Exploring Autonomic Computing

Rodrigo Fernandes de Mello

April 2009

Acknowledgements

Some chapters of this work had the important cooperation of the following authors:

1. Cássio Mantini – on self-healing;

2. Eduardo Alves – on immunologic systems;

3. Marcelo Keese Albertini – on the self-organizing novelty detector network.

Contents

Chapter

1

Basic Concepts

1.1 Definitions

The technological evolution has decisively influenced in the real-world production. Decades ago, companies observed they could invest money on computer hardware and software to automatize their departments. Results were fast noticed, thus, everyday tasks started being better organized and controlled. Afterwards, CEOs observed they could get production information through reports and other tools to support management. Company sectors were already working when managers decided to integrate department softwares and synchronize the whole business. At the same time, ordinary people started taking advantage of personal computers and other electronic devices. Most of those equipments were used to support people's paper work and business.

All this evolution is based on the fact that technology helps controlling mechanical tasks and improves production. This same fact has continuously reinforced the growth of the technological market what has been motivating the development of new types of devices, programming languages, operating systems and interconnection networks. The more technology is available, more users look for ways of integrating them. This happens not just inside companies, but also to end users. Such integration is part of the next step to reduce the production cost and, consequently, increase income. That made some companies to apply around three fourth of the time and deployment costs on system integration.

The same companies have been facing problems to find out high-qualified people for integration tasks. Besides the integration, the maintenance of the current systems is also a big challenge. Companies need to have employees with different qualifications such as database, system and network administrators. All those different points of view need to work together to keep the internal systems running. A relevant information, according to the University of Berkeley, is that the maintenance and integration costs are six times higher than the infrastructure investments. Other investigations by Klein (67) confirm that 80% of the IT budget is applied on maintaining current applications.

This scenario is already unstable and it is becoming unsustainable what may jeopardize the current production environments. Such situation made Paul Horn, vice-president and research director at IBM, to bring a new definition for the necessary technological changes what he called the autonomic computing. Autonomic computing is an emerging concept which aims at making systems capable of self-managing their tasks, avoiding or minimizing the human intervention. Horn proposed this term based on the human nervous system (HNS). HNS is basically divided into the voluntary and involuntary systems. The voluntary is controlled by the humans (such as pain and heat caused by external objects). The involuntary, also known as the autonomic nervous system, takes actions without the explicit human control, such as the heart frequency, digestion, blood circulation and glandular functions.

The involuntary system is still divided into three categories. The first is the sympathetic nervous system (SNS), which accelerates muscles and organs to support the body adaptation. The second, or the parasympathetic nervous system (PNS), slows muscles and organs down, helping to calm people down. The last is the metasympathetic nervous system (MNS), which is represented by the central nervous systems (relative to the brain). In unstable situations, the SNS and PNS may accelerate and slow down at the same time what can influence in body stability.

Autonomic computing considers four main aspects to provide self-managing features to systems (87):

1. self-optimization – this aspect provides the system the ability to monitor and detect its behavior variations. Those variations are employed to maintain and improve performance. For instance, consider a computing environment and a set of tasks to be scheduled. The optimized distribution of tasks is performed, and periodically reevaluated, aiming at keeping good performance levels;

2. self-configuration – according to this aspect, a system must be capable of self-installation and self-configuration. It considers adaptive algorithms which are able to update the system without jeopardizing the normal execution. For instance, consider a running application that modifies a database. When updating it, the database is recreated. In this situation, the self-configuration aspect must guarantee that the current execution is not affected by future updates. Consider another circumstance, where a service receives user requests. In case of an expressive increase in requests, the system could replicate the service and, consequently, be able to fulfill more requests;

3. self-healing – a system must be capable of detecting incorrect execution states, faults and other abnormalities, as well as take them over, minimizing the interference in the normal execution. For instance, consider a system which periodically saves the execution contexts (*checkpoints*) of its processes on the hard disk. In case a process fail, the system can resume its execution from the last saved context, what is transparent to the end user and minimizes the cost of a execution from scratch. A system without such support would restart its execution, what incurs waiting time and additional resource allocation;

4. self-protection – it aims at identifying, detecting and protecting the system against intrusions capable of modifying its normal behavior. In this way, the system must keep its processing and data integrity and still guarantee the consistency of its operations. For instance, if a program or an user acts in an unexpected and intrusive way, the system will detect it and avoid such activities.

To complement those aspects, an autonomic system must be robust, protecting itself from malicious users and applications; easy to use; proactive, this is, take decisions without the explicit user cooperation; transparent, i.e., the users and developers do not notice the system; reversible, i.e., the system must be capable of canceling its operations.

Although recently formalized, autonomic computing has been employed for decades. The RAS (Reliability, Availability and Serviceability) concepts employed by IBM on its mainframes, since 1970, intended to provide autonomic features by exploring hardware redundancy and fault detection. The IBM 4300, introduced in 1982, had self-diagnosis features and transmitted log information, memory dumps, and error codes to IBM. Microsoft Windows as well as some Linux flavors consider auto-update tools such as *apt-get*. Another example is the RAID (Redundant Array of Independent Disks) technology which allows the recovery of data in case of hard disk problems. Data are automatically read or written in another part of the array. Those RAID operations are transparent to end users.

Other specific autonomic initiatives are conducted by Efes Pilsen, Santix and Bankdata (87). Efes Pilsen is a beer company which uses a high available module to manipulate its datawarehouse structure. This module presents self-managing features. Santix is a software integration company which adopts self-optimization techniques to reduce administrators' tasks. Bankdata is a financing institution which considers autonomic software and hardware in data mining operations.

Those examples make evident that the Autonomic Computing concept has been applied for decades, although using different names. Among those names or associated terms are Adaptive Enterprise, proposed by Cap Gemini Ernst & Young; Dynamic Systems Initative, by Microsoft; e-Business on Demand, by IBM; N1, by Sun Microsystems; On-Demand Computing, by EDS; Organic IT, by Forrester Research; Policy-Based Computing, by Gartner; Real-Time Enterprise, by Gartner; Utility Data Center, by Hewlett-Packard.

1.2 Complexity

The same level of complexity that is currently observed in software systems also happens in our society. That has been motivating easier ways of solving everyday tasks. Examples of such complexities are (87):

1. In Netherlands a woman took one week to find the way out of a big shopping mall. She bought food during the day and slept on a bench at night;
2. French farmers started a riot because they did not understand the new agriculture laws (they have blocked roads with tractors and other equipments);
3. Researches indicate that 90% of the video camera features are not used by 95% of the owners (they only exceed that when they have teenagers);
4. When people go to big shopping malls, they frequently forget where they parked;
5. Most of the taxpayers leave the annual refund or rebate for last. According to researches, this happens due to its complexity;
6. Programmers and technicians write manuals, but frequently forget to address common user problems;
7. NASA invested million of dollars to design a pen to work on the space. It would work upside down; gravityless; in any surface, including glass; in very low and high temperatures, up to 300 centigrades (Russians decided to use the pencil);

The situations mentioned make clear how complex is the current society and that people need to find easier ways to tackle problems. The same complexity levels are observed in systems such as (87):

1. 1960 – The American Satelite Corona was lauched after 12 consecutive faults (due to hardware problems);
2. 1962 – Ranger 3 was launched to leave instruments over the moon. Software problems caused a 22.000 miles deviation from the original target (US$ 14 million);
3. 1962 – Mariner I was launched to reach Venus. It lost its trajectory right after leaving earth and it was destroyed. A hiphen was missing in the launching code (US$ 16 million);

4. 1981 – The costs to develop the US Airforce communication and control system was 10 times higher than expected (US$ 3.2 million);

5. 1992 – The London ambulance system had problems due to the lack of tests. False emergencies and duplicated attendance happened (US$ 50 million);

6. 1995 – The official opening of the Denver international airpoirt was postponed in 9 months due to baggage system problems;

7. 1997 – The development of the Statewide Automated Child Support System of California was suspended because it exceded the original costs (US$ 312 million);

8. 1997 – Soyuz was wrongly positioned by the system when coming back to earth (heat shield at the back). This was manually solved;

9. 1999 – The Ariane rocket, by the European Space Agency, was destroyed after launching (US$ 500 million);

10. 2000 – Mars Polar Lander had software problems due to metric conversion. It collided against Mars (US$ 165 million);

11. 2001 – X-Ray disaster in Panama. The X-Ray dosage was wrong and caused health problems (including death).

Another example is the Bank One which bought many small american banks and had difficulty to integrate them all. Softwares were not ready to work together. They concluded the costs to install, upgrade, configure, maintain, optimize, detect problems and provide security were very high to join the whole structure.

Those questions have motivated companies to reduce the IT complexity by (87):

1. The whole infrastructure must be evaluated what includes softwares, resources, processes and operational procedures;

2. Consolidation of hardware and software;

3. Tasks that can possibly be outsourced;

4. Looking for systems that can be automatically updated (releases, bugs, patches, etc.);

5. Better analyse the need for new software developments and the process in which they are developed.

The complexity was also approached by Lou Gerstner who started working at IBM as CEO in 1993. He was the first CEO who was not an employee and a computer specialist. He attempted to focus on the customer and not on the technological complexities. The IBM market value increased from US$ 29 billion of dollars, in 1993, to US$ 181 billion, in march 2002. According to Gerstner, the company needed to be integrated and focused on customers. During his management, the company reduced from 128 CIOs to 1; from 55 datacenters to 12; from 80 web hosting centers to 11; from 31 communication networks to 1; from 16.000 products to 5.200. IBM saved US$ 9 billion and it was 75% faster to launch new products. It also got a growth of 5.5% in the customer satisfaction.

1.3 Products and Autonomic Applications

This section presents examples of systems that include autonomic features (87).

1.3.1 IBM DB2

IBM D2 is a toolkit to support database administration. It executes on several platforms. The recent versions attempt to automatize the deployment, management and configuration of relational databases. IBM DB2 is able to detect and correct a set of problems without the adminitrator's intervention. Those autonomic features are based on the:

1. Health monitor – monitors the system in order to provide performance and reliability metrics. In case of problems, it warns the administrator about the severity and the source of the issue;

2. Configuration advisor – collects system information (about hard disks, CPU, operating systems, database size and usage) to optimize operations.

1.3.2 Intamission Autevo

Intamission is a British company which develops an autonomic toolkit to IBM Autevo. This toolkit is JINI and JavaSpaces-compatible. It works as a memory over the communication system and provides sharing, persistency and transaction support. It is based on the Java RMI technology and it improves the flexibility of information exchange for J2EE applications. It allows the distribution of Servlets and EJBs on multiple application servers.

1.3.3 Space Systems

The new era of space systems must be autonomic, independent and capable of deciding on different situations. The project Milenium by NASA is an example of space system which employs artificial intelligence. They invested in this AI technology because the control from earth takes too long to reach the devices (for instance, the time to reach Mars and come the signal back to earth takes 11 minutes) and those equipments need to take actions based on environmental experiences (soil, temperature, etc.).

A first version of this system was developed. According to NASA, it will be considered in different missions and they estimate that it will reduce the mission costs in 60% (because large teams and complex equipments will not be necessary as nowadays). They also expect to reduce the energy consumption and schedule missions by using the AI system. They have also been designing a module to make remote modifications in the AI system (devices can have their systems updated). NASA is also investing resources in the constellation of ships. They intend to launch different devices which may exchange information one each other. All these investments are due to the fact that NASA intends to explore farthest and unknown places which require autonomic features.

1.4 IT Industry Requirements

The IT industry needs high-qualified labor and also to find ways of reducing costs. That made a huge movement of capital from US to other countries. American companies started to outsource their development sectors to countries like India and China. That modified the North American and European internal investments on technology what made people decide to look for different careers (Computer science is not one of the top careers as it used to be). Nowadays, developed countries do not locally find the minimum human resources to fulfill the administrative and management tasks (87).

Such outsourcing movement has degraded the local employment. As a way of approaching such issue, some companies have been investing on professional certification, careers and education. Some countries have also motivated the immigration as a manner of bringing high-qualified professionals. Those professionals are mainly related to administrative, design and management tasks.

1.5 Autonomic Components

An autonomic system is basically composed of the four main components (87; 90):

1. Managed element – sensors and actuators control the managed element which can be, for instance, a server, a database, a file or any other component;

2. Autonomic Manager – collects, filters and generates reports based on the available data. It also analyses and learns about the managed element. It accumulates knowledge and can also make predictions;

3. Sensors – they provide mechanisms to collect data on the status of managed elements;

4. Actuators – they modify the element state acccording to decisions taken by the autonomic system.

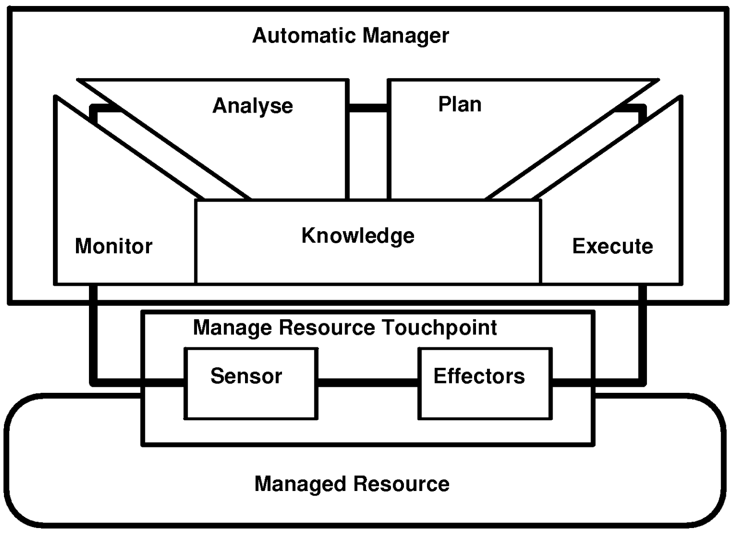


Figure 1.1: Autonomic Manager

In the software context, the main part to be studied is the Autonomic Manager (figure 1.1). The design of an Autonomic Manager must consider the following aspects:

1. Policy determination – the designer needs to know all the specific policies that the Autonomic Manager must follow. Policies define criteria. They are rules such as: backup files after the monthly processing; and, after receiving the user login and password, make the authentication and load his/her environment;

2. Solution Knowledge – different softwares have different ways of installation, update, configuration and maintenance. They also have different formats, parameters and return codes which bring a big complexity to IT. The solution software knowledge must capture configurations, installation requirements and other information to be considered by the self-optimization and self-configuration modules;

3. System administration – single and standard interface to access the autonomic system;

4. Problem determination – this aspect considers different managed element information to figure out solutions. It integrates information types, patterns and data sources to decide on possible actions. The analysis of such information and the use of knowledge allows to determine the specific system problem and how to solve it;

5. Autonomic Monitoring – it provides the ability to obtain and filter sensor data which is considered to take actions;

6. Complexity Analysis – there are different data types and volumes, which come from many sources. Data vary in type, format, size, updating scheme. The Autonomic Manager must detect and analyse which data are relevant for its operations. This step involves learning machine techniques;

7. Transaction Measurement – the system transaction flow must be known what helps to understand when a resource is needed. By using the monitor, the system can relocate the resources and optimize the workflow among Autonomic Managers. For instance, two Autonomic Managers detect high WEB transaction volumes and they cooperate to configure a cluster of computer and also other services to better fulfill user requests.

1.6 Initiatives

There are many initatives in the Autonomic Computing area such as (87):

1. IBM/CISCO – they are working together on self-healing initiatives. This work involves the design and implementation of technologies to detect, log and solve system problems by applying event correlation;

2. IBM/Think Dynamics – IBM bought Think Dynamics which develops toolkits to allocate resources according to service requirements;

3. Sun Microsystems – the company is developing the N1 project which attempts to manage distributed resources such as storages, servers, softwares and networks. The goal is to provide some services for the infrastructure such as virtualization, provisioning and automate business policies;

4. Microsoft/HP – they are developing the Dynamic Data Center project which provisions and manages resources;

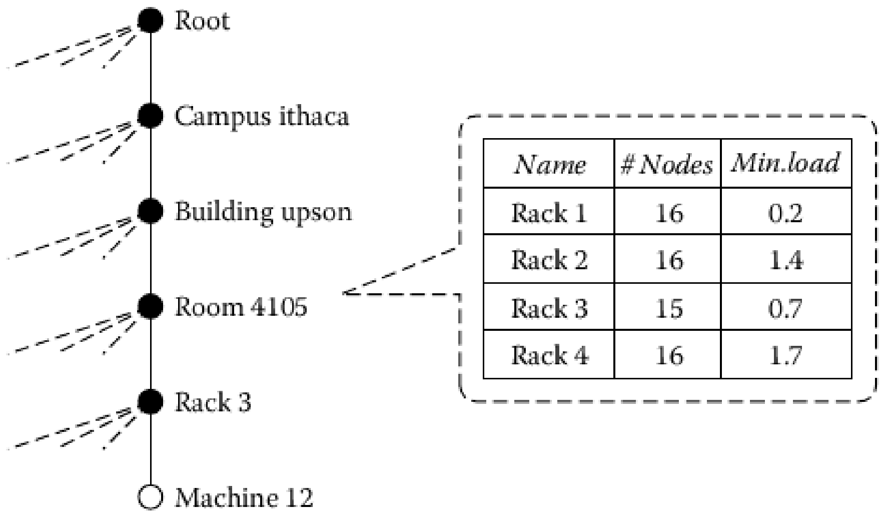
5. Intel – the company is developing the Proactive Computing project which attempts to anticipate user needs and execute operations to fulfill such requirements.

1.7 A System-Wide Perspective

A system may have autonomic components what does not mean that it works as an autonomic system. This reflects the need for integrating knowledge about the whole system. For instance, consider an application and the filesystem running on the same computer. That application access the filesystem. Let's consider that both are implemented as adaptive and autonomic components which attempt to improve performance by using as much main memory as necessary (114).

Now consider a large workload submitted to the application. The application will try to allocate more memory which it taken from the filesystem. The filesystem therefore cannot keep the high throughput. Due to the low throughput, the application autonomic module notices that it does not need that much memory anymore and starts freeing it. Then, the filesystem has more available memory and, consequently, it can improve throughput.

This is a case where autonomic managers are looking for keeping their service levels in a selfish way. They must be orchestrated by a global system module, which decides when and how a resource, in this situation the main memory, is used. Consider a similar situation where multiple autonomic modules execute. They might fight one each other and the overall system performance, availability and security are not met at all. Those circumstances require a global control infrastructure.



Name	*#Nodes*	*Min.load*
Rack 1	16	0.2
Rack 2	16	1.4
Rack 3	15	0.7
Rack 4	16	1.7

Figure 1.2: Global control

That control works as a single-view database and it usually organizes component information in a hierarchical model (figure 1.2), where queries are submitted and act on levels. Queries can, for instance, ask for the processing capacity of all resources in room 3 or the actives nodes in the rack 5. It allows one-shot and continuous queries. One-shot fulfill the query and it finishes. The periodical ones update the query responsible on a time basis. Most of the times, the global control database is not persistent (information is stored on computer main memory and requested when necessary).

The global control also allows queries such as: where is the file foo.txt? Where is the most recent replica of the file test.txt? Which computers have the anti-virus version 1.2.3? Which are the idlest computers? Which computer from cluster A runs a DNS service? Those queries can also feed the knowledge base and support the system control. For instance, the number of file replicas depends on he filesystem workload; the replica distribution depends on where they are more requested.

Different approaches can be used to design and implement global controls such as: distributed (peer-to-peer networks) or centralized schemes, different information update protocols (gossip-based and consistent protocols), distributed hash tables to address where an information is, web services, clone (easier to maintain) and partition (lower cost to keep services consistent and scalable) services on different computers. The global control can also be based on existent software such as the SNMP (Simple Network Management Protocol) implementations and peer-to-peer catalogs (which receive periodical information update). Each approach has its pros and cons, some of them provide scalable systems (peer-to-peer for instance) others are easier to use and well known (SNMP implementations). An example of global control is the Astrolabe (114).

1.8 Design Patterns

Sweitzer & Draper (110) studied and proposed design patterns for autonomic systems. Part of those patterns is presented as follows:

1. the first considers how multiple autonomic managers could deal with the same resource (figure 1.3). The proposed solution (figure 1.4) considers an enterprise service bus to unify autonomic managers requests, synchronize them all and also provide a single interface.

2. the second addresses how resource information can be shared among autonomic managers. The authors propose the design of a database to store configuration management information (figure 1.5). The database would provide a single access interface. By using this pattern, autonomic managers do not need to understand all the resource details to get information, they simply interact with the database. This is an interesting solution, although the solution should provide ways of guaranteeing the database availability by using, for instance, data replication and managing consistency issues.

3. the third design pattern approaches the management of autonomic managers. The proposal is to have higher-level managers to deal with others (figure 1.6). The higher level controls and can combine information from multiple managers.

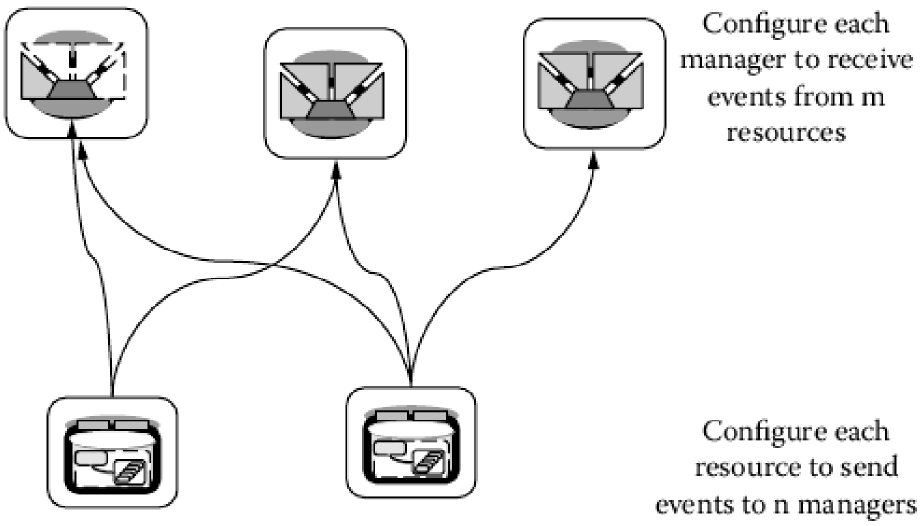

Configure each manager to receive events from m resources
Configure each resource to send events to n managers

Figure 1.3: Pattern: Dealing with multiple resources

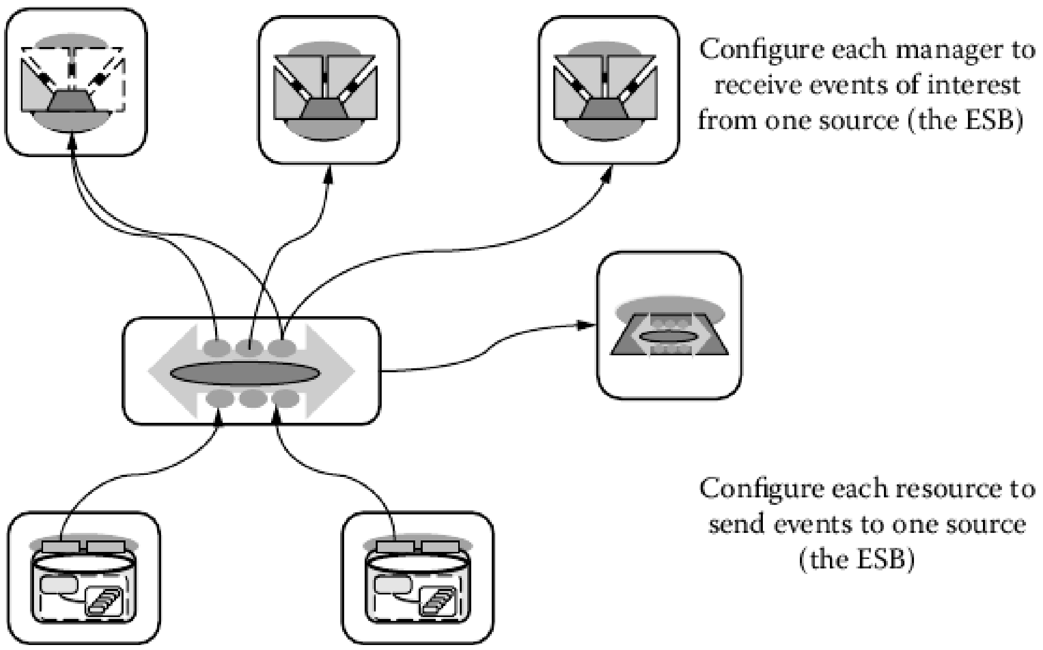

Configure each manager to receive events of interest from one source (the ESB)
Configure each resource to send events to one source (the ESB)

Figure 1.4: Pattern Solution: Dealing with multiple resources

4. in the next pattern, the authors propose the integration of multiple autonomic managers to compose the system workflow (figure 1.7). This approach allows the integration and flow of information to improve resource controlling and coordination among different tasks.

