4) formula developed by Dormand and Prince [Dormand 1980]. Another advantage found in these explicit codes is their easy programming implementation. On the other hand, the utilization of such codes for solving numerically stiff systems may show a remarkably low computational efficiency, i.e. excessively small time steps are required to keep the numerical formula within stable conditions. Implicit formulas (either single or multi-step methods) are used instead for the integration of stiff systems. Despite requiring more number of operations per step, implicit formulas still perform much more efficiently than the explicit ones in the integration of stiff systems.

Numerical codes from the implicit Linear Multi-step Formulas (LMF) and from the single-step implicit Runge-Kutta family are the most used to solve the stiff systems of

[*] The pair notation 4(5) indicates that the integrator computes the solution with an order 4 formula while it uses a solution approximation of order 5 to calculate the local error.

ODEs arising in fluid power circuits. Although LMFs of order greater than 2 cannot retain *A*-stability (a numerical integration formula is called *A*-stable when the numerical stability of the formula is guaranteed for any size of the integration time step). The most popular multi-step codes for stiff equations are based on the backward differentiation formula (BDF) such as the GEAR and LSODE codes. They are part of the public domain library of numerical methods ODEPACK [Hindmarsh 1983]. Both formulas implement variable order and variable step-size and they can solve stiff and non-stiff systems by changing automatically the integration formula to BDF or to Adams methods respectively.

Single-step implicit Runge-Kutta formulas are also widely employed for solving stiff equations. Popular Runge-Kutta codes are: RADAU5 [Axelsson 1969], a fully implicit Runge-Kutta method of order 5 based on the Radau quadrature; and SDIRK4 [Alexander 1977], a diagonally semi-implicit formula of order 4. Both methods are *L*-stable (an integration formula is *L*-stable when it is *A*-stable and, moreover, numerical oscillations artefacts associated to the stiffer modes are extinguished immediately). Modified Rosenbrock formulas [Wolfbrandt 1977] have become of special interest due to its simple implementation and its efficiency. They can be interpreted as a generalization of Runge-Kutta formulas. However, in order to guarantee the numerical stability of the formula, an accurate Jacobian matrix of the system must be provided at every integration step. Popular codes based on the Rosenbrock method are GRK4 [Kaps 1979] and DEGRK [Shampine 1982].

The rapid growth in development and usage of simulation software packages to model and/or to solve differential equations has gradually diminished the interest of engineers towards implementing their own integration routines. Simulation software instead, offers a small choice of numerical integrator formulas. The rest of this section introduces a brief review of some popular simulation packages and the numerical integrators they include.

In general purpose simulation software such as MATLAB/SIMULINK and VisSim, the user provides the model of the system, generally formulated as ODEs. A review of the ODE solvers existing in MATLAB is presented in [Shampine 1997]. For non-stiff equations the software provides an order 2 Runge-Kutta formula (ode23) and an order 4 Dormand Prince formula (ode113) [Dormand 1980]. An explicit multi-step integrator is also supplied for non-stiff systems: the ode113, which is based on the Adams methods. Concerning the integration of stiff equations, MATLAB/SIMULINK includes the ode15s code,

which is a multi-step formula based on the Gear method and the ode23s, a Rosenbrock formula of order 2. VisSim software includes similar implicit and explicit integration formulas as well as the multi-step formulas from the ODEPACK library.

The following simulation software packages are partially or completely specialised in fluid power systems: AMESIM and BathFp make use of a variation of the LSODE integrator. EASY5 and DYMOLA offer a variety of single-step and multi-step codes such as Gear, SDIRK, and RADAU of different orders. The HOPSAN [Krus 1990] software implements a distributed simulation approach which allows partitioning the simulation tasks in parallel. It makes use the transmission line modelling as a method of integration.

DYNAST and DYMOLA are other multi-physics simulation software including special fluid power libraries. These packages can formulate the problems as differential algebraic equations and therefore offer numerical integrators, such as the DASSL formula [Brenan 1996], intended for these types of equations.

3 LUMPED-PARAMETER MODELS OF FLUID POWER COMPONENTS AND SYSTEMS

This Chapter deals with the construction of fluid power circuit simulation models from lumped-parameter models of fluid power components or elements. In the lumped-parameter approach, the mathematical model of a physical system with spatially distributed fields is simplified to single scalars. In this idealization, physical properties of the system such as mass, stiffness, inductance and capacitance are concentrated into single physical elements. The dynamic behaviour of these systems can be described by ODEs, being time the only independent variable.

Simulation models are constructed following an object-oriented methodology and a topology in which fluid power components are grouped into four different categories according to their functionality. This topology allows the interconnection of different fluid power elements of different groups in order to form a more complex fluid power circuit.

3.1 Modelling topology

Fluid power systems can often be represented as a combination of idealized elements, which describe the physical mechanisms of fluid power generation, storage, dissipation and transformation. Based on the previous classification, the topology adopted in this formulation of equations comprises four main element groups:

- Pumps: Provide the power to the system by generating a flow from an external mechanical power source.
- Volumes: Behave as fluid power storage. Volume elements may act as fluid capacitors and fluid inductors. Fluid capacitors store the energy in terms of the fluid pressure, and fluid inductors store the energy by means of the inertial effects of the fluid flow.
- Flow resistors: These elements dissipate fluid power by means of pressure losses.
- Actuators: Convert the hydraulic power into mechanical power.

The achievement of modular and hierarchal properties among the previous elements is considered necessary in order to mathematically formulate fluid power systems in a straight-forward and systematic approach. This is accomplished by means of defining communication ports for each of the elements. Communication ports (represented as circles in Figure 3.1) link two elements of different groups, establishing a two-way interaction between them. Communication ports are intrinsic to each element group and can be classified into three different groups, according to the type of physical variables they transmit.

- Hydraulic ports ○: They are used to connect Volume elements with other hydraulic elements such as pumps, flow resistors and actuators.
- Control ports ⊘: They are employed to input signals (control current *i,* pump displacement V_p, setting pressures…) to those elements admitting controllability such as variable displacement pumps, control valves, adjustable orifices…).
- Mechanical ports ◍: They define a two-way interaction between the fluid power and the mechanical domains by exchanging their dynamic variables.

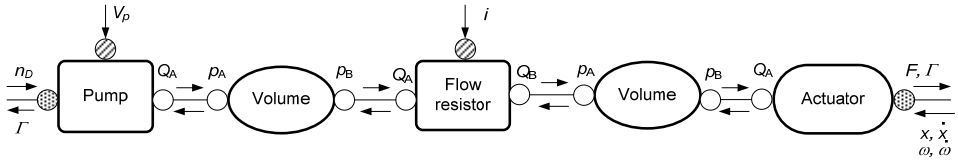


Figure 3.1. Fluid power element groups and communication ports

Fluid power elements connected by hydraulic ports exchange flow and pressure variables between each other. As it can be observed in the above figure, hydraulic ports found in Volumes output pressure variables, while hydraulic ports found in pumps, flow resistors and actuators output flow variables to their adjacent elements.

Mechanical ports allow the possibility to establish a co-simulation between fluid power domains and mechanical domains. In a co-simulation, two different solvers, one dedicated to the mechanical system and the other dedicated to the fluid power system, run in parallel. Since both domains are coupled, components connecting the domains must exchange their dynamic variables at every integration step. In [Larsson 2003], software envi-

ronments to implement co-simulations and stability analyses of coupled problems are investigated. As an example, Figure 3.1 shows a block diagram system of fluid power elements interconnected by their communication ports. From left to right the element blocks represent a variable displacement pump, a short pipeline, an electrically operated proportional valve, hoses, and a hydraulic actuator. The pump and the actuator are connected to a mechanical system by means of their mechanical ports. In the case of the pump, it receives the mechanical variable n_D, which represents the rotational speed of an engine shaft. The interaction is completed when the pump transmits resistance torque variable Γ to the engine. On the other hand, the actuator outputs the piston force F (or vane actuator torque Γ) to the mechanical system, which acquires this variable as an external load. The mechanical simulator then solves the dynamics of the mechanical system and returns the new piston position x and velocity $\dot{x}$ (or angular position ω and angular velocity $\dot{\omega}$) which are then used in the fluid power simulator to calculate the new piston force (or torque).

3.2 Mathematical formulation of fluid power components

This section introduces the equations describing the dynamics of some representative fluid power components. Fluid power components are modelled as lumped-parameter systems and therefore they can be described with ordinary differential equations. The formulation and communication between components follows the topology introduced in the previous section. A mathematical model of the fluid is also discussed.

3.2.1 Fluid bulk modulus

Due to practical considerations, in many applications the main physical fluid properties, such as density and viscosity, are considered invariable to fluid pressure and fluid temperature. The bulk modulus of a liquid (which is a measure of the fluid stiffness) can be substantially reduced by gas or vapour entrapped in the liquid in the form of bubbles. In addition, bulk modulus may be also lowered by mechanical compliance of the fluid container. In [Merritt 1967] a definition of *effective* bulk modulus is proposed and defined as the reciprocal sums of individual bulk modulus of a mixture of air-liquid fluid and the flexible container where the fluid is confined. The equation determining the effective bulk modulus is:

$$\frac{1}{B_{eff}} = \frac{1}{B_l} + \frac{1}{B_c} + \frac{V_g}{V_t}\frac{1}{B_g}, \tag{3.1}$$

where B_l , B_c and B_g are the liquid, container and gas bulk modulus respectively. V_g is the volume of entrapped gas and V_t is the volume of liquid plus entrapped gas.

However, at low pressure levels the volume of entrapped air in the fluid can grow substantially, reducing significantly the density and the stiffness of the fluid. Such changes in the fluid density have a strong influence in the system performance, and therefore systems operating through a wide range of pressure levels require a more accurate definition of fluid bulk modulus than the one proposed in equation (3.1).

A two-phase fluid model is derived in [Nykänen 2000], where the effective bulk modulus of the fluid is determined as a function of the pressure of the fluid, the volumetric fraction of entrapped gas, the bulk modulus of the liquid itself, the elasticity coefficient of the structural expansion of the container enclosing the fluid, and the polytropic gas constant of the entrapped air.

Another approach is presented in equation (3.2), which is a more generally accepted formula to compute the effective oil bulk modulus. Although the formula is not derived from a physical model, the approximation determines the effective bulk modulus as a non-linear function of pressure p. The constants a_B and b_B are given in bar and can be found tabulated for a specific oil.

$$B(p) = (b_B + p)\left[\frac{1}{a_B} - \ln\left(1 + \frac{p}{b_B}\right)\right]. \tag{3.2}$$

3.2.2 Pump elements

Most of fluid power applications have a hydraulic pump as a source of power supply. As mechanical power to fluid power transformers, hydraulic pumps show volumetric and mechanical losses which must be quantified in order to obtain an accurate value of the delivered pump flow.

Several loss models for hydraulic pumps can be found in the literature. Formulas based on coefficients, which might be obtained from data given by the manufacturer, are the most common. In [Wilson 1948] one of the first coefficient models was presented. In that approach the pump volumetric losses took into account laminar leakage and constant leakage flow. [Schlösser 1961] and [Thoma 1969] expanded the work introduced by Wil-

son adding further coefficients which would account for more flow losses and friction sources. More recently, [Dorey 1988] presented a non-linear type of flow model with variable coefficients. In general, the number of coefficients required to accurately describe the losses is about five for the flow loss model and ten for the torque loss model, requiring therefore a considerable amount of experimental data. Other formulas quantifying the losses are based on measurements, from which losses can be tabulated as a function of the system variables. One of the main drawbacks of these approaches is that many measurements are needed to include the entire pump operating range.

In [Huhtala 1996], the so-called two-line model is proposed. In this approach, flow and torque loss models are based on polynomial fittings obtained from measured data of the hydraulic pump. The advantage of this method is that just few measurement points (between 30 and 40) are required and the accuracy obtained is found to be very good in all the range of the pump operation.

According to the modelling topology presented in Section 3.1, pump (or flow generator) elements output the volumetric flow Q as a function of the following inputs: shaft rotational speed n_D, volumetric displacement V_p and pressure at the inlet and outlet ports of the pump.

3.2.3 Volume elements

Two of the most representative fluid power components which behave as fluid energy storing elements are presented next. Such components are a fixed-size volume and a short pipeline, the latter accounting for pressure losses and inertial effects. The equations describing the dynamics of these elements are mainly obtained from the law of conservation of mass in control volumes and they are discussed in [Esqué 2002a].

Fixed-size volume

A fixed-size volume can be considered as a fluid capacitor in which the rate of stored energy (in terms of pressure p) is expressed as a function of the net volumetric flow Q entering a control volume of size V, as Figure 3.2 illustrates.

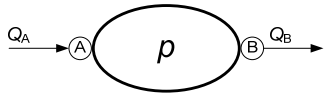


Figure 3.2. Fixed-size volume

The well-known pressure generation equation (3.3), is used to formulate the problem.

$$\frac{dp}{dt} = \frac{B_{eff}(p)}{V}\left(\sum Q - \frac{dV}{dt}\right) \tag{3.3}$$

Since the term $dV/dt = 0$ for a fixed-size volume, fluid capacitance can be identified as being $C = V / B_{eff}(p)$.

Short pipeline

The short pipe model takes into account the pressure losses of the fluid flow across the pipe, and it also accounts for the inertial effects caused by the acceleration of the fluid in the pipe. The model is realized (see Figure 3.3) by two fluid capacitors (fixed-size volume elements) connected by a flow resistor (orifice).

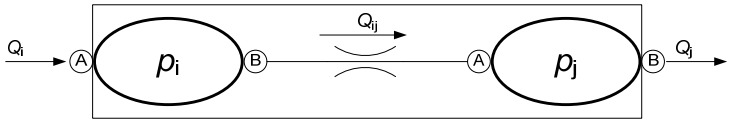


Figure 3.3. Short pipeline

Pressures p_i and p_j generated in the volume elements are determined according to the pressure generation equation (3.3). Energy losses along the pipe are determined by equation (3.4)(c). The sources of those losses are the following:

- A pressure drop due to the laminar flow travelling through a pipe (it is assumed that laminar flow is predominant). The flow equation is given by the Hagen-Poiseuille law, see equation (3.4)(a), which describes the pressure loss Δp_L as a linear function of the volumetric flow, pipe length L, fluid kinematic viscosity υ and density ρ, and inversely proportional to the fourth power of pipe diameter D.

- Minor losses, due to pipe elbows, bends and fittings are given by the turbulent flow equation (3.4)(b). The resistance coefficient ξ is determined experimentally according to the pipe geometry changes. Values for ξ can be found tabulated in [Merritt 1967].

$$\begin{aligned} \Delta p_L &= K_L Q \qquad K_L = \frac{128\nu\rho L}{\pi D^4} \qquad (a) \\ \Delta p_T &= K_T Q^2 \qquad K_T = \frac{8\rho\xi}{\pi^2 D^4} \qquad (b) \\ p_{loss} &= \Delta p_L + \Delta p_T = K_L Q + K_T Q|Q| \qquad (c) \end{aligned} \tag{3.4}$$

The inertial effects of the fluid flow (assuming constant cross-section area, steady pipe and uniform velocity of the fluid in the pipe) are formulated according to Newton's second law applied to a lumped mass of fluid, as showed in the following equation,

$$\dot{Q}_{ij} = \frac{A}{\rho L}\left(\Delta p_{ij} - p_{loss}\right), \tag{3.5}$$

where A is the cross-sectional area of the pipe and ρ is the fluid density.

Collecting and combining equations (3.3), (3.4) and (3.5) a set of ordinary differential equations defining the dynamics of a short pipe is obtained, with state variables $\left(p_i \quad p_j \quad Q_{ij}\right)$

$$\begin{pmatrix} \dot{p}_i \\ \dot{p}_j \\ \dot{Q}_{ij} \end{pmatrix} = \begin{pmatrix} \dfrac{B_{eff}}{½\,AL}\left(Q_i - Q_{ij}\right) \\ \dfrac{B_{eff}}{½\,AL}\left(Q_{ij} - Q_j\right) \\ \dfrac{A}{\rho L}\left[\left(p_i - p_j\right) - \left(K_L Q_{ij} + K_T Q_{ij}\left|Q_{ij}\right|\right)\right] \end{pmatrix} \tag{3.6}$$

3.2.4 Flow resistor elements

Flow resistor elements are characterized by the function $Q = f\left(\Delta p\right)$, where Q is the volumetric flow rate through the resistor and Δp is the pressure loss across the conductor. Fluid resistors dissipate fluid power, exchanging the power loss (Q x Δp) into heat. The above definition assumes that the rate of change in the flow is small enough to ignore the fluid inertia effects. The first flow resistor element introduced in this section is an *orifice*, defined as a sudden and short restriction. The orifice element is the most elemental flow resistor and it can be used as a subcomponent for building more complex flow resistor

models. Figure 3.4 shows a schematic of a spool valve (a flow resistor component), which consists of an orifice subcomponent and a spool dynamics model. The block accounting for the dynamics of the spool passes the cross-sectional flow path area A to the orifice element subcomponent.

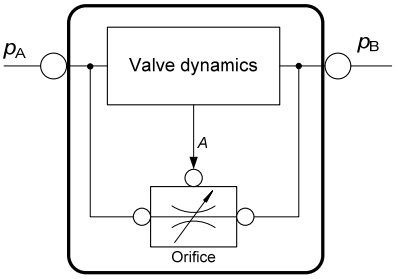


Figure 3.4. Flow resistor component with orifice element as a subcomponent

Orifice

The flow through an orifice is commonly determined by using the turbulent flow equation $Q = K(\Delta p)^{1/2}$. However, its Jacobian evaluation at the origin shows a singularity. Numerical integrators evaluating such Jacobian will fail to obtain a solution. In order to overcome this problem Ellman and Piché [Ellman 1996b] derived the following empirical two-regime fluid flow piece-wise equation

$$\begin{aligned} Q &= C_q A \sqrt{\frac{2\Delta p}{\rho}} && \Delta p > \Delta p_{tr} && (a) \\ Q &= \frac{3A\nu \,\mathrm{Re}_{tr}}{4d}\left(\frac{\Delta p}{\Delta p_{tr}}\right)\left(3 - \frac{\Delta p}{\Delta p_{tr}}\right) && 0 \le \Delta p \le \Delta p_{tr} && (b) \end{aligned} \qquad (3.7)$$

$$with \qquad \Delta p_{tr} = \frac{9\nu^2 \,\mathrm{Re}_{tr}^2 \,\rho}{8DC_q^2},$$

where a laminar flow equation (3.7)(b) describes the fluid flow for pressures lower than a transition pressure level Δp_{tr}. The quadratic spline polynomial provides a smooth transition between laminar and transition regimes and avoids the infinite derivative at the origin.

Pressure relief valve

A pressure relief valve can be a relatively complex element to model accurately. The behaviour of this component is mainly dictated by the valve internal geometry and the dy-

namics of the spool, which is governed by force quantities difficult to determine such as flow, jet, friction, spring and damping forces. A method used for modelling a single-stage pressure relief valve with a damping piston is presented in [Handroos 1990], where the analytical model of the valve is not based on physical parameters but in a reduced number of parameters which can be determined from measured characteristic curves of the valve. In order to achieve that, the model describing the dynamics of the spool is brought in the form of equation (3.8), where p_{ref} is the setting pressure of the valve, p_{in} is the pressure at the inlet port of the valve, K is a variable proportional to the cross-section area of the flow passage between the spool and the outlet port and Q is the volumetric flow through the valve. Parameters c_1 and c_2 are then identified from the typical pressure-flow characteristic curves shown in Figure 3.5 and from the set of equations (3.9).

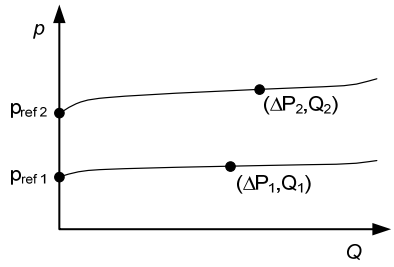


Figure 3.5. Pressure valve characteristic curve

$$\begin{aligned} \ddot{K} &= \left(p_{in} - p_{ref}\right)c_3^2 - 2c_3c_4\dot{K} - c_3^2\left(c_1 + c_2 p_{ref}\right)K \\ Q &= K\sqrt{p_{in}} \end{aligned} \tag{3.8}$$

$$\begin{aligned} c_1 &= \frac{p_1\sqrt{\Delta p_1}}{2Q_1} + \frac{p_2\sqrt{\Delta p_2}}{2Q_2} - \frac{p_1\Delta p_1}{2Q_1\sqrt{\Delta p_1}} - \frac{p_2\Delta p_2}{2Q_2\sqrt{\Delta p_2}} - \frac{\Delta p_2}{2\Delta p_1} - \frac{\Delta p_1}{2\Delta p_2} + 1 \\ c_2 &= \frac{p_1}{2Q_1\sqrt{\Delta p_1}} + \frac{p_2}{2Q_2\sqrt{\Delta p_2}} - \frac{1}{2}c_1\left(\frac{1}{\Delta p_1} + \frac{1}{\Delta p_2}\right) \\ p_1 &= \Delta p_1 - p_{ref1} \qquad\qquad p_2 = \Delta p_2 - p_{ref2} \end{aligned} \tag{3.9}$$

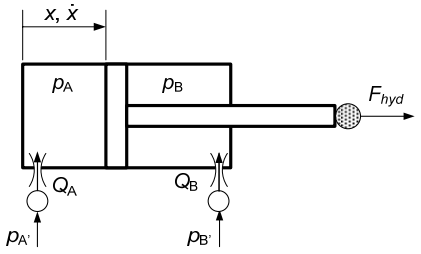
In equation (3.9), p_{ref1} and p_{ref2} are two different valve setting pressures in which the pairs (Δp_1,Q_1) and (Δp_2,Q_2) are measured. Values of the remaining parameters c_3 and c_4 of equation (3.8) are determined with dynamic response measurements of the valve.

3.2.5 Actuator elements

The ultimate purpose of a hydraulic system is to transmit its hydraulic power to mechanical power. Fluid power components providing this interface between mechanical and fluid power domains are mainly cylinder actuators and hydraulic motors. The formulation of a cylinder actuator is derived next.

Cylinder actuator

The formulation of a double acting cylinder actuator (schematized in Figure 3.6) can be written as a combination of the elements presented formerly. Pressures in cylinder chambers p_A and p_B are defined according to the pressure generation equation for volume elements in (3.3). Incoming flows Q_A and Q_B through the cylinder ports are calculated following the orifice flow equation (3.7), where $p_{A'}$ and $p_{B'}$ are the pressures in the adjacent volume elements connected to the cylinder ports. As a transformer element, the hydraulic cylinder communicates its hydraulic piston force F_{hyd} to a mechanical system. The dynamic equations describing the cylinder chambers pressures and the hydraulic piston force as a function incoming volumetric flows are shown in (3.10).



$$\begin{aligned} \dot{p}_A &= \frac{B_{eff}}{A_A x}\left(Q_A - \dot{x}A_A\right) \\ \dot{p}_B &= \frac{B_{eff}}{A_B\left(x_{\max} - x\right)}\left(Q_B + \dot{x}A_B\right) \\ F_{hyd} &= p_A A_A - p_B A_B - F_\mu \end{aligned} \tag{3.10}$$

Figure 3.6. Hydraulic cylinder

Position x and velocity $\dot{x}$ of the piston is determined by the dynamic equations governing the mechanical system connected to the hydraulic cylinder. Figure 3.7 shows a schematic representation of an actuator driving a multi-body mechanism with rotational joints, with joint coordinates q. The hydraulic force F_{hyd} is transmitted to the mechanical system as the joint torque Γ which drives the equations of motion of the mechanical device.

The solution of these equations returns the position x and the velocity $\dot{x}$ of the cylinder rod back to the hydraulic system (3.10).