5. the fifth pattern (figure ??) addresses the control of resource internals. This pattern considers that more complex resources may have internal autonomic managers. For instance, consider an autonomic manager for a database which is capable of recommending the query execution and retrieval according to the best indices.

Those patterns demonstrate the design worries of autonomic systems. As those sys-

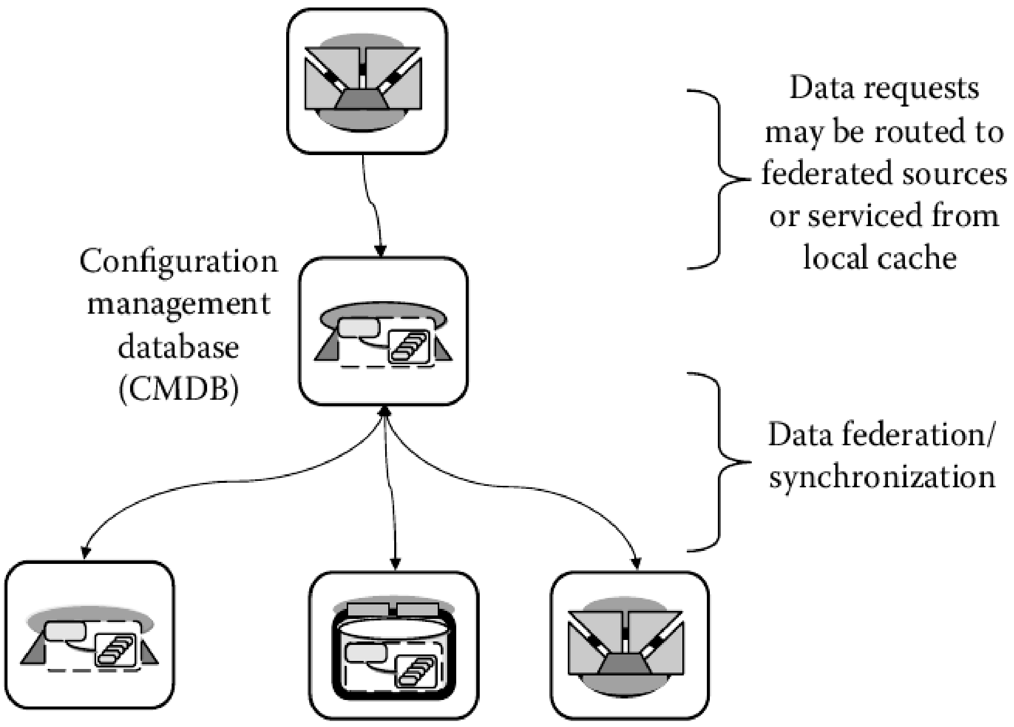

Data requests may be routed to federated sources or serviced from local cache
Configuration management database (CMDB)
Data federation/ synchronization

Figure 1.5: Pattern Solution: Sharing information

Figure 1.6: Pattern Solution: Management of autonomic managers

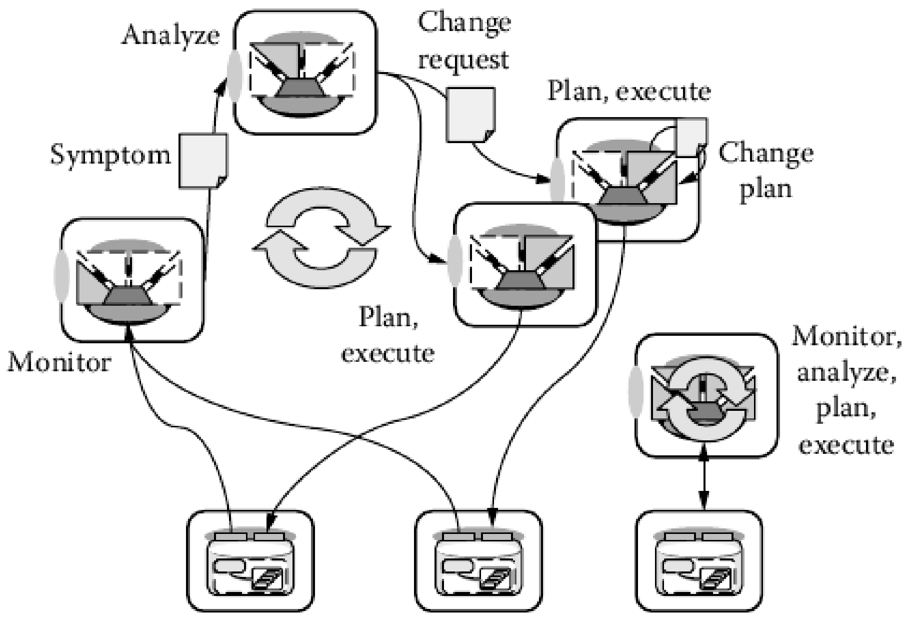


Figure 1.7: Pattern Solution: Integration of multiple managers

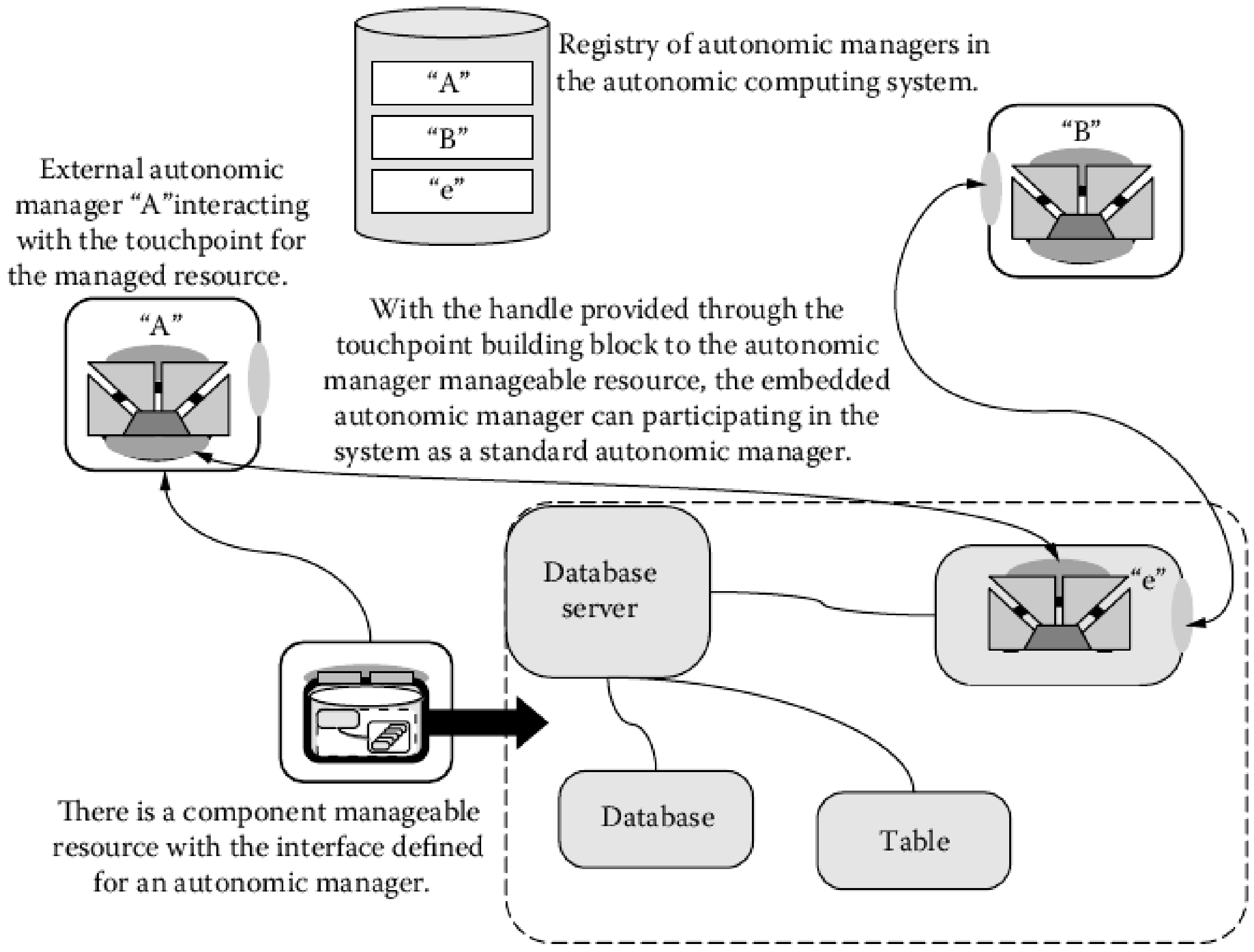

Registry of autonomic managers in the autonomic computing system.
"A"
"B"
"e"
External autonomic manager "A" interacting with the touchpoint for the managed resource.
"A"
"B"
With the handle provided through the touchpoint building block to the autonomic manager manageable resource, the embedded autonomic manager can participating in the system as a standard autonomic manager.
Database server
"e"
Database
Table
There is a component manageable resource with the interface defined for an autonomic manager.

Figure 1.8: Pattern Solution: Autonomic managers for resource internals

tems become more and more adopted, new approaches will be proposed to address other issues. The basic idea behind them is to provide a better and more structured way of desigining and implementing such systems. They consider the most common situations and best practices when developing.

1.9 Research Opportunities

There are several different areas which can take advantage on the autonomic computing concepts. Those areas motivate researches on topics such as (87; 90):

1. Problem determination – when an event is a problem or not. How to characterize the different levels of problems and their sources;

2. Fault tolerance and system take over – detection and analysis of faulty hardware and software modules as well as ways of taking those situations over, without causing damages or jeopardizing the normal system execution;

3. Continuous, predictive and adaptive optimization – extract and analyse data to provide the best optimization as possible in a certain moment and for the future system events;

4. Self-learning and its application on all the autonomic aspects – learn about the system situations, events, etc. and store such information to improve autonomic aspects;

5. Formalization and representation of business policies – ways of defining business policies and how to orchestrate system components to work together;

6. Resource Discovery – how to discover and get information about resources such as capacity, workloads, etc. Such information is useful when optimizing a system, when trying to improve the high availability, etc.;

7. User interface – users need easy and unified interfaces to access systems. Operation interfaces cannot change everytime and be too hard to manage, otherwise it goes back to the huge complexity of the current systems;

8. System context – the system must know the environment where it executes. This includes user interactions, user profiles, system capacity and workload, application characteristics, etc.;

1.10 Other Considerations

Imagine all the electric power plants with different interfaces. Each one with its own voltage and frequency. Now, consider each company manufacturing equipments for customers of a specific power plant. That specificity would bring more and more complexity to the whole system. It would made almost impossible moving out a country region, maybe cities. Standards make everything easier, this is the same recipe followed by those plants. Even in this way, they have some differences (87; 90).

For software companies, things are similar. When possible, softwares must consider standards such as XML, SOAP, WSDL, HTTP, etc. They make easier to integrate different systems. Even considering standards, companies can choose in between proprietary and open standards. The open ones are preferable mainly because the technology is maintained by different companies and not just one.

Besides the standard considerations, there are others related to employment. Many employees may be afraid of losing their positions due to the autonomic features. But, in fact, they could be moved to other sectors and help the companies in the core business. Others might carry on the technical work, because not all the activities can be autonomic. Different autonomic levels can be find:

1. Basic level – every system element is well managed by the IT team. The team monitors and substitutes fault elements;

2. Management level – tools to collect information and synthesize them to support IT management;

3. Predictive level – correlation among system elements, detection of working patterns and prediction to optimize operations;

4. Adaptive level – prediction is used to adapt the system operations;

5. Autonomic level – IT is managed by business policies.

1.11 Final Remarks

This chapter defined and introduced autonomic concepts. It also approached the current society and system complexity as well as the concept of autonomic manager, the main component in a autonomic system. Such component is resposible for grabing, analysing, planning and executing operations on resources. The chapter also presented applications, initiatives, a system-wide perspective, design patterns and, finally, research opportunities. It contains the basic knowledge for reading the next chapters.

Exercises

1. Do you have other examples of software or society complexities in mind?

2. Find out and study other automic systems. What kind of autonomic features do they provide?

3. Study or detect important autonomic features to develop in your area of interest. Why are they relevant?

Chapter 2

Building up an Autonomic System

The main component of an autonomic system is the autonomic manager which is responsible for obtaining information from a resource, analysing it according to the autonomic requirements (drawing decisions), planning such decisions to the real-world circumstances and executing them. Autonomic managers deal with four basic aspects of the autonomic computing: self-optimization, self-configuration, self-healing and self-protection. The steps to extract information, employ it according to those aspects, planning and execution are presented in this chapter.

2.1 Information Extraction

The system information is composed of inputs, outputs and internal states of execution. Such information is provided to the autonomic manager to take decisions. The most updated and accurate the information is, the better will be the results. In hardware systems, the designer may consider monitoring systems to obtain such information. An example is the SMART (Self-Monitoring Analysis and Reporting Technology) which is a hard drive assessing system developed by manufacturers. It captures hard drive information such as read error rate, throughput performance, seek error rate, etc. (106). A similar project is the lm_sensors which provides hardware monitoring tools for Linux (73). This project captures the speed of fans, system voltages, chip and motherboard temperature, fan speed control which depend on the hardware being monitored.

Those monitoring systems follow the same principle. They ask for and obtain information from a black box system and are usually periodical approaches. For instance, consider the Linux software *vmstat* which monitors the computer software providing information such as the number of bytes in and out from main and swap memories. It grabs the system information in a specific time instant and, then, runs again to get more information. All such information can be composed to plot charts such as the CPU and memory usage.

Those monitoring systems frequently consider counters. Counters are variables which account the system utilization in a specific time instant. Let's say we are periodically monitoring hard disk read operations. In a moment, the monitor requests the number of reads and it returns zero. After ten seconds, we ask for it again and the system returns 5 what means that five read operations were executed in between the first and the second monitoring points. This is one of the most typical approaches for obtaining system information.

The monitoring approach grabs the ammount of data and what kind of event has happened in a certain time interval, but it does not inform you, or the autonomic manager, when an event has happened and its full information. This second approach, named event-based monitoring or just event interception, calls a procedure in to inform that something has happened. Let's imagine a interception module on a hard disk that everytime a read operation occurs it interrupts the read and calls a function to inform about it. This approach is more exact as it informs the right moment when the event occurs. The accurate information may help the design of the autonomic system, however, depending on the called procedure, the complexity and the interception cost can become prohibitive.

This is the basic trade off for the information extraction system. The designer needs to evaluate if its possible to monitor and intercepts as well as the best choice for a certain system. Some systems can only be monitored while others intercepted. Given the inherently physical characteristics, hardwares are most likely to be monitored while software systems can be monitored or intercepted.

2.1.1 SMART: Hard drive monitoring

Self-Monitoring, Analysis, and Reporting Technology (SMART) is a monitoring tool introduced by hard disk manufactures which obtain hard disk information to detect and report reliability (106). SMART follows the AT Attachment (ATA), which is an interface standard for storage devices (hard disks, solid-state drivers and CD-ROMs) (7). Each drive manufacturer selects the most important attributes to monitor and define the ranges from them. The attributes has a raw value which depends on the manufacturer and a normalized one which lays in the interval $]1, 253[$, where 1 represents the worst case and 253, the best. Manufactures that implement at least one of the SMART attributes are: Samsung, Seagate, IBM (Hitachi), Fujitsu, Maxtor, Toshiba, Western Digital and ExcelStor Technology.

The SMART attributes and their usual meaning are presented in tables 2.1, 2.2 and 2.3 (106).

Table 2.1: Smart attributes and their meaning

ID	Hex	Attribute name	Better[1]	Critical[2]	Description
01	01	Read Error Rate	$<$	yes	Indicates the rate of hardware read errors that occurred when reading data from a disk surface. A non-zero value indicates a problem with either the disk surface or read/write heads. Note that Seagate drives often report a raw value that is very high even on new drives, and does not thereby indicate a failure
02	02	Throughput Performance	$>$	no	Overall (general) throughput performance of a hard disk drive. If the value of this attribute is decreasing there is a high probability that there is a problem with the disk
03	03	Spin-Up Time	$<$	no	Average time of spindle spin up (from zero RPM to fully operational – millisecs)
04	04	Start/Stop Count		no	A tally of spindle start/stop cycles
05	05	Reallocated Sectors Count	$<$	yes	Count of reallocated sectors. When the hard drive finds a read/write/verification error, it marks this sector as "reallocated" and transfers data to a special reserved area (spare area). This process is also known as remapping, and "reallocated" sectors are called remaps. This is why, on modern hard disks, "bad blocks" cannot be found while testing the surface – all bad blocks are hidden in reallocated sectors. However, as the number of reallocated sectors increases, the read/write speed tends to decrease. The raw value normally represents a count of the number of bad sectors that have been found and remapped. Thus, the higher the attribute value, the more sectors the drive has had to reallocate
06	06	Read Channel Margin		no	Margin of a channel while reading data. The function of this attribute is not specified
07	07	Seek Error Rate	$<$	no	Rate of seek errors of the magnetic heads. If there is a partial failure in the mechanical positioning system, then seek errors will arise. Such a failure may be due to numerous factors, such as damage to a servo, or thermal widening of the hard disk. More seek errors indicates a worsening condition of a disk's surface or the mechanical subsystem, or both. Note that Seagate drives often report a raw value that is very high, even on new drives, and this does not normally indicate a failure
08	08	Seek Time Performance	$>$	no	Average performance of seek operations of the magnetic heads. If this attribute is decreasing, it is a sign of problems in the mechanical subsystem
09	09	Power-On Hours (POH)	$<$	no	Count of hours in power-on state. The raw value of this attribute shows total count of hours (or minutes, or seconds, depending on manufacturer) in power-on state
10	0A	Spin Retry Count	$<$	yes	Count of retry of spin start attempts. This attribute stores a total count of the spin start attempts to reach the fully operational speed (under the condition that the first attempt was unsuccessful). An increase of this attribute value is a sign of problems in the hard disk mechanical subsystem
11	0B	Recalibration Retries	$<$	no	This attribute indicates the number of times recalibration was requested (under the condition that the first attempt was unsuccessful). An increase of this attribute value is a sign of problems in the hard disk mechanical subsystem
12	0C	Power Cycle Count		no	This attribute indicates the count of full hard disk power on/off cycles
13	0D	Soft Read Error Rate	$<$	no	Uncorrected read errors reported to the operating system
184	B8	End-to-End error	$<$	yes	This attribute is a part of HP's SMART IV technology and it means that after transferring through the cache RAM data buffer the parity data between the host and the hard drive did not match

Table 2.2: Smart attributes and their meaning

ID	Hex	Attribute name	Better[3]	Critical[4]	Description
187	BB	Reported Uncorrectable Errors	$<$	no	A number of errors that could not be recovered using hardware ECC (see attribute 195)
188	BC	Command Timeout	$<$	yes	A number of aborted operations due to HDD timeout. Normally this attribute value should be equal to zero and if you have values far above zero, then most likely you have some serious problems with your power supply or you have an oxidized data cable
189	BD	High Fly Writes	$<$	no	HDD producers implement a Fly Height Monitor that attempts to provide additional protections for write operations by detecting when a recording head is flying outside its normal operating range. If an unsafe fly height condition is encountered, the write process is stopped, and the information is rewritten or reallocated to a safe region of the hard drive. This attribute indicates the count of these errors detected over the lifetime of the drive. This feature is implemented in most modern Seagate drives and some of Western Digital's drives, beginning with the WD Enterprise WDE18300 and WDE9180 Ultra2 SCSI hard drives, and will be included on all future WD Enterprise products
190	BE	Airflow Temperature (WDC)	$<$	no	Airflow temperature on Western Digital HDs. Marked as obsolete
190	BE	Temperature Difference from 100	$>$	no	Value is equal to (100 – temp. °C), allowing manufacturer to set a minimum threshold which corresponds to a maximum temperature
191	BF	G-sense error rate	$<$	no	Frequency of mistakes as a result of impact loads
192	C0	Power-off Retract Count	$<$	no	Number of times the heads are loaded off the media. Heads can be unloaded without actually powering off (or Emergency Retract Cycle count – Fujitsu)
193	C1	Load/Unload Cycle Count	$<$	no	Count of load/unload cycles into head landing zone position
194	C2	Temperature	$<$	no	Current internal temperature
195	C3	Hardware ECC Recovered	$>$	no	Time between ECC-corrected errors or number of ECC on-the-fly errors
196	C4	Reallocation Event Count	$<$	yes	Count of remap operations. The raw value of this attribute shows the total number of attempts to transfer data from reallocated sectors to a spare area. Both successful and unsuccessful attempts are counted
197	C5	Current Pending Sector Count	$<$	yes	Number of "unstable" sectors (waiting to be remapped). If the unstable sector is subsequently written or read successfully, this value is decreased and the sector is not remapped. Read errors on the sector will not remap the sector, it will only be remapped on a failed write attempt. This can be problematic to test because cached writes will not remap the sector, only direct I/O writes to the disk
198	C6	Uncorrectable Sector Count	$<$	yes	The total number of uncorrectable errors when reading/writing a sector. A rise in the value of this attribute indicates defects of the disk surface and/or problems in the mechanical subsystem (or Off-Line Scan Uncorrectable Sector Count – Fujitsu)
199	C7	UltraDMA CRC Error Count	$<$	no	The number of errors in data transfer via the interface cable as determined by ICRC (Interface Cyclic Redundancy Check)
200	C8	Write Error Rate/Multi-Zone Error Rate	$<$	no	The total number of errors when writing a sector
201	C9	Soft Read Error Rate	$<$	yes	Number of off-track errors
202	CA	Data Address Mark errors	$<$	no	Number of Data Address Mark errors (or vendor-specific)

Table 2.3: Smart attributes and their meaning

ID	Hex	Attribute name	Better[5]	Critical[6]	Description
203	CB	Run Out Cancel	<	no	Number of ECC errors
204	CC	Soft ECC Correction	<	no	Number of errors corrected by software ECC
205	CD	Thermal Asperity Rate (TAR)	<	no	Number of errors due to high temperaure
206	CE	Flying Height		no	Height of heads above the disk surface. A flying height that's too low increases the chances of a head crash while a flying height that is too high increases the chances of a read/write error
207	CF	Spin High Current	<	no	Amount of surge current used to spin up the drive
208	D0	Spin Buzz		no	Number of buzz routines needed to spin up the drive due to insufficient power
209	D1	Offline Seek Performance		no	Drive's seek performance during its internal tests
211	D3	Vibration During Write		no	Vibration During Write
212	D4	Shock During Write		no	Shock During Write
220	DC	Disk Shift	<	no	Distance the disk has shifted relative to the spindle (usually due to shock or temperature). Unit of measure is unknown
221	DD	G-Sense Error Rate	<	no	The number of errors resulting from externally-induced shock and vibration
222	DE	Loaded Hours		no	Time spent operating under data load (movement of magnetic head armature)
223	DF	Load/Unload Retry Count		no	Number of times head changes position
224	E0	Load Friction	<	no	Resistance caused by friction in mechanical parts while operating
225	E1	Load/Unload Cycle Count	<	no	Total number of load cycles
226	E2	Load 'In'-time		no	Total time of loading on the magnetic heads actuator (time not spent in parking area)
227	E3	Torque Amplification Count	<	no	Number of attempts to compensate for platter speed variations
228	E4	Power-Off Retract Cycle	<	no	The number of times the magnetic armature was retracted automatically as a result of cutting power
230	E6	GMR Head Amplitude		no	Amplitude of "thrashing" (distance of repetitive forward/reverse head motion)
231	E7	Temperature	<	no	Drive Temperature
240	F0	Head Flying Hours		no	Time while head is positioning
250	FA	Read Error Retry Rate	<	no	Number of errors while reading from a disk
254	FE	Free Fall Protection	<	no	Number of "Free Fall Events" detected

Linux boxes usually have the command 'smartctl' installed, which allows to grab information from the vendor specific attributes provided by SMART. The basic commands are list in table ??.

Table 2.4: Smartctl command examples(4; 5)

Command	**Description**
smartctl -i /dev/sda	Disk model/firmware information
smartctl -Hc /dev/sda	Status results. If the disk status is failing, save your data in another one
smartctl -A /dev/sda	Prints only the vendor specific SMART Attributes
smartctl -l error /dev/sda	Shows whether the errors are recent or old
smartctl -l selftest /dev/sda	Report of the self-tests run on the disk

2.1.2 Lm_Sensors: Hardware monitoring tool

This Linux monitoring tool to acquire data on the environmental conditions of computers. The tool obtain information from the speed of fans, system voltages, temperature, fan speed control. The available features depend on the specific hardware manufacturer and configuration. The user may install the package and then proceed loading the necessary modules using the command 'sensors-detect'. Afterwards, one may call the command line 'sensors' and acquire hardware data.

2.1.3 Vmstat: Linux monitoring tool

Vmstat is a Linux tool to monitor system utilization. It reports the information presented in table 2.5. One may run vmstat to peridiocally obtain system information and use it for autonomic purposes. The autonomic system design can run Vmstat by typing 'vmstat' at the Linux command line or periodically run it such as in 'vmstat 1' (runs every second).

Table 2.5: Vmstat information

Processes	
r	The number of processes waiting for run time
b	The number of processes in uninterruptible sleep
Memory	
swpd	the amount of virtual memory used
free	the amount of idle memory
buff	the amount of memory used as buffers
cache	the amount of memory used as cache
inact	the amount of inactive memory (-a option)
active	the amount of active memory (-a option)
Swap	
si	Amount of memory swapped in from disk (/s)
so	Amount of memory swapped to disk (/s)
IO	
bi	Blocks received from a block device (blocks/s)
bo	Blocks sent to a block device (blocks/s)
System	
in	The number of interrupts per second, including the clock
cs	The number of context switches per second
CPU – in percentage of total CPU time	
us	Time spent running non-kernel code (user time, including nice time)
sy	Time spent running kernel code (system time)
id	Time spent idle. Prior to Linux 2.5.41, this includes IO-wait time
wa	Time spent waiting for IO. Prior to Linux 2.5.41, included in idle

2.1.4 Tcpdump: Network monitoring tool

Tcpdump is a tool to monitor network traffic which listens packets on the network interface card (56). It provides a description of packets as well as allows to observe what is inside them. The most common ways of executing Tcpdump are presented in table 2.6. There are other complex ways of using Tcpdump, where it helps to evaluate packet headers and other attributes. It output format varies, depending on the typed command.

Table 2.6: Tcpdump command examples

Command	Description
tcpdump -i eth0	Listens all packets on the network interface card eth0
tcpdump -A -i eth0	Listens and prints the packets out in ASCII format
tcpdump -D	Informs all interfaces which can be monitored
tcpdump -X dst port 80	Listens packets with destination as port 80 and prints their information in ASCII

2.1.5 DLSym: Library interception

A typical Linux program is compiled with the dynamic shared libraries support. When the program calls a function, instead of having the code internally, the program loads a specific shared library and runs the function. This is an usual approach to avoid rewritting functions and provide software reuse.

DLSym is a Linux library which allows to intercept calls to those functions. Consequently, any function in a dynamic shared library can be intercepted, what helps to build monitoring tools.

An example of DLSym code is presented in table 2.7. According to this example, the line *void __attribute__ ((constructor)) _my_init(void)* makes the function *_my_init(void)* be called when any Linux program is launched. This helps to set up some features before executing any program. In this case, we only print a message out.

The main line in *_my_init(void)* is *_sin = dlsym(lib_handle, "sin")*, which loads the original *sin* from the shared library and attributes it to the function pointer *_sin*. This pointer is then used when calling the original function.

The function *double sin(double value)* intercepts any program that calls *sin* and is dynamically linked. This function will print a message and, then, call the original function. Instead of printing a message, we could log any information about it, including the exeuction timestamp, the function parameters and also observe other system variables. This brings the possibility to get the information when a call is executed, what is more precise than monitoring.

The presented example may be compiled, using GCC, as *gcc -D_DEBUG -D_GNU_SOURCE -shared -o file.so file.c -ldl -lc.* Then, you must load the *file.so* using the environment variable *LD_PRELOAD* using the bash command *export LD_PRELOAD=/path/file.so.* Afterwards, any command you launch will show the message printed inside *_my_init(void)* before effectively executing it and any call to *sin* will be intercepted. To cancel the interception you may execute the bash command *unset LD_PRELOAD.*

Table 2.7: DLSym example

```
#include <sys/mman.h>
#include <sys/param.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
#include <syslog.h>
#include <errno.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <sched.h>
#include <dlfcn.h>
#include <sys/socket.h>

#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#include <stdio.h>
#include <sys/time.h>
#include <sys/resource.h>

double (*_sin)(double);

/* This makes the function _my_init be called when starting the program */
void __attribute__ ((constructor)) _my_init(void);

/* Library initialization */
void _my_init(void)
{
        void *lib_handle;
        char buffer[1024];
        char hostname[12];
        int fd;

        lib_handle = RTLD_NEXT;
        if (!lib_handle)
        {
                if (DEBUG)
                        fprintf(stderr, "Opening lib: %s", dlerror());
                exit(1);
        }

        printf("Starting...");

        _sin = dlsym(lib_handle, "sin");
}

double sin(double value) {
        printf("this is another sin function");
        return _sin(value);
}
```

2.1.6 Ptrace: System call interception

Another common program to intercept process signals and system calls is Strace. Strace is very useful to make diagnosis and to debug applications. Sometimes the developers and administrators do not know when certain file is opened, if the right configuration file was loaded, which part of the code is failing. Strace can help to figure out what and when those issues are happening. There are two tipical ways of executing it, the first is by launching it together with the application, such as 'strace ls'. If the program is already running, we can intercept each one of its processes using 'strace -p <PID>', where <PID> is the process id.

Strace is based on the Linux Ptrace function which allows a parent process to intercept its children. In order to make your on interceptor, you may add the following line in your C program '#include <sys/ptrace.h>' and call the function which has the header 'long ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *data)', where the value of request defines the action to be taken, pid defines the process to be intercepted, addr and data are memory pointers which depend on the type of request.

The operation of ptrace is based on the fact that every process makes Linux system calls such as for writing of a file, socket, pipe, etc.: *write(2, "Hello", 5).* When the process calls for a write operation, it basically executes the code in table 2.8. We observe that we need to specify the operation in registers. The type of system call is defined in register eax, the file descriptor is stored in ebx, the memory address in ecx and the number of bytes to be written in edx. This follows a standard and all those parameters can be retrieved. For instance, if we intercepted a write call, we can obtain the file descriptor (ebx), the memory address (ecx) and the number of bytes (edx).

Table 2.8: Assembly code

movl $4, %eax
movl $2, %ebx
movl $hello,%ecx
movl $5, %edx
int $0x80

Table 2.9 presents a sample code to intercept some system calls: open, close, read and write. As in the example using the system call write, we can also get the register values for other system calls. We only need to know what they represent.

2.2 Analysis

The analysis phase considers the four different aspects of autonomic systems. Details on those aspects are presented as follows.

Table 2.9: DLSym example

```
#include <sys/ptrace.h>
#include <sys/types.h>
#include <signal.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/reg.h>
#include <sys/syscall.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <sys/user.h>

void process_syscall (long syscall, int pid)
{
        struct user_regs_struct regs;
        switch(syscall)
        {
                case SYS_open:
                        ptrace(PTRACE_GETREGS, pid, NULL, &regs);
                        printf("open %d\n", (int) regs.eax);
                        break;
                case SYS_read:
                        ptrace(PTRACE_GETREGS, pid, NULL, &regs);
                        printf("read\n");
                        break;
                case SYS_write:
                        ptrace(PTRACE_GETREGS, pid, NULL, &regs);
                        printf("write\n");
                        break;
                case SYS_close:
                        ptrace(PTRACE_GETREGS, pid, NULL, &regs);
                        printf("close\n");
                        break;
                default: break;
        }
}

int main(int argc, char **argv, char **env)
{
        int pid;
        unsigned int val;
        if (argc < 2)
        {
                        fprintf(stderr, "Usage: PID %d\n", argv[0]);
                exit(1);
        }
        pid = atoi(argv[1]);
        printf("Attaching to process: %d\n", pid);

        /* Attach this process to the one to be intercepted */
        ptrace(PTRACE_ATTACH, pid, NULL, NULL);

        while(waitpid(pid, NULL, 0) >= 0)
        {
                long orig_eax;
                val = orig_eax = ptrace(PTRACE_PEEKUSER, pid, 4 * ORIG_EAX, NULL);
                process_syscall(orig_eax, pid);
                val = ptrace(PTRACE_SYSCALL, pid, NULL, NULL);
        }
        return 0;
}
```

2.2.1 Self-Optimization

People frequently face situations where they should or must find the best solution for a problem. Consider, for instance, a company which needs to deal with its logistics. It is responsible for delivering a set of items $\{x_0, x_1, \ldots, x_{n-1}\}$ to the customers $\{c_0, c_1, \ldots, c_{k-1}\}$, where $k \leq n$. The employees might randomly go to the k customers, what may not be the most optimized solution, and, consequently, the company will spend more resources and time. Such a company would be out of the current market.

It should find the best solution, what, in this situation, would be to trace the best path for delivering. Such path would reduce the time and resources consumed (e.g. gas, tires, etc.).

For better understanding the problem, consider an instance where $k = 3$ and $n = 3$. The optimum path would be found after analysing all the possible combinations. In this case, the combinations would be $\{\{c_0, c_1, c_2\}, \{c_0, c_2, c_1\}, \{c_1, c_0, c_2\}, \{c_1, c_2, c_0\}, \{c_2, c_0, c_1\}, \{c_2, c_1, c_0\}\}$ which totalize $k!$ options. Another important question is to avoid the re-evaluation of the same combination, simply because it would increase the cost to find the best path.

Now consider that, for each combination, you must analyse the cost to follow that path. In this case, the cost could be expressed in terms of gas what is relative to the distance. This is, summing up the distances in between deliver points the company would have a very good approximation of the total cost. As the distance increases, the time will probably also increases and the objective of reducing the deliver time would also be addressed.

Consequently, the function $f(c)$ (equation 2.2) would be adopted, which considers the quality or fitness of a possible solution (or combination) c. The term $k = |c|$ represents the size of a solution, $D(c_j, c_w)$ is a function to determine the distance from the source c_j to the destination c_w.

$$f(c) = \sum_{i=0}^{|c|-2} D(c_i, c_{i+1}) \tag{2.1}$$

This simple problem can be extended to consider the initial and final destinations. For instance, consider that the employees must always leave from an initial place, perhaps the company, and arrive at a destination, also the company. In such situation, the function $f(c)$ would add up the cost for leaving the company to the first delivery place and also consider the way back (from the last visited place). This situation would lead to the function presented in equation **??**.

$$f(c) = D(company, c_0) + \sum_{i=0}^{|c|-2} D(c_i, c_{i+1}) + D(c_{|c|-1}, company) \tag{2.2}$$

Such situation could also be modeled as combinations, instead of changing the fitness function. This would create a population such as $\{\{company, c_0, c_1, c_2, company\}, \{company, c_0, c_2, c_1, company\}, \{company, c_1, c_0, c_2, company\}, \{company, c_1, c_2, c_0, company\}, \{company, c_2, c_0, c_1, company\},$

$\{company, c_2, c_1, c_0, company\}\}$. Therefore, the population could be analysed by using equation 2.2, instead.

Now consider the cost to evaluate each solution (or combination). Applying equation **??** on combination $\{company, c_0, c_1, c_2, company\}$, the index i would vary from $0, \ldots, |c| - 2$, what would result in $|c| - 1$, or $k - 1$, operations. This means that, for every candidate solution, $|c| - 1$ distances are computed. Now consider the distance is found on a matrix such as the one presented in table 2.10 and the cost to obtain an element is only a memory access.

Table 2.10: Distances between delivery places

Source/Destination	**company**	c_0	c_1	c_2
company	0.00	12.05	56.12	32.12
c_0	12.05	0.00	87.56	45.21
c_1	56.12	87.56	0.00	6.83
c_2	32.12	45.21	6.83	0.00

In the presented situation, the cost to evaluate one candidate solution would be $|c| - 1$ operations. The evaluation of all the $k!$ would cost $k! \cdot (|c| - 1)$ operations what is similar to $(k - 1) \cdot k!$. Now, consider computers with capacities in terms of million of operations per second and different problem instances. The time costs to find the optimal solutions are presented in table 2.11[7] (where PI and CC stand for problem instance and computer capacity, respectively).

Table 2.11: Cost to evaluate all possible solutions

PI (k)/**CC**	$1,000$	$5,000$	$10,000$	$50,000$	$100,000$	$20 \cdot 10^9$
$k = 20,\ 4 \cdot 10^{19}$	$4 \cdot 10^{10}$ s	$9 \cdot 10^9$ s	$4 \cdot 10^9$ s	$9 \cdot 10^8$ s	$4 \cdot 10^8$ s	$2 \cdot 10^3$ s
$k = 30,\ 7 \cdot 10^{33}$	$1 \cdot 10^{23}$ m	$2 \cdot 10^{22}$ m	$1 \cdot 10^{22}$ m	$2 \cdot 10^{21}$ m	$1 \cdot 10^{21}$ m	$6 \cdot 10^{15}$ m
$k = 40,\ 3 \cdot 10^{49}$	$8 \cdot 10^{36}$ h	$1 \cdot 10^{36}$ h	$8 \cdot 10^{35}$ h	$1 \cdot 10^{35}$ h	$8 \cdot 10^{34}$ h	$4 \cdot 10^{29}$ h
$k = 50,\ 1 \cdot 10^{66}$	$4 \cdot 10^{49}$ y	$9 \cdot 10^{48}$ y	$4 \cdot 10^{48}$ y	$9 \cdot 10^{47}$ y	$4 \cdot 10^{47}$ y	$2 \cdot 10^{42}$ y
$k = 60,\ 4 \cdot 10^{83}$	$1 \cdot 10^{66}$ d	$3 \cdot 10^{65}$ d	$1 \cdot 10^{65}$ d	$3 \cdot 10^{64}$ d	$1 \cdot 10^{64}$ d	$7 \cdot 10^{58}$ d
$k = 70,\ 8 \cdot 10^{101}$	10^{84} c	10^{84} c	10^{83} c	10^{83} c	10^{82} c	10^{77} c

Table 2.12 presents the performance of some processors as well as a pencil a paper approach (**?**). We may notice that the capapcity considered in the previous example is high and even considering such circumstances the problem is not easily solvable yet. Here, assume that solving a problem means to have an algorithm which finds out the best solution at all. In the presented problem, the solution would be the sorthest path to deliver products at different spots.

After this example, it might be clear how hard is to solve such visiting problem. Another point to mention is that the nowadays logistic companies might consider more than 20 places to visit, what confirms the importance of tackling this problem. The question here is how to solve it in acceptable time. Do not forget that the meaning of

[7]The measurements are expressed in seconds (s), minutes (m), hours (h), years (y), decades (d) and centuries (c).

Table 2.12:

Processor	IPS	Year
Pencil and paper (for comparison)	0.0119 IPS	1892
Intel 4004	92 kIPS at 740 kHz	1971
IBM System/370 model 158-3	1 MIPS	1972
Intel 8080	640 kIPS at 2 MHz	1974
VAX-11/780	500 kIPS	1977
Motorola 68000	1 MIPS at 8 MHz	1979
Intel 286	2.66 MIPS at 12 MHz	1982
Motorola 68020	4 MIPS at 20 MHz	1984
ARM2	4 MIPS at 8 MHz	1986
Motorola 68030	11 MIPS at 33 MHz	1987
Intel 386DX	8.5 MIPS at 25 MHz	1988
Motorola 68040	44 MIPS at 40 MHz	1990
Intel 486DX	54 MIPS at 66 MHz	1992
Motorola 68060	88 MIPS at 66 MHz	1994
Intel Pentium Pro	541 MIPS at 200 MHz	1996
ARM 7500FE	35.9 MIPS at 40 MHz	1996
PowerPC G3	525 MIPS at 233 MHz	1997
Zilog eZ80	80 MIPS at 50 MHz	1999
Intel Pentium III	1,354 MIPS at 500 MHz	1999
Freescale MPC8272	760 MIPS at 400 MHz	2000
AMD Athlon	3,561 MIPS at 1.2 GHz	2000
AMD Athlon XP 2400+	5,935 MIPS at 2.0 GHz	2002
Pentium 4 Extreme Edition	9,726 MIPS at 3.2 GHz	2003
ARM Cortex A8	2,000 MIPS at 1.0 GHz	2005
AMD Athlon FX-57	12,000 MIPS at 2.8 GHz	2005
AMD Athlon 64 3800+ X2 (Dual Core)	14,564 MIPS at 2.0 GHz	2005
Xbox360 IBM "Xenon" Triple Core	19,200 MIPS at 3.2 GHz	2005
PS3 Cell BE (PPE only)	10,240 MIPS at 3.2 GHz	2006
AMD Athlon FX-60 (Dual Core)	18,938 MIPS at 2.6 GHz	2006
Intel Core 2 Extreme X6800	27,079 MIPS at 2.93 GHz	2006
Intel Core 2 Extreme QX6700	49,161 MIPS at 2.66 GHz	2006
P.A. Semi PA6T-1682M	8,800 MIPS at 2.0 GHz	2007
Intel Core 2 Extreme QX9770	59,455 MIPS at 3.2 GHz	2008
Intel Core i7 Extreme 965EE	76,383 MIPS at 3.2 GHz	2008
AMD Phenom II X4 940 Black Edition	42,820 MIPS at 3.0 GHz	2009

"solve" is to find out the best solution. However, it is important to mention that some problems have been proved to be so hard to solve that they are considered as a special class. Eventually, algorithm designers may face those problems and prove those problems are too hard to be solved or that they are as hard, i.e. similar, as existent proven-hard problems.

The meaning of hard here is related to the time complexity function of the problem. Such a function usually expresses the maximum time to find the solution. Those problems are addressed by algorithms which present different time complexities. They can be fairly efficient or too ineficient. The distinction of those approaches lays on the fact that the algorithm has a polynomial or exponential time complexity.

Polynomial algorithms are characterized by a time complexity function $O(p(n))$, where n is the input length, this is, the input parameters for the algorithm (such as the number of places and the distance matrix presented before), $p(.)$ is a polynomial-bounded function and $O(.)$ represents the time complexity order. A non-polynomial-bounded algorithm is frequently regarded as an exponential time complexity algorithm (43). This class consumes lots of resources and may not be feasible to solve problems, that is why designers say that a problem is not "well-solved" until a polynomial time algorithm is found. Those problems are also called intractable.

Although exponential algorithms are not desired, they are used in some circumstances. In the previous visiting problem, there will not be any throuble to solve it for $k = 5$ places. This confirms that even exponential approaches are useful and considered for small scale instances. However, they start becoming unacceptable for larger situations.

In fact, there are two classes of intractable problems. The first includes undecidable problems which are proved to not have an answer. Algorithms cannot be found to solve those questions. For instance, Alan Turing studied the problem where given a program and a finite input, he tried to figure out if the program finishes running or runs forever for that input. This problem, known as the halting problem, states that there is no algorithm which always completes and always answers a problem correctly.

This class of intractable problems is not considered here. The second class includes decidable intractable problems. Those problems cannot be solved by polynomial algorithms even considering a nondeterministic computer model which is capable of executing as many as necessary instructions in parallel.

The complexity of problems is approached by Garey & Johnson (43) who state the definition of two stages: guessing and checking. The guessing proposes a solution for a problem instance and the validity checking analyses whether the answer for proposed solution is yes or no. The verification of a solution in polynomial time does not imply it can be solved in polynomial time.

Let's consider the visiting problem presented before. One may guess a visiting order and use a verification procedure to analyse the validity of the path for a question such as 'Is there a path shorter than T?'. Although, the problem carry on having many possible solutions for an instance, the validity checking for only one candidate solution can verified in polynomial time.

Let's consider another problem. Which two numbers are multiplied to result $18,980$? You may take a lot of time to prove which numbers are, but if someone tells you the numbers are 365 and 52 you can easily and fastly verify it. The question here is: Is there another way around to fastly and easier find the answer or we need to guess numbers and verify in polinomial time? This question is still opened in computer science and a 1 million dollars prize is offered by the Clay Mathematics Institute (89). If someone proves there is a way around, so any problem could be solved in polynomial time, what would be a very strong advance for computer science and related areas. The negative proof could also evolve the areas and confirm research proposals.

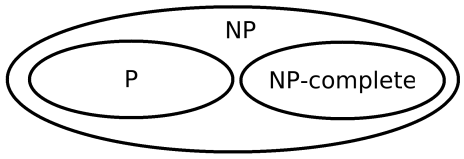


Figure 2.1: NP, P and NP-complete problems

Problem guesses which are verifiable in polynomial time are considered to be in class `NP` (which stands for Nondeterministic Polynomial time). A class of problems inside `NP` are the ones which are proved to be solvable in polynomial time and not just checked. Those problems compose the class `P`. There is still another class contained in `NP` which is called `NP-complete`. The `NP-complete` problems are considered the toughest ones to be solved, because they are probably not in `P`.

The first problem to be proved to be in the `NP-complete` set was the Boolean logic Satisfiability one (43). Other problems came up after, but the basic premise is that they are, in some way, reducible to one another. This leads researchers to prove that problems are very hard to solve by reducing them to `NP-complete` ones, what is the usual way of proving time complexity. At the same time, if someone finds the polynomial time solution for one of those problems, all the others are also solvable in polynomial time, what would guarantee that instead of $\mathtt{P} \neq \mathtt{NP}$, both classes would be equal as well as the `NP-complete` one, this is $\mathtt{P} = \mathtt{NP} = \mathtt{NP\text{-}complete}$.

Those difficult problems are addressed by using exact or approximation approaches. Exact methods explore and evaluate all the possibilities to provide the best answer. They can be very time consuming, depending on the problem and its instance. For small instances, they might offer acceptable time complexity. However, for large instances, such as to the visiting problem, they are prohibitive. Those large instances are usually addressed by approximation algorithms which consider a trade off between the solution quality and the time complexity. They attempt to find good or fair solutions (what sounds very subjective), at an acceptable time constraint (what is also subjective).

The approximation algorithms can be classified into the construtive and the local search approaches. The first generates solutions from scratch. The second starts with an initial solution and makes small perturbations on it, until converging to an aceptable one.

Approximation algorithms are based on heuristics. Heuristics produces acceptable solutions for practical problems without any proof of correctness. They result in higher performance approaches, however they do not guarantee the solution quality. A generalization of a heuristics is the metaheuristics. Metaheuristics is a heuristics method which approaches a class of problems. Metaheuristics are more like patterns to address different problems.

Different methods are classified as metaheuristics such as Hill Climbing, Greedy Search, Tabu Search, Simulated Annealing, Genetic Algorithms and Ant Colony Optimization. They propose different approaches to find good solutions for very complex problems.

Trying the guess and check approach

The visiting problem: unknown cost function

The first approach to the guess-and-check algorithm is presented in table 2.13. It randomly generates the distance matrix and the candidate solutions. After proposing each candidate, the function *valid* is called to evaluate if the solution restrictions are respected. The basic restriction is that each customer must be visited only once and every customer has to be visited.

To better understand how we explore the solution space, let's consider figure 2.2. We propose arbitrary solutions and, afterwards, evaluate them in order to know the cost, in terms of the total travelled distance. This implies that we do not know a function to describe the travelled distance.