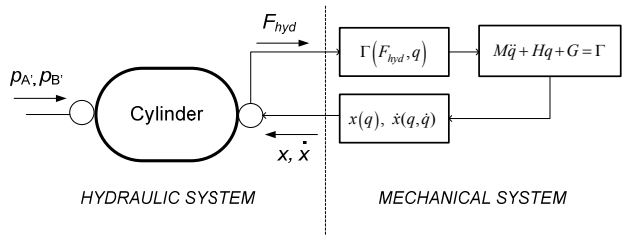


Figure 3.7. Exchange of hydraulic and mechanical variables between a hydraulic cylinder and a mechanical system

In applications where the external load F_{ext} acting on the cylinder is a known variable, the acceleration of the cylinder piston can be determined with Newton's second law. Adding this equation to (3.10), a set of a set of four ordinary differential equations is formed, with $\left(p_A \;\; p_B \;\; x \;\; \dot{x}\right)$ as state variables:

$$\begin{pmatrix} \dot{p}_A \\ \dot{p}_B \\ \dot{x} \\ \ddot{x} \end{pmatrix} = \begin{pmatrix} \dfrac{B_{eff}}{A_A x}\left(Q_A - \dot{x}A_A\right) \\ \dfrac{B_{eff}}{A_B\left(x_{\max} - x\right)}\left(Q_B + \dot{x}A_B\right) \\ \dot{x} \\ \dfrac{p_A A_A - p_B A_B - F_\mu - F_{ext}}{m} \end{pmatrix} \tag{3.11}$$

The friction force F_μ occurring in the cylinder seals is a highly non-linear phenomenon which depends on the pressure difference between the chambers and piston velocity $\dot{x}$. In [Olsson 1996], a relatively simple and realistic friction model is presented, where the friction force is given by

$$\begin{aligned} g\left(\dot{x}\right) &= \frac{1}{\sigma_0}\left(F_C + \left(F_S - F_C\right)e^{-\left(\dot{x}/\dot{x}_s\right)}\right) \\ \dot{z} &= \dot{x} - \frac{\left|\dot{x}\right|}{g\left(\dot{x}\right)}z \\ F_\mu &= \sigma_0 z + \sigma_1 \dot{z} + b\dot{x} \end{aligned} \tag{3.12}$$

where F_C and F_S are the Coulomb and static friction forces respectively, $\dot{x}_s$ is the transient velocity from static to Coulomb friction regimes and b is the viscous friction coefficient. The state variable z is related to the deformation of the cylinder seal. F_s/σ_0 is the maximum seal deformation and σ_1 is a damping term. Another well accepted friction model, derived from (3.12) by neglecting the dynamics associated to the deformation of the sealings, is the following:

$$F_\mu = \operatorname{sgn}(\dot{x})\left(F_C + (F_S - F_C)e^{-\left(\dot{x}/\dot{x}_s\right)} \right) + b\dot{x} \tag{3.13}$$

4 NUMERICAL INTEGRATION OF ODEs ARISING IN FLUID POWER SYSTEMS

Backward Differentiation Formula (BDF) belongs to the family of linear multi-step formulas (LMF). They were the first formulas proposed to solve stiff differential equations. Gear [Gear 1971] implemented BDFs in a variable order and variable step size code called DIFSUB. This code has shown to be efficient in solving general stiff differential equations. The code is still in use, together with improved versions of DIFSUB such as GEAR and LSODE codes. Despite the claimed good efficiency of these multi-step methods, the following drawbacks are found when the ODE equations to be integrated are originated in fluid power systems:

- The main disadvantage of multi-step methods was found by Dahlquist when analyzing their stability [Dahlquist 1963]. His theorem states that no linear multi-step method of order greater than 2 can be *A*-stable.
- The stability region of BDF methods decreases as the order of the formula increases. For many stiff problems, in which the eigenvalues do not have large imaginary components, the numerical stability is not compromised. However, ODEs whose Jacobians have eigenvalues with large imaginary parts, make the non *A*-stable BDF formulas very inefficient for such problems (the formula has to reduce excessively the integration step size in order to achieve stability conditions).
- BDF methods lack of accuracy near discontinuities, which are very common in fluid power systems.
- The above codes (DIFSUB, GEAR and LSODE) are implemented with fixed-coefficients (i.e. previous calculated points of the solution are equally spaced). Gear [Gear 1974] showed that codes based on fixed-coefficient implementation must restrict the frequency in which the step-size is changed during the integration in order to remain stable.

In the following sections of this chapter, the advantages of single-step methods over LMF are discussed. The computational efficiency of both numerical integration approaches is analyzed and compared in Section 4.1. Stability properties of single-step methods are investigated in Section 4.2. In Section 4.3, the equations of order conditions determining the order of accuracy and stability properties of Rosenbrock formulas are derived. Finally, the last section of this chapter illustrates how the analytic form of the Jacobian matrix can be automatically generated for any given system before the numerical integration starts. The advantages of using the analytical form of the Jacobian – over a Jacobian obtained by numerical techniques – are shown in Chapter 5.

4.1 On the efficiency of implicit Linear Multi-step and implicit Runge-Kutta formulas

Implicit Runge-Kutta formulas (RKF) are one-step methods with a multi-stage scheme that can be of high order and still retain *A*-stability. They do not present either the disadvantages of BDF listed above. The main problem associated to RKFs is that the computation of the solution requires the solution of a non-linear system of equations, which is excessively expensive to compute when compared to the costs involved in solving multi-step methods. In this section, the computational costs involved in the solution of multi-steps and one-step methods are analyzed and compared.

The general form of a linear multi-step formula (LMF) was introduced in Section 1.1.2 as

$$\sum_{i=0}^{k} \alpha_i \boldsymbol{y}_{n+1-i} = h \sum_{i=0}^{k} \beta_i \boldsymbol{f}\left(\boldsymbol{y}_{n+1-i}\right). \tag{4.1}$$

LMF computing the numerical solution $\boldsymbol{y}_{n+1}$ to the exact solution $\boldsymbol{y}(x_n + h)$ can be rewritten by grouping in a constant C the terms calculated in previous solution points, yielding

$$\boldsymbol{y}_{n+1} = h\beta_0 \boldsymbol{f}\left(\boldsymbol{y}_{n+1}\right) + \boldsymbol{C}. \tag{4.2}$$

Each step in (4.2) requires the solution of a non-linear system $\boldsymbol{\phi}(\boldsymbol{y}) = \boldsymbol{0}$, where $\boldsymbol{\phi}(\boldsymbol{y}_{n+1}) = \boldsymbol{y}_{n+1} - h\beta_0 \boldsymbol{f}\left(\boldsymbol{y}_{n+1}\right) - \boldsymbol{C}$. This is usually done by a modified form of Newton itera-

tion. The iterative Newton scheme computes in its *m*-th iteration the approximation to the solution $\boldsymbol{y}_{n+1}$ as $\boldsymbol{y}_{n+1}^{(m+1)} = \boldsymbol{y}_{n+1}^{(m)} + \Delta\boldsymbol{y}_{n+1}^{(m)}$, with $\Delta\boldsymbol{y}_{n+1}^{(m)}$ given by the linear system

$$\left(\boldsymbol{I} - h\beta_0\boldsymbol{J}\right)\left(\Delta\boldsymbol{y}_{n+1}^{(m)}\right) = -\boldsymbol{\phi}\left(\boldsymbol{y}_{n+1}^{(m)}\right), \tag{4.3}$$

being $\boldsymbol{J}$, the Jacobian matrix of the function $\boldsymbol{f}$ evaluated at $\boldsymbol{y}_n$, and $\boldsymbol{I}$ the $N \times N$-dimensional identity matrix. The iteration scheme is repeated until a convergence criterion is achieved. The cost involved in solving a *N*-dimensional ODE system $y_j' = f_j\left(y_1, \ldots, y_N\right)$ with $j = 1 \ldots N$, by means of a linear multi-step method is then determined from the Newton iteration scheme (4.3). The cost to solve each iteration *m* consists on:

- One function evaluation of $\boldsymbol{f}$.
- Evaluation of the Jacobian $\boldsymbol{J}$ and *LU*-factorization of the iteration matrix $\left(\boldsymbol{I} - h\beta_0\boldsymbol{J}\right)$. The *LU*-factorization of the iteration matrix is very costly, $O\,(N^3/3)$ operations. Fortunately, the same iteration matrix is employed for all Newton iterations required in one integration step.

The costs in each integration step can be reduced by using the same iteration matrix for a few number of steps. Some BDF codes use this approach in those cases where the Jacobian varies slowly from step to step.

On the other hand, the formulation of one-step implicit RKF is formed with intermediate stages $\boldsymbol{k}_i$. The *s*-stage implicit RKF has the form

$$\begin{aligned} &\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\sum_{i=1}^{s} b_i\boldsymbol{k}_i \\ &\text{where } \boldsymbol{k}_i = \boldsymbol{f}\left(\boldsymbol{y}_n + h\sum_{j=1}^{s} a_{ij}\boldsymbol{k}_j\right), \qquad i = 1, \ldots, s, \end{aligned} \tag{4.4}$$

where a_{ij} and b_i are free-choice parameters, whose values will determine the solution accuracy of the formula and its numerical stability. The implementation of fully implicit RKF is, by far, more costly than BDF methods. At each integration step, the following non-linear algebraic system of $s \times N$ equations must be solved:

$$\boldsymbol{\phi}_i\left(\boldsymbol{k}\right) = \boldsymbol{k}_i - \boldsymbol{f}\left(\boldsymbol{y}_n + h\sum a_{ij}\boldsymbol{k}_j\right), \qquad i = 1, \ldots, s, \tag{4.5}$$

being *N* the dimension of the system $\boldsymbol{y}' = \boldsymbol{f}\left(\boldsymbol{y}\right)$, and *s* the number of stages. The non-linear system is solved again with a modified Newton iteration scheme

$$\boldsymbol{k}_i^{(m+1)} = \boldsymbol{k}_i^{(m)} + \Delta\boldsymbol{k}_i^{(m)}, \qquad i = 1,...,s\,, \tag{4.6}$$

where the incremental values $\Delta\boldsymbol{k}_i^{(m)}$ are obtained by solving the $s \times N$-dimensional linear system

$$\begin{pmatrix} \boldsymbol{I} - ha_{11}\boldsymbol{J} & -ha_{12}\boldsymbol{J} & \cdots & -ha_{1s}\boldsymbol{J} \\ -ha_{21}\boldsymbol{J} & \boldsymbol{I} - ha_{22}\boldsymbol{J} & \cdots & -ha_{2s}\boldsymbol{J} \\ \vdots & \vdots & \ddots & \vdots \\ -ha_{s1}\boldsymbol{J} & -ha_{s2}\boldsymbol{J} & \cdots & \boldsymbol{I} - ha_{ss}\boldsymbol{J} \end{pmatrix} \begin{pmatrix} \Delta\boldsymbol{k}_1^{(m)} \\ \Delta\boldsymbol{k}_2^{(m)} \\ \vdots \\ \Delta\boldsymbol{k}_s^{(m)} \end{pmatrix} = \begin{pmatrix} -\boldsymbol{\phi}_1\left(\boldsymbol{k}^{(m)}\right) \\ -\boldsymbol{\phi}_2\left(\boldsymbol{k}^{(m)}\right) \\ \vdots \\ -\boldsymbol{\phi}_s\left(\boldsymbol{k}^{(m)}\right) \end{pmatrix} \tag{4.7}$$

The cost of a Newton iteration applied to a fully implicit RKF method is:

- s function evaluations of $\boldsymbol{f}$.
- Solution of the linear system (4.7), whose *LU*-factorization requires $O\left((sN)^3/3\right)$ operations. Usually, the Jacobian $\boldsymbol{J}$ is evaluated once for all the iterations within one integration step, and therefore, only one *LU*-factorization of the $s \times N$-dimensional iteration matrix $\boldsymbol{I} - h(\boldsymbol{A} \times \boldsymbol{J})$ is required at every integration step.

Such excessive linear algebra costs can be reduced by using Butcher's technique [Butcher 1976], that exploits the special structure of the iteration matrix in (4.7). By transforming the matrix $\boldsymbol{A}^{-1}$ to a Jordan canonical form, the *LU*-factors can be solved in diagonal blocks and therefore the *LU*-factorization of the iteration matrix is reduced. However, implicit RKF methods are still far from being competitive (in terms of efficiency) to BDF methods, since the latter only requires $O(N^3/3)$ operations to solve each iteration of the Newton scheme.

One way to reduce significantly the computational costs of implicit RKF is found in the *semi-implicit* Runge-Kutta formulas (SIRK) [Alexander 1977], also named *diagonally-implicit* Runge-Kutta formulas (DIRK). SIRK formulas are a particularization of implicit RKF in which matrix $\boldsymbol{A}$ is lower triangular (i.e. $a_{ij} = 0$ for $i < j$). As a result, the stages $\boldsymbol{k}_i$ in (4.4) can be solved successively i = 1, 2,…, s with only one N-dimensional non-linear system $\left(\boldsymbol{I} - ha_{ii}\boldsymbol{J}\right)$ to be solved at each stage. The computational cost per step in DIRK methods is reduced from $O((sN)^3/3)$ to $O(sN^3/3)$ operations.

A SIRK method with all the diagonal terms of matrix $\boldsymbol{A}$ equal ($a_{ii} = \gamma$ for i = 1,…,s) is called *singly diagonally implicit* (SDIRK) method:

$$\begin{aligned} &\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\sum_{i=1}^{s} b_i \boldsymbol{k}_i \\ &\text{where } \boldsymbol{k}_i = \boldsymbol{f}\left(\boldsymbol{y}_n + h\left(\sum_{j=1}^{i-1} a_{ij}\boldsymbol{k}_j + \gamma \boldsymbol{k}_i \right) \right), \qquad i = 1,...,s \end{aligned} \quad . \tag{4.8}$$

The advantage of having a lower triangular $\boldsymbol{A}$ matrix with equal diagonal terms is that all stages $\boldsymbol{k}_i$ can be successively solved by using the same *LU*-factorization of the iteration matrix $(\boldsymbol{I} - h\gamma\boldsymbol{J})$. The dominant costs of a SDIRK method are therefore reduced to *s* function evaluations per iteration and one *LU*-factorization of an *N*-dimensional iteration matrix per integration step, yielding to $O(N^3/3)$ operations.

In the above presented LMF and RK implicit methods, the Newton scheme is iterated several times at each integration step until certain convergence criterion is met. A new class of implicit methods, which have the advantage of not using an iteration scheme, is presented next. Rosenbrock formulas introduce the Jacobian term directly into the integration formula instead. Rosenbrock methods were first introduced in [Rosenbrock 1963] and they can be interpreted as the application of only a single Newton iteration at each stage k_i of a SIRK formula. This yields to the following formulation:

$$\begin{aligned} &\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \sum_{i=1}^{s} b_i \boldsymbol{k}_i \\ &\text{where } \boldsymbol{k}_i = h\boldsymbol{f}\left(\boldsymbol{y}_n + \sum_{j=1}^{i-1} a_{ij}\boldsymbol{k}_j \right) + h\boldsymbol{J}a_{ii}\boldsymbol{k}_i, \qquad i = 1,...,s \end{aligned} \quad . \tag{4.9}$$

Modified Rosenbrock methods (also known as ROW-methods, or generalized RKFs) can be seen as a generalization of (4.9), since they introduce linear terms of $\boldsymbol{J}\boldsymbol{k}_j$ to the stages $\boldsymbol{k}_i$ for *j*=1..*i*. By doing this, more freedom is obtained when establishing order conditions of accuracy and stability properties [Wolfbrandt 1973, Kaps 1979]. The modified Rosenbrock formula is written as

$$\begin{aligned} &\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \sum_{i=1}^{s} b_i \boldsymbol{k}_i \\ &\text{where } \boldsymbol{k}_i = h\boldsymbol{f}\left(\boldsymbol{y}_n + \sum_{j=1}^{i-1} a_{ij}\boldsymbol{k}_j \right) + h\boldsymbol{J}\sum_{j=1}^{i} \gamma_{ij}\boldsymbol{k}_j, \qquad i = 1,...,s \end{aligned} \quad . \tag{4.10}$$

Again, a formula with $\gamma_{ii} = \gamma$ is of special interest, since all stages $\boldsymbol{k}_i$ can be solved by using the same *LU*-factorization of $(\boldsymbol{I} - h\gamma\boldsymbol{J})$ for *i* = 1..*s*, as next equation sows:

$$\left(\boldsymbol{I}-h\gamma\boldsymbol{J}\right)\boldsymbol{k}_i = h\boldsymbol{f}\left(\boldsymbol{y}_n + \sum_{j=1}^{i-1} a_{ij}\boldsymbol{k}_j\right) + h\boldsymbol{J}\sum_{j=1}^{i-1}\gamma_{ij}\boldsymbol{k}_j, \qquad i = 1,...,s\ . \tag{4.11}$$

The fact that Rosenbrock methods do not require the solution of non-linear systems make them, potentially, very efficient for the integration of stiff systems of ODEs. The Rosenbrock method in (4.10) is the formula proposed in this thesis for the integration of ODEs arising in fluid power systems. So far, it seems that Rosenbrock formulas have not being used in fluid power systems, although they have proved to be very effective in other applications and in the solution of test equations for low-moderate accuracy requirements.

4.2 Stability properties of one-step methods

The stability of a numerical integration formula can be studied by analyzing the behaviour of the local and global errors of the solution. The global error (also called *true* error) ε_n of a numerical solution at the point t_n is defined as $\varepsilon_n = y_n - y(t_n)$, where y_n is the approximated solution computed by the formula, and $y(t_n)$ is the exact solution (usually unknown). The local error $\delta_h(t_n)$ of the solution is the error of the numerical formula after the integration of a single step starting from an exact solution $y(t_n)$. Therefore, the global error can be seen as a cumulative error, where errors add up during the integration, while the local error only measures the error made at each step.

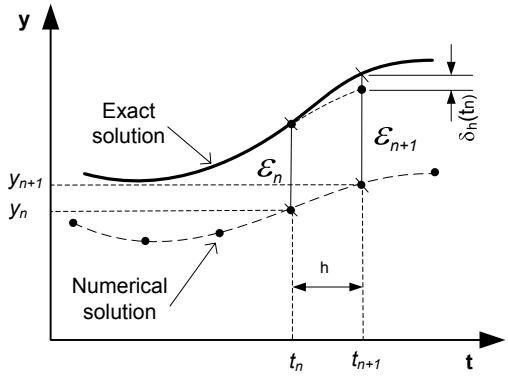


Figure 4.1. Local and global errors of a numerical solution

The stability of a one-step integration method is guaranteed when the error inequality (4.12) holds.

$$\|\varepsilon_{n+1}\| \le \|\varepsilon_n\| + \|\delta_h(t_n)\|. \tag{4.12}$$

The inequality states that the global error ε of the solution does not grow unboundedly as the integration advances.

There is not a stability analysis of RKF methods dealing with non-linear and stiff ODEs. Instead, to characterize and analyze the stability of a numerical method, the following test equation is used:

$$y' = \lambda y, \qquad with \;\; y(t_0) = y_0, \;\; \lambda \in \mathbb{C}, \;\; \mathrm{Re}(\lambda) \le 0. \tag{4.13}$$

The advantage of this test equation is that its analytical solution is known $y(t) = y_0 \exp(\lambda t)$. The behaviour of a numerical method in solving the test problem (4.13) can be extrapolated to predict its behaviour in solving a non-linear equation of the type $y' = f(y)$ [Gupta 1985], since this can be approximated by

$$y' = \frac{\partial f}{\partial y}(y - y_0) + f(y_0) \tag{4.14}$$

over a small interval [t_0, $t_0 + h$]. The term $f(y_0)$ seldom affects the stability and therefore $\partial f / \partial y$ may approximate λ. When dealing with a system of equations $\partial f / \partial y$ is then the Jacobian matrix, and the system $\boldsymbol{y}' = \boldsymbol{J}(\boldsymbol{y} - \boldsymbol{y}_0) + \boldsymbol{f}(\boldsymbol{y}_0)$ can be transformed to a set of equations $y' = \lambda(y - y_0) + f(y_0)$, where λ is an eigenvalue of $\boldsymbol{J}$. As a result, the test problem (4.13) can be used as a good model for analyzing the stability of numerical methods solving the general case $\boldsymbol{y}' = \boldsymbol{f}(\boldsymbol{y})$.

The test equation (4.13) applied to the forward Euler method $y_{n+1} = y_n + hf(y_n)$ yields to (4.15) when substituting function f by λy.

$$\frac{y_{n+1}}{y_n} = 1 + h\lambda = R. \tag{4.15}$$

R is the ratio of computed solutions at t_{n+1} and t_n and it is known as the *stability function* or the *amplification factor* of the numerical method. The stability function R of the test equation exact solution is

$$R = \frac{y(t_{n+1})}{y(t_n)} = e^{h\lambda}. \tag{4.16}$$

For all values of λ which make the exact solution of $y' = \lambda y$ stable, i.e. $\mathbb{R}(\lambda)<0$, the stability function R of the exact solution holds $|R(h\lambda)| \leq 1$. This becomes then the condition of stability for which the solution of a numerical method does not grow unbounded. Imposing the condition $|R(h\lambda)| \leq 1$ to the Euler method in (4.15), it turns that Euler method is stable if and only if $|1 + h\lambda| \leq 1$. Therefore Euler method is stable for values of $h\lambda = [-2, 0]$, assuming that $\lambda \in \mathbb{R}$.

Stability regions can be represented graphically for each numerical method. In the general case, where $\lambda \in \mathbb{C}$ are the eigenvalues of the Jacobian matrix of a set of equations, the stability regions are plotted in the complex $h\lambda$ plane, and show the values of $h\lambda$ which make the numerical method stable, i.e. regions where $|R(h\lambda)| \leq 1$.

In Figure 4.2 the stability regions (shaded in gray) of some numerical integration formulas are plotted. As stated above, the forward Euler method is stable whenever $|1 + h\lambda| \leq 1$, which is the inner area of a circle or radius 1 and centre in (-1,0) in the complex $h\lambda$ plane. This method is therefore not suited for the integration of stiff ODEs having large negative eigenvalues λ_i, since very small step sizes h should be required in order to fit the method within its stability region. It is desirable then that numerical methods to be used for the integration of stiff ODEs should be stable for a large region in the left half-plane ($h\lambda$ with negative real part), since the left-half plane is the location of all the eigenvalues λ which makes the exact solution of stable too.

Numerical methods whose stability region comprises the entire left-half $h\lambda$ plane ($|R(h\lambda)| \leq 1$ for all real λ) are called *A*-stable. In other words, *A*-stable methods are stable for any positive time step h whenever λ has a negative real part. According to the stability regions plotted in Figure 4.2, the trapezoidal rule, with $R(h\lambda) = (1 + \frac{1}{2}h\lambda)/(1 - \frac{1}{2}h\lambda)$, is stable in the whole left-hand side $h\lambda$ plane, while the backward Euler formula, with $R(h\lambda) = 1/(1 - h\lambda)$, is stable for all $h\lambda$ values except for the unit circle cantered at (1,0). Both integrations formulas are therefore *A*-stable.

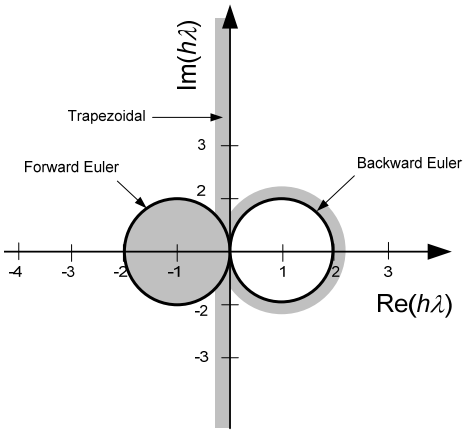


Figure 4.2. Stability regions of some simple one-step formulas.

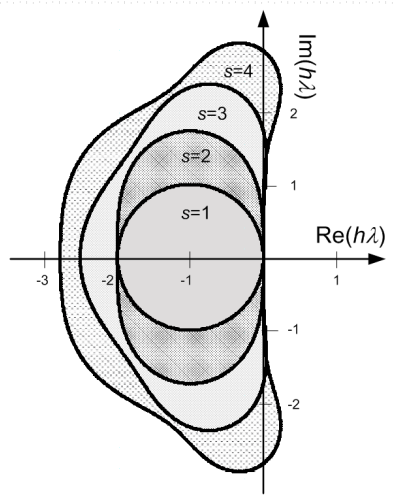


Figure 4.3. Stability regions of explicit RKFs, with *p*=*s*.

A-stability might not be a sufficient condition for numerical methods dealing with stiff systems of ODEs. Another desired property for the numerical method is that its stability function accomplish $|R(h\lambda)| \ll 1$ as $h\lambda \to -\infty$. This requirement can be understood when solving the test equation (4.13) with a very large negative λ. Since $-1/\lambda$ is the time constant of the exact solution, a $\lambda \to -\infty$ means that the solution decays exponentially to zero immediately. This asymptotic behaviour should be then also reflected in the numerical formula, yielding

$$\frac{y_{n+1}}{y_n} \to 0 \qquad \text{as} \qquad h\lambda \to -\infty . \tag{4.17}$$

Numerical methods not satisfying the previous condition cannot damp out fast enough the solutions of very stiff ODEs. As a consequence, stability and accuracy problems may arise. An example of such behaviour is shown in Figure 4.4, where a stiff system is integrated by means of two different methods:

- Backward Euler formula: $y_{n+1} = y_n + hf(y_{n+1})$, with $R(h\lambda) = 1/(1-h\lambda)$
- Trapezoidal rule: $y_{n+1} = y_n + \frac{h}{2} f(y_n + y_{n+1})$, with $R(h\lambda) = \frac{(1+\frac{1}{2}h\lambda)}{(1-\frac{1}{2}h\lambda)}$

From their stability regions, plotted in Figure 4.2, it can be concluded that both methods are *A*-stable and therefore, for any size of the integration step *h*, the numerical stability should be granted. Nonetheless, it will be seen in the following integration example that *A*-stability does not always lead to an optimum integration when stiff systems are present. The stiffness in the hydraulic circuit example in Figure 4.4 arises due to the size difference of the two volumes, where $V_2/V_1 = 100$ (the same ratio is observed between the eigenvalues λ_1 and λ_2 of the Jacobian matrix). Due to this stiffness, slow asymptotic convergence problems may arise. Examining again the stability functions of the formulas, the trapezoidal method shows the following asymptotic behaviour $|R(-\infty)| = 1$, while in the backward Euler method $|R(-\infty)| = 0$. The asymptotic behaviour of the trapezoidal method is therefore undesired when integrating stiff systems. This can be seen in the plot of Figure 4.4, where the solution of the pressure p_1 provided by the trapezoidal rule results in an oscillatory solution. Despite these oscillations, the solution is considered to be stable, as long as the oscillations gradually vanish. On the other hand, the backward Euler method provides a satisfactory numerical solution without numerical oscillations around the transient response.