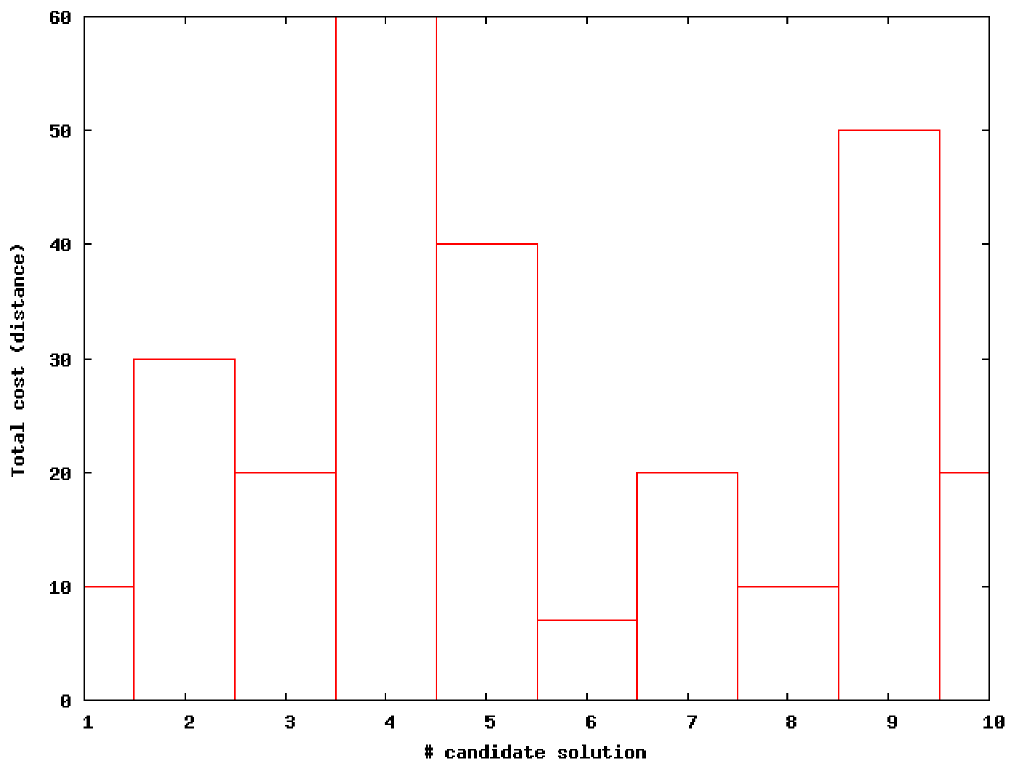


Figure 2.2: Cost for candidate solutions

After the first approach, we proposed the code in table 2.14 which proposes an initial candidate solution and, then, we make perturbations on it to propose new candidates. This approach consumes less processing time, as it does not need to be validated on each iteration.

Evolving the code, we present a new approach in table 2.15 which does not require the function *valid*, because the initial solution is proposed using the allowed values (customer identifications). Then, afterwards, they are evaluated and modified to propose new solutions.

Table 2.13: First approach to the visiting problem

```
#include <stdlib.h>
#include <stdio.h>

#define MAX 10000
#define LIMIT 20

long int m[LIMIT][LIMIT];
int candidate[LIMIT];
int isset[LIMIT];

int valid(int *candidate) {
        int i;

        for (i = 0; i < LIMIT; i++) {
                isset[i] = 0;
        }

        for (i = 0; i < LIMIT; i++) {
                if (isset[candidate[i]] > 0)
                        return 0;
                else
                         isset[candidate[i]]++;
        }

        return 1;
}

int main(int argc, char *argv[]) {
        int i, j;
        double sum;
        long int r;

        srandom(time(NULL));

        // Generating the matrix
        // if x,y == 0 -> not valid -> matrix
        for (i = 0; i < LIMIT; i++) {
                for (j = 0; j < LIMIT; j++) {
                        while ((r = random()) == 0);
                        m[i][j] = r;
                }
        }

        // Guessing
        for (i = 0; i < MAX; i++) {
                // Guessing.. effectively
                do {
                        for (j = 0; j < LIMIT; j++)
                                candidate[j] = random() % LIMIT;
                } while (!valid(candidate));

                sum = 0.0;
                // evaluate it
                for (j = 0; j < LIMIT-1; j++) {
                        sum += m[candidate[j]][candidate[j+1]];
                }

                printf("Distance: %f\n", sum);
        }

        return 0;
}
```

The code was still improved (table 2.16) to propose solutions better than a threshold T. This approach does not impose limits to the number of guesses and checks, on the other hand, it proposes and evalutes until reaching a good solution.

After guessing and proposing good solutions, we might face the situation where a very

Table 2.14: Second approach to the visiting problem

```
#include <stdlib.h>
#include <stdio.h>
#define MAX 10000
#define LIMIT 20
#define CHANGES 1

long int m[LIMIT][LIMIT];
int candidate[LIMIT];
int isset[LIMIT];

int valid(int *candidate) {
        int i;

        for (i = 0; i < LIMIT; i++)
                isset[i] = 0;

        for (i = 0; i < LIMIT; i++) {
                if (isset[candidate[i]] > 0)
                        return 0;
                else
                         isset[candidate[i]]++;
        }

        return 1;
}

int main(int argc, char *argv[]) {
        int i, j, x, y;
        double sum;
        long int r, aux;

        srandom(time(NULL));
        // Generating the matrix
        for (i = 0; i < LIMIT; i++) {
                for (j = 0; j < LIMIT; j++) {
                        while ((r = random()) == 0);
                        m[i][j] = r;
                }
        }

        do {
                for (j = 0; j < LIMIT; j++)
                        candidate[j] = random() % LIMIT;
        } while (!valid(candidate));

        // Guessing
        for (i = 0; i < MAX; i++) {
                sum = 0.0;
                // evaluate it
                for (j = 0; j < LIMIT-1; j++) {
                        sum += m[candidate[j]][candidate[j+1]];
                }

                printf("Distance: %f\n", sum);

                for (j = 0; j < CHANGES; j++) {
                        x = random() % LIMIT;
                        y = random() % LIMIT;

                        aux = candidate[x];
                        candidate[x] = candidate[y];
                        candidate[y] = aux;
                }
        }

        return 0;
}
```

Table 2.15: Third approach to the visiting problem

```
#include <stdlib.h>
#include <stdio.h>

#define MAX 10000
#define LIMIT 70
#define CHANGES 50

long int m[LIMIT][LIMIT];
int candidate[LIMIT];
int isset[LIMIT];

int main(int argc, char *argv[]) {
        int i, j, x, y;
        double sum;
        long int r, aux;

        srandom(time(NULL));

        // Generating the matrix
        // if x,y == 0 -> not valid -> matrix
        for (i = 0; i < LIMIT; i++) {
                for (j = 0; j < LIMIT; j++) {
                        while ((r = random()) == 0);
                        m[i][j] = r;
                }
        }

        // Guessing.. effectively
        for (j = 0; j < LIMIT; j++)
                candidate[j] = j;

        // Guessing
        for (i = 0; i < MAX; i++) {
                sum = 0.0;
                // evaluate it
                for (j = 0; j < LIMIT-1; j++) {
                        sum += m[candidate[j]][candidate[j+1]];
                }

                printf("Distance: %f\n", sum);

                // Changing
                for (j = 0; j < CHANGES; j++) {
                        x = random() % LIMIT;
                        y = random() % LIMIT;

                        aux = candidate[x];
                        candidate[x] = candidate[y];
                        candidate[y] = aux;
                }
        }

        return 0;
}
```

good solution has appeared during iterations. The other approaches do not save the best candidate for last. The next approach (code in table 2.17) saves and presents it at the end.

The visiting problem: known cost function

Last section has presented an optimization approach based on an unknown cost function. In such circumstances, you propose candidate solutions without any idea about how the cost function works. In this section, we consider a known function to represent the

Table 2.16: Fourth approach to the visiting problem

```
#include <stdlib.h>
#include <stdio.h>

#define LIMIT 70
#define CHANGES 2

double T = 52000000000;
long int m[LIMIT][LIMIT];
int candidate[LIMIT];
int isset[LIMIT];

int main(int argc, char *argv[]) {
        int i, j, x, y;
        double sum;
        long int r, aux;

        srandom(time(NULL));

        // Generating the matrix
        // if x,y == 0 -> not valid -> matrix
        for (i = 0; i < LIMIT; i++) {
                for (j = 0; j < LIMIT; j++) {
                        while ((r = random()) == 0);
                        m[i][j] = r;
                }
        }

        // Guessing... effectively
        for (j = 0; j < LIMIT; j++)
                candidate[j] = j;

        // Guessing
        do {
                sum = 0.0;
                // evaluate it
                for (j = 0; j < LIMIT-1; j++) {
                        sum += m[candidate[j]][candidate[j+1]];
                }

                printf("Distance: %f\n", sum);

                // Changing
                for (j = 0; j < CHANGES; j++) {
                        x = random() % LIMIT;
                        y = random() % LIMIT;

                        aux = candidate[x];
                        candidate[x] = candidate[y];
                        candidate[y] = aux;
                }
        } while (sum > T);

        return 0;
}
```

cost function.

Assume you ordered, in some way, all possible solutions. Let each value of axis x in $\mathbb{Z}$ represent a candidate solution and the plot of figure 2.3 the continuous version for the cost function. Consider we have an initial solution $i \in \mathbb{Z}$ where $i = 1$. We could use the gradient descent to walk on the curve and find a local or global minimum. In order to do that, we made the regression of the curve in figure 2.3 and obtained the equation 2.3. Afterwards, to compute the gradient descent, we need to find the derivative for it, which is presented in equation 2.4. Having an initial point in x-axis and this derivative, we can

Table 2.17: Fifth approach to the visiting problem

```
#include <stdlib.h>
#include <stdio.h>
#include <values.h>

#define LIMIT 10
#define CHANGES 2

long int m[LIMIT][LIMIT];
int candidate[LIMIT];
int isset[LIMIT];
double bestDistance = MAXDOUBLE;
int bestSolution[LIMIT];

int main(int argc, char *argv[]) {
        int i, j, x, y, generation;
        double sum;
        long int r, aux;

        srandom(time(NULL));

        // Generating the matrix
        // if x,y == 0 -> not valid -> matrix
        for (i = 0; i < LIMIT; i++) {
                for (j = 0; j < LIMIT; j++) {
                        while ((r = random()) == 0);
                        m[i][j] = r;
                }
        }

        // Guessing... effectively
        for (j = 0; j < LIMIT; j++)
                candidate[j] = j;

        // Guessing
        for (generation = 0; generation < 1000; generation++) {
                sum = 0.0;
                // evaluate it
                for (j = 0; j < LIMIT-1; j++) {
                        sum += m[candidate[j]][candidate[j+1]];
                }

                if (sum < bestDistance) {
                        bestDistance = sum;
                        for (j = 0; j < LIMIT-1; j++)
                                bestSolution[j] = candidate[j];
                }

                printf("New distance: %f\n", sum);

                // Changing
                for (j = 0; j < CHANGES; j++) {
                        x = random() % LIMIT;
                        y = random() % LIMIT;

                        aux = candidate[x];
                        candidate[x] = candidate[y];
                        candidate[y] = aux;
                }
        }

        printf("The best solution has distance %f and the following order:\n", bestDistance);

        for (j = 0; j < LIMIT-1; j++)
                printf("%d\n", bestSolution[j]);

        return 0;
}
```

walk down those curve regions and find a minimimum (code in table 2.18).

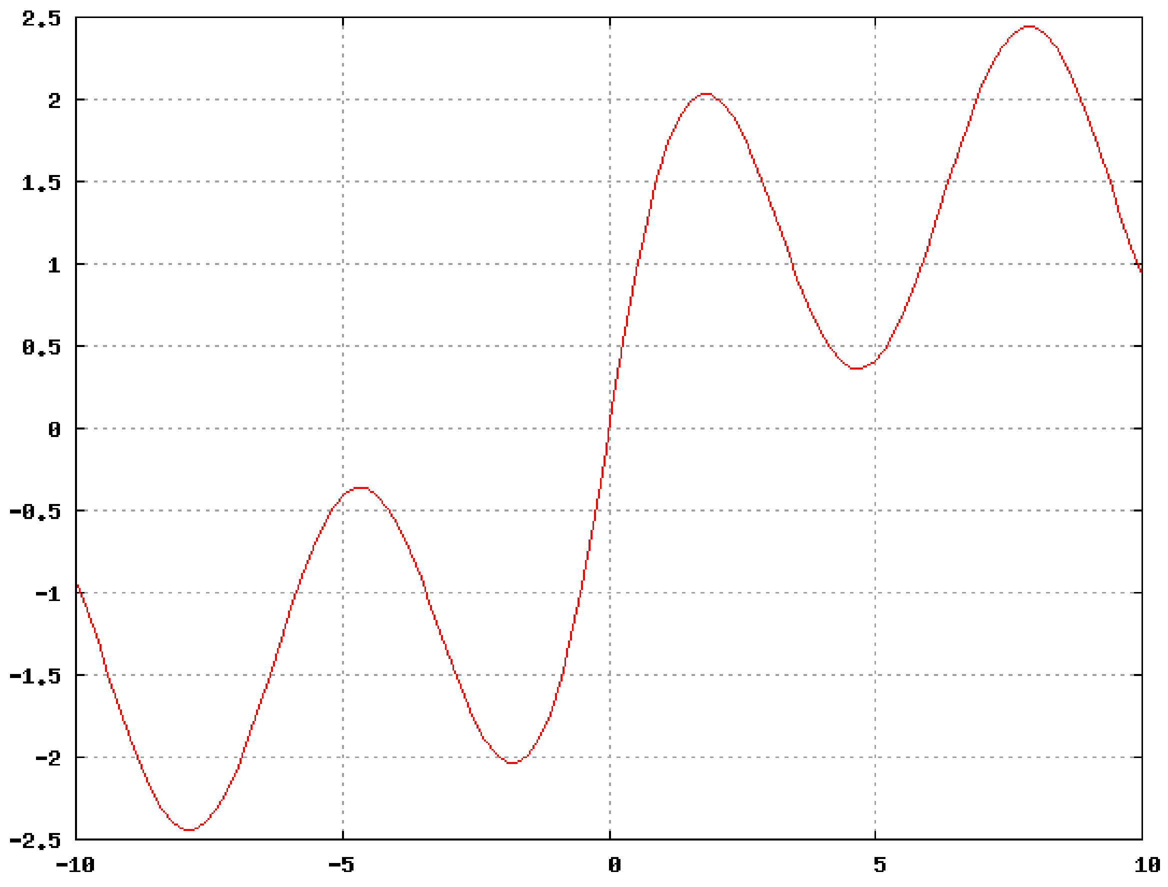


Figure 2.3: Ordered solutions

$$f(x) = \arctan x + \sin x \tag{2.3}$$

$$f(x) = \frac{1.0}{1.0 + x^2} + \cos x \tag{2.4}$$

This approach requires to have the cost function, what is not always available. Another problem is that it gives a minimum according to the initial point in x-axis. In this case, if we start with 1.0, we will tend to $x = -1.80$. Starting with 7.0, we tend to $x = 4.67$. This implies that even knowing the function, we have to select an initial candidate and, then, walk down to find a good solution.

Another approach would be by guessing the initial candidate and, then, call the gradient descent to find a good solution. The code from table **??** presents such approach. It uses guesses for the initial candidate and keeps the best solution to present at the end.

Table 2.18: Gradient descent

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

/* f(x) = atan(x)+sin(x) */
/* f'(x) = 1/(1+x**2) + cos x */

int main(int argc, char *argv[]) {

        if (argc != 2) {
                printf("usage: %s x\n", argv[0]);
                exit(0);
        }

        double xNew = atof(argv[1]);
        double xOld = xNew + 1;
        double eps = 0.01; // step size
        double derivative = 0.0;
        double precision = 0.0000001;
        int i;

        while (fabs(xNew - xOld) > precision) {
                xOld = xNew;
                derivative = 1.0/(1.0+pow(xNew,2.0))+cos(xNew);
                xNew = xNew - eps*derivative;
                printf ("Going to %lg\n", xNew);
        }

        return 0;
}
```

Examples of Problem Formulation

The following sections present examples on how to formulate and formalize optimization problems.

Distributed Scheduling Problem

According to the formalization by Garey & Johnson (42), we characterize the distributed scheduling problem optimization as follows. Be the set A of parallel applications $A = \{a_0, a_1, \ldots, a_{k-1}\}$ and the function $\mathbf{tam}(.)$, which defines the number of processes that compose an application. Thus, each one of the k applications is composed of a different number of processes, i.e., $\mathbf{tam}(a_0) = 5$, $\mathbf{tam}(a_1) = 8$ etc. Consider, then, that a set P contains all processes of all k parallel applications. In this way, the number of elements in P is equal to $|P| = \sum_{i=0}^{k-1} \mathbf{tam}(a_i)$.

Each process $p_j \in P$, where $j = 0, 1, \ldots, |P| - 1$, contains particular features, here named behavior, of resource utilization: CPU, memory, input and output. Consequently, every process requires different ammounts of resources provided by a set V of computers from the distributed environment (where $|V|$ defines the number of computers). Besides that, each computer $v_w \in V$, where $w = 0, 1, \ldots, |V| - 1$, has different capacities in terms of CPU, main memory access latency and storage capacity, secondary memory read-and-write throughput, and network interface with a certain bandwidth, latency and overhead to pack messages.

Table 2.19: Guess and Check for Gradient descent

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

// f(x) = atan(x)+sin(x)
// f'(x) = 1/(1+x**2) + cos x

void gdescent(double xNew, double prec, double e, double *x, double *y) {
        double xOld = xNew + 1;
        double eps = e; // step size
        double derivative = 0.0;
        double precision = prec;
        int i;

        while (fabs(xNew - xOld) > precision) {
                xOld = xNew;
                derivative = 1.0/(1.0+pow(xNew,2.0))+cos(xNew);
                xNew = xNew - eps*derivative;
                printf ("Going to %lg\n", xNew);
        }

        *x = xNew;
        *y = atan(xNew) + sin(xNew);
}

int main(int argc, char *argv[]) {
        double x, y;
        double bestX, bestY = 10000000;
        int i;
        double signal = 1;

        srandom(time(NULL));

        for (i = 0; i < 1000; i++) {
                gdescent(signal*random(), 0.0001, 0.001, &x, &y);
                signal *=-1;

                if (y < bestY) {
                        bestY = y;
                        bestX = x;
                }
        }

        return 0;
}
```

Computers in V are connected by different communication networks. All this environment can be modeled by a non-directed graph $G = (V, E)$, where each vertex represents a computer $v_w \in V$ and the communication channels in between vertices are edges $\{v_w, v_m\} \in E$. Those channels have properties as bandwidth and latency associated.

The optimization problem consequently consists in scheduling the set P of processes over the graph vertices G in order to minimize the overall execution time of each application in A. This time is characterized by the sum of all operation costs involved in processing, memory accesses, hard disk and network.

To simplify the understanding, consider a problem instance with two parallel applications composed of the processes in table 2.20 (MI represents the million of instructions executed by processes; ML and ME are, respectively, the number of Kbytes by second read from and write to the main memory; HDL and HDE are, respectively, the number of Kbytes by second read from and write to the hard disk – secondary memory; NETR and NETE are, respectively, the number of Kbytes by second received and sent through the network – in this situation, the sender, for NETR, and the destination, for NETE, are also presented).

Table 2.20: Parallel application process behavior

Application 0							
Process	CPU (MI)	ML Kb/s	ME Kb/s	HDL Kb/s	HDE Kb/s	NETR Kb/s	NETE Kb/s
p_0	1,234	123.78	0.00	78.21	0.00	12.50 – p_1	532.12 – p_1
p_1	1,537	23.54	89.45	0.00	12.30	532.12 – p_0	12.50 – p_0
Aplication 1							
Process	CPU (MI)	ML Kb/s	ME Kb/s	HDL Kb/s	HDE Kb/s	NETR Kb/s	NETE Kb/s
p_2	1,221	823.78	70.00	78.21	543.00	10.92 – p_3	321.12 – p_4
p_3	1,137	223.54	179.45	324.00	212.31	423.12 – p_4	10.92 – p_2
p_4	2,237	23.54	17.45	12.00	0.00	321.12 – p_2	423.12 – p_3

Table 2.21: Computer features

Computer	CPU (MIPS)	ML Kb/s	ME Kb/s	HDL Kb/s	HDE Kb/s
v_0	1,200	100,000	40,000	32,000	17,000
v_1	2,100	120,000	50,000	42,000	19,000
v_2	1,800	100,000	30,000	22,000	9,000
v_3	1,700	95,000	20,000	25,000	11,000
v_4	2,500	110,000	60,000	62,000	30,000

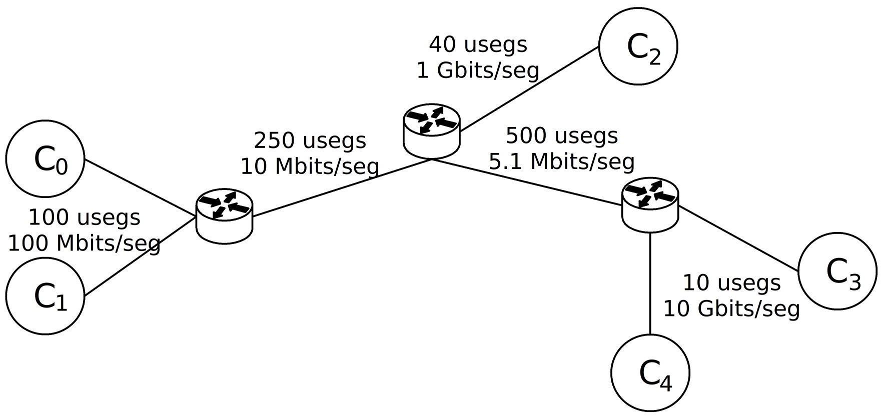


Figure 2.4: Example of network interconnection

Let the environment be composed by the five computers described in table 2.21 (where MIPS represents the processing capacity in million of instructions per second; ML and ME are, respectively, the main memory read-and-write throughput, in Kbytes per second; HDL and HDE are, respectively, the hard disk read-and-write throughput, in Kbytes per second – secondary memory). Such computers, in V, are interconnected according to figure 2.4, which also presents the average network bandwidth and latency. Besides that, consider the scheduling operator defined by $\propto$.

The distributed scheduling problem consists in defining on which computer v_w each process p_j will be placed on, considering the resource capacities and workloads. An example of solution for this instance is given by $p_0 \propto v_0$, $p_1 \propto v_1$, $p_2 \propto v_2$, $p_3 \propto v_3$ e $p_4 \propto v_4$. In this way, for each problem instance, we must schedule $|P|$ processes on an environment composed of $|V|$ computers, consequently, the universe of possible solutions is equal to $|V|^{|P|}$. For the previously presented instance, the problem has a solution universe equals to $5^5 = 3.125$. Real problem instances can consider, for instance, 1.024 computers and 64 applications, containing 512 processes each. In this situation, the solution space would be equal to $1.024^{64 \cdot 512} = 1.024^{32.768}$.

In this context, we may observe that the problem solution space is exponential and, therefore, we must propose alternative approaches capable of providing good solutions in acceptable computing time. There is no polinomial-time algorithm to optimally solve this problem (this is, to find the best solution at all), what allows to characterize it as intractable (42). Given this fact, we may adopt algorithms that explore part of the solution space and find, by using guesses and checkings, a good candidate in non-deterministic polinomial time (42).

After characterizing the problem as intractable, it is important to understand how hard it is. In order to know it, consider the equivalent NP-complete problem approached by Lageweg & Lenstra (71), and presented by Garey & Johnson (42). This problem defines the process scheduling over m processors as follows:

Instance: *Let the set of tasks T, a number $m \in \mathbb{Z}^+$ of processors, for each task $t \in T$ a length $l(t) \in \mathbb{Z}^+$ and a weight $w(t) \in \mathbb{Z}^+$, and a positive integere K.*

Question: *Is there a scheduling σ over m processors for T, where the sum, for every $t \in T$, of $(\sigma(t) + l(t)) \cdot w(t)$ is not higher than K?*

De fato: *In this context, instead of considering m processors, we assume m computers with specific capacities, and T, instead of representing tasks, is the set of processes. The length $l(t)$ defines the process duration, in this situation, the resource consumption. Besides that, we mention that according to Garey & Johnson (42) the problem stays NP-complete for any instance where $m \geq 2$. The problem can be solved in polinomial time only if the processes duration are identical, what is not expected in the presented problem, because there are great variation in application profiles.*

Distributed Data Access Optimization Problem

Consider the distributed data access optimization problem as: Consider the set A of parallel applications running on a large scale distributed environment, where $A = \{a_0, a_1, \ldots, a_{k-1}\}$. Let the function $tam(.)$ define the number of processes that composes each parallel application. In this way, each one of the k applications is composed of a different number of processes, for instance, $tam(a_0) = 5$, $tam(a_1) = 8$ etc. Then, assume that the set P constains all processes of all k applications (i.e., the number of elements in P is equal to $h = \sum_{i=0}^{k-1} tam(a_i)$), and that such processes are previously distributed over the available computing resources, according to any scheduling criterion.

Each process $p_i \in P$, where $i = 0, 1, \ldots, h$, contains specific characteristics, here defined as behavior, of resource utilization: CPU, memory, input and output. Consequently, every process requires different ammounts of resources which are provided by the set of computers V. Besides that, every computer $v_j \in V$, where $j = 0, 1, \ldots, w - 1$, has different capacities in terms of processing power, main memory access latency and store capacity, secondary memory read-and-write throughput, and network interface with specific bandwidth, latency and overhead to pack messages.

Computer in V are interconnected by different network technologies. All this environment can be modeled by a non-directed graph $G = (V, E)$, where each vertex represents a computer $v_j \in V$ and the communication channels in between vertices are edges $\{v_j, v_m\} \in E$. Those channels present bandwidth and latency properties related. Also consider that there is a set F of files, where $F = \{f_0, f_1, \ldots, f_{z-1}\}$, which are accessed through read-and-write operations executed by processes in P.

Assume that the processes are previously scheduled, the distributed data access optimization problem consists in distributing the z files over the vertices of G in order to minimize the cost function that every process $p_i \in P$ has to access them. This function C_{p_i}, defined in equation 2.5, is composed of the storage cost (or distribution) of files $s(f_y, v')$ over the elements $v' \in V'$, where $V' \subset V$, summed up to the access cost $d(f_y, v')$ which defines the time a computer $v \in V$ needs to obtain the file from another $v' \in V'$, multiplied by the file utilization rate which is $u(f_y)$ (the storage and access cost depend on the latency and bandwidth properties associated to the edges of G).

$$C_{p_i} = \sum_{f_y \in F, v' \in V'} s(f_y, v') + \sum_{f_y \in F, v' \in V'} d(f_y, v') \cdot u(f_y) \tag{2.5}$$

This model can still consider the files as atomic or divisible in fixed chunks of size s. If atomic files are considered, two processes p_i and p_r, that perform operations on them, and that are located in distant environment regions (where the distance is characterized by the low bandwidth and high communication latency) would aggravate the selection of the vertices (computers) where the file would be placed on. For optimization purposes, we assume that those processes can access chunks of a same file, consequently, every file $f_y \in F$ is composed by a set of q chunks of fixed size s, where $f_y = \{t_0, t_1, \ldots, t_{q-1}\}$. In this way, the processes p_i and p_r can access, simultaneously, different chunks of the file f_y and, the optimization problem could predict those accesses in order to improve the system efficiency.

In this context, the data access optimization problem consists in distributing the chunks of files $f_y \in F$ over the environment composed of the set V of computers with heterogeneous capacities and interconnection, aiming that every process $p_i \in P$ has the lowest possible cost to access file chunks.

To simplify the understanding, consider a problem instance with two parallel applications composed of the processes in table 2.22 (MI represents the million of instructions executed by processes; ML and ME are, respectively, the number of Kbytes by second read from and write to the main memory; HDL and HDE are, respectively, the number of Kbytes by second read from and write to the hard disk – secondary memory; NETR and NETE are, respectively, the number of Kbytes by second received and sent through the network – in this situation, the sender, for NETR, and the destination, for NETE, are also presented).

Table 2.22: Parallel application process behavior

Application 0							
Process	CPU (MI)	ML Kb/s	ME Kb/s	HDL Kb/s	HDE Kb/s	NETR Kb/s	NETE Kb/s
p_0	1,234	123.78	0.00	78.21	0.00	12.50 – p_1	532.12 – p_1
p_1	1,537	23.54	89.45	0.00	12.30	532.12 – p_0	12.50 – p_0
Aplication 1							
Process	CPU (MI)	ML Kb/s	ME Kb/s	HDL Kb/s	HDE Kb/s	NETR Kb/s	NETE Kb/s
p_2	1,221	823.78	70.00	78.21	543.00	10.92 – p_3	321.12 – p_4
p_3	1,137	223.54	179.45	324.00	212.31	423.12 – p_4	10.92 – p_2
p_4	2,237	23.54	17.45	12.00	0.00	321.12 – p_2	423.12 – p_3

Table 2.23: Computer features

Computer	CPU (MIPS)	ML Kb/s	ME Kb/s	HDL Kb/s	HDE Kb/s
v_0	1,200	100,000	40,000	32,000	17,000
v_1	2,100	120,000	50,000	42,000	19,000
v_2	1,800	100,000	30,000	22,000	9,000
v_3	1,700	95,000	20,000	25,000	11,000
v_4	2,500	110,000	60,000	62,000	30,000

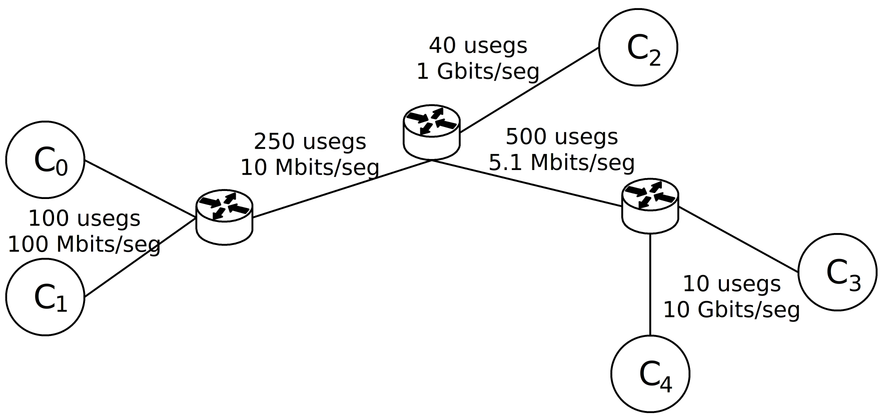


Figure 2.5: Example of network interconnection

Consider that those processes are scheduled over the set V which contains the five computers described in table 2.23 (where MIPS represents the processing capacity in million of instructions per second; ML and ME are, respectively, the main memory read-and-write throughput, in Kbytes per second; HDL and HDE are, respectively, the hard disk read-and-write throughput, in Kbytes per second – secondary memory). Consider the scheduling operator defined by $\propto$, a scheduling example is given by $p_0 \propto v_0$, $p_1 \propto v_1$, $p_2 \propto v_2$, $p_3 \propto v_3$ and $p_4 \propto v_4$. The computer in V are interconnected according to figure 2.4, which also presents the average bandwidth and latency for each communication channel.

Still consider that all the five processes of the two parallel applications access the same file f_y, which is composed of the chunks $\{c_0, c_1, \ldots, c_9\}$. Consequently, in order to optimally solve the problem, we must explore all the possibilities of chunks distribution over the 5 computers and evaluate the access cost for each process p. In this situation, we would make permutations to form all possible distribution solutions. To exemplify, consider the solution $s_n \in S$ where the subset of chunks $\{t_0, \ldots, t_6\}$ is allocated on the computer v_0 and the others, i.e. $\{t_7, t_8, t_9\}$, on the computer v_3. The set of all possible solutions S for any problem instance is given by n^r, where n is the number of computers and r is the number of file chunks. Consequently, in the presented instance, the solution space is equals to $5^{10} = 9.765.625$.

We may observe the problem in real-world circumstances. Currently, files vary from KBytes to Gbytes. Assume, for illustration purposes, that the file chunks have around 32 Kbytes and that the target environments are composed of more than 256 computers, thus, we may infer the problem scale according to table 2.24. The table presents solution spaces for a single file distribution. In real situations, we expect to deal with million of computers storing million of files, where each file can vary from Kbytes to Gbytes.

Table 2.24: Solution space to distribute file chunks

File size (MBytes)	Computers	Number of Chunks	Solution Space
10.0	256	320	$4 \cdot 10^{770}$
100.0	256	3200	$2 \cdot 10^{7706}$
1024.0	256	32768	$1 \cdot 10^{78913}$
10.0	512	320	$9 \cdot 10^{866}$
100.0	512	3200	$4 \cdot 10^{8669}$
1024.0	512	32768	$2 \cdot 10^{88777}$
10.0	1024	320	$1 \cdot 10^{963}$
100.0	1024	3200	$9 \cdot 10^{9632}$
1024.0	1024	32768	$3 \cdot 10^{98641}$

Besides existing a large search space, for each possible solution we still need to compute a function to evaluate the changes in data access cost for each process $p_i \in P$, what is defined in equation 2.5. However, in order to optimize the average cost M for every process, we must apply equation 2.6, where h is the number of processes in set P. In this way, the time complexity of the evaluation of each possible solution $s_n \in S$ is in the order of $O(n^2)$.

$$M = \frac{\sum_{p_i \in P} C_{p_i}}{h} \tag{2.6}$$

In this context, we may observe that the problem solution space is exponential and, therefore, we must consider approaches capable of finding good solutions in accpetable time. We must also observe that there is no known polinomial-time algorithm to optimally solve the problem, what allows to characterize it as intractable (42). Given this fact, we may adopt algorithms to explore part of the solution space and find (by using guesses and checkings) a good candidate in non-deterministic polinomial time (42).

After characterizing the problem as intractable, we must understand how hard it is. For that, consider the equivalent NP-complete problem studied by Sickle & Chandy (104) and also presented by (42). It defines the allocation of multiple file copies on a distributed environment as follows:

Instance: *Consider a graph $G = (V, E)$, for each $v \in V$ a utilization $u(v) \in Z^+$, a storage cost $s(v) \in Z^+$ and a positive integer K.*

Question: *Is there a subset $V' \subset V$ in which every $v \in V$ has $d(v)$, defined as the number of edges in the shortest path in G of v for any member in V', according to equation $\sum_{v \in V'} s(v) + \sum_{v \in V} d(v) \cdot u(v) \leq K$? The goal here is to distribute all file copies on computers in V' aiming at minimizing the access time for any $v \in V$.*

This problem is equivalent and presents a subtle difference to the one previously stated. In the latter, multiple copies of the same file are distributed while the one stated considers chunks of a same file. However, we can reduce the first problem in the second, where the processes $p_i \in P$ would access different chunks of a same file f_y, what allows to characterize it as NP-complete (104).

2.2.2 Self-Configuration

Different techniques have been proposed and developed to adapt systems to environmental conditions. Such adaptation can improve system performance, fix errors, provide fault tolerance, security, etc. Mckinley *et al.* (81) present two general approaches to provide adaptation to software systems: parameter and compositional adaption. The parameter approach considers changes in system variables to modify the software behavior, e.g., the TCP protocol adjusts the window of packets according to the network congestion parameter. The compositional considers the exchange of an algorithm or the structure to improve the system response to the environment. Compositional adaption allows to modify the system to adopt new algorithms and structure which were not considered in the design phase.

The challenges involved in the second approach motivated Mckinley *et al.* (81) to address three technogical issues for compositional adaptation: separation of concerns, computational reflection and component-based design.

Separation of concerns attempts to separate the business logic from cross-cutting concerns such as fault tolerance, security, performance, quality of service, etc. Basically, system vendors want to provide those functionalities to their product modules. This separation simplifies the maintenance, development and system evolution, while helps in reuse. Nowadays, a widely adopted technology to obtain separation of concerns is the aspect-oriented programming, which helps to separate cross-cutting concerns during the development phase. The code used to implement a cross-cutting concern is called aspect. The good point with aspect-oriented programming is that the concerns can be written and managed independently of the application code.

The second technogical issue, named computational reflection, makes applications capable of reasoning about and modify itself. This issue is composed of two activities: introspection and intercession. The introspection makes the application capable of observing its own behavior and intercession enables the application to act and modify such behavior. Introspection would, for instance, support the accounting of method calls while intercession would help to add new classes to the system. Reflection is tipically provided by the programming language, e.g., Java, or by a middleware.

The last technogical issue is called component-based design, which is based on component interfaces. Well specified interfaces can be followed by different software vendors. Each vendor may implement it in components with particular functionalities. As those components follow the same 'contract', they can be statically (compile and link time) or dynamically (runtime) binded to the interface. The specific binded component will be the one to execute the invoked methods.

Besides those three issues, Mckinley *et al.* (81) still considers other influences and supports for adaptations. The first is the development of middlewares and their services which present well defined interfaces and a layered structure to separate different implementation levels. Many of the cross-cutting concerns have been developed by middleware projects such as J2EE, CORBA, Java RMI and .NET. Developers know that those platforms offer and deals with those concerns, then they basically code the business logic and use the functionalities provided, such as the supports for transactions, authentication, security, etc.

There are still other factors which motivate the self-configuration aspect such as design patterns. Those patterns document solutions for well-known situations and help to approach them. They tipically consider best practices which help to maintain and extend softwares.

Mckinley *et al.* (81) still classifies two possible self-configuration approaches: static and dynamic. The first is less flexible, being tipically hardwired in the system. This approach is employed in the development (hardcoded), compile/link (such as AspectJ) and load (e.g. Java Virtual Machine loading a specific class) time phases. The second considers adaption on runtime, what can tune (adapt parameters or features) or modify (add new functionalities, e.g., algorithms) the system properties.

After defining important subject in the area, it is also relevant to define four key challenges for self-configuration: assurance, security, interoperability and decision making. The assurance attempts to guarantee, at different levels, the correctness of the adapted system which can be obtained from the selection of verified and validated components, and automatical code generation using specifications. The security issue deals with how vulnerable is a system after being adapted. The interoperability aims at providing the integration of third-party mechanisms and make them work cooperatively to meet the application requirements. The last challenge, named decision making, attempts to detect and define how and when a system must be modified to better meet the environmental needs. Decision making grabs sensored information and analizes it to recompose the system.

Now we present some examples on self-configuration. The first is presented in table 2.25, which prints out the type of a class and a primitive type.

Table 2.25: Discovering types

```
import java.util.*;

public class DiscoverTypes {

        public static void main(String args[]) {
                Vector v = new Vector();
                boolean b = false;

                Class c1 = v.getClass();
                Class c2 = boolean.class;

                System.out.println(c1);
                System.out.println(c2);
        }
}
```

The next example (table 2.26) receives the full name of a class as parameter, loads it and prints its fields and the respective modifiers out.

Table 2.27 presents a code to obtain the methods and return types for any class. The user must provide the full class name as parameter.

Examples on tables 2.28 and 2.29 call String and Integer methods. The first calls the empty constructor while the second invokes a specific constructor for the class Integer.

Table 2.30 receives class names as parameter and stores in a Vector. Then, the user call a method which is searched in the Vector. If a class contains it, then it is called.

Table 2.31 presents the source code for a Web Server. It waits for browser commands and tries to load the specific class to deal with such commands. An example of class to deal with the HTTP command GET is presented in table 2.32. When the browser requests a web page using HTTP GET, the class (table 2.32) is loaded and the method

Table 2.26: Loading a class and printing its fields and modifiers

```
import java.lang.reflect.*;
import java.util.*;

public class FullName {
        public static void main(String args[]) throws Exception {
                Class str = Class.forName(args[0]);
                Field[] fields = str.getFields();

                for (int i = 0; i < fields.length; i++) {
                        System.out.println(fields[i].getType()+", "
                                        +fields[i].getName());

                        if (Modifier.isFinal(fields[i].getModifiers()))
                                System.out.println("\tfinal");
                        if (Modifier.isNative(fields[i].getModifiers()))
                                System.out.println("\tnative");
                        if (Modifier.isPrivate(fields[i].getModifiers()))
                                System.out.println("\tprivate");
                        if (Modifier.isProtected(fields[i].getModifiers()))
                                System.out.println("\tprotected");
                        if (Modifier.isPublic(fields[i].getModifiers()))
                                System.out.println("\tpublic");
                        if (Modifier.isStatic(fields[i].getModifiers()))
                                System.out.println("\tstatic");
                }
        }
}
```

Table 2.27: Loading the class and printing information about its methods

```
import java.lang.reflect.Method;
import java.lang.reflect.Type;
import static java.lang.System.out;

public class MethodTest {
        public static void main(String args[]) throws Exception {
                Class c = Class.forName(args[0]);
                Method[] methods = c.getDeclaredMethods();

                for (Method m : methods) {
                System.out.println("Method: Name: "+m.getName()+
                                " Return Type: "+m.getReturnType());
                Class[] pType = m.getParameterTypes();

                for (int i = 0; i < pType.length; i++)
                        System.out.println("Parameter: Type: "+pType[i]);
                }
        }
}
```

execute is called. The class GET deals with HTML and PHP files. We observe the system can accept new commands only by creating new classes. It can also support other script languages such as Python and Ruby, only by extending the functionalities of class GET.

2.2.3 Self-Healing

The self-healing aspect aims at detecting incorrect execution states, faults and other abnormalities, as well as take them over, avoiding or minimizing the interference in the normal system execution. This aspect is closely related to the fault tolerance area, what motivated the study of two taxonomies proposed by Gärtner (45) and Laprie & Randell (72).

Table 2.28: Calling a String method: using the empty constructor

```
import java.lang.reflect.*;
import java.util.*;

public class CallMethod {
        public static void main(String args[]) {
                try {
                Class c = Class.forName("java.lang.String");
                Object instance = c.newInstance();
                String methodStr = "concat";
                Class[] argTypes = new Class[] { String.class };

                Method method = c.getDeclaredMethod(methodStr, argTypes);
                String margs[] = new String[]{ "testing" };
                Object a = method.invoke(instance, (Object []) margs);

                System.out.println(a);
                } catch (Exception e) { e.printStackTrace(); }
        }
}
```

Table 2.29: Calling an Integer method: using a non-empty constructor

```
import java.lang.reflect.*;
import java.util.*;

public class CallMethod2 {
        public static void main(String args[]) {
                try {
                Class c = Class.forName("java.lang.Integer");
                Object instance = c.getConstructor(new Class[]{String.class}).newInstance(new String("1"));
                String methodStr = "toString";

                Method method = c.getDeclaredMethod(methodStr);
                Object a = method.invoke(instance);

                System.out.println(a);
                } catch (Exception e) { e.printStackTrace(); }
        }
}
```

Gärtner (45) and **?**) evaluated terminologies of the fault tolerance area and observed irregular and different definitions to the same addressed aspects. Gärtner (45) emphasizes the importance in unify and clearly define the terms of that area, because they would simplify the compreehension of inherent problems as well as tackle them. This aspect motivated the defintion of terms such as fault, error, failure, and also ways of modeling faults. The author considers an asynchronous distributed system model, as it is the closest to real-world environments. Besides that, it better generalizes the system modeling, also modeling synchronous environments.

According to this model, a set of processors V, interconnected through communication channels in E, are organized according to a graph $G = (V, E)$. Every element $v \in V$ presents computational capacity [8] and different workloads. Each channel $e \in E$, where $e = \{v_i, v_j\}$, presents arbitrary delays and different bandwidths in between vertices. Consider that a set of processes P, with $|P| > 1$ elements, was allocated over the subset $V' \subset V$,

[8]The term capacity means the performance of vertex resources. Being the vertex a processor, the term refers to the processing power – million of instructions per second. Being the vertex a computer, the term involves the processing, memory, hard disk and network capacities.

Table 2.30: Storing class names and calling methods

```
import java.lang.reflect.*;
import java.util.*;

public class Example {
        private Vector classes;

        public Example() { classes = new Vector(); }

        public void loadClass(String className) { classes.add(className); }

        public void call(String method, Class[] argTypes, Object[] paramValues) throws Exception {
                for (int i = 0; i < classes.size(); i++) {
                String className = (String) classes.elementAt(i);
                Class c = Class.forName(className);
                Object instance = c.getConstructor().newInstance(); // classes must have empty constructors

                Method m = c.getDeclaredMethod(method, argTypes);
                if (m != null) {
                        System.out.println("Calling "+className+"."+method);
                        Object a = m.invoke(instance, paramValues);
                        System.out.println("Result is: "+a);
                }
                }

        }

        public static void main(String args[]) throws Exception {
                Example e = new Example();
                e.loadClass("java.lang.String");
                e.call("concat", new Class[]{String.class}, new Object[]{new String("test String")});
                e.call("toString", new Class[]{}, new Object[]{});
        }
}
```

where $|V'| > 1$. Processes in P communicate by exchanging messages, using the channels in E. In order to approach a specific event, consider that a process p_i which sends a message m to p_j, and that there is an edge or set of edges capable of connecting them, at the instant t. Given the different capacities of the elements in V, the channels in E and their respective workloads, nothing can be guaranteed about the excat time interval k needed to deliver m to p_j, processing of such message, generation and send of a reply.

On this model, Gärtner (45) and Laprie & Randell (72) formalize the terms fault, error and failure. A fault represents a undesirable transition in between system states. An error is caused by a fault, which takes the system to an incorrect state. An error may lead to a failure, which indicates that the system has diverged from the specified behavior. To better understand those terms, consider figure 2.6 where a finite state machine is presented, which models the process behavior. The set of states is formed by $E = \{E_1, E_2, E_3\}$, wich present associated transitions. Those transitions can be activated acionadas by a specific input, given by the alphabet $\{0, 1\}$, ot by any unexpected inputs contained in set T. Still consider that the states E_1 and E_2 are correct and, consequently, follow the process specification. E_3 represents an erroneous state, which diverges from the specification.

The transition from state E_1 to E_2 can occur in the expected manner, i.e., given by the input 0 or by an input $t_0 \in T$, where $T = \{t_0, t_1\}$. In case of t_0, the machine

Table 2.31: Self-Configuring Web Server

```
import java.lang.reflect.*;
import java.net.*;
import java.io.*;

public class WebServer {
        private ServerSocket ss;
        private int port;

        public WebServer(int port) throws Exception {
                this.port = port;
                this.ss = new ServerSocket(this.port);
        }

        public void run() throws Exception {
                while (true) {
                        // listening
                        Socket s = ss.accept();
                        BufferedReader br = new BufferedReader(
                                                new InputStreamReader(
                                                s.getInputStream()));
                        PrintStream ps = new PrintStream(s.getOutputStream());
                        String str = null;
                        String[] cmd = null;

                        System.out.println("Reading...");
                        do {
                                str = br.readLine();

                                if (cmd == null)
                                        cmd = str.split(" ");

                                System.out.println(str);
                        } while (str.trim().length() > 0);

                        System.out.println("Command");
                        for (int i = 0; i < cmd.length; i++)
                                System.out.println(cmd[i]);

                        //cmd[0] -> command

                        Class c = Class.forName(cmd[0]);
                        Object o = c.newInstance();
                        Class argType[] = new Class[] { String.class};
                        String mstr = "execute";
                        Method m = c.getDeclaredMethod(mstr, argType);
                        Object result = m.invoke(o,
                                        (Object []) new String[] { cmd[1] });

                        System.out.println("Writing...");
                        ps.println(result);

                        s.close();
                }
        }

        public static void main(String args[]) throws Exception {
                WebServer ws = new WebServer(80);
                ws.run();
        }
}
```

goes to a correct state, although, we may say that such transition is motivated by an unexpected event. This event is called fault. Now consider a transition from state E_2 to E_3 after the input n_1. This event indicates a fault, as previously defined, however, in this circumstance the machine was taken to an erroneous state, where the processing and generation of incorrect answer(s) will happen. The unexpected state E_3 represents

Table 2.32: Class to deal with the command HTTP GET

```
import java.io.*;

public class GET {
        private String fileName;
        private String buffer;
        private final String htdocs = "/htdocs";

        public String execute(String param) throws Exception {
                if (param.equals("/"))
                        fileName = "index.html";
                else
                        fileName = param;

                buffer = "HTTP/1.1 200 OK\nContent-Type: text/html; charset=UTF-8\n\n";
                String str = "";

                if (fileName.indexOf("html") >= 0) {
                        FileReader fr = new FileReader(fileName);
                        BufferedReader br = new BufferedReader(fr);

                        do {
                                str = br.readLine();
                                if (str != null)
                                        buffer += str;
                        } while (str != null);

                        fr.close();
                } else if (fileName.indexOf("php") >= 0) {
                        Process p =
                                Runtime.getRuntime().exec(
                                                "/usr/bin/php "+htdocs+fileName);
                        BufferedReader b =
                                new BufferedReader(
                                new InputStreamReader(
                                        p.getInputStream()));
                        do {
                                str = b.readLine();
                                if (str != null)
                                        buffer += str;
                        } while (str != null);
                }

                return buffer;
        }
}
```

an error, while the result generated by the processing of E_3 (which diverges from the specification) is a failure.