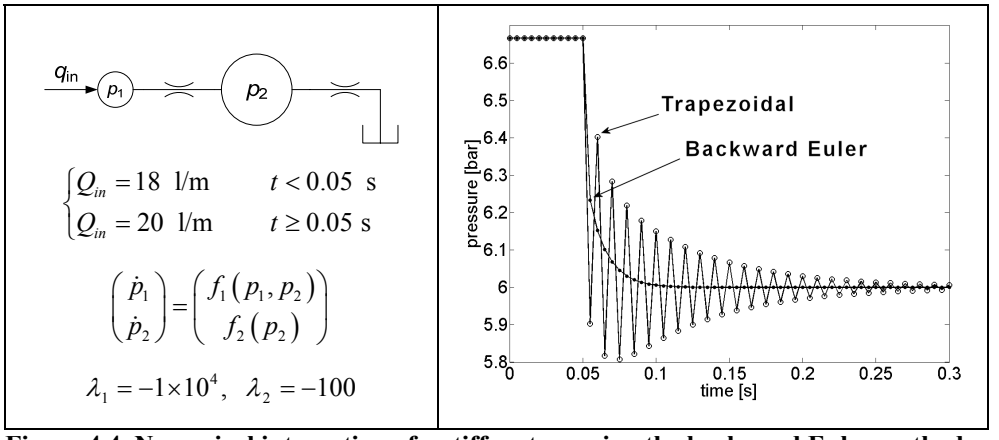


Figure 4.4. Numerical integration of a stiff system using the backward Euler method and the trapezoidal rule.

A-stable methods with maximally damped behaviour $|R(h\lambda)| = 0$ as $h\lambda \to -\infty$ are called *L*-stable. The advantages of *L*-stable methods over non *L*-stable ones have been shown in the previous example. The oscillations shown in non *L*-stable methods can only

be reduced or avoided by making the integration step small enough, which makes the integration less efficient.

The need of an *L*-stable formula for the integration of stiff systems of ODEs is therefore justified if integration efficiency is sought. Advantages of low order *L*-stable Rosenbrock formula over other SIRK methods are also pointed out in [Piché 1996], where the robust stability properties of the method are praised. In [Esqué 2002b] the stability of *L*-stable Rosenbrock formula with a variable step size is tested in the dynamic simulation of some hydraulic components. The formula shows, again, excellent stability against discontinuities and numerical stiffness.

4.3 Equations of conditions for modified Rosenbrock formulas

The overall efficiency of a numerical formula should not be only measured in terms of the number of operation required to advance one step. As it has been showed in the previous section, an efficient integration is also subject to the stability properties of the formula and to the nature of the system to be integrated. Since the characteristics of the system equations governing the fluid power circuits are well-known, an efficient numerical integration formula for those applications should retain the following properties:

- Computationally inexpensive linear algebra
- Excellent stability properties
- Able to detect and handle discontinuities
- Simple implementation of a step size predictor, based on a local error estimation

The Rosenbrock method (4.10) is a good candidate to compete with the current numerical methods used in the integration of fluid power systems. The main characteristics of Rosenbrock methods are listed next:

- Reduced computational costs: the method can achieve full accuracy and stability with a single Newton iteration of an *N*-dimensional iteration matrix per integration step.
- Rosenbrock methods can be *L*-stable.
- Error estimators can be embedded into the formula with almost no extra numerical costs.

The main drawback of Rosenbrock methods is that not only the evaluation of the full Jacobian must be provided at every integration step, but also this Jacobian must be computed accurately. Otherwise, the numerical stability of the Rosenbrock formula might be affected

by inaccuracies in the Jacobian. This difficulty is dealt in Section 4.4, where a systematic approach for obtaining the analytic expression of the Jacobian matrix for any given hydraulic system is presented.

This Section focuses on the Rosenbrock formulas introduced in (4.10). Conditions concerning the order of accuracy of the formula and its stability are also presented.

4.3.1 Order conditions

The free parameters a_{ij}, γ_{ij}, b_i of the s-stage Rosenbrock formula

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \sum_{i=1}^{s} b_i \boldsymbol{k}_i$$

$$\text{where } \boldsymbol{k}_i = h\boldsymbol{f}\left(\boldsymbol{y}_n + \sum_{j=1}^{i-1} a_{ij}\boldsymbol{k}_j\right) + h\boldsymbol{J}\sum_{j=1}^{i}\gamma_{ij}\boldsymbol{k}_j, \quad i = 1,...,s$$

will determine the order of accuracy of the integration formula and they will also characterize its numerical stability. A method is of order p if the local error $y_n - y(x_{n-1} + h) = O\left(h^{p+1}\right)$, where y_n is the numerical solution, y is the exact solution and h is the integration step size. Order conditions for the free parameters can be determined by differentiation [Hairer 1996] or by applying the theory of Butcher series [Hairer 1974, Kaps 1981]. In the first approach, the order conditions for the free parameters are determined when comparing the Taylor series of the test equation exact solution to the Taylor series of the numerical method. By doing so, the conditions for the free parameters are defined by equalling the derivative terms of both Taylor expansions. This differentiation approach is straightforward for establishing order conditions of lower Rosenbrock formulas. For higher orders conditions the formulation becomes too complex (formulas of order 6 and 8 require 37 and 200 order conditions respectively) and a new form of representation is needed. This new notation consists of a graphical representation called *labelled trees*. The vertices of labelled trees represent summation indices and the derivative terms of the Taylor expansions can be represented and built with this notation. Labelled trees have been used in the derivation of order conditions of Runge-Kutta methods in [Hairer 1993]. In [Kaps 1981] labelled trees are applied to the Butcher series theory to derive high order Rosenbrock methods.

The table below lists the order conditions for the free parameters of Rosenbrock methods up to order 4, where $\beta_{ij} = a_{ij} + \gamma_{ij}$. An expansion of this table with order condition up to order 6 can be found in [Kaps 1981].

Table 4.1. Order conditions for Rosenbrock formulas (Source: [Kaps 1981])

Order	Order condition
1	$\sum_j b_j = 1$
2	$\sum_{j,k} b_j \beta_{jk} = \frac{1}{2} - \gamma$
3	$\sum_{j,k,l} b_j a_{jk} a_{jl} = \frac{1}{3}$ $\sum_{j,k,l} b_j \beta_{jk} \beta_{jl} = \frac{1}{6} - \gamma - \gamma^2$
4	$\sum_{j,k,l.m} b_j a_{jk} a_{jl} a_{jm} = \frac{1}{4}$ $\sum_{j,k,l.m} b_j a_{jk} \beta_{kl} a_{jm} = \frac{1}{8} - \frac{\gamma}{3}$ $\sum_{j,k,l.m} b_j \beta_{jk} a_{kl} a_{km} = \frac{1}{12} - \frac{\gamma}{3}$ $\sum_{j,k,l.m} b_j \beta_{jk} \beta_{kl} \beta_{lm} = \frac{1}{24} - \frac{\gamma}{2} + \frac{3\gamma^2}{2} - \gamma^3$

4.3.2 Stability conditions

Recalling from Section 4.2, one-step methods applied to the scalar test equation (4.13) can be expressed as $y_{n+1} = R(h\lambda) y_n$, where R is the stability function of the formula. The method is then stable if and only if $|R(h\lambda)| \le 1$. If the previous inequality is valid for any arbitrary $h\lambda$, with Re(λ)<0, then the formula is called *A*-stable. If in addition of being *A*-stable, the stability function of the formula also accomplish $|R(h\lambda)| = 0$ as $h\lambda \to -\infty$, then the formula is *L*-stable. In this section, the stability functions of RKF and Rosenbrock methods are presented. They will be used later to determine the free parameters that make Rosenbrock formulas *L*-stable.

The stability function of an explicit *s*-stage RKF:

$$\begin{aligned} k_1 &= f\left(y_n\right) \\ k_2 &= f\left(y_n + h a_{21} k_1\right) \\ k_3 &= f\left(y_n + h\left(a_{31} k_1 + a_{32} k_2\right)\right) \\ &\cdots \\ k_s &= f\left(y_n + h\left(a_{s1} k_1 + \ldots + a_{s,s-1} k_{s-1}\right)\right) \\ y_{n+1} &= y_n + h\left(b_1 k_1 + \ldots + b_s k_s\right) \end{aligned} \tag{4.18}$$

can be explicitly computed from (4.18) as the following polynomial of degree $\leq s$:

$$R\left(h\lambda\right) = 1 + h\lambda \sum_j b_j + \left(h\lambda\right)^2 \sum_{j,k} b_j a_{jk} + \left(h\lambda\right)^3 \sum_{j,k,l} b_j a_{jk} a_{kl} + \cdots . \tag{4.19}$$

The above stability function shows that explicit RK methods cannot be *A*-stable, since its region of stability $\left|R\left(h\lambda\right)\right| \leq 1$ is bounded. Figure 4.3 plots the stability regions of *s*-stage explicit RKFs for *s* = 1...4.

The *s*-stage implicit RKF (4.4) applied to the test equation (4.13) yields the stability function (4.20) [Stetter 1973], in which *R* becomes a rational function whose numerator and denominator polynomials are function of the free parameters.

$$R\left(z\right) = \frac{P\left(z\right)}{Q\left(z\right)} = \frac{\det\left(\boldsymbol{I} - z\boldsymbol{A} + z\boldsymbol{1}\boldsymbol{b}^T\right)}{\det\left(\boldsymbol{I} - z\boldsymbol{A}\right)} \tag{4.20}$$

In the above stability function, z = $h\lambda$, $\boldsymbol{I}$ is the identity matrix, $\boldsymbol{A}$ is the matrix of RK coefficients a_{ij}, $\boldsymbol{b}$ is the vector of RK elements b_i and $\boldsymbol{1}$ is a vector of ones. Implicit RKF are *A*-stable if the degree of the polynomial *P* is not larger than the degree of *Q*.

In SDIRK methods (4.8) – a particularization of implicit RK formulas with a lower triangular $\boldsymbol{A}$ matrix and equal diagonal terms – the denominator of the rational function *R* simplifies to $Q\left(z\right) = \left(1 - \gamma z\right)^s$. The polynomial *P* can be rewritten [Kaps 1979] as

$$P\left(z\right) = \sum_{k=0}^{s} L_k^{(s-k)}\left(\tfrac{1}{\gamma}\right)\left(-\gamma z\right)^k , \tag{4.21}$$

where $L_k^{(m)}\left(x\right) = \sum_{i=0}^{k} \left(-1\right)^i \frac{\left(k+m\right)!}{\left(m+i\right)!\ \left(k-i\right)!} \frac{x^i}{i!}$ is the *m*-th derivative of the *k*-degree Laguerre polynomial, which outputs a polynomial with real coefficients and one variable *x*.

Rosenbrock methods of the form (4.10) applied to the scalar test equation show the same stability function *R* as SDIRK methods, that is

$$R(z)=\frac{1}{(1-\gamma z)^{s}}\sum_{k=0}^{s}L_{k}^{(s-k)}\left(\tfrac{1}{\gamma}\right)(-\gamma z)^{k}. \tag{4.22}$$

This stability function R of Rosenbrock formulas is then uniquely determined by the parameter γ, and therefore the regions of stability can be defined as a function of this parameter, as Table 4.2 displays.

Table 4.2. Values of the free parameter γ, for which *A*- and *L*- stability are achieved. (Source: [Hairer 1996])

	Rosenbrock formulas, with $p = s$		**Stiffly Accurate Rosenbrock formulas, with $p = s$-1**
Order *s*	***A*-stability**	***L*-stability**	***A*- and *L*-stability**
2	1/4…∞	$(2\pm\sqrt{2})/2$	$(2-\sqrt{2})/2$ … $(2+\sqrt{2})/2$
3	1/3…1.06857902	0.43586652	0.18042531…2.18560010
4	0.39433757…1.28057976	0.57281606	0.22364780…0.57281606
5	0.24650519…0.36180340 0.42078251…0.47326839	0.27805384	0.24799464…0.67604239
6	0.28406464…0.54090688	0.33414237	0.18391465…0.33414237

An interesting group of Rosenbrock formulas are the so-called Stiffly Accurate formulas. These methods are built by imposing the following condition on the free parameters

$$\begin{aligned} &a_{si}+\gamma_{si}=b_{i} \qquad \text{for} \qquad i=1,\ldots,s \\ &a_{s}=1 \end{aligned}. \tag{4.23}$$

The above conditions force the numerical solution y_n to be exactly the same as the last internal stage s. The benefit of imposing such condition is that the highest coefficient of P(z) in (4.21) becomes zero and therefore, the stability function R of a stiffly accurate Rosenbrock formula always satisfies $R(z)=0$ at $z\to-\infty$, i.e. the formula becomes L-stable.

4.4 Analytical form of the Jacobian matrix

One of the drawbacks of Rosenbrock methods lies in that they need to be provided with the instantaneous value of the full Jacobian matrix of the system at every integration step. Such demand can affect negatively the computational efficiency of Rosenbrock meth-

ods. The construction of the Jacobian matrix from a system of ODEs can, in some cases, become the most time consuming task involved in the numerical integration. In [Esqué 2005], it is shown that the computation of the Jacobian, by means of numerical approximation, can account up to 40% of the computational costs required to advance one step in the numerical integration. The test was performed by integrating middle-sized fluid power circuits (from 5 to 20 state variables) using a single-step Rosenbrock numerical formula, which required two function evaluations per step.

Yet another disadvantage of Rosenbrock methods arises due to the fact that the Jacobian matrix in the integration formula itself, rather than Jacobian being part of a Newton iteration scheme. This implies that the accuracy of the provided Jacobian affects both solution accuracy and the numerical stability of the integration formula. The computation of an accurate Jacobian for each integration step might become too computationally expensive and, in some cases, an accurate or realistic Jacobian might even be difficult to calculate, for example near discontinuities. The latter could even make the numerical integration unstable.

The calculation of an analytical form of the Jacobian matrix – as a function of the state variables of the system – guarantees an accurate evaluation of the Jacobian at each integration point. In addition, such evaluation of the analytical Jacobian is also less computationally expensive than obtaining the Jacobian by numerical differentiation. However, the derivation of an analytical Jacobian from the set of ODEs is not always a straightforward task and in most cases the associated symbolic manipulation can be a tedious and error-prone task to perform, especially when the dimension of the system is relatively large. The modelling approach presented in Chapter 2 makes possible to obtain the analytical Jacobian matrix of a fluid power circuit in a systematic way. The process can be automated by an algorithm, able to generate a subroutine containing the algebraic expression of the Jacobian matrix. This subroutine will then return the numerical evaluation of the Jacobian at a specific integration point when called by the numerical integration formula.

Figure 4.5 shows a flowchart of the tasks involved during the numerical integration of a system of ODEs by means of a Rosenbrock formula. The pre-process described above, in which the analytical Jacobian matrix is computed, is also illustrated. Next Section describes in more detail how the analytical Jacobian matrix of a fluid power circuit is calculated from a formal definition of the fluid power circuit.

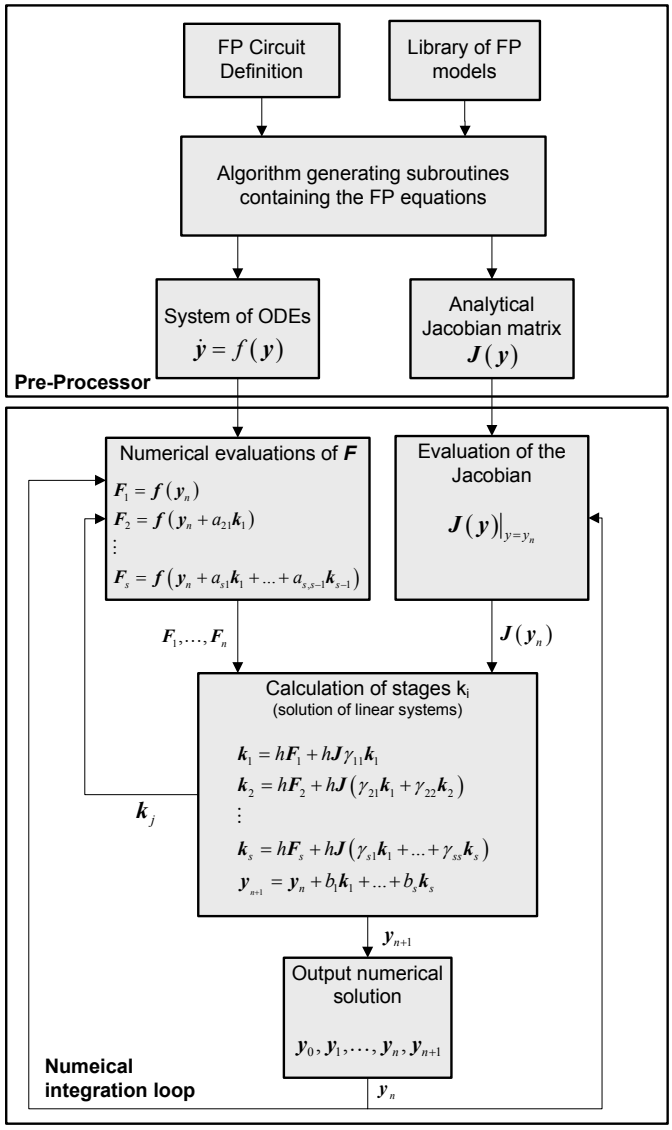


Figure 4.5. Flowchart of the tasks involved in a numerical integration with a Rosenbrock formula

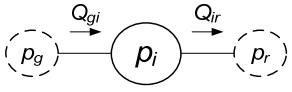
4.4.1 Jacobian of individual components

For each dynamic model of a fluid power component stored in the simulation models *library*, information of its Jacobian matrix must be also available. This section illustrates how Jacobians of individual components are calculated, stored, and indexed so that they can be retrieved and assembled into the full Jacobian matrix of a fluid power circuit.

In the following example, illustrated in Table 4.3, the analytical Jacobian of a volume element is presented. The Jacobian of a single component is derived from the set of ODEs describing its dynamics (Chapter 2). In this case this is the pressure generation equation $\dot{p}_i = F_i\left(p_g, p_i, p_r\right)$ of the *volume* component V_i with two hydraulic ports. These hydraulic ports will be connected to other volume elements (dashed circles) when assembled into a larger system. As showed in the middle row of Table 4.3, Jacobian elements J_{ii}, J_{ig}, and J_{ir} are determined analytically by derivation of F_i with respect to the state variables p_i, p_g and p_h respectively. The indexing of the state variables with subscripts (i, g, h) determines the position of these Jacobian elements within the full Jacobian matrix of the system (an $N \times N$ -dimensional matrix, where N is the number of state variables). The indexing of the state variables establishes the connections between the different ports of the fluid power elements. The indexing is carried out during the definition of the fluid power circuit model.

Table 4.3. Jacobian of a volume component

Formulation as a system of ODEs



$$\dot{p}_i = F_i = \frac{B_i\left(p_i\right)}{V_i} \times \left[Q_{gi}\left(p_g, p_i\right) - Q_{ir}\left(p_i, p_r\right) \right]$$

Jacobian elements

$$J_{ii} = \frac{\partial F_i}{\partial p_i} = \frac{1}{V_i} \times \left(\frac{\partial B_i}{\partial p_i} \times \left(Q_{gi} - Q_{ir} \right) + B_i \times \left(\frac{\partial Q_{gi}}{\partial p_i} - \frac{\partial Q_{ir}}{\partial p_i} \right) \right)$$

$$J_{ig} = \frac{\partial F_i}{\partial p_g} = \frac{B_i}{V_i} \times \frac{\partial Q_{gi}}{\partial p_g} \qquad J_{ir} = \frac{\partial F_i}{\partial p_r} = -\frac{B_i}{V_i} \times \frac{\partial Q_{ir}}{\partial p_r}$$

Location of the Jacobian elements in the full Jacobian matrix

	1	…	*g*	…	*i*	….	*r*	…	*N*
1	0	…	0	…	0	…	0	…	0
⋮	⋮		⋮		⋮		⋮		⋮
i	0	…	J_{ig}	…	J_{ii}	…	J_{ir}	…	0
⋮	⋮		⋮		⋮		⋮		⋮
N	0	…	0	…	0	…	0	…	0

The expressions of the volumetric flow rate Q_{gi} and Q_{ir} are known from Section 3.2.4 to be

$$\begin{cases} Q_{gi} = C_q A \sqrt{\dfrac{2\left(p_g - p_i\right)}{\rho}} & \left(p_g - p_i\right) > \Delta p_{tr} \\ Q_{gi} = \dfrac{3A\nu \,\mathrm{Re}_{tr}}{4d}\left(\dfrac{\left(p_g - p_i\right)}{\Delta p_{tr}}\right)\left(3 - \dfrac{\left(p_g - p_i\right)}{\Delta p_{tr}}\right) & 0 \le \left(p_g - p_i\right) \le \Delta p_{tr} \end{cases}, \tag{4.24}$$

with their partial derivatives with respect to pressure variables expressed as

$$\begin{cases} \dfrac{\partial Q_{gi}}{\partial p_g} = \dfrac{C_q A}{\sqrt{2\rho\left(p_g - p_i\right)}} & \left(p_g - p_i\right) > \Delta p_{tr} \\ \dfrac{\partial Q_{gi}}{\partial p_g} = \dfrac{3A\nu \,\mathrm{Re}_{tr}\left(2\left(p_g - p_i\right) - 3\Delta p_{tr}\right)}{4d\Delta p_{tr}} & 0 \le \left(p_g - p_i\right) \le \Delta p_{tr} \end{cases} \tag{4.25}$$

$$\frac{\partial Q_{gi}}{\partial p_i} = -\frac{\partial Q_{gi}}{\partial p_g}$$

The algebraic expression of the pressure dependent bulk modulus $B_i\left(p_i\right)$ is formed from equation (3.2), as

$$B_i\left(p_i\right) = \left(b_B + p_i\right)\left[\frac{1}{a_B} - \ln\left(1 + \frac{p_i}{b_B}\right)\right], \tag{4.26}$$

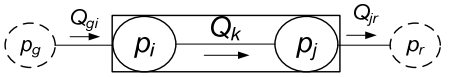
and its partial derivative with respect to pressure p_i is

$$\frac{\partial B_i}{\partial p_i} = -\ln\left(\frac{p_i + b_B}{b_B}\right) - \frac{a_B - 1}{a_B} \tag{4.27}$$

Table 4.4 and Table 4.5 show the construction of the analytical Jacobian elements associated to a pipeline and a cylinder actuator respectively.

Table 4.4. Jacobian of a pipeline component

Formulation as a system of ODEs



$$\dot{p}_i = F_i = \frac{2B_i(p_i)}{AL} \times \left[Q_{gi}(p_g, p_i) - Q_k \right]$$

$$\dot{p}_j = F_j = \frac{2B_j(p_j)}{AL} \times \left[Q_k - Q_{jr}(p_j, p_r) \right]$$

$$\dot{Q}_k = F_k = \frac{A}{\rho L}\left(p_i - p_j - K_L Q_k - K_T Q_k |Q_k| \right)$$

Jacobian elements

$$J_{ii} = \frac{\partial F_i}{\partial p_i} = \frac{2}{AL} \times \left(\frac{\partial B_i}{\partial p_i} \times \left(Q_{gi} - Q_k \right) + B_i \times \frac{\partial Q_{gi}}{\partial p_i} \right)$$

$$J_{jj} = \frac{\partial F_j}{\partial p_j} = \frac{2}{AL} \times \left(\frac{\partial B_j}{\partial p_j} \times \left(Q_k - Q_{jr} \right) - B_j \times \frac{\partial Q_{jr}}{\partial p_j} \right)$$

$$J_{kk} = \frac{\partial F_k}{\partial Q_k} = -\frac{A}{\rho L} \times \left(K_L + K_T |Q_k| + K_T Q_k \operatorname{sgn}(Q_k) \right)$$

$$J_{ik} = \frac{\partial F_i}{\partial Q_k} = -\frac{2B_i}{AL} \qquad J_{jk} = \frac{\partial F_j}{\partial Q_k} = \frac{2B_j}{AL} \qquad J_{ki} = \frac{\partial F_k}{\partial p_i} = \frac{A}{\rho L} \qquad J_{kj} = \frac{\partial F_k}{\partial p_j} - \frac{A}{\rho L}$$

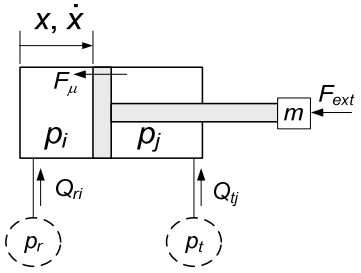
$$J_{ig} = \frac{\partial F_i}{\partial p_g} = \frac{2B_i}{AL} \times \frac{\partial Q_{gi}}{\partial p_g} \qquad J_{jr} = \frac{\partial F_j}{\partial p_r} = -\frac{2B_j}{AL} \times \frac{\partial Q_{jr}}{\partial p_r}$$

Location of the Jacobian elements in the full Jacobian matrix

	1	…	*g*	…	*i*	*j*	*k*	….	*r*	…	*N*
1	0	…	0	…	0	0	0	…	0	…	0
⋮	⋮		⋮		⋮	⋮	⋮		⋮		⋮
i	0	…	J_{ig}	…	J_{ii}	0	J_{ik}	…	0	…	0
j	0	…	0	…	0	J_{jj}	J_{jk}	…	J_{jr}	…	0
k	0	…	0	…	J_{ki}	J_{kj}	J_{kk}	…	0	…	0
⋮	⋮		⋮		⋮	⋮	⋮		⋮		⋮
N	0	…	0	…	0	0	0	…	0	…	0

Table 4.5. Jacobian of a cylinder actuator component

Formulation as a system of ODEs



$$\dot{p}_i = F_i = \frac{B_i\left(p_i, x\right)}{A_i x}\left[Q_{ri}\left(p_r, p_i\right) - \dot{x}A_i\right]$$

$$\dot{p}_j = F_j = \frac{B_j\left(p_j, x\right)}{A_j\left(x_{\max} - x\right)}\left[Q_{tj}\left(p_t, p_j\right) + \dot{x}A_j\right]$$

$$\dot{x} = F_k = \dot{x}$$

$$\ddot{x} = F_l = \frac{p_i A_i - p_j A_j - F_\mu\left(\dot{x}\right) - F_{ext}}{m}$$

Jacobian elements

$$J_{ii} = \frac{\partial F_i}{\partial p_i} = \frac{1}{A_i x} \times \left(\frac{\partial B_i}{\partial p_i} \times \left(Q_{ri} - \dot{x}A_i\right) + B_i \times \frac{\partial Q_{ri}}{\partial p_i} \right)$$

$$J_{jj} = \frac{\partial F_j}{\partial p_j} = \frac{1}{A_j\left(x_{\max} - x\right)} \times \left(\frac{\partial B_j}{\partial p_j} \times \left(Q_{tj} + \dot{x}A_j\right) + B_j \times \frac{\partial Q_{tj}}{\partial p_j} \right)$$

$$J_{ll} = \frac{\partial F_l}{\partial \dot{x}} = -\frac{1}{m} \times \frac{\partial F_\mu}{\partial \dot{x}} \qquad J_{ik} = \frac{\partial F_i}{\partial x} = \frac{Q_{ri} - \dot{x}A_i}{A_i x^2} \times \left(\frac{\partial B_i}{\partial x} x - B_i \right) \qquad J_{il} = \frac{\partial F_i}{\partial \dot{x}} = -\frac{B_i}{x}$$

$$J_{jl} = \frac{\partial F_j}{\partial \dot{x}} = \frac{B_j}{x_{\max} - x} \qquad J_{jk} = \frac{\partial F_j}{\partial x} = \frac{Q_{tj} + \dot{x}A_j}{A_j\left(x_{\max} - x\right)^2} \times \left(\frac{\partial B_j}{\partial x} \times \left(x_{\max} - x\right) + B_j \right)$$

$$J_{kl} = \frac{\partial F_k}{\partial \dot{x}} = 1 \qquad J_{li} = \frac{\partial F_l}{\partial p_i} = \frac{A_i}{m} \qquad J_{lj} = \frac{\partial F_l}{\partial p_j} = -\frac{A_j}{m}$$

$$J_{ir} = \frac{\partial F_i}{\partial p_r} = \frac{B_i}{A_i x} \times \frac{\partial Q_{ri}}{\partial p_r} \qquad J_{jt} = \frac{\partial F_j}{\partial p_t} = \frac{B_j}{A_j\left(x_{\max} - x\right)} \times \frac{\partial Q_{tj}}{\partial p_t}$$

Location of the Jacobian elements in the full Jacobian matrix

	1	…	*i*	*j*	*k*	*l*	….	*r*	…	*t*	…	*N*
1	0	…	0	0	0	0	…	0	…	0	…	0
⋮	⋮		⋮	⋮	⋮			⋮		⋮		⋮
i	0	…	J_{ii}	0	J_{ik}	J_{il}	…	J_{ir}	…	0	…	0
j	0	…	0	J_{jj}	J_{jk}	J_{jl}	…	0	…	J_{jt}	…	0
k	0	…	0	0	J_{kk}	0	…	0	…	0	…	0
l	0	…	J_{li}	J_{lj}	0	J_{ll}	…	0	…	0	…	
⋮	⋮		⋮	⋮	⋮			⋮		⋮		⋮
N	0	…	0	0	0		…	0	…	0	…	0

4.4.2 Construction of the full Jacobian matrix

The analytical form of the full Jacobian matrix can be defined by collecting the Jacobian elements associated to each of the individual fluid power components of the system [Esqué 2005]. The algorithm making this task must be provided with a formal definition of the hydraulic circuit. In this formal definition:

- The fluid power components of the system are listed and their state variables are indexed with integers, going from 1 to N (as shown in the circuit of Figure 4.6).
- Connections between different components are defined by pairing their ports. The above index notation is used to describe these connections.