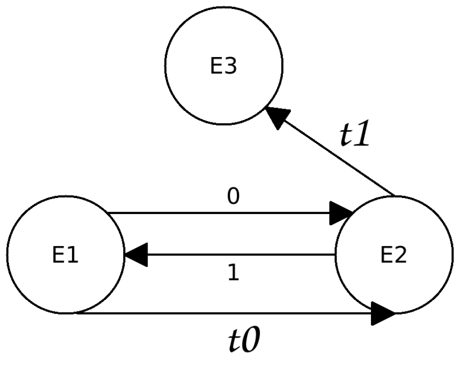


Figure 2.6: Set of states for a process

Gärtner (45) defines the specified states of a system as invariants. In the previously modeled process, the invariants are given by the set $I = \{E_1, E_2\}$, where $I \subset E$. There is, however, a universe set U which contains all elements in I and other states that can be reached by unexpected events $t \in T$. Consequently, we have $I \subset E \subset U$, where the result of $U - I$ contains all states not specified by the system, or errors, which can lead to failures. The subset given by $N = (U - I) \cap E$ contains all expected erroneous states and reacheable by faulty transitions in T. We can, therefore, say that the system is T-tolerant, because it is ready and deals with each one of the faults in T.

After this formalization, we may observe that a fault can take the system to erroneous states and, consequently, to failures. In this context, the origin or cause of an error or failure is a fault, what motivates the study and project of fault-tolerant systems.

After formalizing those terms, we need to characterize two properties that allow to classify fault treatment approaches. These properties, presented by Gärtner (45), are safety and liveness. The first is fulfilled by systems that present a set w of executions, given by $X = \{x_0, x_1, \ldots, x_{w-1}\}$, which does not break the invariants in I. The second, or liveness, guarantees that the system carries on executing.

In formal terms, we can characterize both properties according to the set of states visited during system execution. Safety guarantees that only states in I are visited during execution. Liveness guarantees that, eventually, the system goes to any state in U.

Table 2.33: Classification of the Fault Tolerance Approaches (45)

	Live	*Not Live*
Safe	*Masking*	*Fail Safe*
Not Safe	*Non-Masking*	*None*

These two properties help to define the four approaches for fault tolerance (45) (table 2.33): Masking, Fail safe, Non-masking and None. The first is composed of systems that carry on executing safely. In this situation, faults are masked by the system. The second type, or Fail safe, guarantees that the system stops its execution in a safe state. This approach was considered in the design of the terrestrial control of the space rocket Ariane 5 (**?**). This system was designed to mask single faults, and stop in a safe state in case of several faults. In this circumstance, the launching (liveness) is less important than safety. In non-masking, the system carries on executing, however it can, eventually, visit insecure states. However, there are investigations which intend to find algorithms (**?**) capable of returning the system to a safe state. The fourth and last approach, named None, does not guarantee the continuity of the exeuction nor the safety, this approach does not provide any type of fault tolerance at all.

The four presented approaches basicaly define levels of fault tolerance. In general, researches that prioritize safety are related to detection (), while the ones that mostly consider liveness are focused on stabilizing the system in case of faults ().

Laprie & Randell (72) still emphasize complementary terms to the fault tolerance area:

service delivery, outage and shutdown. Service delivery refers to the period in which the system acts as specified. Service outage is the period in which the system acts in an unexpected manner. The last, named service shutdown, refers to the period in which the system is intentionaly down.

To better classify faults, Laprie & Randell (72) propose a taxonomy that identifies eight aspects: creation phase or occurrence, system boundaries, cause, dimension, objective, intention, capability and persistence. The creation phase or occurrence deals with the moment in which the fault was introduced in the system. This aspect is divided into faults durante inserted during the software development and the ones introduced in the production environment.

A fault can also happen inside the system boundaries, i.e. internal, or generated by an external system. This external system may propagate its errors and faults and, therefore, generate faults in the first.

The phenomena responsible for generating faults can be natural or caused by humans. The natural ones do not count on human participation, such as noise in communication channels and power outage. The ones caused by humans include cases where the responsible did not act as expected, what characterizes omission; or cases where the bad usage has generated the fault, these are named commission faults.

The dimenson refers to the cause factor of the fault, what can be in hardware or software. The objective refers to the manner in which the fault was introduced, i.e., if it was inserted in a malicious way or not. The developer's, operator's, administrator's or any other user's intention is also considered in that taxonomy. Those actors can introduce faults in a deliberate way. Deliberate faults are the ones inserted by bad decisions or intentions. The non-deliberate ones are caused by non-intentional actions, where operators, administrators or developers do not expect to have problems.

The capability identifies if a fault was generated in an accidental way or by incompetent labor. The last aspect, named persistence, defines if faults are transients, this is, it happens but it is bounded in time, ou permanents, i.e., they are continuous in time.

The presented taxonomy was resultant from studies conducted in between 1990 and 2004. It emphasizes the main terms and classes observed by a committee and summarized by Laprie & Randell (72).

The fault tolerance area has been looking for and motivating other investigations on system reliability (72). Those investigations include: fault prevention, removal and forecasting. The first aims at preventing the occurrence or introduction of a fault in the system. The second looks for minimizing the number of severe faults. The third looks for estimating the current number of faults, future incidences and consequences. These three approaches have been explored in order to support the fault tolerance area.

2.2.4 Self-Protection

The self-protection aspect aims at identifying, detecting and protecting the system against events capable of modifying its normal behavior. Those events are caused by malicious or non-malicious situations. The malicious include users or programs developed to cause damage, while the non-malicious are caused by wrong configuration, non-professional labor, etc. This aspect also needs to keep the processing and data integrity, confidentialty as well as guarantee the consistency of operations. The presented definition makes clear that this aspect is closely related to security (R & Mathur; 80).

There are a set of steps involved in self-protection approaches, they basically consider the information source and analysis. The most usual information sources results from the host and network monitoring. Host monitoring collects information from the host computer components such as the file system, network events and system calls. Changes in the data and in the file system access pattern may indicate intrusive behavior. Network events may be intercepted to support the detection of attacks in user and kernel processes as well as in the network stack itself. System calls may be intercepted to obtain process behavior information. Network monitors intercept packets on the communication links by setting the network interface card in promiscuous mode. Because of such configuration, the host computer where the network monitor runs may also be susceptible to attacks such as in case of a 'ping-of-death'.

All the information captured is processed by the intrusion detection system (IDS) which is responsible for tasks such as analysing (apply different indentification approaches), planning (indicate or suggest solutions) and execute (apply the planning) (17).

The analysis step may involve two different techniques: signature or pattern detection, and anomaly detection. The signature detection looks for patterns similar to the ones stored in a knowledge base. Its main drawback is that non-cataloged patterns cannot be detected. The anomaly detection identify attacks by analysing behavior changes. Different models may be employed in this approach such as statistical and artificial neural networks which can help to determine normal and unexpected or unknown system behavior. Outliers are classified and can be considered for further analysis or simply potential attacks. This approach can be used to detect unseen attacks, however it fails in detecting ilegal operations when they do not enough differ from the normal system behavior.

The planning step considers whether is possible to apply the indicated changes and if they will not cause any damage to the system. The exeuction step apply the changes in the environment.

Axelsson (8) proposes a taxonomy of intrusion detection techniques which considers three first-level classes: anomaly, signature and compound detectors. The anomaly class looks for abnormalities in information. This class attempts to classify normal and unexpected system behavior. According to the author, most of the techniques in this class considers self-learning or programmed approaches. Self-learning systems learn about the normal behavior and, after samples, they are capable of determining outliers. Those self-learning techniques classify data in a batch or time series approach. In the batch one, data is available as a picture, they simply classify by segmenting data based on rules or statistics. Time series appproaches consider the system variations on time to detect intrusions. Programmed approaches have pre-defined rules determined by designers. Anything different from those rules it classified as abnormal. Most of those techniques employ simple statistics (average, standard deviation and variance) and information thresholds.

In the signature class, an operation is defined as intrusion if it is listed in a knowledge base. This class is typically programmed, where the designer define explicit rules to determine intrusion. Four different approaches are listed in this class: state-modeling, expert-system, string matching and simple rule-based. In the state modeling, the designer encodes all the system states related to an intrusive situation. The expert-system approach considers rules to describe intrusive behavior which can also work together with other tools to create new rules. String-matching is just looks for similar string patterns in the system information. It is the least flexible approach. The last, the simple rule-based, are the same as the expert-system one, although the designer must specify the rules.

The compound detectors consider signature-inspired techniques to look for patterns in the system and also in the intruder behavior. They model the system activities as well as the possible intruder operations. According to the author, they are the most proiminent detectors surveyed. They basically consider self-learning techniques to get information about behavior modifications.

Axelsson (8) also proposes another taxonomy to address characteristics that are not directly related to intrusion detection principles. However, such taxonomy helps to understand how the current techniques approach the system protection. The taxonomy considers the time of detection, granularity of data processing, source of audit data, response to detect intrusions, locus of data processing, locus of data collection, security and degree of inter-operability.

The time of detection considers if the IDS works online or not. In the first case, they detect and act on the environment as fast as possible. In the second approach, they are started by administrators. The granularity of data processing considers if data is manipulated as a continuous stream or in batch. The source of audit data defines where the information comes from. The usual sources are the host computer and the network links. Response of detected intrusions can be passive of active. The passive only nofity the responsible to take decisions. The active decides on the system operations. Locus of data processing is related to where the information is analysed. If it is processed in a central or distributed architecture. The locus of data collection is also related to location. It classifies detectors which collects information and stores them in centralized or decentralized architectures. The security class addresses the attacks against the own detection system. The degree of inter-operability defines integration levels (or inter-operability levels) among different intrusion detection systems.

The two proposed taxonomies were employed by the author to classify different detector such as Haystack (105), MIDAS (98), IDES (75), W&S (113), Comp-Watch (32), NSM (49), NADIR (55), Hyperview (28), DIDS (108), ASAX (46), USTAT (53), DPEM (68), IDIOT (70), NIDES (29), GrIDS (109), CSM (118), Janus (44), JiNao (58), EMERALD (95) and Bro (91).

2.3 Planning and Execution

After deciding on what to do, the autonomic manager must plan how to apply the necessary changes, if any. Those changes may influence one or many resources as well as act on other managers. Sometimes, the change happens on only one command, others it depends on several interactions. For instance, consider that an optimizer decided to transfer some process workloads to other computers. The planning step would involve the steps to save the context of different processes and transfer them to the target nodes. This step should also consider the processes resuming.

In another situation, consider a Web system which needs to be reconfigured to fulfill more user requests. Let's assume that a metaheuristic has run which decided the best target nodes to place new Web system instances on. Here, the planning would involve the steps to launch those instance on the right nodes.

After having the steps, they have to be executed to apply the expected changes. Sometimes, the execution cannot be performed instantaneouly and needs to be scheduled. Let's get back to the Web system example. Consider the number of instances is not necessary anymore and one of them was selected to go down. Such operation could be executed while it is meeting a user request.

2.4 Final Remarks

This chapter has presented the main component in an autonomic system: the autonomic manager. Its four aspects were presented: information extraction, analysis, planning and execution. The first is responsible for obtaining resource data, the second analyzes such information according to autonomic goals, then a list of steps is defined to be executed.

Exercises

1. Choose an issue of your area of interest to approach considering the autonomic aspects.

(a) How and when to extract information? Implement it.

(b) What would you consider about self-optimization? Formalize the problem to be addressed and analyse its complexity.

(c) What would you consider about self-configuration? Do you have any idea on how to approach it?

(d) What would you consider about self-healing? Do you have any idea on how to approach it?

(e) What would you consider about self-protection? Do you have any idea on how to approach it?

(f) How would you address the planning and execution?

(g) Implement the autocorrelation function and employ it to predict data points generated by the following equations (generate a trainning sample and consider $x > 0$):

   i. $f(x) = \sin(x)$
   ii. $f(x) = x \cdot \sin(x)$
   iii. $f(x) = \frac{\sin(x)}{x}$
   iv. $f(x) = \cos(x)$

Chapter

3

Basic Statistics

Statistics is the area responsible for organizing, describing, analising and interpreting data generated by experiments, studies or any other source. According to Magalhães & de Lima (77), Statistics is divided into three large branches: Descriptive Statistics, Probability and Inference. The first is usually applied in preliminary analysis. Its function consists in summarizing data and draw the first impressions on it. Probability employs mathematics to study the uncertainty of events. Inference is usually applied when we need to study a system that we have only samples. It helps to draw conclusions on data without having all information.

Before defining the basic concepts on Statistics, we need to define the term variable (77). Variable is the evaluated characteristic or property. Let's imagine that we want to quantify the gender of a group. The variable would be gender, while it could accept two possibilities: female and male.

Variables which assume numerical values are called quantitatives. Non-numerical variables are named qualitatives. The quantitative ones can be discrete or continuous. Discrete variables are enumerable or finite, while continuous assume intervals in real numbers. The qualitative ones can present an order, such as size (which can be small, medium and large), which are named ordinals. Others do not have any order, such as gender, which are called nominals.

After those definitions, we start by exploring some basic concepts as follows.

3.1 Summary measurements

The descriptive statistics (101) looks for ways of characterizing data by using global tendencies such as average, dispersion, etc. The most traditional way of describing a dataset is by computing its average. Consider a variable X and its observations $x_0, x_1, \ldots, x_{n-1}$. The average for such observations is given by equation ??;

$$\mu = \frac{x_0, x_1, \ldots, x_{n-1}}{n} = \frac{\sum_{i=0}^{n-1} x_i}{n} \tag{3.1}$$

We usually employ averages when dealing with day-by-day tasks. Let's assume someone asks you about the time to be served by a bank teller. You will usually answer the question informing the average time. The average is a good first observation on data.

Besides the average, there is the median which is the central element of an ordered set. Having the same variable X, we would sort all its observations and choose the central one. If the set contains an even number of elements, we may select the two central elements and compute their average.

The mode is the most frequent observation of a variable X. Let's imagine that 8 out of 10 observations present the value 5, then, this is the mode.

All these measurements are relevant to get the first impression on a dataset, but they are not conclusive. The observations can be very heterogeneous and the average still keeps a good value. For instance, consider a chocolate powder company. Consider that the average weight for the cans must be 200 grams. Observing a sample, we may obtain this average, however you may have 10 grams in a can while 390 in another. In this situation, the mode could be a good indicator, it would help to evaluate which is the most frequent weight.

Consider a person evaluating two malls where he/she will buy all the Christmas gifts. The average, median and mode for the prices of both malls are presented in table 3.1. The second mall, Mall in All, presents the lowest average price. However, its median price is higher, what implies that 50% of the prices are greater (or equal) to USD 170, while other 50% are lower (or equal) to USD 170. The most common price at the second mall is USD 170, while in the first is USD 130. In this circumstance, we conclude that the Super Mall may present stores with higher prices which make a distortion in the average while the Mall in All has a better distribution of prices. If you choose the Super Mall because you know it, you might have chosen it because you know that the mode is good and you will purchase products at the most frequent price. However, if you choose Mall in All you might have selected it because you know the prices are better distributed and you do not really know the individual stores in it.

The first mall is a good option for people who know the exactly store to go, as the prices are not that well distributed. The second is a good option when you have no idea about the stores and what they provide.

Sometimes we do not have the observations for the variable X, we only have the frequency or the probability of them. Table 3.2 presents an example of the probabilities for the values of X. In such circumstance, the average would be computed by using

Table 3.1: Prices at the two malls in US Dollars

Mall	**Super Mall**	**Mall in All**
average	220	200
median	150	170
mode	130	170

equation 3.2, what would result in $\mu = 100 \cdot 0.1 + 120 \cdot 0.2 + 150 \cdot 0.5 + 190 \cdot 0.2 = 147$. The mode would be given by the most frequent value, which is 150, while the median would be in between 120 and 150. In this case, the median would be $\frac{median-120}{0.3} = \frac{150-120}{0.5}$, then $median = 138$, which is the point where the data is equally splitted.

$$\mu = \sum_{i=0}^{n-1} x_i \cdot p_i \tag{3.2}$$

Table 3.2: Probabilities of the variable X

X	100	120	150	190
p	0.1	0.2	0.5	0.2

The average, median and mode do not give a good idea about how observations of X are distributed. A certain variable, let's assume the mall prices again, can be very disperse. When going to the mall, the customer may find a high average price, just because a small set of products is very expensive.

A typical way of computing the dispersion is by evaluating the deviation from the median or the average. Assume the following observations for X: $0, 1, 1, 2, 2, 2, 3, 7$. The median is 2, while the average is 2.25. The deviation from the median is computed by $x_i - median$, where we would obtain $-2, -1, -1, 0, 0, 0, 1, 5$. If we compute the average deviation we would obtain $\frac{2}{8}$. We may notice that the negative values reduce the influence of the positive ones, what gives the false impression of low variation. Thus, we could start computing the same deviation using $|x_i - median|$. We would have the following deviations from the median: $2, 1, 1, 0, 0, 0, 1, 5$. The average deviation, or median deviation, for this different approach is $\frac{10}{8}$. In the same way we compute the median deviation, we can compute the average deviation using $|x_i - \mu|$.

The average and median deviation consider the absolute value which makes complex the study of its mathematical properties. This is what has motivated the proposal of different approaches such as the variance and the standard deviation. The variance of the observations of X is given in equation 3.3 or by its reduced form presented in equation 3.4 (this avoids subtractions). The standard deviation is given in equation 3.5

$$\sigma^2 = \frac{\sum_{i=0}^{n-1} (x_i - \mu)^2}{n} \tag{3.3}$$

$$\sigma^2 = \frac{\sum_{i=0}^{n-1} {x_i}^2}{n} - \mu^2 \tag{3.4}$$

$$\sigma = \sqrt{\sigma^2} \tag{3.5}$$

Having the frequency or probability for each observation, such as in table 3.2, we compute the variance as presented in equation 3.6.

$$\sigma^2 = \sum_{i=0}^{n-1} (x_i - \mu)^2 \cdot p_i \tag{3.6}$$

3.2 Frequency, Box-plots and Histograms

When analysing a variable, we usually create a frequency table for observations. An example is given in table 3.3, which presents how long, in terms of years, people work for the same company. Firstly, we might observe that there is a column named occurrences, which defines the number of people who fit in the specific category. Then, we have the relative frequency for the occurrences and the acumulated one. The accumulated frequency allows to define thresholds in data which helps to answer questions such as the percentage of people with up to 5 years in the company (38.88% in this case). The accumulated frequency is computed when we are able to sort the observations out. Nominal variables would not take advantage of that: such as gender.

Table 3.3: Frequency table of the number of years people work for the same company

Number of years	**occurences**	**frequency**	**acumulated frequency**
1	5	$\frac{5}{54} = 0.09259$	0.09259
2	9	$\frac{9}{54} = 0.16666$	0.25925
3	4	$\frac{4}{54} = 0.74074$	0.33333
4	2	$\frac{2}{54} = 0.03703$	0.37037
5	1	$\frac{1}{54} = 0.18518$	0.38888
6	5	$\frac{5}{54} = 0.09259$	0.48148
7	9	$\frac{9}{54} = 0.16666$	0.64814
8	10	$\frac{10}{54} = 0.18518$	0.83333
9	7	$\frac{7}{54} = 0.12962$	0.96296
10	2	$\frac{2}{54} = 0.03703$	1.00000
Total	54	1.00	–

In the presented example, the number of years is discrete, however other variables are continuous. However, in such situations we can define ranges for observations. For instance, the salary of the same people in the company could be expressed as presented in table **??** The symbol $\vdash$ is used to include the low value and exclude the higher one. This same technique can be used to study discrete variables, creating intervals for them.

An example of those discrete-based intervals is the number of brand new cars purchased by people in a country (considering the decade).

Table 3.4: Frequency table of salaries in company (USD Dollars)

Salary	Occurences	Frequency	Accumulated Frequency
$1,250 \vdash 1,500$	21	$\frac{21}{54} = 0.38888$	0.38888
$1,500 \vdash 1,750$	12	$\frac{12}{54} = 0.22222$	0.61111
$1,750 \vdash 2,100$	9	$\frac{9}{54} = 0.16666$	0.77777
$2,100 \vdash 2,500$	6	$\frac{6}{54} = 0.11111$	0.88888
$2,500 \vdash 2,750$	4	$\frac{4}{54} = 0.07407$	0.96296
$2,750 \vdash 3,500$	2	$\frac{2}{54} = 0.03703$	1.00000
Total	54	1.00	–

The frequency tables can be represented in a graphical way by using histograms. The histogram for the salary study (table 3.4) is presented in figures 3.1 and 3.2. The first presents the occurence histogram and, the second, the frequency density histogram. Both help to observe which intervals concentrate observations.

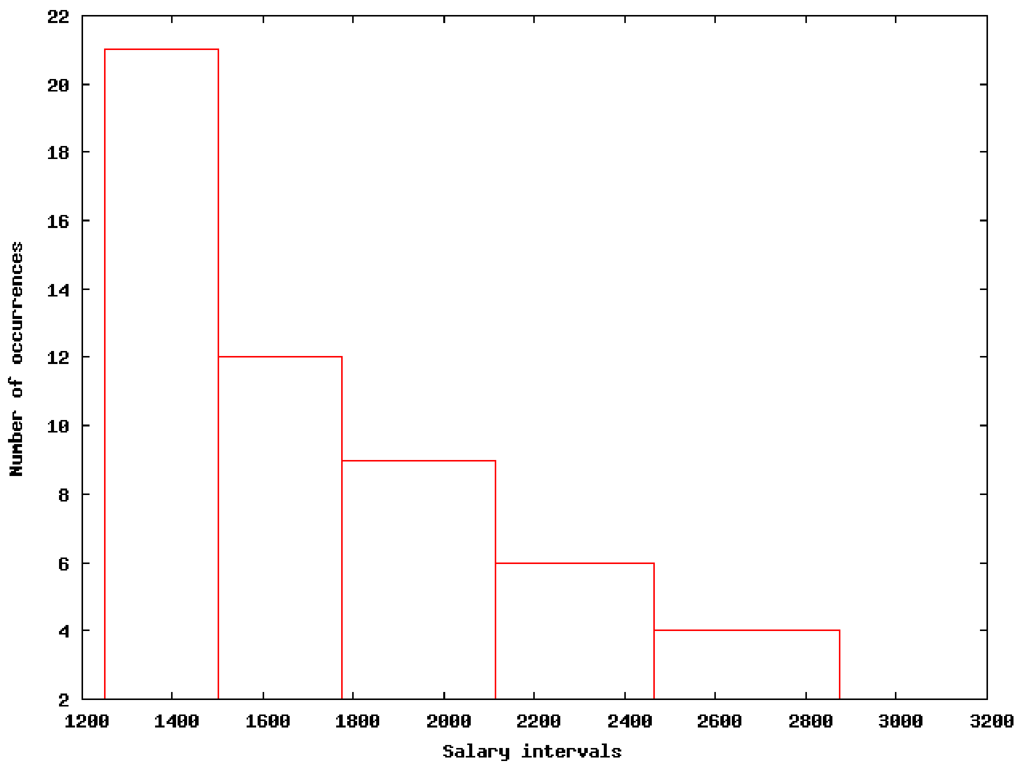


Figure 3.1: Histogram of the occurrences of the variable salary

The accumulated frequency is also presented in figure 3.3. This is useful to observe the percentage of people who have incomes up to an upper limit.

Having the frequency density histogram (figure 3.2), we can, for instance, find and position the average and median. To find the median, we must firstly remeber it is positioned in the middle of the distribution. This means that the median of the histogram

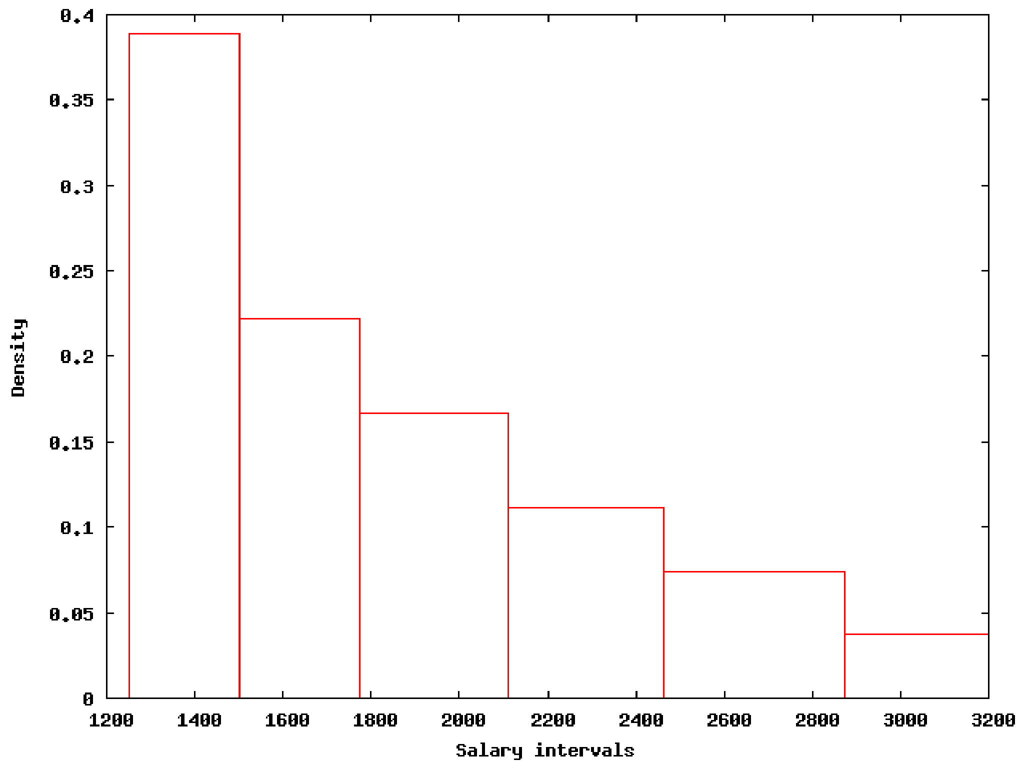


Figure 3.2: Histogram of the frequency density of the salary

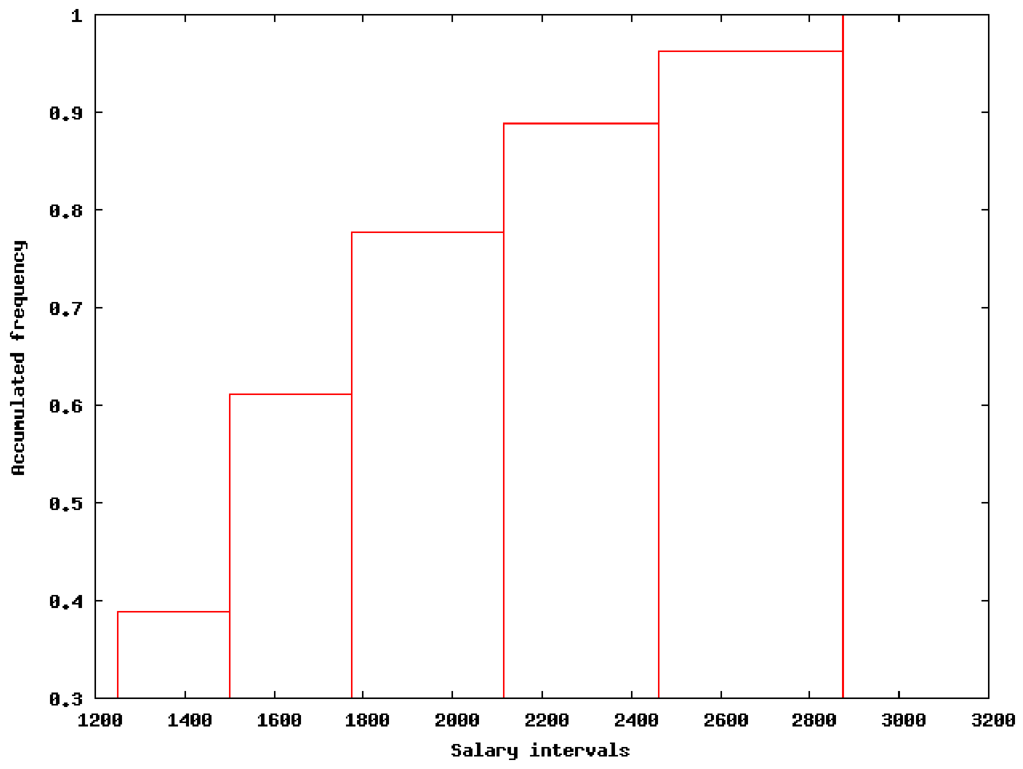


Figure 3.3: Histogram of the accumulated frequency of the salary

of figure **??** is in the second bar $[1500, 1750[$. The same can be observed in figure 3.3, which presented the accumulative density. We might observe that the second bar contains around 60% of the data (this bar adds up the first and second intervals of the density function).

On this observation, we can compute the median by using the following equation $\frac{median - lower\_interval\_value}{missing\_density} = \frac{higher\_interval\_value - lower\_interval\_value}{interval\_density}$, in this case, it would be $\frac{median-1500}{0.5-0.38888} = \frac{1750-1500}{0.22222}$, what results in $median = 1,625$. The resulting median is presented in figure 3.4.

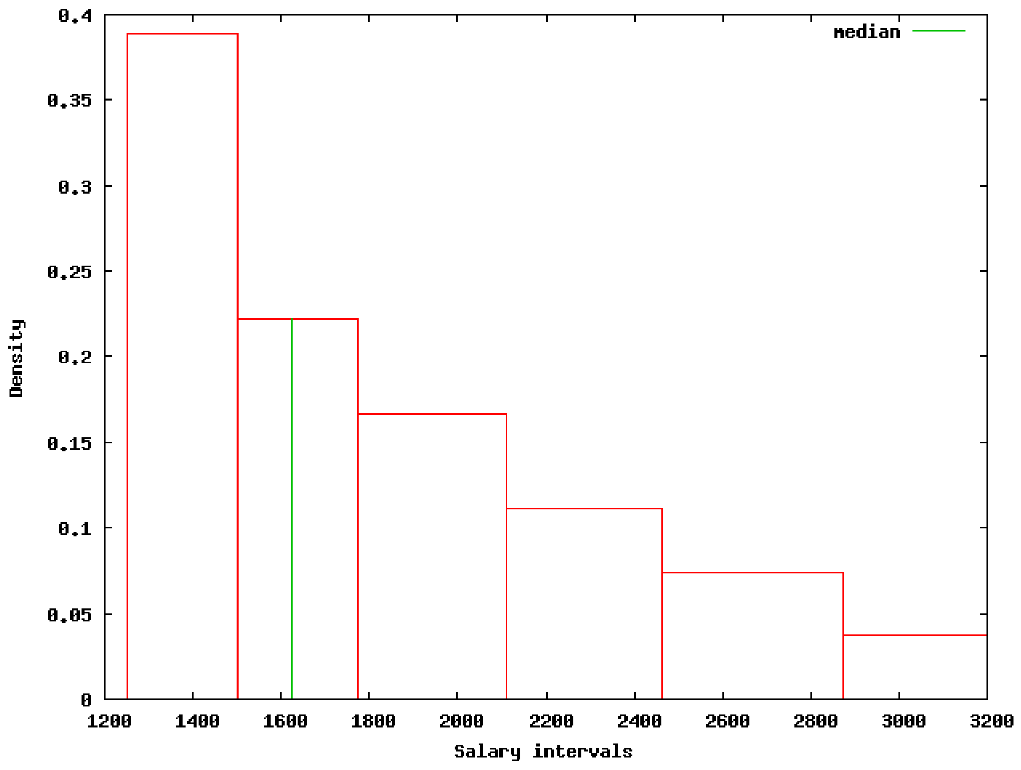


Figure 3.4: Histogram of the frequency density of the salary and the median

We can also compute the average for the density histogram following equation 3.2. The computation is given by $(1500 - 1250) \cdot 0.38888 + (1750 - 1500) \cdot 0.22222 + (2100 - 1750) \cdot 0.16666 + (2500 - 2100) \cdot 0.11111 + (2750 - 2500) \cdot 0.07407 + (3500 - 2750) \cdot 0.03703) = 1801.79250$. Figure 3.6 presents the density histogram with its median and average.

The median splits a dataset up at the middle. Thus, we know the central element and that 50% of the information is lower and 50% is higher than it. The median is also called the second quantile. This is a statistical concept to divide datasets according to their distributions. There are still the first and the third quantiles. The first divides the first 25% of data from the other 75%. The third divides the last 25% of data. The second and third quantiles can be computed in the same way we do for the median. The difference is that we consider a threshold of 25% for the first and 75% for the second.

Looking at figure 3.2, we might observe that the first 25% of data is in the first interval, i.e. $[1250, 1500[$. In this situation we compute the first quantile as $\frac{q_1-1250}{0.25} = \frac{1500-1250}{0.38888}$ what results in $q_1 = 1312.50$. The same is computed for the third quantile, which happens in the

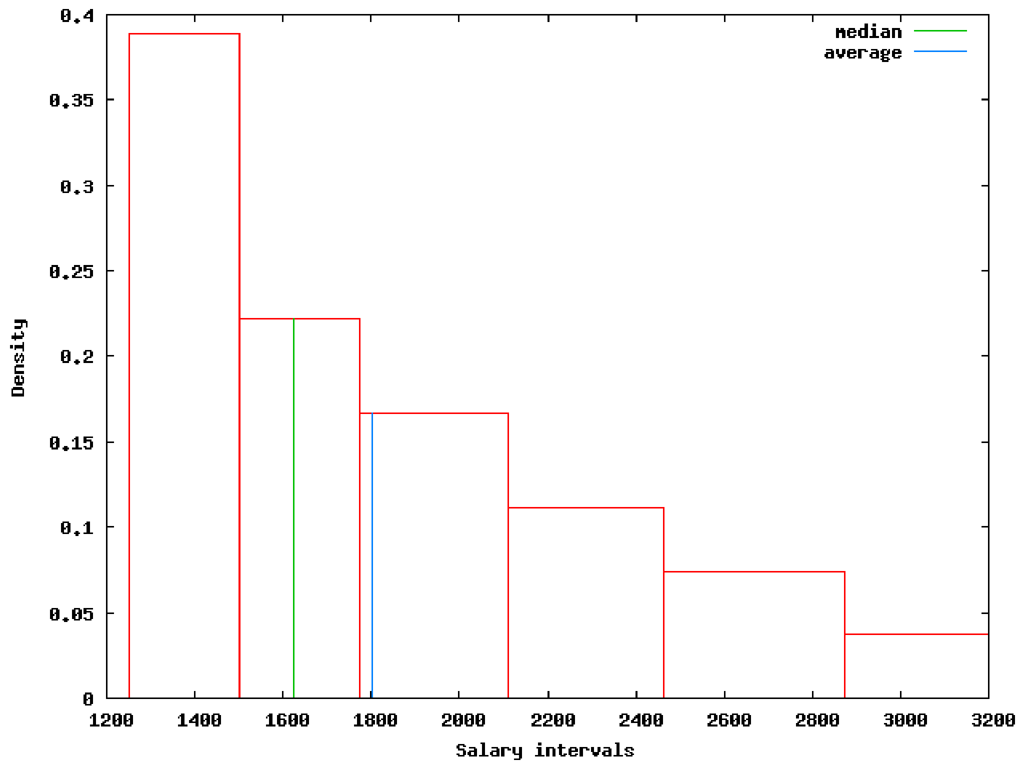


Figure 3.5: Histogram of the frequency density of the salary, the median and the average

third interval, i.e. $[1750, 2100[$, where $\frac{q_3-1750}{0.75-0.61111} = \frac{2100-1750}{0.16666}$, what results in $q_3 = 2041.68$.

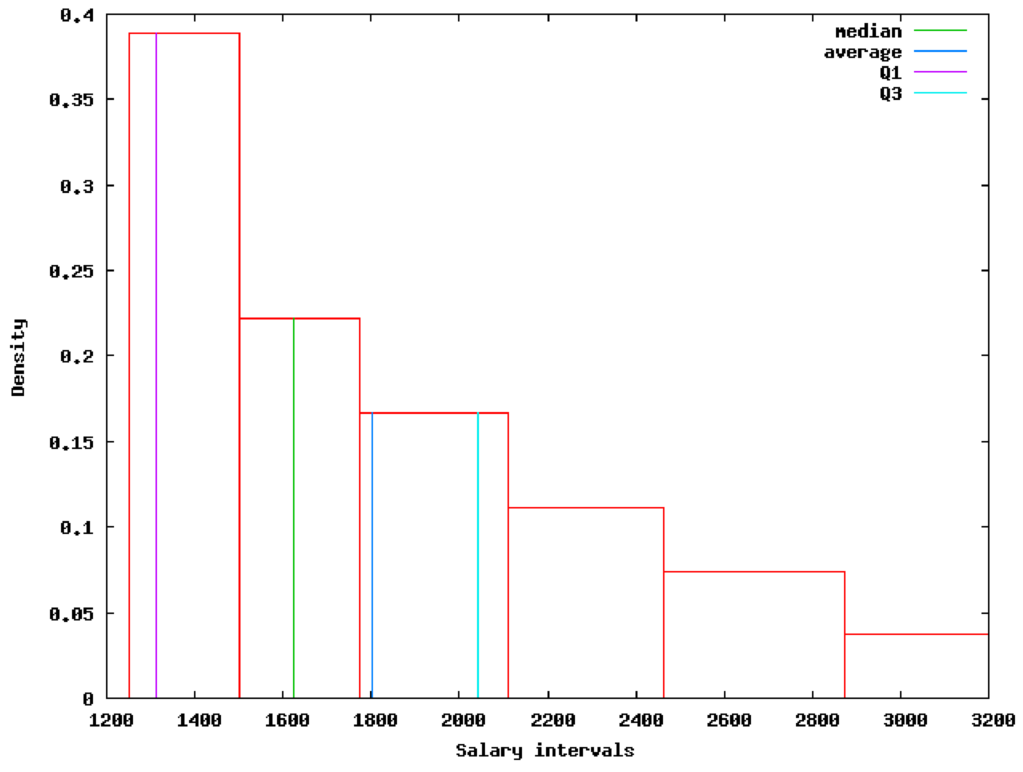


Figure 3.6: Histogram of the frequency density of the salary, the median, average, first and third quantiles

We can firstly compare the median and average. We may observe that both assume different values. The median split the dataset up, while the average tends to assume higher values. This average tendency is explained because of high values in the dataset, which moves the measurement to the right. By the first quantile, we notice that 25% of data is concentrated in a narrow interval, i.e. $[1250, 1312.50[$. The two next intervals with 25%, in between the first quantile and the median, and also in between the median and the third quantile, are wider. The last 25% of data is in an even wider interval, i.e. $[2041.68, 3500[$. From that, we observe that 25% of people (up to the first quantile) earns around the same ammount of money, while there other 25% (last quantile on) where the salary presents high variation. There is another approach more indicated to observe data distribution in between quantiles, this is the Box plot. The Box plot for the same situation is presented in figure 3.7. It helps to observe the data concentration in between $[1250, q_1[$. $[q_1, median[$ concentrates a little more than $[median, q_3[$. The interval $[q_3, 3500[$ contains the most dispersed salaries.

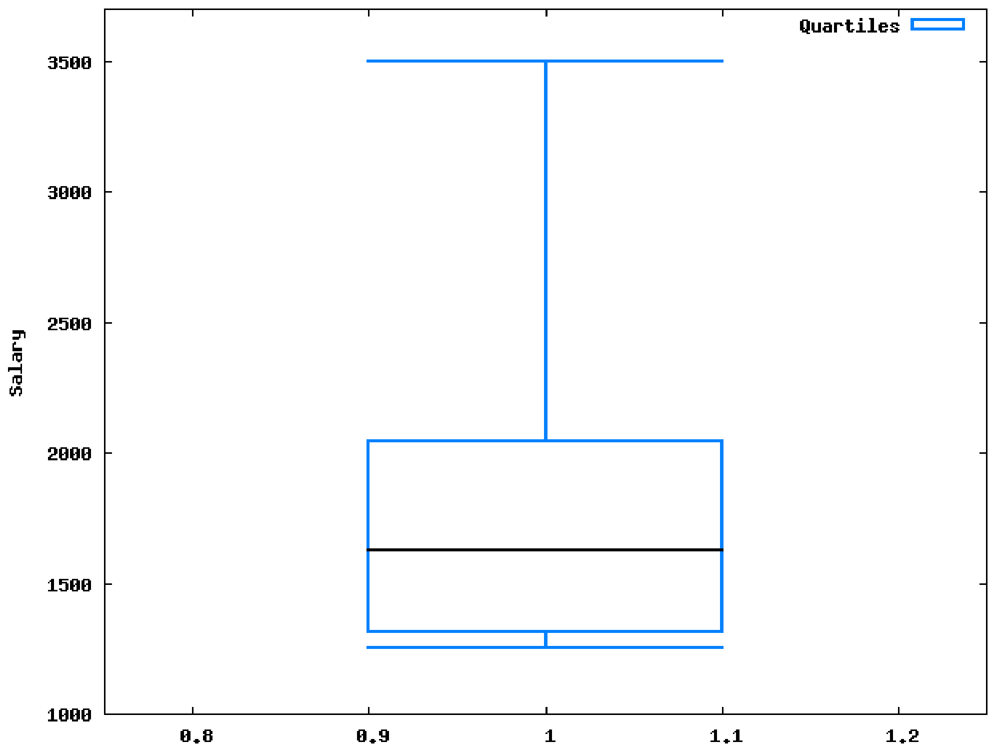


Figure 3.7: Box plot for the variable salary

If we consider two companies, we could use the Box plot to compare them (figure 3.8). The first Box plot, at the left, is the same previously presented. The second Box plot presents a lower median, however 25% of data is concentrated in between $[1450.50, 1550[$. There are higher salaries in the second company, but they are more dispersed (third quantile up to the end). We also observe that 25% of data is in between $[median, q_3[$ which is more dispersed than the same in the first company. The interval $[1000, q_1[$ is much less concentrated than the first company.

Concluding, a very proactive person could invest his/her carrer in the second company. A person who needs guarantees and stability would probably go for the first company.

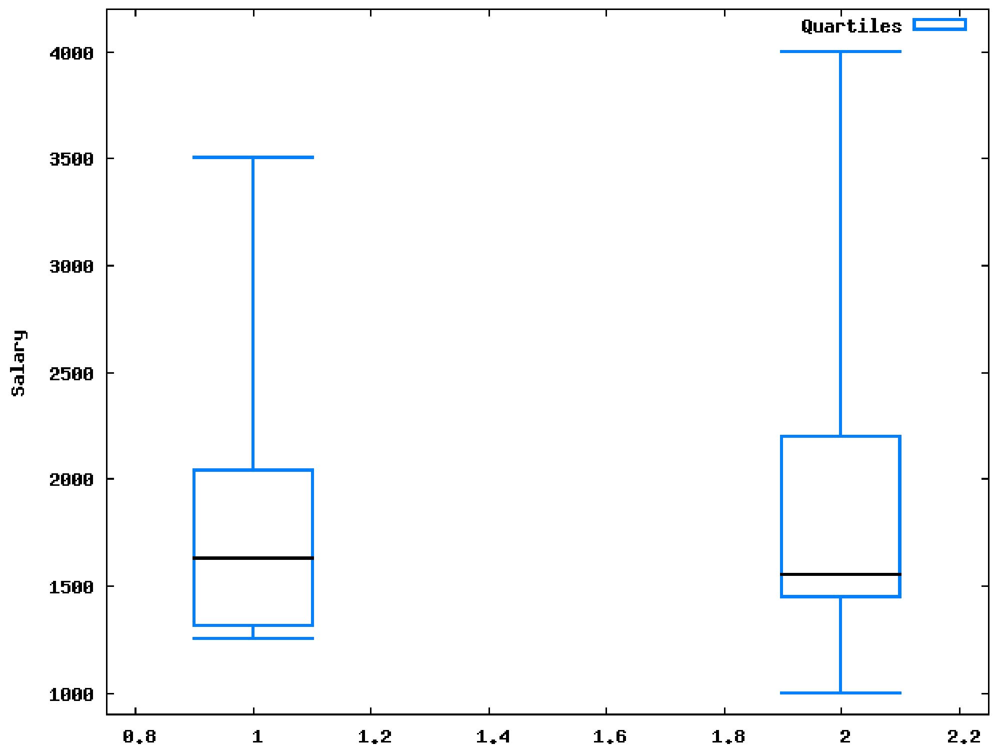


Figure 3.8: Box plot for two companies

3.3 Employing histograms on datasets

We may extract information from different sources. We can also generate it using functions. Let's consider the generation of a Normal distribution function which is presented in algorithm 1. This algorithm was used to generate 100,000 points, using $location = 0$ (μ) and $scale = 1$ (σ) (figure 3.9).

Algoritmo 1 Algorithm to generate the Normal distribution

```
1: double $location$; {This variable defines the average}
2: double $scale$; {This variable defines the standard deviation}
3: double $r, \theta$;
4: $r = \sqrt{-2.0 \cdot \ln random()}$;
5: $\theta = 2.0 \cdot \Pi \cdot random()$;
6: $ret = location + scale \cdot r \cdot \cos\theta$;
7: return $ret$;
```

From figure 3.9, we might observe that all observations are in $[-5, 5]$. Then, we generated the histogram of occurences for all observations which is presented in figure