In order to illustrate the processes involved in building the full Jacobian matrix from a generic circuit, the example shown in Figure 4.6 is followed:

i) The system of ODEs is formed as $\dot{y}_i = F_i(y_1, y_2, ..., y_{11})$ for i = 1…11, where the vector of state variables $\boldsymbol{y} = \left[p_1 \;\; p_2 \;\; x_3 \;\; x_4 \;\; p_5 \;\; p_6 \;\; Q_7 \;\; p_8 \;\; p_9 \;\; Q_{10} \;\; p_{11} \right]^T$ is formed according to the given indexes. The functions F_i are obtained from the *library* of fluid power elements (Tables 4.3-5) by assigning the subscript indexes $i, j, k,\ldots$ according to the state variable vector y. For example, the state variable y_i with index i = 11 corresponds to the pressure p_{11} of a volume component with hydraulic ports connected to state variables 6 and 9. According to the *volume* component formulation stored in the library, the ODE associated to this fluid power component is then defined as

$$\dot{p}_{11} = F_{11} = \frac{B_{11}(p_{11})}{V_{11}} \times \left[Q_{6,11}(p_6, p_{11}) - Q_{11,9}(p_{11}, p_9) \right] \tag{4.28}$$

ii) The same approach is used to determine the Jacobian elements J_{ij} of the individual fluid power components in the system. The algorithm reads the formal description of every component, it accesses the library, it assigns the indexes, and it returns the analytic expressions of the Jacobian elements.

iii) The full Jacobian matrix is formed by assigning the Jacobian elements J_{ij} to the i-th row and j-th column of the full Jacobian matrix.

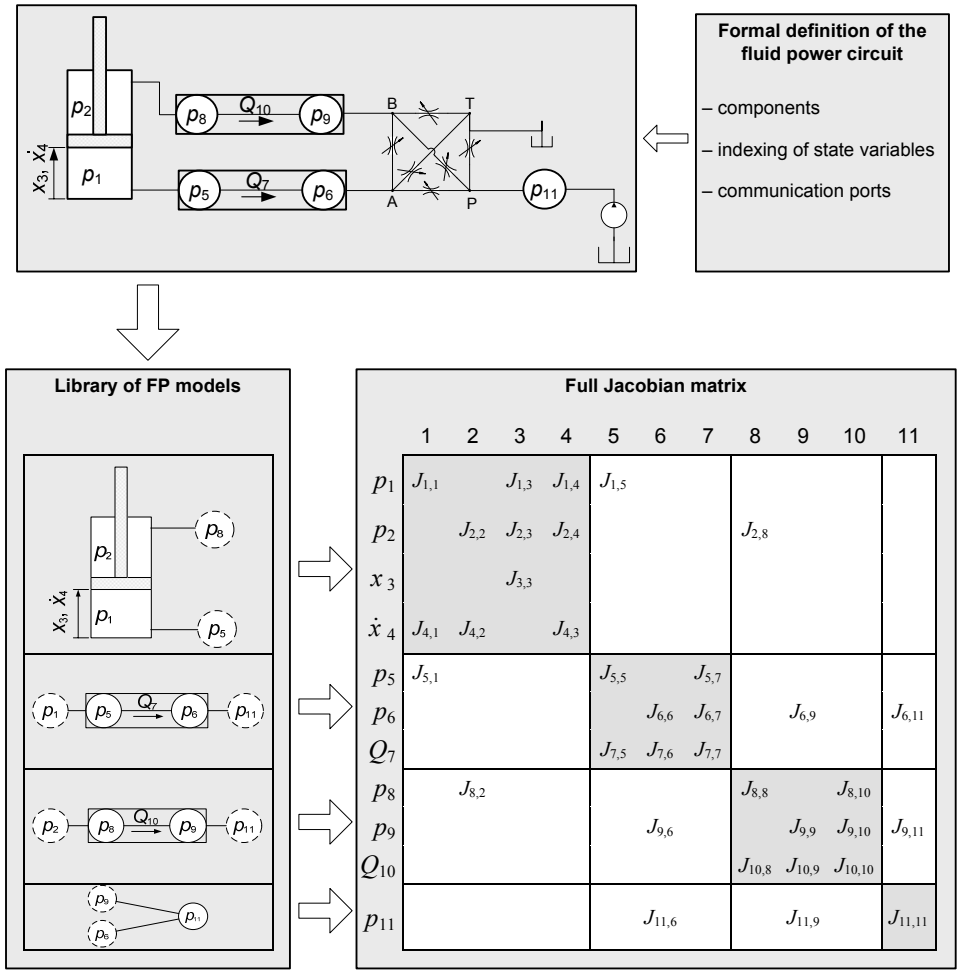


Figure 4.6. Construction of the analytical Jacobian matrix for a generic hydraulic circuit

5 PERFORMANCE OF ROSENBROCK FORMULAS

In this chapter Rosenbrock formulas are tested and compared against other numerical integration methods. The performance of Rosenbrock formulas are evaluated in two different types of simulation: real-time and offline simulations. In real-time simulations, the simulation time matches the clock time and numerical integration is performed with a constant step size throughout the entire simulation. Offline simulations usually advance with integration steps of different sizes, which are controlled by an error estimator. Real-time and offline simulations require numerical integration formulas with different properties.

At the end of this chapter, the advantages of using Jacobian matrices obtained analytically over the ones obtained by numerical approximation are discussed. It will be shown that the first approach enhances stability and accuracy, while it reduces the computational time of the integration.

5.1 Real-time simulations

Real-time simulations are found in many applications where simulated results need to be computed and acquired in real-time (clock time). Numerical integration methods for real-time simulations must show the following properties:

- Advance the integration with a constant step size h in order to provide solutions at equidistant time intervals.
- Computational efficiency: CPU time required to advance the integration of the system $y' = F(y)$ from $y(t)$ to $y(t+h)$ must be less or equal than the clock time interval h.
- Good numerical stability properties in order to integrate successfully stiff equations with a given fix step size h.
- Provide an acceptable solution accuracy.

Explicit Runge-Kutta formulas of low order of accuracy are often used in real-time simulations. They have the advantage of having a simple algebraic formulation and therefore fast computational times. However, it is known that the main drawback of explicit formulas is their poor numerical stability when integrating stiff systems of ODEs.

Singly diagonally implicit Runge Kutta (SDIRK) formulas of the Rosenbrock class have the disadvantage of being more costly in terms of computational operations. They need to form a Jacobian in each integration step and solve a linear system of equations for each stage in every integration step. An analysis of the computational costs involved in implicit Runge-Kutta formulas was already presented in Section 4.1. Concerning numerical stability, SIRK formulas have clear advantages over the explicit RK ones, especially when the formulas have to deal with numerically stiff systems.

Table 5.1 shows all numerical formulas (explicit and semi-implicit) used in the real-time integration tests carried out in this section. A short description of each formula is given below.

Table 5.1. Real-time integration algorithms (FE = function evaluations)

Method family	Order of accuracy	Method	Properties
RK Fully-Implicit	Order 5	**RADAU5**	Used for obtaining the approximate exact solution
RK-Explicit	Order 3	**ODE23**	3 FE
	Order 5	**DOPRI5**	6 FE
ROSENBROCK	Order 2	**ROS2**	2 FE; 2 Stages
		ROS2p	2 FE; 2 Stages
		ODE23s	2 FE; 2 Stages
	Order 3	**ROS3p**	2 FE; 3 Stages
		RODAS3	3 FE; 4 Stages

RADAU5 [Hairer 1996] is an implicit Runge-Kutta code based on the 3-satge Radau IIA method [Butcher 1964b].

ODE23 and DOPRI5 are explicit Runge-Kutta formulas. The first is a 2(3) pair of Bogacki & Shampine [Bogacki 1989] and the latter is a 5(4) pair by Dormand & Prince [Dormand 1980]. Both formulas ore offered as ODE solvers in the MATLAB® software package.

ROS2 is an *L*-stable second-order two-stage SDIRK formula proposed by Verwer [Verwer 1999] to solve partial differential equations arisen in photochemical dispersion problems. In his paper, Verwer highlighted the good stability properties of the formula when using large integration step sizes.

ROS2p is a SDIRK *L*-stable two-stage second-order method proposed by Piché & Ellman [Piché 1994]. The formula was compared to other popular SIRK formulas by means of integrating a simple fluid power circuit test. The authors claimed that that the use of ROS2p was the best choice for simulating that particular numerically stiff test.

ODE23s is another second-order *L*-stable Rosenbrock formula by [Shampine 1997] which is also offered as a numerical integration solver for stiff systems in the MATLAB/Simulink® package. The integrator can be used as a MATLAB command (*ode23s*) or it can be chosen from the Simulink solvers list.

ROS3p [Lang 2000] is a third-order *A*-stable method with stability function $\left|R(\infty)\right| \approx 0.73$. It is realized with three stages and only two function evaluations. Lang derived this efficient solver claiming that it retained its third-order accuracy when solving stiff non-linear parabolic problems.

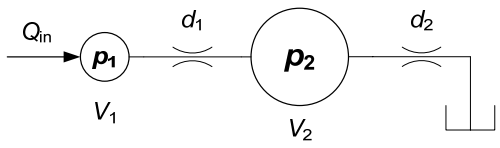
RODAS3 [Sandu 1996] was designed following the same principles as the *L*-stable Rosenbrock solver RODAS4 [Hairer 1991] but reducing its order from four to three.

5.1.1 Test circuits

Fluid power tests circuits are used to evaluate the performance of the numerical methods listed in Table 5.1, among them the Rosenbrock formulas. The performance of each method will be measured in terms of a) numerical accuracy (by comparing the numerical solution to an approximate exact solution) and b) the computational time required to execute the integration.

Circuit #1: (stiffness)

This is a simple fluid power circuit consisting of two volumes V_1 and V_2. A variable flow Q_{in} is entering V_1 and going to V_2 through an orifice of diameter d_1, and from V_2 to a tank through another orifice d_2. The main characteristic of this circuit is the relative size of the volumes with respect to each other, which is of two orders of magnitude. This characteristic makes this circuit to behave as stiff and therefore it constitutes a simple but yet a challenging integration problem. The schematic of the circuit as well as the size of its components is shown in Figure 5.1.



$V_1 =$ 0.05 l
$V_2 =$ 1 l
$d_1 =$ 4 mm
$d_2 =$ 4 mm
$Q_{in} = 40 + 10 \times \sin(2\pi \times 10t)$ l/min

Figure 5.1. Test circuit #1: Two-volume two-orifice fluid power circuit

Circuit #2: (friction and discontinuities)

This test circuit contains a cylinder actuator controlled with a 4/3 proportional valve. As shown in the schematic of Figure 5.2, the system also comprises pipes (whose mathematical model considers fluid flow inertia) and a pressure relieve valve. The mathematical formulation of all these fluid power components is presented in detail in Chapter 3. The most relevant physical parameters of the circuit are shown in Table 5.2.

The numerical integration algorithm solving this test has to overcome the following two difficulties: mechanical friction and discontinuities. The cylinder seal friction is modelled in this test with static, Coulomb and viscous friction forces components (see equation (3.13)). The friction behaviour is highly non-linear and also adds relatively fast transients to the overall system, especially when the cylinder piston oscillates around an equilibrium position. Discontinuities might originate during the commanding of the proportional valve. Fast openings and closures of the valve induce abrupt and even discontinuous changes in volumetric fluid flows. Another source of discontinuities is found in the pressure relief valve, whose spool operates with small constant times (of the order of milliseconds).

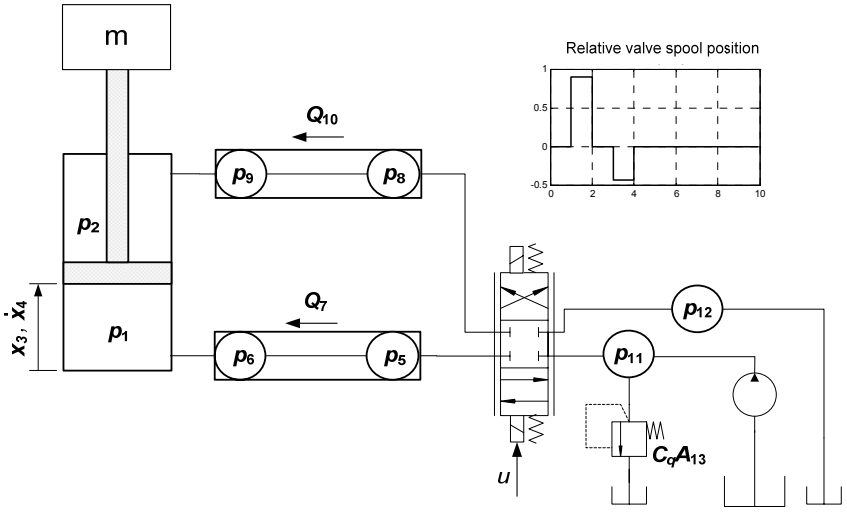
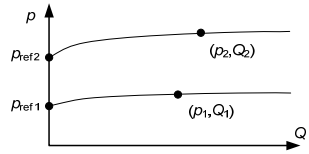


Figure 5.2. Test circuit #2

Table 5.2. Parameter values of the simulation model sketched in Figure 5.2

Cylinder		Pipes		PRV			
$ø_p/ø_r$ - stroke	50/28–500 mm	L	3 m				
m	1000 kg	D	10 mm				
Fc	200 N	B	10^7 MPa				
Fs	500 N	ξ	100				
b	500 N m^{-1} s^2	Other					
$\dot{x}_s$	20 mm s^{-1}	V_{11}	1 l	pref1	10 Mpa	pref2	12 Mpa
$ø_{ports}$	9.5 mm	V_{12}	1 l	p1	11 Mpa	p2	13 Mpa
		$ø_{orifices}$	4 mm	Q1	10 l/min	Q2	60 l/min

5.1.2 Numerical tests

Test 1

In this test, the Circuit #1 of Figure 5.1 is solved using all the numerical integration algorithms listed in Table 5.1. In addition, each of these numerical codes will integrate the circuit employing five different integration step sizes, ranging from 0.25 to 10 milliseconds.

The input volumetric flow Q_{in} follows a sinusoidal profile with a frequency of 10 Hz and with an amplitude of 10 l/min. After t = 0.8 s the input flow is kept constant. The volu-

metric flow profile is shown in the left plot of Figure 5.3. The numerical integration of the Circuit #1, resulting from the previous input flow, gives the solutions p_1 and p_2, shown in the right-hand side plot of Figure 5.3. This numerical integration was carried out employing the RADAU5 algorithm, a Runge-Kutta fully-implicit order-5 method, using variable step size. The level of accuracy was set by imposing relative and absolute error tolerances of 10^{-8}. The numerical solution given by RADAU5 can be considered, for our purposes, as an approximation to the exact solution.

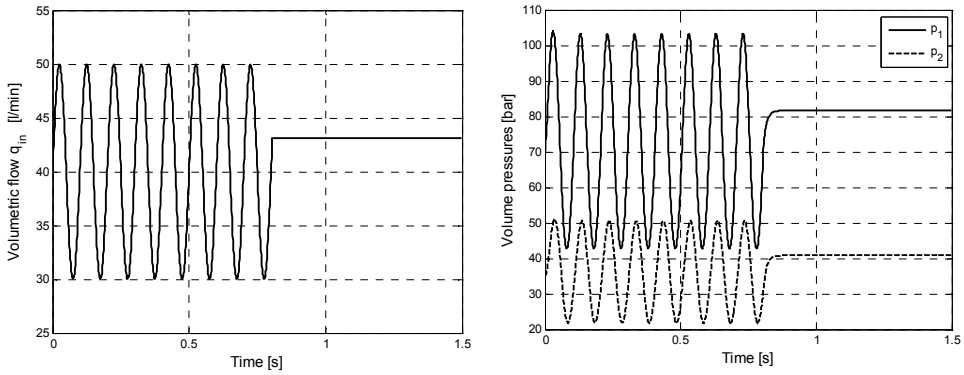


Figure 5.3. Test 1: Incoming volumetric flow (left-hand side) and approximate exact solution (right-hand side)

The accuracy of the numerical integration is obtained when comparing the solution $\boldsymbol{y}(x)$ of that formula to the exact solution $\overline{\boldsymbol{y}}(x)$ obtained from the RADAU5 formula. Since the latter integration has advanced with variable step size, the solutions at the integration time points $x = 0, h, 2h \ldots$ have been calculated in the RADAU5 integration by means of interpolation, implemented in the subroutine "contr5". The integration relative error (a scalar) of each component i of the solution $\boldsymbol{y} = \begin{bmatrix} p_1 & p_2 \end{bmatrix}^T$ is obtained using the root-mean-square (RMS):

$$\boldsymbol{e}^{(i)} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(\frac{y_j^{(i)} - \overline{y}_j^{(i)}}{\overline{y}_j^{(i)}}\right)^2} \tag{5.1}$$

where n is the total number of integration steps (length of the solution). Finally, the integration error accounting all the components of the solution is computed as the norm of vector

e. This relative error, obtained from the numerical integration of Test 1, is plotted in Figure 5.4 using a logarithmic scale. The error has been computed for all numerical formulas listed in Table 5.1. The experiment has been repeated using five different integration step sizes *h*. Numerical integrations, using certain *h*, which did not succeed (due to stability problems) can be identified in the plot as the ones which have not an error bar. Bars with green highlighted borders represent minimum errors among all formulas integrating with same integration step *h*.

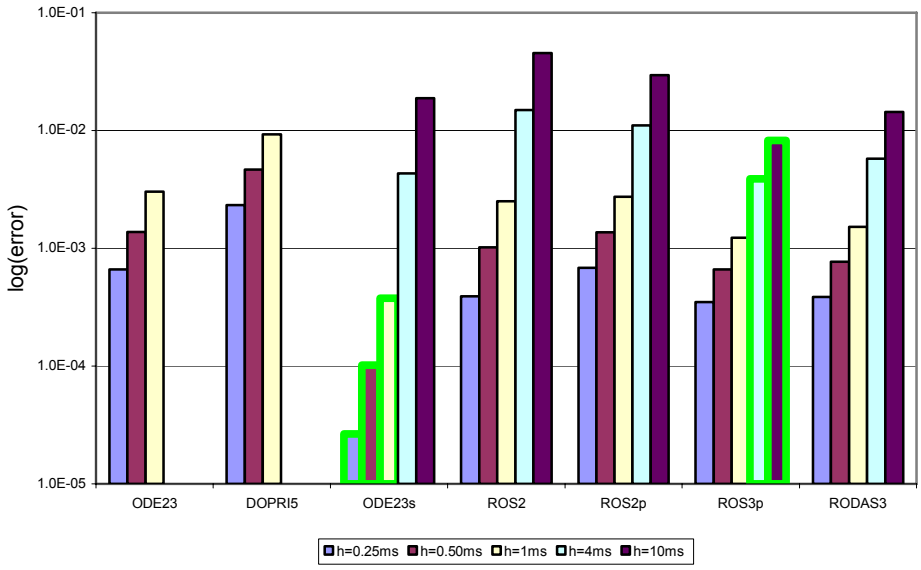


Figure 5.4. Test 1: Relative error of the numerical solutions (an absence of error bar means that the numerical integration failed)

Analysis of Test 1:

- In this first test, volume sizes and orifice diameters have been chosen (values displayed in Figure 5.1) with the purpose of making the system not too stiff, i.e. the numerical problem can also be solved with explicit integration formulas.
- The above figure shows that Rosenbrock formulas provide more accurate solutions than the explicit formulas. Rosenbrock formulas advancing the integration with h = 4 ms provide similar accuracy as explicit formulas using h = 1 ms.

- Among the Rosenbrock formulas, ODE23s gives the best accuracy for integration steps h = 0.25..1 ms. For larger h, the solution provided by ROS3p is more accurate than the rest of Rosenbrock formulas.
- Explicit methods suffer from instability and fail to integrate the test problem when using integration step sizes of 4 and 10 ms.

Test 2

This test is based on the same fluid power circuit as the previous Test 1. However the stiffness of the system has been increased by decreasing the size of V_1 from 0.05 to 0.01 l. Flow and pressure transients have also been augmented by enlarging the orifice diameter d_1 to 6 mm and by amplifying the sinusoidal input flow $Q_{in} = 60 + 40 \times \sin(2\pi \times 10t)$ l/min. The profile of the new input flow Q_{in} and plots of the approximate exact solution p_1, p_2 of this test are shown in Figure 5.5.

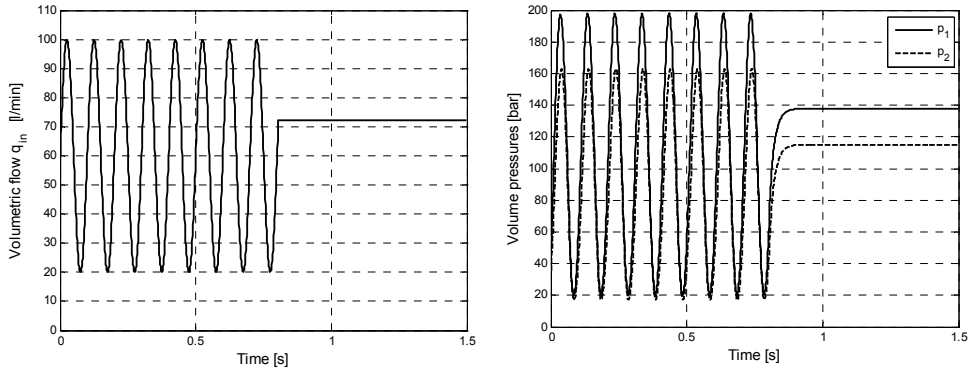


Figure 5.5. Test 2: Incoming volumetric flow (left-hand side) and approximate exact solution (right-hand side)

The accuracy of the numerical formulas of Table 5.1 is displayed in the bar diagram of Figure 5.6 for different fixed-size integration step h.

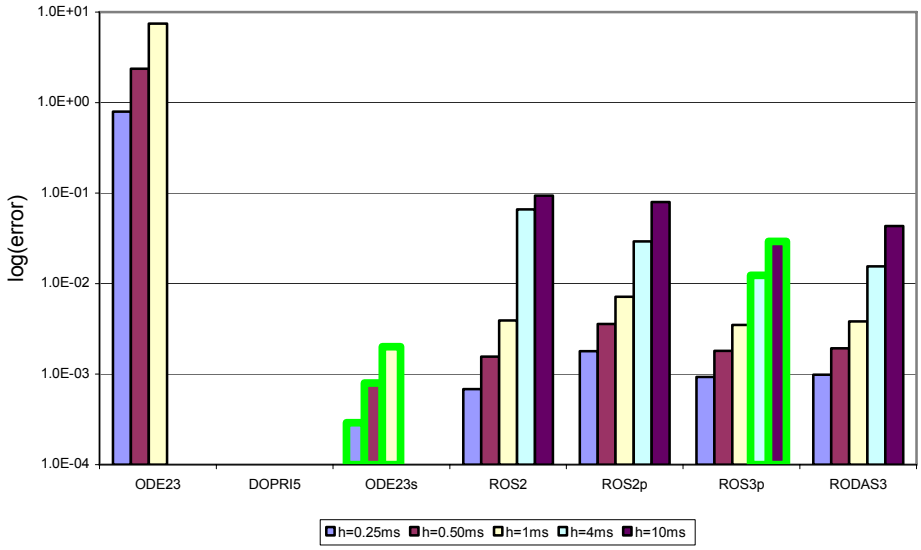


Figure 5.6. Test 2: Relative error of the numerical solutions (an absence of error bar means that the numerical integration failed)

Analysis of Test 2:

- The numerical stiffness of the system is increased, when compared to previous Test 1.
- The lack of stability of ODE23, caused by the system stiffness, is responsible of the high errors in the solution (more than 100% of relative error). DOPRI5 did not provide a satisfactory solution for any of the integration step sizes.
- Among the Rosenbrock formulas, ODE23s still is the more accurate formula for h = 0.25..1 ms although it is not capable of integrating successfully the numerical problem for h = 4 and h = 10 ms.

Test 3

This numerical test consists on the numerical integration of the fluid power circuit model displayed in Figure 5.2. This circuit includes fluid power components which are present in many fluid power applications, such as pipelines, control and pressure valves, linear actuators. Another characteristic of this test is that the dimension N of the system of ODEs has increased from N=2 to N=13.

Figure 5.7 shows the approximate exact solution of the Test 3 when solved with the RADAU5 formula. The hydraulic circuit is fed with a constant flow source of 25 l/min and the hydraulic actuator is controlled by a proportional valve whose spool is driven by signal *u*. As it can be seen in Figure 5.2, the profile of this signal is discontinuous and therefore this can induce stability problems to the numerical integrator. A similar discontinuous behaviour is observed in the pressure relief valve, whose flow passage (plotted in Figure 5.7 as *CqA*) changes abruptly. The same figure also shows that the cylinder seal friction, with its large oscillations, might also cause numerical stability problems to the solver.

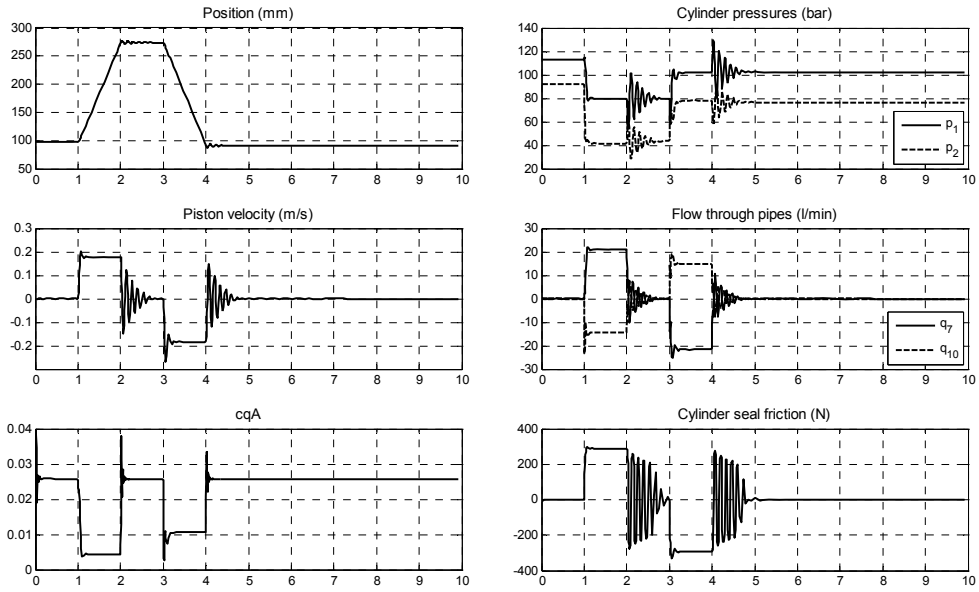


Figure 5.7. Exact solution to the Test Circuit #3 in Figure 5.2

The accuracy of the numerical solution provided by the integration formulas in Table 5.1 is shown in Figure 5.8. The bars show the maximum relative error of the cylinder piston position, when compared to the approximate exact solution.

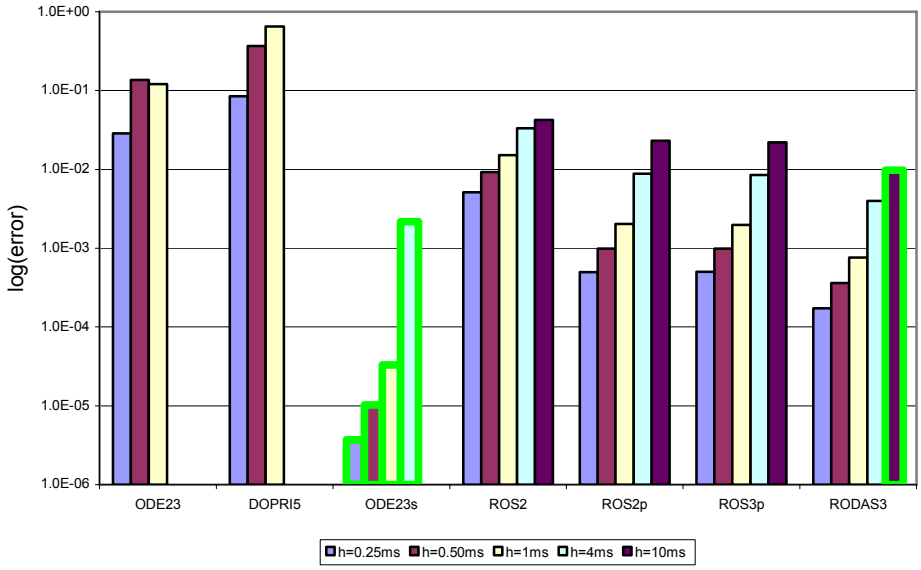
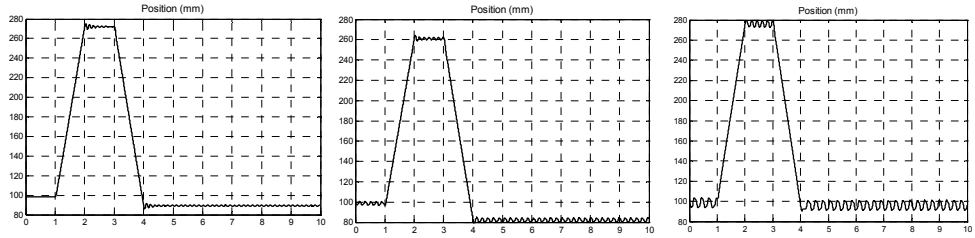


Figure 5.8. Relative error of the numerical solutions (an absence of error bar means that the numerical integration failed)

Analysis of Test 3:

- It can be seen that explicit formulas ODE23 and DOPRI5 fail to integrate the system for the larger step sizes of $h = 4$ and $h = 10$ ms.
- Explicit formulas can produce a numerical solution with a maximum integration step size of 1 ms. Nonetheless, numerical oscillations (typically shown by explicit methods when solving stiff systems) can be seen in the solution, even for the smaller $h = 0.25$ ms (see Figure 5.9)
- Similarly to the previous tests, ODE23s fails the integration when using the larger integration step sizes. In this case the solution becomes unstable for $h = 10$ ms while the rest of Rosenbrock formulas are providing a successful numerical solution.



h = 0.25 ms h = 0.5 ms h = 1 ms

Figure 5.9. Numerical oscillations in the solution shown by explicit ODE23 formula

5.1.3 Computational time

The computational time required to advance the integration one step of size h, is of considerable importance in the real-time simulations. A common practice in real-time simulations is to increase the size of the integration step h if the computational processor unit is not able to provide the required $1/h$ solutions per unit time. By increasing the integration step size, not only the accuracy of the solution decreases but also the stability of the numerical integration is affected. The latter increases the risk of a simulation *crash* (i.e. numerical solution does not converge, resulting in an overflow). The computational time required to advance one step the integration depends on many factors of different nature:

- Processor unit speed
- Size of the mathematical model (dimension of the system of ODEs to be solved)
- Formulation of the numerical integration method. In the case of single-step implicit methods (see Section 4.1 for a detailed formulation and its associated computational costs):
    - Number of Newton iterations per step
    - Number of function and Jacobian evaluations per step
    - Number of stages (backsolves or solutions of linear system) per step

Computational times of the numerical integration formulas employed in the previous tests have been measured and they are presented in Figure 5.10. All measurements have run in the same computer (equipped with a 2.0 GHz Intel Core Duo processor) and under the same conditions. These results must be analyzed taking into account the two different types of numerical methods employed. In one hand, the explicit methods (ODE23 and DOPRI5) are theoretically the fastest since they do not use Jacobians nor they need to solve algebraic

linear systems. Their computational efficiency is approximately proportional to the number of function evaluations. On the other hand, Rosenbrock formulas (ODE23s, ROS2, ROS2p, ROS3p and RODAS3) need to build one Jacobian matrix and solve multiple linear systems (as many as stages) in each integration step.

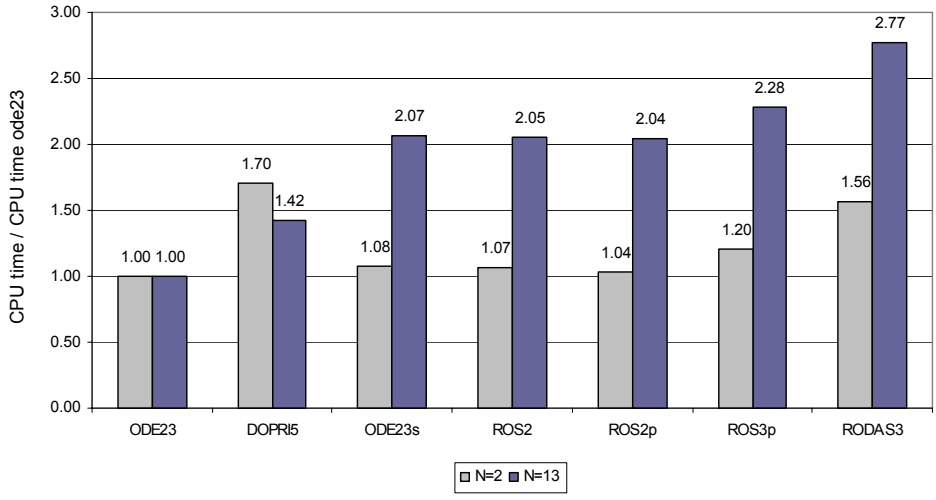


Figure 5.10. CPU time employed to integrate Circuit #1 (*N*=2) and Circuit #2 (*N*=13), relative to CPU time employed by formula ODE23

The above figure displays the computational time required to integrate Circuit #1 (with dimension *N*=2) and Circuit #2 (with dimension *N*=13) employing the explicit and implicit numerical formulas in Table 5.1. Computational times (CPU time) in Figure 5.10 are relative values with respect to the CPU time employed by ODE23, the fastest algorithm. For the ODE23 formula, the absolute CPU time required to advance one integration step was 1.6 μs and 14.0 μs for *N*=2 and *N*=13 respectively.

For ODE systems of *N*=2, Rosenbrock formulas of order 2 only require 4-8% more CPU time that explicit formulas of the same order, while order 3 Rosenbrock formulas need 20-56% more CPU time to advance the integration step. Differences in CPU time, between explicit and implicit formulas, are more accentuated when integrating larger systems of ODEs. In this case, for a system of dimension *N*=13, the CPU time required to solve one integration step employing an order 2 Rosenbrock formula doubles when compared to the time employed by ODE23.

5.1.4 Conclusions of real-time integration tests

Low order SDIRK Rosenbrock formulas have clear advantages over classical explicit RK formulas when employed as numerical solvers of real-time integrations. These advantages are reflected in their superior numerical stability and accuracy. The drawback of SDIRK Rosenbrock formulas is that the computational costs do not grow linearly with the dimension o the system (as explicit formulas do). However, Rosenbrock methods can overcome this problem by employing larger integration step sizes (if the real-time application allows it). On the other hand, explicit RK formulas may be forced to use smaller (than required) integration step sizes in order to keep the numerical integration stable.

5.2 Offline simulations

Offline simulations can be understood as those simulations whose execution time is not synchronized with the clock time. Usually, full power of digital computer's CPU is used to execute the numerical integration.

All of the numerical formulas used in these offline simulation tests (listed in Table 5.3) have an embedded error estimator, which is used within the algorithm to accept or reject the solution after each integration step. At the same time, the embedded error estimator is also used to predict the size of the next integration step. The criterion used to accept or reject an integration step is based on the comparison between the estimated error and an error tolerance provided by the user. The first five formulas Table 5.3 have already been described in previous Section 5.1. For the rest of integration formulas considered in these tests, a brief description follows below:

ROS4 [Hairer 1991] is a SDIRK *L*-stable Rosenbrock formula implemented with 4 stages and four function evaluations per step. The formula is a 4(3) pair[*].

RODAS4, a Rosenbrock formula from [Hairer 1991] is based on a stiffly accurate pair 4(3), where both formulas are *L*-stable. It is however more computationally expensive than its predecessors, since it requires six function evaluations and six backsolves (solution of linear systems) per integration step.

LSODE, the Livermore Solver written by Hindmarsh [Hindmarsh 1983, Radhakrishnan 1993], is a Backward Differentiation Formulas (BDF) belonging to the family

[*] The pair notation 4(3) indicates that the integrator computes the solution with an order 4 formula while it uses a solution approximation of order 3 to calculate the local error.

of Linear Multi-step Formulas. Properties and formulation of BDF were already discussed in Chapter 4 and in Section 2.2. The LSODE formula uses two different methods, a BDF formula for stiff problems and an Adams-Moulton formula for non-stiff problems. Both integration formulas belong to the family of linear multi-step formulas. LSODE implements these methods in the way that the method order can vary (from 1 to 12 for the Adams formula and from 1 to 5 for the BDF) during the integration.

Table 5.3. Offline integration algorithms (FE = function evaluations)

Method family	Order of accuracy	Method	Properties
RK Fully-Implicit	Order 5	**RADAU5**	Used for obtaining the approximate exact solution
RK-Explicit	Order 3	**ODE23**	3 FE
	Order 5	**DOPRI5**	6 FE
ROSENBROCK	Order 2	**ODE23s**	2 FE; 2 Stages
	Order 3	**RODAS3**	3 FE; 4 Stages
	Order 4	**ROS4**	4 FE; 4 Stages
		RODAS4	6 FE; 6 Stages
Multi-Step (BDF)	Variable	**LSODE**	-

5.2.1 Numerical tests

Test 1

This test consists on the numerical integration of the Circuit #1 in Figure 5.1. The sizes of volumes are V_1 = 0.01 l and V_2 = 10 l, and the diameters of both orifices are set to 4 mm. The volumetric flow entering to V_1 changes from 60 to 30 l/min and then from 30 to 60 l/min following a step function, introducing therefore a discontinuous signal in the system. The shape of the volumetric flow function entering V_1 is plotted in the left-hand side of Figure 5.11. The right-hand side of this figure shows the approximate exact solution (obtained with the RADAU5 integration formula) of the volume pressures p_1 and p_2.

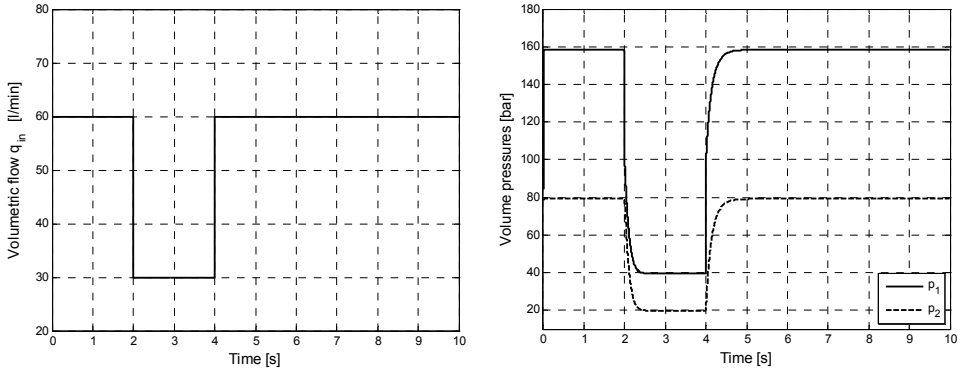


Figure 5.11. Test 1: Incoming volumetric flow (left-hand side) and approximate exact solution (right-hand side)

Number of integration steps and accuracy of the solution is plotted in Figure 5.12 for each of the numerical formulas and for different error tolerances. In the figure, each pair (accuracy – number of steps) is represented by a dot. The test problem is integrated four times for each numerical formula, using in each integration a different relative error tolerance: 10^{-1}, 10^{-2}, 10^{-3} and 10^{-4}. However, the plot shows only three integration results (dots) for the LSODE formula. This means that LSODE could not succeed in the numerical integration of the test problem for one of the given error tolerances. In particular, LSODE failed to complete the integration when an error tolerance of 10^{-2} was required.

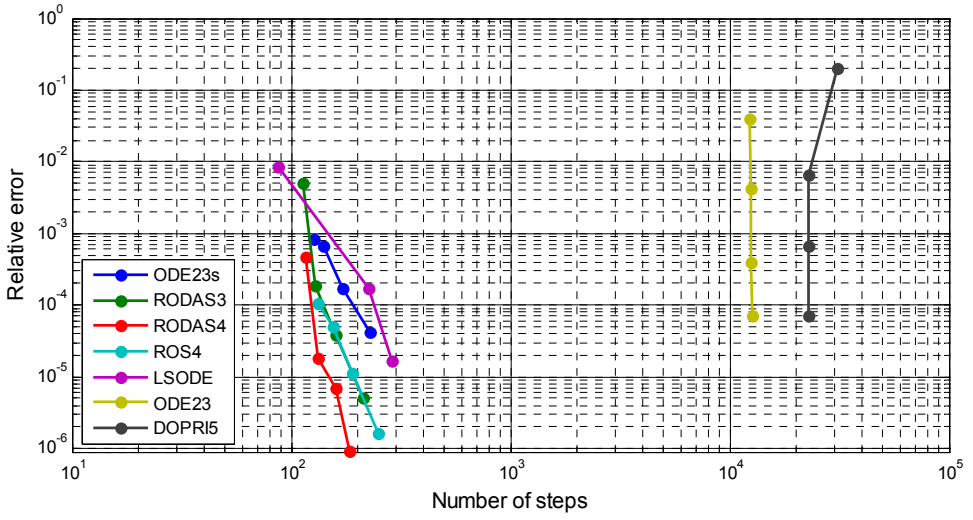


Figure 5.12. Test 1: Performance of the numerical integration in terms of relative error and number of integration steps

Analysis of Test 1:

- From the performance plot in Figure 5.12, it can be seen that Rosenbrock methods outperform the LSODE formula not only in terms of accuracy but also in that they required fewer integration steps.
- A significant behaviour of the explicit methods, related to the difficulty they encounter when integrating stiff systems, is observed in Figure 5.12: no matter the error tolerance imposed to the integration, ODE23 and DOPRI5 formulas always employ a similar amount of integration steps, when the expected behaviour is to use more number of steps as more accuracy is demanded. The explanation of this behaviour can be found in the integration step size predictor embedded in the explicit formula. Normally, step size predictors will adjust the integration step in order to satisfy that the solution accuracy is kept below a given error tolerance. However, when integrating stiff systems, the step size predictor will have to adjust the integration step in order to keep numerical oscillations and other instabilities under control. This normally requires the use of many smaller integrations steps than when controlling simply the accuracy of the solution.
- The use of relatively large error tolerances for integrating ODEs with discontinuities might lead to numerical instabilities, localized in the vicinity of those discontinuity points. This occurrence has been observed in this numerical test, particularly when using a relative error tolerance of 0.1. Upper plots of Figure 5.13 show some clear deviations of the numerical solution, provided by ODE23s and LSODE, with respect to the exact solution. ODE23s shows a clear deviation of a single solution point at t=2 s, which can easily be identified as a numerical error. However, the LSODE solution displays a larger error region around t=2 s with a maximum relative error of 14% at t=2.1 s and a mean relative error of 5% in the time range t = [2, 3] s.
- The relatively large stiffness of this system is expected to introduce numerical oscillations in the solution when explicit numerical formulas are employed to integrate the system. These numerical oscillations are especially visible when large integration step sizes or large tolerances are employed in the integration. The lower plots of Figure 5.13 show this phenomena occurring when ODE23 and DOPRI5 explicit formulas are used with a given relative error tolerance of 0.1. It has been observed that, by employing smaller error tolerances, these numerical oscillations do not show up anymore in the

DOPRI5 formula. Nonetheless, when smaller error tolerances are applied to ODE23, the oscillations do not totally damp out but instead their amplitude is reduced.

- For that particular test, it was found that RODAS3, RODAS4 and ROS4 were able to perform the numerical integration for all given error tolerances without any type of numerical instability or significant deviation from the exact solution.

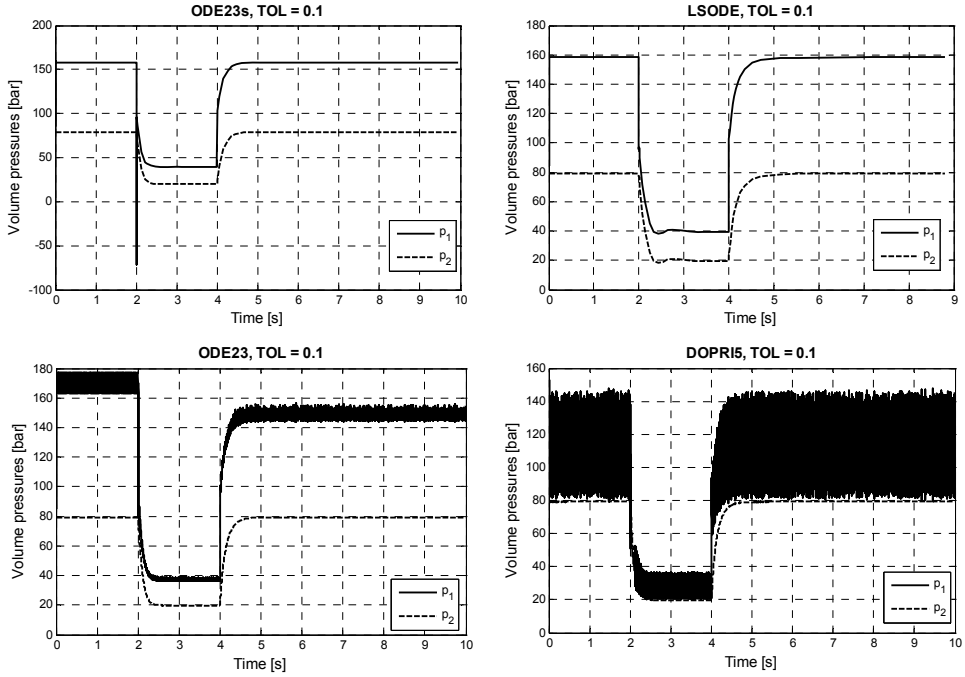


Figure 5.13. Test 1: Effect of discontinuities and numerical stiffness on the solution given by some numerical integrators.

Test 2

In this numerical test, the same Test Circuit #1 is employed but the incoming volumetric flow to volume V_1 is now a sinusoidal signal characterized by a frequency of 4 Hz and an amplitude of 20 l/min. This flow profile is shown in the left-hand side plot of Figure 5.14. The size of the volumes are V_1 = 0.02 l and V_2 = 10 l, and the size of the orifices are d_1 = 6 mm and d_2 = 4 mm.

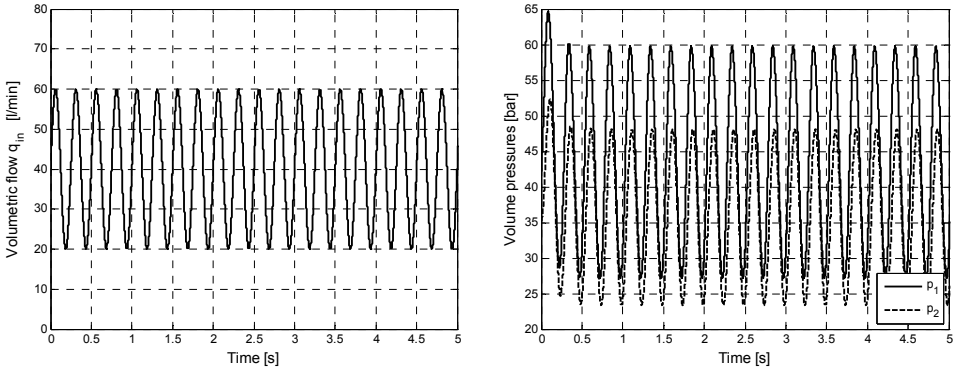


Figure 5.14. Test 2: Incoming volumetric flow (left-hand side) and approximate exact solution (right-hand side)

The performance (accuracy – number of steps) of the numerical integration formulas for this test is shown in Figure 5.15. For these tests the same set of given error tolerances, as in the previous test, are used. As it can be observed in the plot, the numerical formula ODE23s only succeed in three of the four numerical integrations. This failure occurred when a relative error tolerance of 10^{-1} was imposed in the numerical integrator.

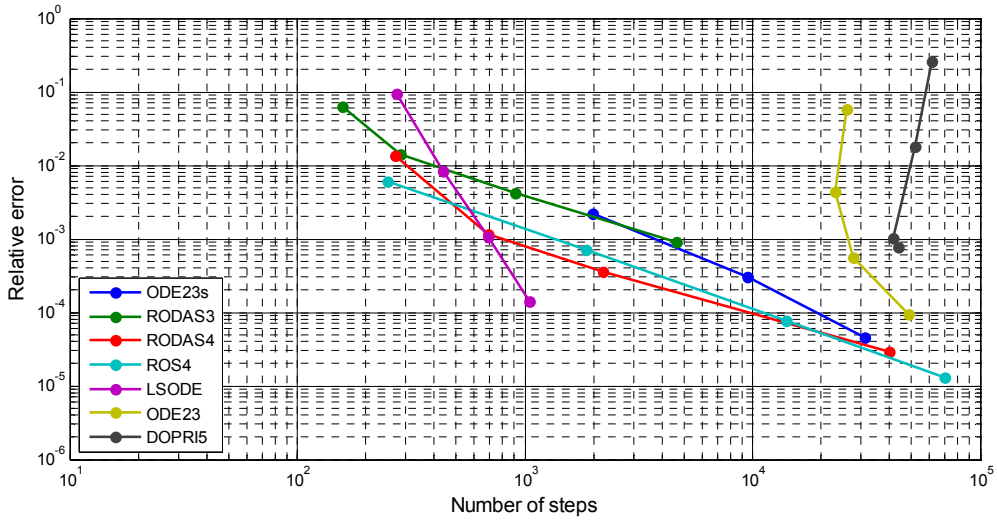


Figure 5.15. Test 2: Performance of the numerical integration in terms of relative error and number of integration steps

Analysis of Test 2:

- Practically all numerical integration algorithms succeed to complete the numerical test. The only exception was the Rosenbrock formula ODE23s, which failed to integrate the test problem for a tolerance of 10^{-1}.
- All Rosenbrock methods show a similar pattern in the performance plot of Figure 5.15. They all show the same ratio (slope) between relative error and number of steps. Performance curve of LSODE shows that this formula is capable of providing higher accuracy solutions employing less integration steps than Rosenbrock formulas. On the other hand, for lower accuracy demands (tolerances of 10^{-2} and larger), some of the Rosenbrock formulas appear to be more computationally efficient than LSODE.
- As it occurred in the previous numerical test, LSODE formulas suffer again at low accuracy tolerances. Figure 5.16, in its upper plot, shows the numerical solution provided by the LSODE integrator for a relative error tolerance of 0.1. When compared to the exact solution (plotted in Figure 5.14), it can be seen that the sinusoidal shape of solution p_1 does not show constant amplitude. Furthermore, exceptionally high peaks arise randomly. This problem still persists when integrating the numerical test with LSODE using a tolerance of 10^{-2}: although the accuracy of the solution improves substantially, random pressure peaks (up to 180 bar) are still visible in the solution. Rosenbrock formulas have not shown these problems when solving the test problem with low accuracy demands.
- Explicit formulas ODE23 and DOPRI5, as expected, suffer again due to the stiffness of the system. The lower plots of Figure 5.16 display the numerical oscillations of the solution p_1. These numerical oscillations still persist for tolerances of 10^{-2} and finally disappear for tolerances smaller or equal than 10^{-3}.

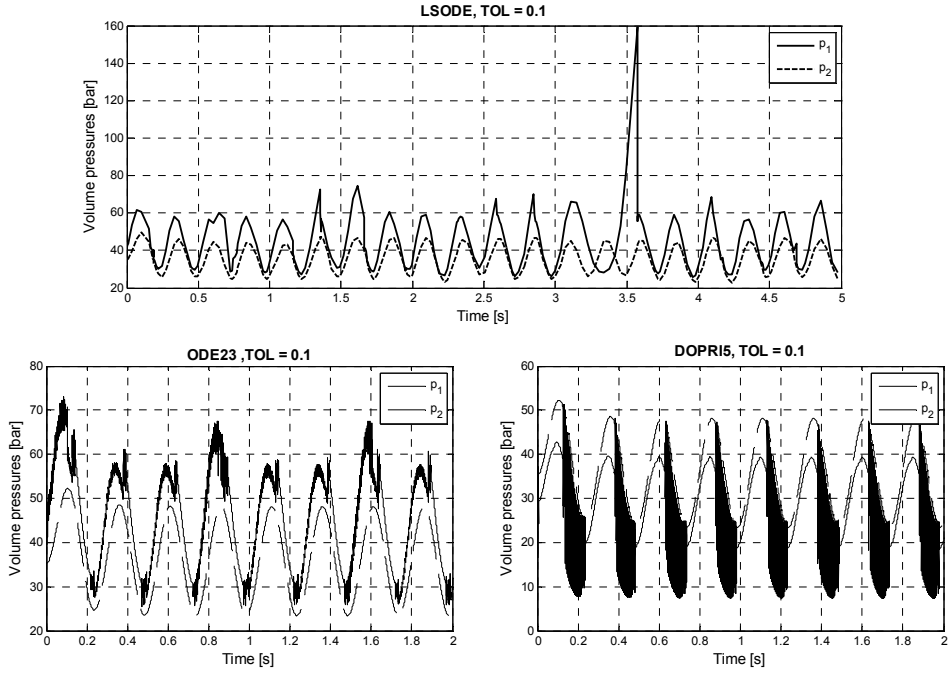


Figure 5.16. Test 2: Numerical integration difficulties found in some of the algorithms

Test 3

In this test problem, Circuit #2 described in Figure 5.2 is employed. As stated previously, this circuit differs from the previous one in that a) its size is relatively much larger (dimension 13 versus dimension 2) and b) new non-linearities and discontinuities are introduced (pressure relief valve, control proportional valve, mechanical seal friction). The main dimensions and component characteristics of the fluid power circuit were presented in Table 5.2. In Figure 5.7 the approximate exact solution of a selected number of variables is plotted.

Due to the different nature of the state variables to be integrated (position, velocity, flow, pressure, forces…), special attention is required when imposing error tolerances to these variables. Different absolute and relative error tolerances have been employed for different variables, depending on their scaling and the numerical magnitude of their solution. As in the previous tests, four different levels of required accuracy are used. In order to keep an analogy with the previous tests, the four levels of error tolerances will be named 10^{-1}, 10^{-2}, 10^{-3} and 10^{-4}, from the least to the most accurate tolerance.

The plot in Figure 5.17 shows the maximum relative error of the cylinder piston position for the different numerical formulas and for the four different levels of required accuracy. As it can be observed, many numerical formulas have failed to succeed in the numerical integration of the test problem: ODE23s and ROS4 failed for error tolerance $tol = 10^{-1}$, LSODE failed twice for $tol = 10^{-1}$ and $tol = 10^{-2}$. Performance levels of DOPRI5 and ODE23 for $tol = 10^{-3}$ and $tol = 10^{-4}$ have been omitted in the plot due to the high number steps they required (more than 300 000).

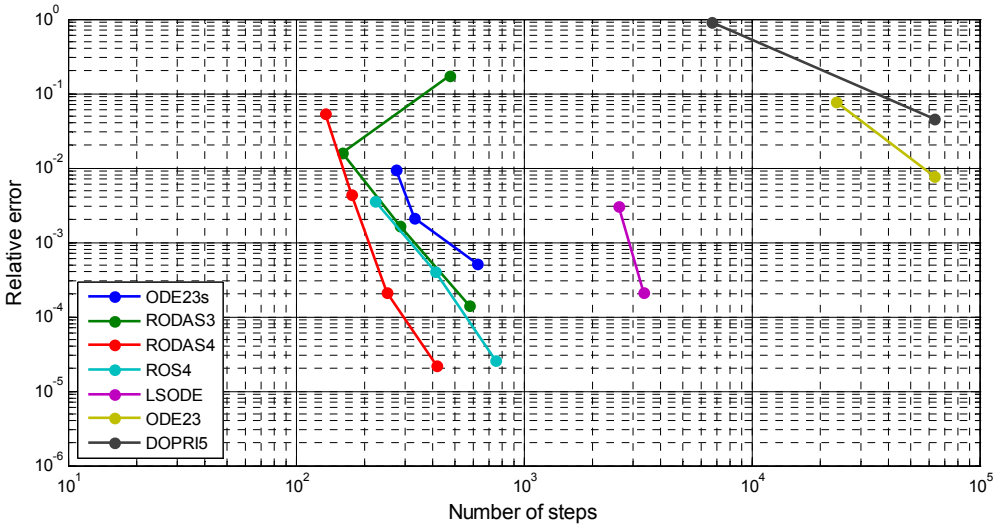


Figure 5.17. Test 3: Performance of the numerical integration in terms of relative error and number of integration steps

Analysis of Test 3:

- Figure 5.17 clearly shows that performance pairs (relative error – number of steps) of Rosenbrock formulas indicate that these have a clear advantage over LSODE and explicit formulas, both in terms of computational efficiency and stability.
- As already stated in previous chapters, multi-step methods (LSODE) are prone to have difficulties when dealing with discontinuities. This is the case of the LSODE formula, which can complete successfully the numerical integration only in two of the four error tolerances, and using approximately ten times more integration steps than some of the Rosenbrock formulas.
- Among the Rosenbrock formulas, higher order formulas (ROS4 and RODAS4) seem to be more efficient for high accuracy solutions (small error tolerances).

- RODAS3 with *tol* = 0.1 suffer from numerical oscillations after simulation time *t* = 4 s. As a consequence the relative error rises up to 15% and error predictor formula tries to correct this behaviour by reducing the integration step size. This is reflected in the plot, where RODAS3 (with *tol* = 0.1) uses more integration steps than expected.
- Numerical solutions provided by explicit methods show, as in previous numerical tests, the typical numerical oscillations which result from integrating stiff systems. This leads (as shown in Figure 5.17) to lower solution accuracies and much larger number of integration steps. Figure 5.18 plots the numerical solution of the piston position (left-hand side plot) and cylinder chamber pressures (right-hand side plot) given by ODE23 when using a relative error tolerance of 10^{-1}.

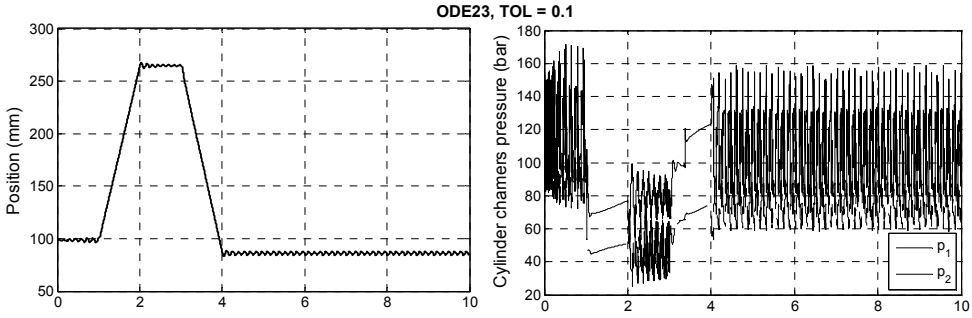


Figure 5.18. Test3: Numerical oscillations in the solution. Solver ODE23. Tol = 0.1

Test 4 (generic numerical test problems from literature)

In this section the numerical formulas are being tested against two additional numerical tests problems. These numerical tests are commonly used in the literature, among many others, to evaluate the efficiency of generic first order differential equations.

Van der Pol's equation:

The solution of Van der Pol's equation (5.2) is a periodic non-linear oscillation where small oscillations are amplified (unstable) and large oscillations are damped. The rate at which the damping factor changes is defined by the constant μ .

$$\ddot{y} + \mu\left(y^2 - 1\right)\dot{y} + y = 0, \qquad \mu > 0. \tag{5.2}$$

For this numerical test, μ has been chosen to be $\mu = 10^6$. This value ensures very stiff conditions and extremely fast transients in the solution *y*, as it can be seen in the left-

hand side plot of Figure 5.19. The performance of the numerical formulas, when integrating the problem (5.2), is shown in the right-hand side of Figure 5.19. The vertical axis quantifies the mean value of relative error of solution y from t = 0..12 s. Each numerical integration formula has computed the solution using four different orders of accuracy, which are defined by using both relative error (RTOL) and absolute error (ATOL) tolerances as

$$TOL = ATOL + RTOL \cdot |y_n|, \tag{5.3}$$

where $RTOL = 10^{-n}$ and $ATOL = RTOL$, with n = 1, 2, 3, 4. y_n is the numerical solution computed at the previous step.

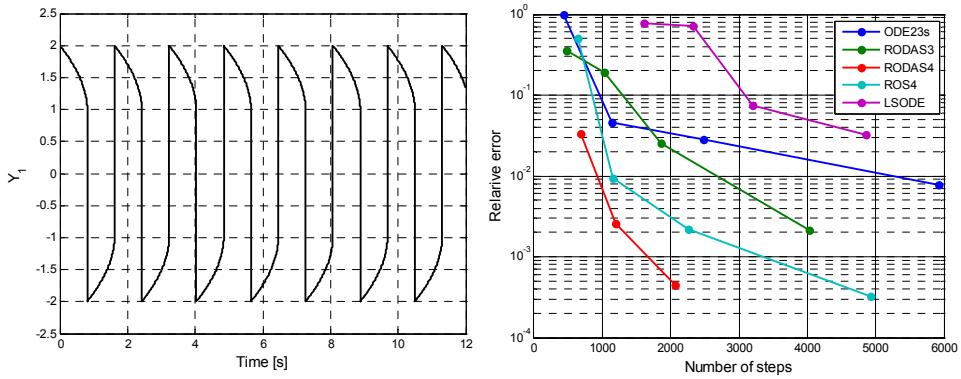


Figure 5.19. Solution of Van der Pol's equation (left-hand side). Performance of the numerical integrators solving Van der Pol's equation (right-hand side).

Analysis:

- Explicit formulas ODE23 and DOPRI5 were not able to integrate such stiff second order system for any of the requested accuracy tolerances. Reducing the constant μ (and therefore the stiffness) to a value of $\mu = 10^4$ made the explicit formulas to succeed.
- Among Rosenbrock and multi-step formulas, they all completed the numerical integration except RODAS4, when a level of accuracy of 10^{-1} was required.
- High order Rosenbrock formulas (RODAS4 and ROS4) provide better accuracy than the lower order ones.
- Van der Pol's equation causes difficulties to the LSODE formula, which cannot provide enough accurate results even though it employs smaller integration steps than the Rosenbrock family methods. Error plots for LSODE and ROS4 solutions are shown in Figure 5.20: when imposing a relative accuracy of 10^{-2}, the numerical integration per-

formed by LSODE shows that its accuracy surrounding the fast transients is worsened as the integration advances. However, right-hand side plot of the figure shows that the accuracy of the numerical solution provided by ROS4 is just affected in the very near surrounding of the transient points.

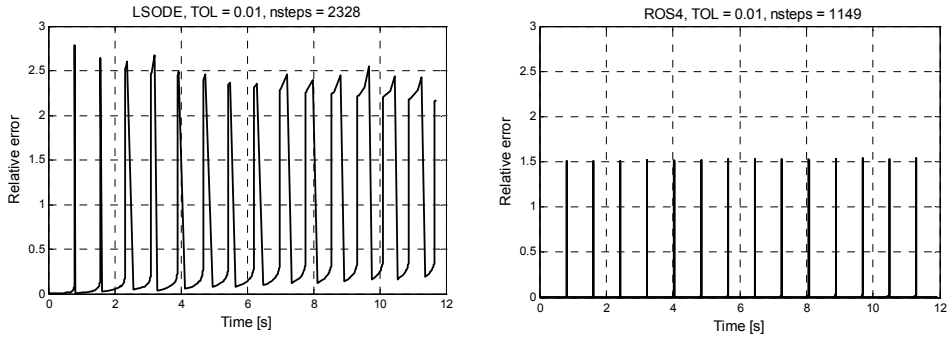


Figure 5.20. Accuracy of LSODE and ROS4 formulas in the integration of Van der Pol's equation with *tol* = 0.01

Hires' equation:

This is a stiff system of 8 non-linear ordinary differential equations proposed by Schäfer [Schäfer 1975]. The equation describes high irradiance responses of photo-morphogenesis by means of chemical reaction involving eight reactants. The system is formulated in (5.4) and its numerical solution is shown in the left-hand side plot of Figure 5.21.

$$\dot{y} = f(y), \qquad y(0) = y_0, \tag{5.4}$$

with

$$f(y) = \begin{pmatrix} -1.71y_1 & +0.43y_2 & +8.32y_3 & +0.0007 & & \\ 1.71y_1 & -8.75y_2 & & & & \\ -10.03y_3 & +0.43y_4 & +0.035y_5 & & & \\ 8.32y_2 & +1.71y_3 & -1.12y_4 & & & \\ -1.745y_5 & +0.43y_6 & +0.43y_7 & & & \\ -280y_6y_8 & +0.69y_4 & +1.71y_5 & -0.43y_6 & +0.69y_7 & \\ 280y_6y_8 & -1.81y_7 & & & & \\ -280y_6y_8 & +1.81y_7 & & & & \end{pmatrix},$$

$$y_0 = (1,0,0,0,0,0,0,0.0057)^T.$$

The performance of the numerical formulas when integrating the problem (5.4) is shown in the right-hand side of Figure 5.21. The vertical axis indicates the maximum relative error found in the integration interval t = 0...370 s. Each numerical integration formula has computed the solution successfully using four different orders of accuracy: RTOL = 10^{-n} , ATOL = 10^{-4}RTOL , with n = 1, 2, 3, 4.

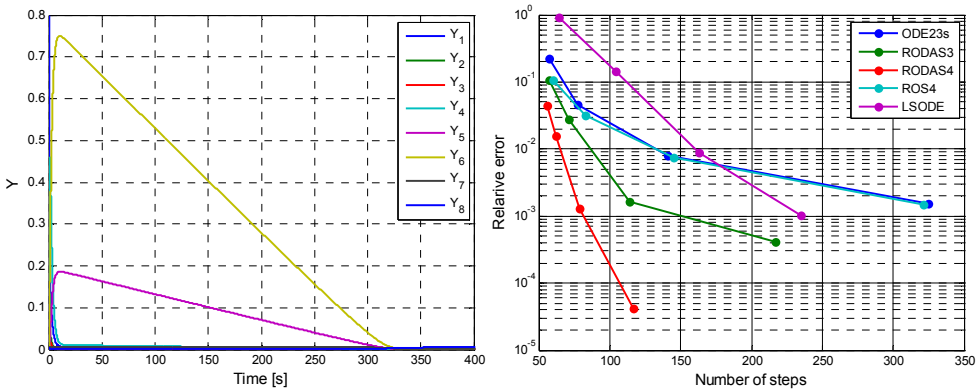


Figure 5.21. Solution of Hairer's equation (left-hand side). Performance of numerical integrators solving Hairer's equations (right-hand side)

Analysis:

- Due to the lower stiffness of Hirer's equations, when compared to the previous tests, all explicit and implicit formulas succeed in performing the numerical integration in the interval t = 0...370 s. However, performance results of ODE23 and DOPRI5 are not shown in Figure 5.21 due to the high number of integration steps employed by this formulas (approx. 5000 and 10000 steps respectively)
- Among Rosenbrock formulas, RODAS4 and RODAS3 show better performance levels than the rest of formulas, especially in the higher accuracy solutions.
- LSODE shows a poor performance for the two lowest levels of accuracy. This can be clearly seen in the right-hand side plot of Figure 5.21, where the relative error of the LSODE solution is considerably larger than the other solution errors provided by Rosenbrock formulas.

5.2.2 Conclusions of offline integration tests

The previous tests have shown that popular explicit RK formulas are clearly not the best option for the integration of stiff systems or systems with discontinuities. Such formulas, with reduced stability properties, need in some cases of extremely small integration step sizes in order to keep the integration under stable conditions.

All Rosenbrock formulas have shown similar behaviour in their performance during the tests. Concerning the multi-step LSODE code, the problems announced previously in Chapter 4 regarding multi-step BDF formulas have been confirmed, such as the lack of accuracy near discontinuities (Test 1 and Test 4) and stability problems for low orders of accuracy (Test 2 and Test 3). It has been found that, in general, Rosenbrock formulas have provided substantially better results (accuracy, efficiency and stability) in all tests performed.

5.3 Analytical and numerical Jacobians

As previously stated, the numerical integration of numerically stiff systems of ODEs is better accomplished using implicit integration methods. One of the main differences between implicit and explicit methods is that the former solver requires the Jacobian matrix of the ODE system. The need of a Jacobian evaluation at each integration step (or at sampled intervals) raises the following concerns: how the Jacobian matrix is formed? And what are the associated computational costs?

Generally, two different approaches are used to form the Jacobian matrix. Jacobian evaluations can either be done numerically (e.g. by finite differences) or symbolically, through the analytical expression of the Jacobian. The use of the analytical expression of the Jacobian has clear advantages [Esqué 2005]:

- It evaluates the Jacobian more accurately, leading to better solution accuracies and better numerical stability.
- Evaluation of Jacobian, from its analytical form, is less computationally expensive than using numerical techniques. This leads to significantly better simulation performance.

Among the implicit ODE solvers, the Rosenbrock formulas are especially sensitive to the accuracy of the Jacobian. This is due to the fact that Rosenbrock formulas only use a

single Newton iteration to solve the non-linear set of equations. Although the numerical tests presented in this chapter have made use of the analytical Jacobian approach, the same tests have been repeated employing a Jacobian evaluation by numerical differentiation (5.5) instead. The tests have shown that Rosenbrock formulas, using a numerically-obtained Jacobian, are still able to integrate successfully most of the tests. Nonetheless, the accuracy of these solutions is in general worsened. In some other cases, the results provided by Rosenbrock integration with a numerically-obtained Jacobian were unacceptable. This can be seen in Figure 5.22, where numerical integration of the test circuit in Figure 5.2 is performed with RODAS3 using the analytical and the numerical approaches for the Jacobian evaluation. Results are compared to the approximate exact solution provided by the code RADAU5. The results show that the solver employing the analytical Jacobian provides a solution closely matching the exact solution. However, when the same solver evaluates the Jacobian numerically, the accuracy of the solution obtained is unacceptable

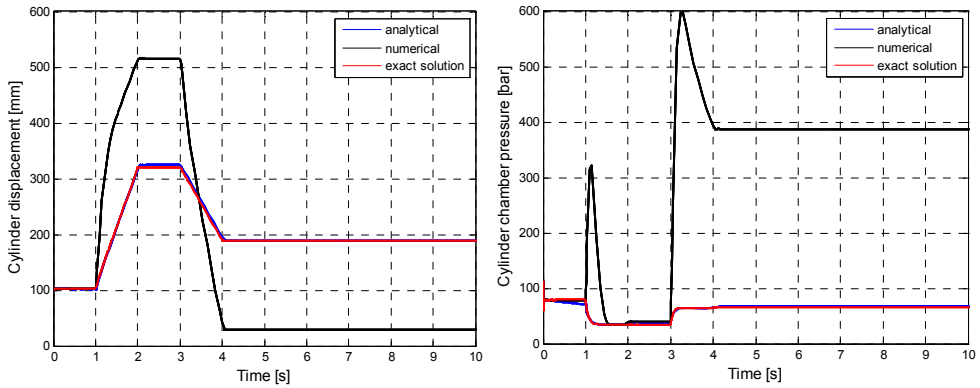


Figure 5.22. Comparison of numerical integrations of Test Circuit #2 when employing numerical and analytical Jacobians

The computational cost (in terms of CPU time) involved in the formation of the Jacobian matrix is yet another concern. In general, an evaluation of the Jacobian is needed in every integration step of implicit Rosenbrock methods. Among all operations required to advance one step with a Rosenbrock formula, only the computation of the Jacobian matrix can take up to 40% of the total CPU time [Esqué 2005]. The rest of computation (60 % of CPU time) is dedicated to function evaluations, solution of linear systems and other minor operations.

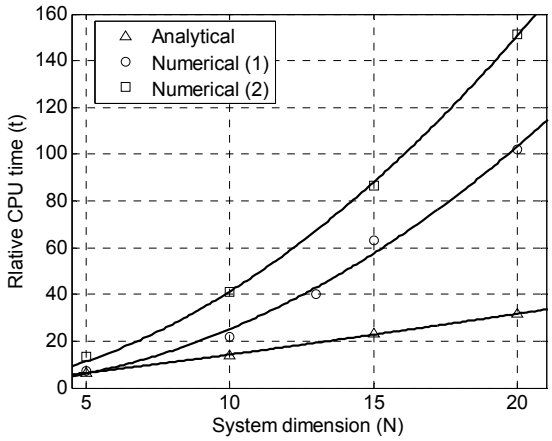
In terms of computational costs, the use of an analytical expression of the Jacobian matrix shows, once again, advantages over the use of numerically-obtained Jacobians. The following test confirms the above. In this test, numerical integration of fluid power circuits of dimensions N = 5, 10, 15, 20 are performed using different methods to evaluate the Jacobian.:

- Analytical Jacobian: The Jacobian matrix is determined symbolically before the integration starts. The Jacobian is then evaluated as a function of the state variables of the system [Section 4.4].
- Numerical Jacobian (1): Instantaneous value of the Jacobian matrix J of the system $\dot{y} = F(y)$ is computed numerically by finite differences as:

$$J_{ij} = \left(F_i\left(y_j + \Delta\right) - F_i\left(y_j\right)\right) / \Delta \tag{5.5}$$

- Numerical Jacobian (2): The subroutine NUMJAC, from the MATLAB ODE suite is employed to evaluate the Jacobian numerically. NUMJAC is an implementation of a robust scheme due to Salane [Salane 1986] for the approximation of partial derivatives.

Figure 5.23 plots the pairs (CPU time (t) – system dimension (N)) obtained after conducting all the integrations. The purpose of this plot is to show the computational time of the RODAS3 Rosenbrock formula as a function of the system dimension and for the different Jacobian evaluation techniques described above. From the analysis presented in Chapter 4, it is known that the time required to integrate a system of ODEs of order N with an implicit formula is aN^{α}, where a is a constant, and the power term α is a function of the algebraic formulation of the numerical formula (including the Jacobian computation). As shown in Figure 5.23, the power curve $t = aN^{\alpha} + b$ fits reasonably well (with R-square > 0.99) to all three series of integrations. Computational costs associated to the different Jacobian evaluation techniques are given by the exponential term α. The lowest value of α is clearly obtained when the solver uses the analytical form of the Jacobian.



$$t = aN^{\alpha} + b$$

Power terms α of the curve fitting:

Analytical.: $\alpha = 1.15$

Numerical (1): $\alpha = 2.04$

Numerical (2): $\alpha = 2.00$

Figure 5.23. Determining the costs in evaluating the Jacobian, as a function of the system dimension *N*.

Because of its better accuracy, stability and computational performance, Jacobian evaluation from its analytical expression should be performed, if possible. However there are situations where symbolic information of the dynamics of the system is not completely available. This is often the case when computer software packages are used to construct the model and to perform the simulation employing the software's own numerical integration solvers. Chapter 3 of this thesis has presented a model topology and a systematic formulation of fluid power elements and systems from which the analytical form of the Jacobian matrix can be automatically derived by an algorithm, as a pre-process prior the starting of the integration, and without the need of any manual symbolic manipulation.

6 CONCLUSIONS

Numerical problems are commonly present in the numerical simulation of fluid power circuits. Such numerical integration difficulties are due to the nature and characteristics of the physics governing fluid power systems, such as a) the highly non-linear behaviour of certain physical phenomenon (e.g. fluid compressibility, turbulent flow) or components (e.g. cylinder seal friction forces), b) numerically stiffness due to big differences in response times of different variables, and c) discontinuities due to digital control signals and due to limited displacement of actuators. All these characteristics can certainly cause stability and accuracy problems to the numerical integration algorithms.

Simulation applications (such as virtual prototyping, hardware-in-the-loop, man-in-the-loop simulators, offline computer-based simulations) are extendedly used in research and in industrial fields. Fluid power engineers posses a deep understanding of the physics and dynamics linked to fluid power systems. This allows them to mathematically formulate the problems and construct simulation models. However, engineers might not be enough acquainted with the theory behind the numerical integration of differential equations and, in many other cases they might even lack the criteria to choose a proper numerical integration formula based on the characteristics of the simulation models to be solved. As a consequence, general-purpose simulation software is often employed in order to numerically integrate the generated differential equations. A drawback in this practice is that these software packages often offer a limited number of numerical integration solvers to choose from. The choice of a wrong numerical solver usually leads the engineer to take some corrective actions which, in many cases, degrades the simulation performance. Some of these actions are: a) reducing the size of the integration step size in order to correct stability problems or to gain more accuracy in the solution. Nonetheless, this action also leads to an important increase of computational time, which can be critical in a real-time application. b) When real-time simulations are not achievable because of the heavy computational times,

simulation models are often simplified (compromising the level of accuracy) or the integration step size is increased (now compromising both the solution accuracy and the numerical stability of the method). Poor numerical stability properties of the solver can cause numerical oscillations in the solution of a stiff system. These oscillations sometimes are wrongly interpreted as a physical behaviour.

From the above exposed, it can be concluded that a general knowledge on the properties of numerical integration methods is essential for the choice of an appropriate numerical integrator. The choice will be made based on the characteristics and the estimated behaviour of the simulation model. In this thesis, a series of *L*-stable Rosenbrock formulas, derived from the semi-implicit Runge Kutta family, have been proposed for the numerical integration of fluid power circuits. The implementation of these Rosenbrock formulas in a programming language is simple and straightforward when compared to general implicit or multi-steps methods. Rosenbrock formulas are implicit single-step formulas which do not require the solution a non-linear algebraic system. Instead, the stages are solved consecutively as unknowns of a linear system. In addition, no special modifications of the code are required in order to adapt the formula to systems containing discontinuities. Discontinuities can become an issue in the more complex integration methods like the Linear Multi-step Formulas.

Numerical simulations have been conducted in order to evaluate the proposed Rosenbrock formulas and to compare their performance to other popular codes. Most of the numerical tests have been built employing systems of ODEs originating in fluid power circuit simulation models due to the special characteristics found in fluid power systems. Other test problems popularly used for evaluating generic numerical integrators of ODEs have also been employed.

Concerning the real-time simulations, the following conclusion can be extracted:

- In the real-time simulation of systems with fast dynamics, where a relatively small sampling time (i.e. integration step size) is required, explicit formulas are commonly employed due to their computational simplicity (no Jacobian evaluations, no need of solving linear or non-linear algebraic systems). Nonetheless, explicit formulas show very clear limitations when dealing with stiff systems. These limitations have already been observed in this thesis when comparing the solution accuracy provided by ex-

plicit and implicit Rosenbrock formulas. Implicit Rosenbrock formulas have provided solutions with much better levels (in most cases several orders of magnitude) of accuracy than explicit formulas of the same order and for the same integration step size.

- Another limitation found in explicit formulas is due to their poor numerical stability. This fact has also been clearly observed in the performance tests carried out in Chapter 5. Numerical stability problems might be solved or partially solved by reducing the step size of the explicit integration, although the computational costs of the integration are then increased by the same ratio.

- The stability shown by the tested Rosenbrock formulas is certainly superior: Rosenbrock formulas have remained still stable even when using integration step sizes up to ten times larger than the largest possible step size used in explicit formulas. Rosenbrock formulas have also provided much better accuracy. In most cases the solution accuracy provided by Rosenbrock integration with time step h has been better than the one provided by explicit methods using h/10.

- The outstanding stability and accuracy properties or Rosenbrock formulas can make them faster than explicit formulas. As computational costs of explicit formulas grow linearly with the dimension N of the system, computational costs associated to implicit formulas can be as high as of the order of $O\left(N^3\right)$. Nonetheless, and as it has been shown in the numerical tests, the proposed Rosenbrock formulas have shown computational costs of the order $O\left(N^{1.15}\right)$. These reduced computational costs are also due to the fact that an analytical form of the Jacobian matrix has been used to obtain its numerical evaluation at each integration step. The proposed Rosenbrock formulas are, among all implicit formulas, one of the less computationally demanding, in terms of the number of operations required to advance one step the integration.

Conclusions with respect to offline simulations are exposed next:

- In offline simulations, integration formulas normally make use of an adaptive integration step size according to an estimation of the local integration error. By means of controlling the integration error, the formulas are also implicitly detecting numerical instabilities and therefore they can reduce these instabilities by reducing the size of the integration step. However, instabilities cannot be always avoided, especially if the

numerical formula employed is not suited for the particular characteristics and behaviour of system being solved. For example, numerical formulas which are not good at detecting discontinuities might turn unstable despite being able to control the local error. Another case is found in the integration of numerically stiff system by those formulas which do not have *L*-stability properties. Under these conditions, high frequency oscillations are seen in the numerical solution. The embedded local error estimator in the formula can partially or totally damp these oscillations if smaller error tolerances are used. However this leads to an inefficient way of solving the numerical problem.

- Numerical integrators for the offline simulation of fluid power circuits need to have excellent stability properties and also need to perform efficiently. The efficiency is measured as the rate between the accuracy of the solution and the number of integration steps. Bad stability properties of the integration formula not only can lead to simulation *crashes* (i.e. error of the solution grows unbounded, causing a computational overflow) but they can also have an important effect on the overall efficiency of the integration. Efficiency is degraded when the error predictor of the formula has to reduce the integration step size in order to mitigate numerical instabilities arisen during the integration. Formulas perform efficiently when a change in the integration step size is solely targeted to control the error of the numerical solution.

- The proposed family of Rosenbrock formulas have proven, in general, to hold excellent stability properties throughout all the numerical tests performed in this research. These numerical tests have been performed on fluid power circuits showing discontinuities, highly non-linear behaviour, and numerical stiffness.

- As it could be expected, the efficiency shown by explicit Runge-Kutta formulas during the offline simulation tests is considerably degraded due to their limited stability under numerically stiff conditions. As a consequence, explicit formulas have required approximately between 100 and 1000 times more integration steps than the implicit formulas. Even employing such small step sizes, numerical oscillations have still been visible in many of the provided solutions. Although showing a limited stability, explicit Runge-Kutta formulas have been able to complete all numerical tests, proving that these formulas can detect and handle discontinuities and non-linearities. This can

explain why explicit Runge-Kutta formulas are still popular and are commonly used as a first choice for solving any type of ordinary differential equation system.

- As a representative of the multi-step Backward Differentiation Formulas, the code LSODE has also proven to be very efficient and stable during those numerical integration tests containing no discontinuities. Nonetheless, when discontinuities were introduced, some of the tests could not be successfully solved due to stability crashes. LSODE also suffered from poor solution accuracy when dealing with highly non-linear systems.

- Semi-implicit *L*-stable Rosenbrock formulas, in special those of order 3 and 4, have proved their polyvalence throughout all the numerical tests. Their excellent stability properties, not only allowed them to complete successfully all of the integration tests, but their stability also awarded them with a very good efficiency rate. These good results support therefore the theoretical analysis of Rosenbrock formulas carried out in Chapter 4.

- The main drawback of Rosenbrock formulas is that an accurate Jacobian matrix needs to be formed at each numerical integration step, while other implicit formulas may just require a crude approximation or even might only need a Jacobian evaluation after certain interval of steps. The supply of a Jacobian at each integration step is sometimes necessary in order to guarantee the numerical stability of the formula. This is caused by the fact that Rosenbrock formulas only make use of a single Newton iteration for solving the non-linear systems at each stage. The need for an accurate Jacobian evaluation at each integration step can imply important additional computational costs.

A systematic approach for constructing an analytical Jacobian matrix of the system, prior to the numerical integration, has been presented. This has shown the following advantages:

- The numerical evaluation of a Jacobian matrix from its analytical form requires significantly less computational costs than evaluating the Jacobian using numerical approximations.

- The analytical expression of the Jacobian matrix provides an accurate Jacobian evaluation. The Jacobian accuracy contributes positively to the numerical stability of the numerical integration formula and also the accuracy of the solution.
- From a simulation model built according to the topology introduced in Chapter 3, it is possible to obtain the Jacobian matrix of the system in its analytical form and without the need of any symbolic manipulation by the user. This task can be executed automatically by means of an algorithm, which collects partial Jacobian definitions of each component and assembles them into the full Jacobian matrix.

Finally, a modelling topology for the systematic construction of dynamic simulation models of fluid power circuits has also been presented in this thesis. With this methodology, fluid power elements are interconnected through communication ports, where physical variables are exchanged. Elements and subsystems retain modular and hierarchical properties. During his research, the author has contributed to the development of a fluid power library containing more than 30 simulation models of fluid power elements [Esqué 2003b]. The construction of a fluid power circuit is easily defined by using a formal description of the components and their port interconnections. An algorithm is then used process this formal description and to derive the system of ordinary differential equations and its analytical Jacobian matrix. This constitutes the pre-process prior to the numerical integration of the system.

6.1 Summary of conclusions

- Explicit integration algorithms are typically used in real-time simulation and implicit algorithms in stiff offline simulation. Rosenbrock formulas have shown superior stability, accuracy and efficiency in both cases.
- The special characteristics found in fluid power systems require that numerical integration tests need to be performed in specific fluid power simulation models. This provides the optimum conditions to evaluate the different algorithms.

- Rosenbrock methods are a special class of implicit integration formulas requiring an accurate evaluation of the Jacobian matrix at each integration step.
- A systematic way to provide an accurate numerical evaluation of Jacobian to the Rosenbrock formula employing relatively low computational costs has been presented.

REFERENCES

[Alexander 1977] Alexander, R. (1977), "Diagonally implicit Runge-Kutta methods for stiff O.D.E.'s" SIAM J. Numer. Anal. Vol 14, pp. 1006-1021

[Axelsson 1969] Axelsson, O. (1969), "A Class of A-stable methods", BIT Vol 9, pp. 185-199

[Bogacki 1989] Bogacki, P., Shampine, L.F., (1989), "A 3(2) pair of Runge-Kutta formula", Math. Lett. Vol 2, pp. 1.9

[Boucher 1986] Boucher, R.F., Kitsios, E.E. (1986), "Simulation of Fluid Network Dynamics by Transmission Line Modelling", Proc Instn Mech Engrs Vol 200 No C1, pp. 21-29

[Brenan 1996] Brenan, K.A., Campbell, S.L., Petzold, L.R. (1996), "Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations", SIAM, 1996

[Burton 1994] Burton, J.D., Edge, K.A., Burrows, C.R. (1994), "Modeling Requirements for the Parallel Simulation of Hydraulic Systems", ASME Journal of Dynamic Systems, Measurement, and Control Vol 116, pp. 137-145

[Butcher 1964a] Butcher, J.C. (1964), "Implicit Runge-Kutta Processes", Math Comput Vol 18, pp. 50-64

[Butcher 1964b] Butcher, J.C. (1964), "Integration processes based on Radau quadrature formulas", Math. Comput. Vol 18, pp. 233-244

[Butcher 1976] Butcher, J.C. (1976), "On the implementation of Runge-Kutta implicit methods", BIT Vol 16, pp. 237-240

[Ceschino 1963] Ceschino, F., Kuntzmann, J. (1963), "Problèmes différentiels de conditions initials (methods numériques)", Dunod Paris, 372 p.

[Dahlquist 1963] Dahlquist, G. (1963), "A special stability problem for linear multistep methods", BIT Vol 3 pp. 27-43

[Dorey 1988] Dorey, R. (1988), "Modelling of Losses in Pumps and Motors", Proceedings of First Bath International Fluid Power Workshop, Bath, UK.

[Dormand 1980] Dormand, J.R., Prince, P.J. (1980), "A Family of Embedded Runge-Kutta formulae", J Comp Appl Math Vol 6, pp. 19-26

[Ellman 1996a] Ellman, A., Käppi, T., Vilenius, M. (1996), "Simulation and analysis of hydraulically-driven boom mechanism", Proceedings of Ninth Bath International Fluid Power Workshop, Bath, UK.

[Ellman 1996b] Ellman A., Piché R. (1996), "A modified orifice flow formula for numerical simulation". Fluid Power Systems and Technology, Collected papers of 1996 ASME IMECE, Atlanta, pp. 59-63

[Esqué 2002a] Esqué, S., Ellman, A. (2002), "Pressure Build-Up in Volumes". Bath Workshop on Power Transmission and Motion Control, PTMC 2002, Bath, UK. Published in the book Power Transmission and Motion Control, edited by C.R. Burrows and K.A. Edge, Professional Engineering Publishing Limited. London, UK, pp. 25-38

[Esqué 2002b] Esqué, S., Ellman, A., Piché, R. (2002), "Numerical integration of pressure build-up volumes using an L-stable Rosenbrock method". Proceedings of the 2002 ASME International Mechanical Engineering Congress and Exposition, New Orleans, Louisiana, USA

[Esqué 2003a] Esqué, S., Raneda, A., Ellman, A. (2003), "Techniques for studying a mobile hydraulic crane in virtual reality", International Journal of Fluid Power Vol 4 No 2 pp. 25-34

[Esqué 2003b] Esqué, S. (2003), "WINSIMU, Software for modelling and simulation of fluid power systems", International Journal of Fluid Power Vol 4 No 2 pp. 67-71

[Esqué 2005] Esqué, S., Ellman, A. (2005), "An efficient numerical method for solving the dynamic equations of complex fluid power systems". Bath Workshop on Power Transmission and Motion Control, PTMC 2005, Bath, UK. Published in the book Power Transmission and Motion Control, edited by C.R. Burrows and K.A. Edge, Professional Engineering Publishing Limited. London, UK, pp. 179-191

[Fehlberg 1969] Fehlberg, E. (1969), "Low-order Classical Runge-Kutta Formula with Step Size Control and their Application to Some Heat Transfer Problems", NASA Technical Report 315, extract published in Computing Vol 6, pp. 61-71

[Gear 1971] Gear, C.W. (1971), "Algorithm 407 – DIFSUB for solution of ordinary differential equations", Commun. ACM 14, 3 (Mar.), pp. 185-190

[Gear 1974] Gear, C.W. (1974), "Multirate methods for ordinary differential equations", Tech. Rep. UIUCDCS-74-880, Dept. of Computer Science, University of Illinois

[Gupta 1985] Gupta, G.K., Sacks-Davis, R., Tischer, P.E. (1985) "A Review of Recent Developments in Solving ODEs", ACM Computing Survey Vol 117 No 1, pp. 5-47

[Hairer 1974] Hairer, E., Wanner, G. (1974), "On the Butcher group and general multivalue methods", Comput J Vol 13, pp. 1-15

[Hairer 1991] Hairer, E., Wanner, G. (1996), "Solving ordinary differential equations II: Stiff and Differential-Algebraic Problems", Springer-Verlag

[Hairer 1993] Hairer, E., Nørsett, S.P., Wanner, G. (1993), "Solving ordinary differential equations I: Nonstif problems", Springer-Verlag

[Hairer 1996] Hairer, E., Wanner, G. (1996), "Solving ordinary differential equations II: Stiff and Differential-Algebraic Problems", Springer-Verlag

[Handroos 1990] Handroos, H., Vilenius, M. (1990), "The utilization of experimental data in modelling of hydraulic single stage pressure control valves". ASME journal of dynamic systems, measurements and control, Vol 112 No 3

[Hindmarsh 1983] Hindmarsh, A.C. (1983), "ODEPACK, A Systematized Collection of ODE Solvers", in Scientific Computing, R. S. Stepleman et al. (eds.), North-Holland, Amsterdam, 1983 (vol. 1 of IMACS Transactions on Scientific Computation), pp. 55-64

[Huhtala 1996] Huhtala, K. (1996), "Modelling of Hydrostatic Transmission – Steady State, Linear and Non-Linear Models", PhD Dissertation, Tampere University of Technology, Acta Polytechnica Sacandinavica. 101 p.

[Kamke 1942] Kamke, E. (1942), "Differentialgleichungen, Lösungsmethoden und Lösungen", Becker & Erler, Leipzig, 642p.

[Kaps 1979] Kaps, P., Rentrop, P. (1979), "Generalized Runge-Kutta Methods of Order Four with Stepsize Control for Stiff Ordinary Differential Equations", Numer. Math. 33, pp. 55-68

[Kaps 1981] Kaps, P., Wanner, G. (1981), "A Study of Rosenbrock-Type Methods of High Order", Numer Math Vol 38, pp. 279-298

[Kitsios 1986] Kitsios, E.E., Boucher, R.F. (1986), "Transmission Line Modelling of a Hydraulic Position Control System", Proc Instn Mech Engrs Vol 200 No B4, pp. 229-236

[Krus 1990] Krus, P., Jansson, A., Palmberg, J-O. and Weddfelt, K. (1990), "Distributed Simulation of Hydromechanical Systems", Proceedings of Third Bath International Fluid Power Workshop, Bath, UK.

[Lang 2000] Lang, J., Verwer, J.G. (2000), "ROS3P – An accurate Third-Order Rosenbrock Solver Designed for Parabolic Problems", Report MAS-R0013, CWI-National Research Institute for Mathematics and Computer Science, Stichting Mathematisch Centrum, The Netherlands.

[Larsson 2003] Larsson, J. (2003), "Interoperability in Modeling and Simulation", PhD Dissertation, Linköping University, Sweden. 178 p.

[Layton 1998] Layton, R.A. (1988), "Principles of Analytical System Dynamics", Springer, Mechanical Engineering Series.

[Merritt 1967] Merritt, H.E. (1967), "Hydraulic Control Systems". John Wiley & Sons, Inc. New York. 358 p.

[Nykänen 2000] Nykänen T., Esqué S., Ellman A. (2000), "Comparison of Different Fluid Models", Bath Workshop on Power Transmission and Motion Control, PTMC 2000, Bath, UK. Published in the book Power Transmission and Motion Control, edited by C.R. Burrows and K.A. Edge, Professional Engineering Publishing Limited. London, UK, pp. 101-110.

[Olsson 1996] Olsson, H. (1996), "Control Systems with Friction", PhD Dissertation, Lund Institute of Technology, Sweden. 172 p.

[Piché 1994] Piché, R., Ellman, A. (1994), "Numerical integration of fluid power circuit models using two-stage semi-implicit Runge-Kutta methods", Proc Instn MEch Engrs Vol 208, pp. 167-175

[Piché 1995] Piché, R., Ellman, A. (1995), "A Fluid Transmission Line Model for Use with ODE Simulators", Proceedings of Eighth Bath International Fluid Power Workshop, Bath, UK

[Pollmeier 1996] Pollmeier, K., Burrows, C.R., Edge, K.A. (1996), "Partitioned Simulation of Fluid Power Systems – an Approach for Reduced Communication Between Processors", Proc Instn Mech Engrs Vol 210, pp. 221-230

[Radhakrishnan 1993] Radhakrishnan, K., Hindmarsh, A. (1993), "Description and Use of LSODE, the Livermore Solver for Ordinary Differential Equations", NASA Reference Publication 1327 / Lawrence Livermore National Laboratory Report UCRL-ID-113855, 124p.

[Rosenbrock 1963] Rosenbrock, H.H. (1963), "Some general implicit processes for the numerical solution of differential equations", Comput J Vol 5 pp. 329-330

[Salane 1986] Salane, D.E. (1986), "Adaptive Routines for Forming Jacobians Numerically", Tech. Report SAND86-1319, Sandia National Laboratories, Albuquerque, NM, USA.

[Sandu 1996] Sandu, A., Verwer, J.G., Blom, J.G., Spee, E.J., Carmichael, G.R. (1996), "Benchmarking stiff ODE solver for atmospheric chemistry problems. II: Rosenbrock solvers", Report NM-R9614, CWI-National Research Institute for Mathematics and Computer Science, Stichting Mathematisch Centrum, The Netherlands.

[Schlösser 1961] Schlösser, W. (1961), "Mathematical Model for Displacement Pump and Motors", Hydraulic Power Transmission April 1961 pp. 252-257 and May 1961, pp. 324,328

[Schäfer 1975] Schäfer, E. (1975), "A new approach to explain "high irradiance responses of photomorhogenesis on the basis of phytochrome". J. of Math. Biology, Vol 2, pp. 41-56

[Shampine 1982] Shampine, L.F. (1982), "Implementation of Rosenbrock methods", ACM Transactions of Mathematical Software, Vol 8 No 2, pp. 93-113

[Shampine 1997] Shampine, L.F., Reichelt, M.W. (1997), "The MATLAB ODE Suite", SIAM J Sci Comput Vol 18 No 1, pp. 1-22

[Stecki 1986a] Stecki, J.S., Davis, D.C. (1986), "Fluid Transmission Lines - Distributed Parameter Models. Part1: a review of the state of the art", proc Instn Mech Engrs Vol 200 No 4, pp. 215-228

[Stecki 1986b] Stecki, J.S., Davis, D.C. (1986), "Fluid Transmission Lines - Distributed Parameter Models. Part2: comparison of models", proc Instn Mech Engrs Vol 200 No 4, pp. 229-236

[Stetter 1973] Stetter, H.J. (1973), "Analysis of discretization methods for ordinary differential equations", Springer, Berlin.

[Thoma 1969] Thoma, J. (1969), "Mathematical Models and Effective Performance of Hydrostatic Machines and Transmisions", Hydraulic and Pneumatic Power, Nov. 1969, pp. 642-651

[Verwer 1999] Verwer, J.G., Spee, E.J., Blom, J.G. and Hundsdorfer, W., (1999), "A second-order Rosenbrock method applied to photochemical dispersion problems", SIAM J. Sci. Comput. Vol 20, No. 4, pp. 1456-1480

[Wilson 1948] Wilson, W.E. (1948), "Performance criteria for positive displacement pumps and fluid motors", ASME Semi-annual meeting 1948, paper No 48-SA-14

[Wolfbrandt 1973] Wolfbrandt, A. (1977), "A study of Rosenbrock Processes with Respect to order conditions and stiff stability", Thesis, Chalmers Univ. of Technology, Goteborg, Sweden

[Wolfbrandt 1977] Wolfbrandt, A. (1977), "A Study of Rosenbrock Processes with Respect to Order Conditions and Stiff Stability", Ph.D. dissertation, Chalmers Univ of Technology, Goteborg, Sweden

APPENDIX A

FORTRAN routines for the offline numerical integration of fluid power systems.

Solvers: RODAS3 and RODAS4 (see Section 5.2 for a description of the solvers)

Driver of the numerical integration using RODAS3 or RODAS4

```
PARAMETER (NVAR=13)       ! NVAR = Dimension of the ODE system

REAL*8 X, Y, XEND, RTOL, ATOL, ITOL, H, HMIN, HMAX, HFIX
EXTERNAL FUN, JAC
DIMENSION Y(NVAR), ATOL(NVAR), RTOL(NVAR)
INTEGER nsteps, IJAC, AUTON, i, N

IJAC = 1     ! IJAC = 1 -> Jacobian is provided
             ! IJAC = 0 -> Jacobian is computed internally by finite differences
AUTON = 0    ! AUTON = 1 -> Autonomous system of ODEs
             ! AUTON = 0 -> Non-autonomous system of ODEs

X =0.D0           ! Integration starting time
XEND = 10.0D0     ! Integration ending time
H = 1.D-6         ! Initial step size
HMIN = 1.D-8      ! Minimum integration step size allowed
HMAX = .1D0       ! Maximum integration step size allowed

i=1
DO WHILE (i.le.NVAR)
    RTOL(i) = 1.D-1               ! Num. integration Relative error tolerance
    ATOL(i) = 1.D-1               ! Num. integration Absloute error tolerance
    i=i+1
END DO

N=NVAR
CALL IC(Y,N)! Returns initial conditions vector Y(0) of Y'=F(Y)

! Executes the numerical integraion routine by using the integrator RODAS3 or RODAS4
CALL RODAS3(N,X,Y,XEND,FUN,JAC,H,HMIN,HMAX,HFIX,RTOL,ATOL,ITOL,nsteps,IJAC,AUTON)
! CALL RODAS4(N,X,Y,XEND,FUN,JAC,H,HMIN,HMAX,HFIX,RTOL,ATOL,ITOL,nsteps,IJAC,AUTON)

! Writes numerical solution vector (Y) in an ASCII file for each integration step
CALL write_solution(nsteps,N)

STOP
END
```

RODAS3

```
SUBROUTINE RODAS3(N,X,Y,XEND,FUNC,JACOB,H,HMIN,HMAX,HFIX,RTOL, ATOL, ITOL, nsteps,IJAC,
        AUTON)

INTEGER N
REAL*8 RPAR, FAC, ITOL
REAL*8 X, Y(N), YNEW(N), Y1(N), Ye(N), E(N,N), DY(N), DFY(N,N)
REAL*8  H, HMIN, HMAX, XEND, ATOL, RTOL, HFIX
```

```
REAL*8 er, erk(N), hnew,q, gamma
REAL*8 a21,a31,a32,a41,a42,a43
REAL*8 c21,c31,c32,c41,c42,c43
REAL*8 m1,m2,m3,m4,m5,m6,me1,me2,me3,me4
REAL*8 C2, C3, C4
REAL*8 K1(N),K2(N),K3(N),K4(N)
INTEGER nsteps, rej, rejcount, i, j, IJAC, AUTON
INTEGER IP(N)
EXTERNAL FUNC,JACOB

DO I=1,N
    DO J=1,N
        DFY(I,J)=0.D0
        E(I,J) =0.D0
    END DO
END DO

! Coefficients (free parameters) of the Rosenbrock formula
gamma = 0.5D0
a21 = 0.D0
a31 = 2.D0
a32 = 0.D0
a41 = 2.D0
a42 = 0.D0
a43 = 1.D0

c21 = 4.D0
c31 = 1.D0
c32 = -1.D0
c41 = 1.D0
c42 = -1.D0
c43 = -2.666666666666667D0

m1 = 2.D0
m2 = 0.D0
m3 = 1.D0
m4 = 1.D0

me1 = 2.D0
me2 = 0.D0
me3 = 1.D0
me4 = 0.D0

IF (AUTON.EQ.0) THEN
    C2 = 0.D0
    C3 = 1.D0
    C4 = 1.D0
ELSE IF (AUTON .EQ. 1) THEN
    C2=0.D0
    C3=0.D0
    C4=0.D0
END IF

q=3.D0   ! q is the order accuracy of the Rosenbrock formula

! Initialization of parameters
nsteps=0
rej=0
rejcount=0

DO WHILE (X.LT.XEND)

    IF (rej.eq.0) THEN
        IF (IJAC .EQ. 0) THEN
!       JACA returns the Jacobian matrix DFY evaluated at the point Y(X) by means of
!       numerical differentiation
        call JACA(N,X,Y,DFY,FUNC)
        ELSE IF (IJAC .EQ. 1) THEN
!       JACOB returns the Jacobian matrix DFY evaluated at the point Y(X) from its
```

```
!       analytical form
        call JACOB(N,X,Y,DFY)
        END IF
!   skips jacobian computation for the new solution if step has been rejected (req==1)
    END IF

    FAC = 1.D0/(h*gamma)
    DO i = 1,N
        DO j = 1,N
        E(i,j) = -DFY(i,j)
        END DO
        E(i,i) = E(i,i) + FAC
    END DO
!   Triangularization of matrix E by Gaussian eliminatiion
    CALL DEC(N,N,E,IP,INFO)

!   Returns DY, the evaluation of function F(Y) at point (X,Y)
    CALL FUNC(N,X,Y,DY)
    DO i=1,N
        K1(i) = DY(i)
    END DO
!   Solution of linear system E*X = K1. Output: K1 = solution vector X
    CALL SOL(N,N,E,K1,IP)
    DO i=1,N
        K2(i) = DY(i)+c21/h*K1(i)
    END DO
    CALL SOL(N,N,E,K2,IP)
    DO i=1,N
        YNEW(i) = Y(i)+a31*K1(i)+a32*K2(i)
    END DO
    CALL FUNC(N,X+C3*h,YNEW,DY)
    DO i=1,N
        K3(i) = DY(i)+c31/h*K1(i)+c32/h*K2(i)
    END DO
    CALL SOL(N,N,E,K3,IP)
    DO i=1,N
        YNEW(i) = Y(i)+a41*K1(i)+a42*K2(i)+a43*K3(i)
!       Ye = Solution of different order of accuracy for error estimation
        Ye(i) = YNEW(i)
    END DO
    CALL FUNC(N,X+c4*h,YNEW,DY)
    DO i=1,N
        K4(i) = DY(i)+c41/h*K1(i)+c42/h*K2(i)+c43/h*K3(i)
    END DO
    CALL SOL(N,N,E,K4,IP)

    DO i=1,N
!       Y1 = Solution of RODAS3
        Y1(i) = Ye(i) + K4(i)
    END DO

!   Local error estimation and prediction of new integrration step size
    ER = 0.D0
    CALL ERROR(N,Y,Y1,YE,H,HMAX,HMIN,ATOL,RTOL,q, er,hnew,erk)
    IF (er.LE.1.D0) THEN
        rej=0           ! Integration is accepted
    ELSE IF  (er.GT.1.D0) THEN
        rej=1           ! Integration is rejected
    END IF
    IF (rej.EQ.0 .OR. rejcount.EQ.100) THEN
        nsteps = nsteps+1
        X = X + h
!       Solution vector Y1 at point X is stored in the memory
        CALL STORE_SOLUTION(X,Y1,h,er,rejcount,nsteps,N, erk)
        rejcount=0
        DO j=1,N
        Y(j)=Y1(j)
        END DO
    ELSE
```

```
        rejcount=rejcount+1
    END IF
    h=hnew

END DO

RETURN
END
```

RODAS4

```
SUBROUTINE RODAS4(N,X,Y,XEND,FUNC,JACOB,H,HMIN,HMAX,HFIX,RTOL, ATOL, ITOL, nsteps,
        IJAC, AUTON)

INTEGER N
REAL*8 RPAR, FAC, ITOL
REAL*8 X, Y(N), YNEW(N), Y1(N), Ye(N), E(N,N), DY(N), DFY(N,N)
REAL*8  H, HMIN, HMAX, XEND, ATOL, RTOL, HFIX
REAL*8 er, erk(N), hnew,q, gamma
REAL*8 a21,a31,a32,a41,a42,a43,a51,a52,a53,a54,a61,a62,a63,a64,a65
REAL*8 c21,c31,c32,c41,c42,c43,c51,c52,c53,c54,c61,c62,c63,c64,c65
REAL*8 m1,m2,m3,m4,m5,m6,me1,me2,me3,me4,me5,me6
REAL*8 C2, C3, C4, C5, C6
REAL*8 K1(N),K2(N),K3(N),K4(N),K5(N),K6(N)
REAL*8 DELT, XDELT, DY1(N), FX(N)
INTEGER nsteps, rej, rejcount, i, j, IJAC, AUTON
INTEGER IP(N)
EXTERNAL FUNC,JACOB

DO I=1,N
    DO J=1,N
        DFY(I,J)=0.D0
        E(I,J) =0.D0
    END DO
END DO

! Coefficients (free parameters) of the Rosenbrock formula
gamma = 0.25D0
a21 = 0.1544000000D1
a31 = 0.9466785232D0
a32 = 0.2557011578D0
a41 = 0.3314825181D1
a42 = 0.2896124002D1
a43 = 0.9986419144D0
a51 = 0.1221224447D1
a52 = 0.6019134331D1
a53 = 0.1253708333D2
a54 = -0.6878860364D0
a61 = a51
a62 = a52
a63 = a53
a64 = a54
a65 = 0.1000000000D1
c21 = -0.5668800000D1
c31 = -0.2430093338D1
c32 = -0.2063598669D0
c41 = -0.1073528206D0
c42 = -0.9594562000D1
c43 = -0.2047028614D2
c51 = 0.7496443504D1
c52 = -0.1024680392D2
c53 = -0.3399990354D2
c54 = 0.1170890893D2
c61 = 0.8083246972D1
c62 = -0.7981132631D1
c63 = -0.3152159434D2
c64 = 0.1631930543D2
```

```
c65 = -0.6058818238D1
m1 = a61
m2 = a62
m3 = a63
m4 = a64
m5 = a65
m6 = 1.D0
me1 = a61
me2 = a62
me3 = a63
me4 = a64
me5 = 1.D0
me6 = 0.D0

IF (AUTON.EQ.0) THEN
    C2=0.386D0
    C3=0.21D0
    C4=0.63D0
    C5=1.D0
    C6=1.D0
ELSE IF (AUTON .EQ. 1) THEN
    C2=0.D0
    C3=0.D0
    C4=0.D0
    C5=0.D0
    C6=0.D0
END IF


q=4.D0   ! q is the order accuracy of the Rosenbrock formula

! Initialization of parameters
nsteps=0
rej=0
rejcount=0

DO WHILE (X.LT.XEND)

    IF (rej.eq.0) THEN
        IF (IJAC .EQ. 0) THEN
!       JACA returns the Jacobian matrix DFY evaluated at the point Y(X) by means of
!       numerical differentiation
        call JACA(N,X,Y,DFY,FUNC)
        ELSE IF (IJAC .EQ. 1) THEN
!       JACOB returns the Jacobian matrix DFY evaluated at the point Y(X) from its
!       analytical form
        call JACOB(N,X,Y,DFY)
        END IF
!   skips jacobian computation for the new solution if step has been rejected (req==1)
    END IF

    FAC = 1.D0/(h*gamma)
    DO i = 1,N
        DO j = 1,N
        E(i,j) = -DFY(i,j)
        END DO
        E(i,i) = E(i,i) + FAC
    END DO
!   Triangularization of matrix E by Gaussian eliminatiion
    CALL DEC(N,N,E,IP,IER)

!   Returns DY, the evaluation of function F(Y) at point (X,Y)
    CALL FUNC(N,X,Y,DY,RPAR,IPAR)
    DO i=1,N
        K1(i) = DY(i)
    END DO
!   Solution of linear system E*X = K1. Output: K1 = solution vector X
    CALL SOL(N,N,E,K1,IP)
    DO i=1,N
```

```
        YNEW(i) = Y(i)+a21*K1(i)
    END DO
    CALL FUNC(N,X+C2*H,YNEW,DY,RPAR,IPAR)
    DO i=1,N
        K2(i) = DY(i)+c21/h*K1(i)
    END DO
    CALL SOL(N,N,E,K2,IP)
    DO i=1,N
        YNEW(i) = Y(i)+a31*K1(i)+a32*K2(i)
    END DO
    CALL FUNC(N,X+C3*H,YNEW,DY,RPAR,IPAR)
    DO i=1,N
        K3(i) = DY(i)+c31/h*K1(i)+c32/h*K2(i)
    END DO
    CALL SOL(N,N,E,K3,IP)
    DO i=1,N
        YNEW(i) = Y(i)+a41*K1(i)+a42*K2(i)+a43*K3(i)
    END DO
    CALL FUNC(N,X+C4*H,YNEW,DY,RPAR,IPAR)
    DO i=1,N
        K4(i) = DY(i)+c41/h*K1(i)+c42/h*K2(i)+c43/h*K3(i)
    END DO
    CALL SOL(N,N,E,K4,IP)
    DO i=1,N
        YNEW(i) = Y(i)+a51*K1(i)+a52*K2(i)+a53*K3(i)+a54*K4(i)
    END DO
    CALL FUNC(N,X+C5*H,YNEW,DY,RPAR,IPAR)
    DO i=1,N
        K5(i) = DY(i)+c51/h*K1(i)+c52/h*K2(i)+c53/h*K3(i)+c54/h*K4(i)
    END DO
    CALL SOL(N,N,E,K5,IP)
    DO i=1,N
        YNEW(i) = YNEW(i)+K5(i)
    END DO
    CALL FUNC(N,X+C6*H,YNEW,DY,RPAR,IPAR)
    DO i=1,N
        K6(i) = DY(i)+c61/h*K1(i)+c62/h*K2(i)+c63/h*K3(i)+c64/h*K4(i)+c65/h*K5(i)
    END DO
    CALL SOL(N,N,E,K6,IP)

    DO i=1,N
!       Ye = Solution of different order of accuracy for error estimation
        Ye(i) = YNEW(i)
!       Y1 = Solution of RODAS4
        Y1(i) = Ye(i) + K6(i)
    END DO

!   Local error estimation and prediction of new integrration step size
    ER = 0.D0
    CALL ERROR(N,Y,Y1,YE,H,HMAX,HMIN,ATOL,RTOL,Q, er,hnew,erk)
10  IF (er.LE.1.D0) THEN
        rej=0           ! Integration is accepted
    ELSE IF  (er.GT.1.D0) THEN
        rej=1           ! Integration is rejected
    END IF
    IF (rej.EQ.0 .OR. rejcount.EQ.100) THEN
        nsteps = nsteps+1
        X = X + h
!       Solution vector Y1 at point X is stored in the memory
        CALL STORE_SOLUTION(X,Y1,h,er,rejcount,nsteps,N,erk)
        rejcount=0
        DO j=1,N
         Y(j)=Y1(j)
        END DO
    ELSE
        rejcount=rejcount+1
    END IF
    h=hnew
```

```
END DO

RETURN
END
```

Linear Algebra subroutines

DEC and **SOL** are linear algebra routines for the decomposition and back-substitution of linear systems. They are public codes available from different sources (e.g. from http://www.unige.ch/~hairer/prog/stiff/decsol.f).

Routine DEC performs a matrix triangularization by Gaussian elimination.
Routine SOL gives the solution of a linear system A*X = B, where A is the triangularized matrix obtained from DEC.

Function and Jacobian evaluation Routines

```
SUBROUTINE FUNC(N,X,Y,F)
! Subroutine FUNC evaluates the function F from the ODE system Y'=F(X,Y)
! Analytical form of function F is to be defined by the user below
!
! INPUT:
!   N: dimension of the system Y'=F(X,Y)
!   X: independent variable
!   Y: vector of solutions at point X
!
! OUTPUT:
!   F: evaluation of function F (Y'=F(X,Y)) at (X,Y)


INTEGER N
REAL*8 X, Y, F
DIMENSION Y(N),F(N)

! Analytical definition of F as a function of X and Y

F(1) =
F(2) =

RETURN
END


SUBROUTINE JACOB(N,X,Y,DFY)
! Subroutine JACOB evaluates the Jacobian matrix of the ODE system Y'=F(X,Y)
! Analytical form of the Jacobian is to be defined by the user below
!
! INPUT:
!   N: dimension of the system Y'=F(X,Y)
!   X: independent variable
!   Y: vector of solutions at point X
!
! OUTPUT:
!   DFY: Jacobian evaluation at (X,Y)

INTEGER N
```

```
REAL*8 X, Y(N), DFY(N,N)

DFY(1,1)=
DFY(1,2)=
DFY(2,1)=
DFY(2,2)=

RETURN
END
```

APPENDIX B

FORTRAN routines for the numerical integration in real-time of fluid power systems.

Solvers: ROS2p and ROS3p (see Section 5.1 for a description of the solvers)

Driver of the numerical integration using ROS2p or ROS3p

```
PARAMETER (NVAR=13)       ! NVAR = Dimension of the ODE system

REAL*8 X, Y, XEND, HFIX
EXTERNAL FUN, JAC
DIMENSION Y(NVAR)
INTEGER nsteps, IJAC, AUTON, N

IJAC = 1     ! IJAC = 1 -> Jacobian is provided
             ! IJAC = 0 -> Jacobian is computed internally by finite differences
AUTON = 0    ! AUTON = 1 -> Autonomous system of ODEs
             ! AUTON = 0 -> Non-autonomous system of ODEs

X =0.D0           ! Integration starting time
XEND = 10.0D0     ! Integration ending time
HFIX = 1.D-3      ! Integration fix step size

N=NVAR
CALL IC(Y,N)! Returns initial conditions vector Y(0) of Y'=F(Y)

! Executes the numerical integraion routine by using the integrator RODAS3 or RODAS4
CALL ROS2p(N,X,Y,XEND,FUN,JAC,HFIX,nsteps,IJAC,AUTON)
! CALL ROS3p(N,X,Y,XEND,FUN,JAC,HFIX,nsteps,IJAC,AUTON)

! Writes numerical solution vector (Y) in an ASCII file for each integration step
CALL write_solution(nsteps,N)

STOP
END
```

ROS2p

```
SUBROUTINE ROS2p(N,X,Y,XEND,FUNC,JACOB,HFIX,nsteps,IJAC,AUTON)

INTEGER N
REAL*8 X, Y(N), YNEW(N), Y1(N), E(N,N), DY(N), DFY(N,N)
REAL*8 H, XEND, HFIX, hnew
REAL*8 gamma, a21, c21, m1, m2, C2
REAL*8 K1(N),K2(N)
INTEGER nsteps, i, j, IJAC, AUTON
INTEGER IP(N)
EXTERNAL FUNC,JACOB

DO I=1,N
    DO J=1,N
        DFY(I,J)=0.D0
        E(I,J) =0.D0
    END DO
```

```
END DO

! Coefficients (free parameters) of the Rosenbrock formula
gamma = 1.D0-1.D0/DSQRT(2.D0)
a21 = (DSQRT(2.D0)-1.D0)/2.D0
c21 = 0.D0
m1 = 0.D0
m2 = 1.D0

IF (AUTON.EQ.0) THEN
    C2=a21
ELSE IF (AUTON .EQ. 1) THEN
    C2=0.D0
END IF

q=2.D0  ! q is the order accuracy of the Rosenbrock formula

! Initialization of parameters
nsteps=0
rej=0
rejcount=0

DO WHILE (X.LT.XEND)

    IF (IJAC .EQ. 0) THEN
!       JACA returns the Jacobian matrix DFY evaluated at the point Y(X) by means of
!       numerical differentiation
        call JACA(N,X,Y,DFY,FUNC)
    ELSE IF (IJAC .EQ. 1) THEN
!       JACOB returns the Jacobian matrix DFY evaluated at the point Y(X) from its
!       analytical form
        call JACOB(N,X,Y,DFY)
    END IF

    FAC = 1.D0
    DO i = 1,N
        DO j = 1,N
         E(i,j) = -h*gamma*DFY(i,j)
        END DO
        E(i,i) = E(i,i) + FAC
    END DO
!   Triangularization of matrix E by Gaussian eliminatiion
    CALL DEC(N,N,E,IP,INFO)

!   Returns DY, the evaluation of function F(Y) at point (X,Y)
    CALL FUNC(N,X,Y,DY,RPAR,IPAR)
    DO i=1,N
        K1(i) = h*DY(i)
    END DO
!   Solution of linear system E*X = K1. Output: K1 = solution vector X
    CALL SOL(N,N,E,K1,IP)
    DO i=1,N
        YNEW(i) = Y(i)+a21*K1(i)
    END DO
    CALL FUNC(N,X+C2*h,YNEW,DY,RPAR,IPAR)
    DO i=1,N
        K2(i) = h*DY(i)
    END DO
    CALL SOL(N,N,E,K2,IP)
    DO i=1,N
!       Y1 = Solution of ROS2p
        Y1(i) = Y(i)+m2*K2(i)
    END DO

    hnew = hfix
    nsteps = nsteps+1
    X = X +h
!   Solution vector Y1 at point X is stored in the memory
    CALL STORE_SOLUTION(X,Y1,h,nsteps,N)
```

```
    DO j=1,N
        Y(j)=Y1(j)
    END DO
    h=hnew

END DO

RETURN
END
```

ROS3p

```
SUBROUTINE ROS3p(N,X,Y,XEND,FUNC,JACOB,HFIX,nsteps,IJAC,AUTON)

REAL*8 X, Y(N), YNEW(N), Y1(N), E(N,N), DY(N), DFY(N,N)
REAL*8  H, XEND, HFIX, hnew
REAL*8 gamma
REAL*8 a21,a31,a32
REAL*8 c21,c31,c32
REAL*8 m1,m2,m3
REAL*8 C2, C3
REAL*8 K1(N),K2(N),K3(N)
INTEGER nsteps, i, j, IJAC, AUTON
INTEGER IP(N)
EXTERNAL FUNC,JACOB

DO I=1,N
    DO J=1,N
        DFY(I,J)=0.D0
        E(I,J) =0.D0
    END DO
END DO

! Coefficients (free parameters) of the Rosenbrock formula
gamma   = 7.886751345948129D-1
a21 = 1.267949192431123D0
a31 = 1.267949192431123D0
a32 = 0.D0
c21  = -1.607695154586736D0
c31 = -3.464101615137755D0
c32 = -1.732050807568877D0
m1  = 2.D0
m2  = 5.773502691896258D-1
m3  = 4.226497308103742D-1

IF (AUTON.EQ.0) THEN
    C2 = 1.D0
    C3 = 1.D0
ELSE IF (AUTON .EQ. 1) THEN
    C2=0.D0
    C3=0.D0
END IF

q=3.D0  ! q is the order accuracy of the Rosenbrock formula

! Initialization of parameters
nsteps=0
rej=0
rejcount=0

DO WHILE (X.LT.XEND)

    IF (IJAC .EQ. 0) THEN
!       JACA returns the Jacobian matrix DFY evaluated at the point Y(X) by means of
!       numerical differentiation
```

```
        call JACA(N,X,Y,DFY,FUNC)
    ELSE IF (IJAC .EQ. 1) THEN
!       JACOB returns the Jacobian matrix DFY evaluated at the point Y(X) from its
!       analytical form
        call JACOB(N,X,Y,DFY)
    END IF

    FAC = 1.D0/(h*gamma)
    DO i = 1,N
        DO j = 1,N
         E(i,j) = -DFY(i,j)
        END DO
        E(i,i) = E(i,i) + FAC
    END DO
!   Triangularization of matrix E by Gaussian eliminatiion
    CALL DEC(N,N,E,IP,INFO)

    !   Returns DY, the evaluation of function F(Y) at point (X,Y)
    CALL FUNC(N,X,Y,DY,RPAR,IPAR)
    DO i=1,N
        K1(i) = DY(i)
    END DO
!   Solution of linear system E*X = K1. Output: K1 = solution vector X
    CALL SOL(N,N,E,K1,IP)
    DO i=1,N
        YNEW(i) = Y(i)+a21*K1(i)
    END DO
    CALL FUNC(N,X+h*C2,YNEW,DY,RPAR,IPAR)
    DO i=1,N
        K2(i) = DY(i)+c21/h*K1(i)
    END DO
    CALL SOL(N,N,E,K2,IP)
    DO i=1,N
        K3(i) = DY(i)+c31/h*K1(i)+c32/h*K2(i)
    END DO
    CALL SOL(N,N,E,K3,IP)

    DO i=1,N
!       Y1 = Solution of ROS3p
        Y1(i) = Y(i)+m1*K1(i)+m2*K2(i)+m3*K3(i)
    END DO

    hnew = hfix
    nsteps = nsteps+1
    X = X +h
!   Solution vector Y1 at point X is stored in the memory
    CALL STORE_SOLUTION(X,Y1,h,nsteps,N)
    DO j=1,N
        Y(j)=Y1(j)
    END DO
    h=hnew

END DO

RETURN
END
```

Linear Algebra subroutines

DEC and **SOL** are linear algebra routines for the decomposition and back-substitution of linear systems. They are public codes available from different sources (e.g. from http://www.unige.ch/~hairer/prog/stiff/decsol.f).

Routine DEC performs a matrix triangularization by Gaussian elimination.

Routine SOL gives the solution of a linear system A*X = B, where A is the triangularized matrix obtained from DEC.

Function and Jacobian evaluation Routines

```
SUBROUTINE FUNC(N,X,Y,F)
! Subroutine FUNC evaluates the function F from the ODE system Y'=F(X,Y)
! Analytical form of function F is to be defined by the user below
!
! INPUT:
!   N: dimension of the system Y'=F(X,Y)
!   X: independent variable
!   Y: vector of solutions at point X
!
! OUTPUT:
!   F: evaluation of function F (Y'=F(X,Y)) at (X,Y)


INTEGER N
REAL*8 X, Y, F
DIMENSION Y(N),F(N)

! Analytical definition of F as a function of X and Y

F(1) =
F(2) =

RETURN
END


SUBROUTINE JACOB(N,X,Y,DFY)
! Subroutine JACOB evaluates the Jacobian matrix of the ODE system Y'=F(X,Y)
! Analytical form of the Jacobian is to be defined by the user below
!
! INPUT:
!   N: dimension of the system Y'=F(X,Y)
!   X: independent variable
!   Y: vector of solutions at point X
!
! OUTPUT:
!   DFY: Jacobian evaluation at (X,Y)

INTEGER N
REAL*8 X, Y(N), DFY(N,N)

DFY(1,1)=
DFY(1,2)=
DFY(2,1)=
DFY(2,2)=

RETURN
END
```