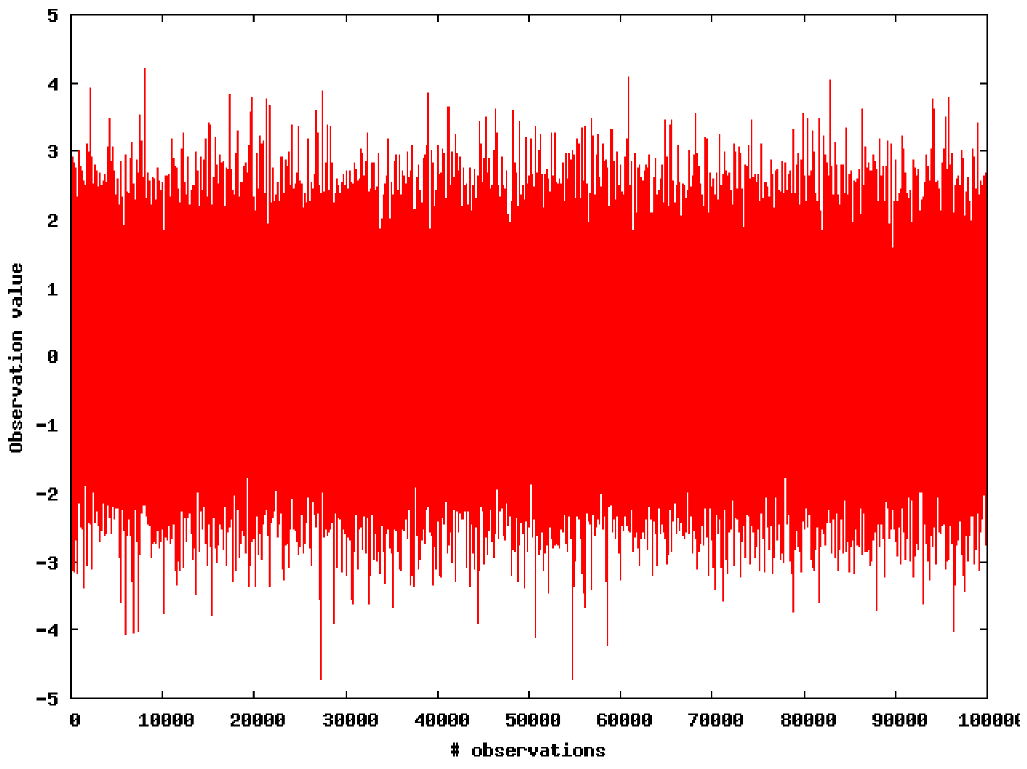


Figure 3.9: The 100,000 observations

3.10. We used it to generate the histogram of frequency density, which is showed in figure 3.11.

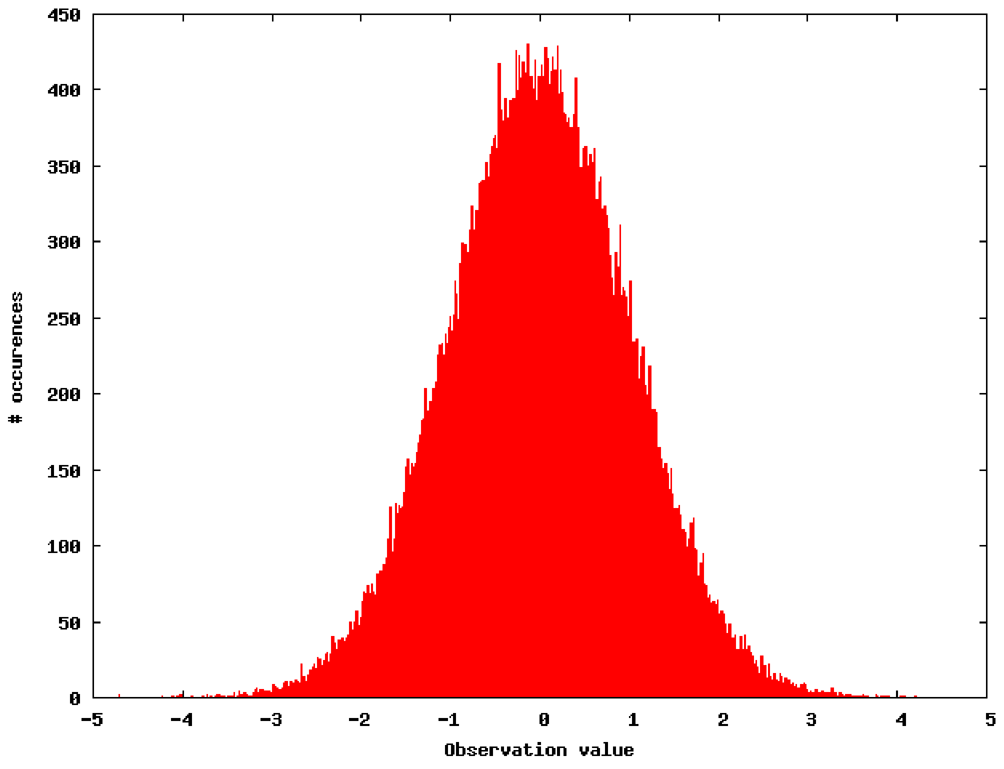


Figure 3.10: Histogram of occurences

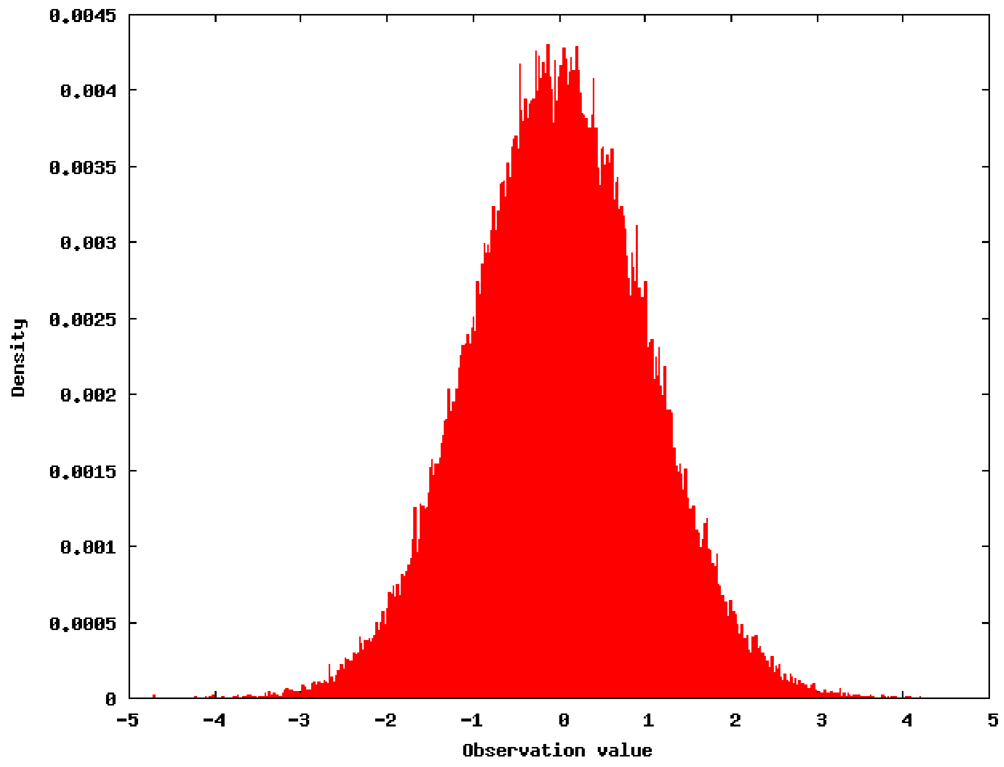


Figure 3.11: Histogram of frequency density

On this last histogram, we will present some statistical techniques. We will firstly compute the average, median and standard deviation, which are $\mu = -0.00032$, $median = -0.00181$, $\sigma = 0.99838$. We observe the median and average roughly assume the same value, this happens because the Normal distribution splits the observations up in 50% higher and lower than the parameter location (this is, μ). We also observe that the computed average is very close to the one defined by the parameter location. The same is noticed for the standard deviation, which is almost the parameter scale, i.e. 1.

Now, we will translate the histogram of frequency density[1] on a curve to simplify the study (figure 3.12). Then, we integrate the area below the curve in the interval $[\mu-\sigma, \mu+\sigma]$ and obtain the following result $\int_{\mu-\sigma}^{\mu+\sigma} f(x)dx = 0.67977$. If we go back to books on the Normal distribution, we will observe that the area in such interval is around 68%, what is not different here. We also conclude that most of the observations are in such interval.

We can integrate wider intervals such as $\int_{\mu-2.0\cdot\sigma}^{\mu+2.0\cdot\sigma} f(x)dx = 0.95391$, $\int_{\mu-3.0\cdot\sigma}^{\mu+3.0\cdot\sigma} f(x)dx = 0.99727$, $\int_{\mu-4.0\cdot\sigma}^{\mu+4.0\cdot\sigma} f(x)dx = 0.99989$. Observe that as wider the interval is, more observations we include in it. However, the most representative integration in relation to the number of standard deviations is the first, because it includes around 68% of data. This happens because the Normal distribution concentrates the larger number of occurrences around the parameter location (μ).

Let's continue to analyse the histogram. Now, we have two histograms as presented

[1]All the histograms were generated using the code presented in table 3.5.

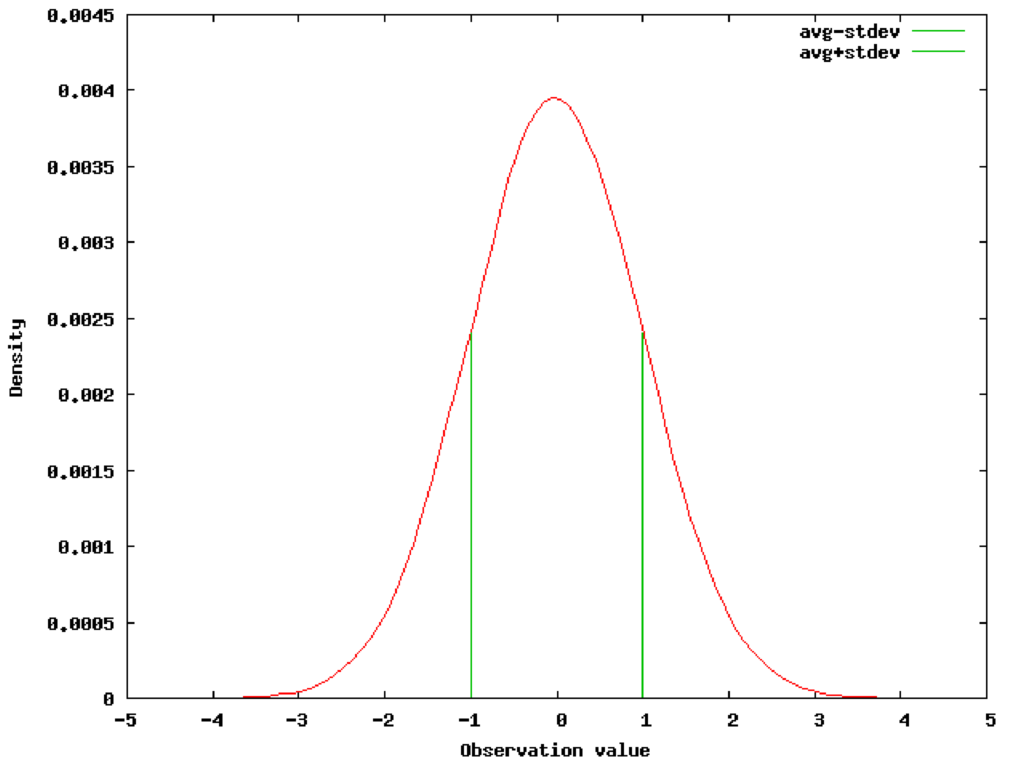


Figure 3.12: Curve based on the histogram of frequency density

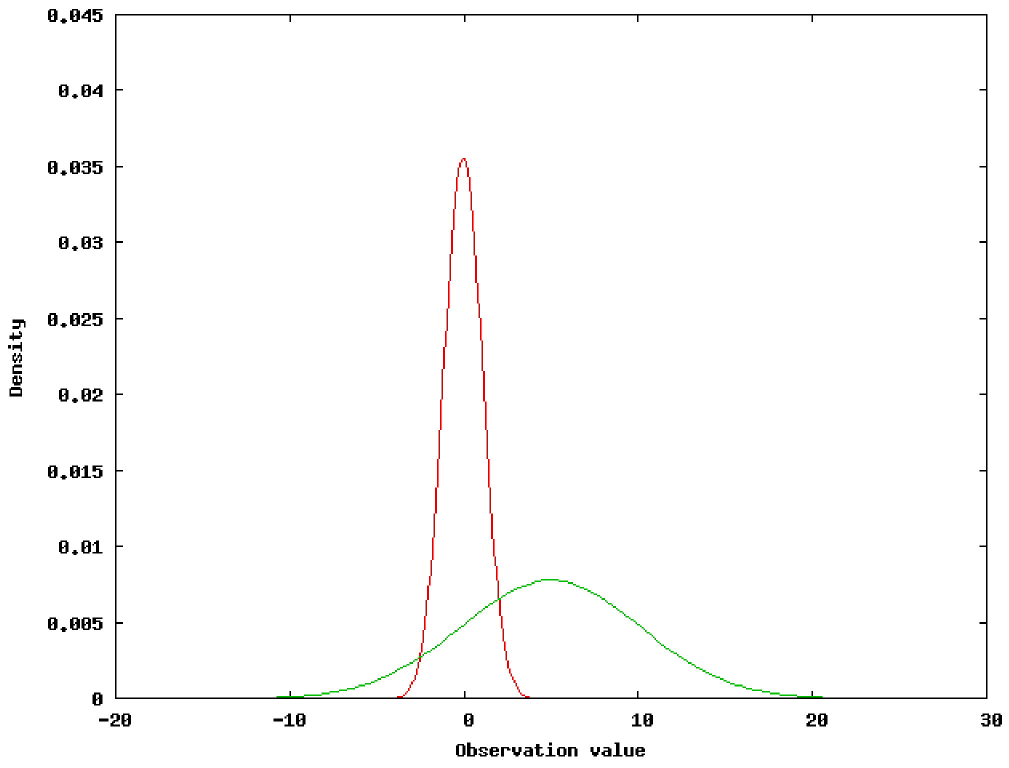


Figure 3.13: Two histograms of frequency density

in figure 3.13. The narrower Normal has the following parameters: $\mu = 0$ and $\sigma = 1$. The wider Normal has the following parameters: $\mu = 5$ and $\sigma = 5$. Consider that both define a variable where the higher, the better. If we provide the averages to people, everybody will choose the second, i.e. $\mu = 5$. However, having the standard deviations and the histograms plotted, we might observe that their observations assume similar values. Even worse, as the standard deviation of the second is higher, i.e. $\sigma = 5$, then it may generate lower values. In such circumstance, the gambler would probably choose the wider Normal, because there is chance of obtaining higher values. The conservatives would choose the narrower, because they would lose less. If we integrate the area of intersection, we could obtain the percentage that both assume the same values.

3.4 Hypothesis tests

3.4.1 One-tail test: distributions with known variance

Assume that a group of 30 people had the LDL cholesterol measured before and after a treatment. The group has presented the two Normal distributions depicted in figure 3.14. The Normal distribution $\mu = 190$ and $\sigma = 40$ was obtained before the treatment, and $\mu = 140$ and $\sigma = 40$ afterwards.

The objective now is to have evidences if the treatment was effective. If the results are close to 190, the treatment was not. However, if we notice that the observations tend to 140, then we confirm good results.

In such circumstance, we have people of the same group being evaluated twice, before and after a treatment. In this case, we might observe that there is a dependency among observations (same people). In those intervention situations, we consider the Paired T-test to evaluate the treatment efficiency or, basically, if the observations have changed.

In this circumstance, we consider a hypothesis test to evaluate if there is a significative change after the treatment. In order to do that, we consider the two following hypothesis:

$$H_0 = \text{the treatment is not efficient} \tag{3.7}$$

$$H_a = \text{the treatment is efficient} \tag{3.8}$$

This formalization defines the null hypothesis H_0 as: the 30 people did not present any significative measurement change. The second hypothesis, or alternative hypothesis, assumes that such people have presented a blood change and the treatment is efficient. Such hypothesis can also be written as:

$$H_0 : \mu = 190 \tag{3.9}$$

$$H_a : \mu = 140 \tag{3.10}$$

Table 3.5: Histogram generator – Java source code

```
import java.util.*;
import java.io.*;

public class Histogram {
        public static void main(String args[]) throws Exception {
                BufferedReader config = new BufferedReader(new FileReader(args[0]));
                BufferedReader histogram = new BufferedReader(new FileReader(args[1]));

                Vector range = new Vector();
                Vector value = new Vector();

                String str = null;

                do {
                        str = config.readLine();
                        if (str != null) {
                                range.add(new Double(str.trim()));
                                value.add(new Integer(0));
                        }
                } while (str != null);

                value.add(new Integer(0));

                do {
                        str = histogram.readLine();
                        boolean added = false;
                        int i = 0;

                        while (!added && i < range.size() && str != null) {
                                Double d = (Double) range.elementAt(i);
                                if ((new Double(str.trim())).doubleValue() <= d.doubleValue()) {
                                        Integer count = (Integer) value.elementAt(i);
                                        value.removeElementAt(i);
                                        value.add(i, new Integer(count.intValue() + 1));
                                        added = true;
                                }
                                i++;
                        }

                        if (!added && i == range.size() && str != null) {
                                Integer count = (Integer) value.lastElement();
                                value.remove(value.indexOf(value.lastElement()));
                                value.add(new Integer(count.intValue() + 1));
                        }

                }while (str != null);

                double lrange = 0;

                for (int i = 0; i < range.size(); i++) {
                        Double d = (Double) range.elementAt(i);
                        Integer v = (Integer) value.elementAt(i);

                        System.out.println(lrange+" -| "+d.doubleValue()+" -> "+v.intValue());

                        lrange = d.doubleValue();
                }

                Integer last = (Integer) value.lastElement();
                System.out.println(lrange+" -| inf -> "+last);
        }
}
```

Then, the null hypothesis assumes no change, while the alternative one considers that the sample average is equal to 140. However, the best way of describing them is by using the following:

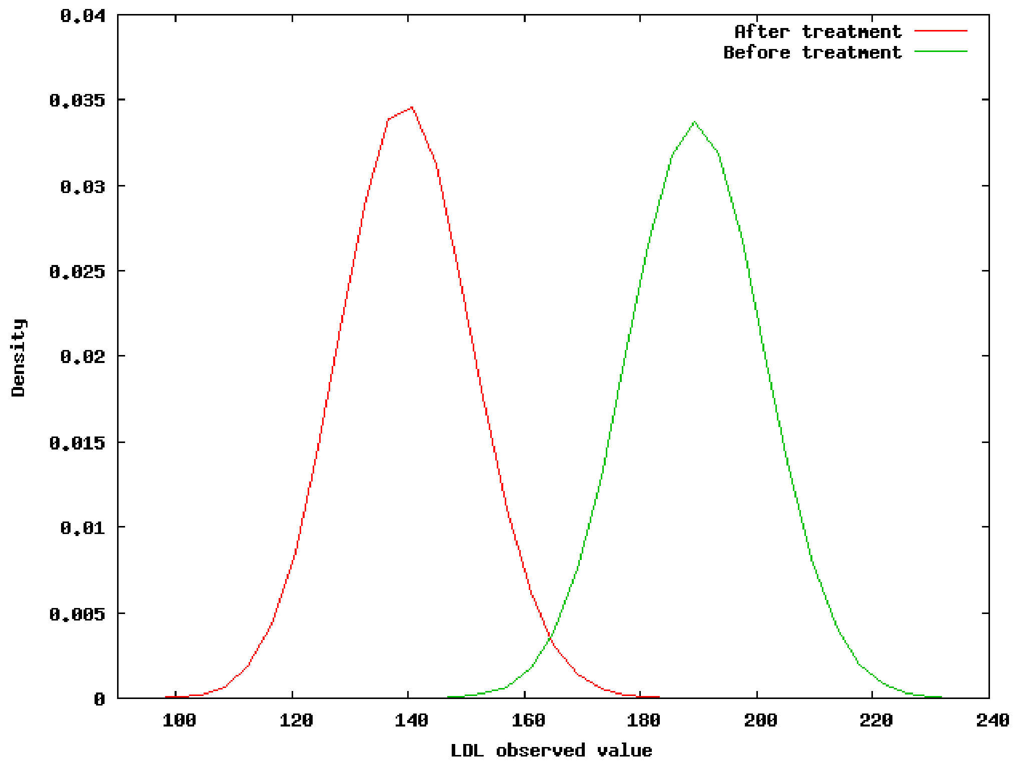


Figure 3.14: The frequency density of LDL before and after treatment

$$H_0 : \mu = 190 \tag{3.11}$$
$$H_a : \mu \leq 190 \tag{3.12}$$

This last manner is more precise, because any significative change to a lower value depicts that the treatment is efficient. In this situation, as we wish a lower average, we use an one-tail test to evaluate the hypothesis. The one tail is clearly recognized when we see this $\leq$ in the alternative hypothesis, what means that the lower the value is, the more H_a is accepted. If we consider that the treatment could worsen the disease, then we would consider a two-tail test as follows:

$$H_0 : \mu = 190 \tag{3.13}$$
$$H_a : \mu \neq 190 \tag{3.14}$$

As you may notice, the null hypothesis always describes no change on data. Before applying the hypothesis test, we need to address the two types of errors that might occur:

1. Type I error – reject hypothesis H_0, when it is true;
2. Type II error – do not reject H_0, when it should be.

Those errors can also be expressed in terms of probability as follows:

$$\alpha = P(\text{type I error}) = P(\text{reject} H_0 | H_0 \text{is true}) \tag{3.15}$$

$$\beta = P(\text{type II error}) = P(\text{not reject} H_0 | H_0 \text{is false}) \tag{3.16}$$

For the one-tail LDL situation, we could interpret the errors as follows:

$$\alpha = P(\text{accept the treatment as valid when it is not}) \tag{3.17}$$

$$\beta = P(\text{reject the treatment when, in fact, it is efficient}) \tag{3.18}$$

It is also important to notice that no problem occurs when we reject H_0 and it is false or when we accept it and it is true.

Figure 3.15 presents the LDL blood distributions and the respective influence of the type I and II errors. When we define x_c, we therefore select an area to confirm the hypothesis H_0 or reject it. In this case, H_0 defines that there is no change in people blood after the treatment, this is, they keep having $\mu = 190$. If new blood average is lower than x_c, then we reject H_0 and, therefore, accept H_a, this means that the treatment was efficient. Based on this example, you might observe that as we reduce the area α, by moving x_c to left, we increase β and vice versa. This means there is a trade-off in between accepting or not the null hypothesis. The ideal situation would be having both areas equal to zero.

Once we consider this limitation, we might observe that most of researchers pay more attention on the type I error, once it confirms that something has significantly changed from the initial situation. Then, they define the probability of α, or its area, as the significance level of the test. Researchers usually define an area for α (data under such area confirms the rejection of H_0) and based on such assumption they find x_c. Equation 3.19 is employed to find x_c where μ_{new} is the new average after treatment, μ is the average of the distribution considered in H_0, which is 190 here, σ is the standard deviation and n is the sample size, in this case $n = 30$. The term $z_c(\alpha)$ defines the number of standard deviations from average μ to consider when integrating the area α.

$$z_c(\alpha) = \frac{\mu_{new} - \mu}{\sigma/\sqrt{n}} \tag{3.19}$$

By applying equation 3.19, we would have $z_c(\alpha) = \frac{x_c - 190}{40/sqrt30}$. People usually consider $\alpha = 0.05$ what means that if the new average tends, in this case, to the extreme left of the distribution and considers only 5% of data, the new average does not belongs to the distribution of H_0. If we integrate 5% of the area of the Normal (using the code in table 3.6), we would notice that it is 1.64 standard deviation from the average 190, this

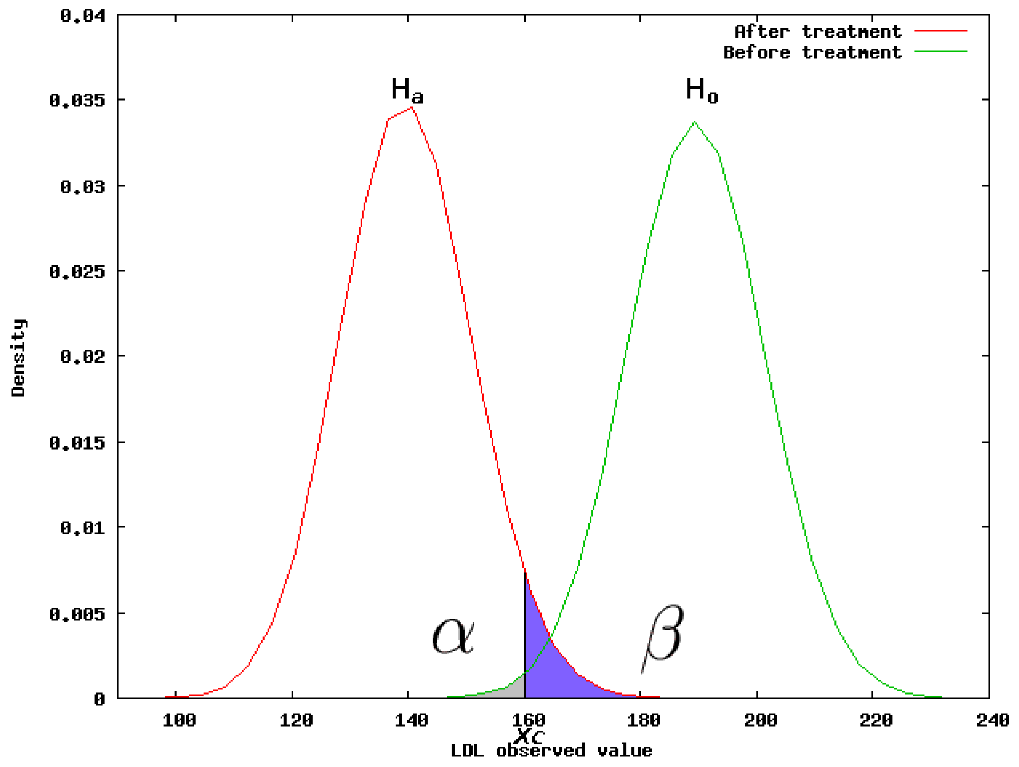


Figure 3.15: The frequency density of LDL: Two types of errors

means that $z_c(\alpha = 0.05) = 1.64$. As we consider data to the left of H_0, then we use the minus signal what would be -1.64. In such circumstance, we would have the following equation $x_c = 190 + z_c(\alpha) \cdot \frac{40}{\sqrt{30}}$, therefore, $x_c = 190 - 1.64 \cdot \frac{40}{\sqrt{30}}$ where $x_c = 178.02$. Any value lower than 178.02 would be under the area α and would not be considered in the distribution $\mu = 190$. If the new average is lower than that, we can reject H_0 and, therefore, accept H_a, what means that the treatment was efficient. Researchers usually consider $\alpha \in [0.01, 0.05]$ to confirm if the sample average has significantly changed. If so, they confirm the alternative hypothesis. In this case, the new average is 140, consequently, the treatment has modified the blood measurement to a value lower than 178.02 what means we can reject H_0 and accept H_a. Accepting the alternative hypothesis, we confirm the treatment efficiency.

We can also compute the type II error by using the same equation 3.19. In such situation we want to compute the integral of the area β of distribution $\mu = 140$ and $\sigma = 40$. The first step is to find z_c what means the number of standard deviations that the critical point x_c is from the average. So, we apply the following equation $z_c(\beta) = \frac{x_c - 140}{40/sqrt30}$. Considering we have $x_c = 178.02$, therefore, $z_c(\beta) = \frac{178.02-140}{40/sqrt30}$ what results in $z_c(\beta) = 5.206$, this means that the area β is to the right side of the point $\mu + 5.206$ standard deviations. If the integrate such area β we might conclude $\beta < 1.0 \cdot 10^{-5}$. The area β contains the observations of the new distribution which are wrongly refered as belonging to $\mu = 190$. Then, in this case, there is a small probability $\beta < 1.0 \cdot 10^{-5}$ to classify an observation of the new distribution into the old one.

Table 3.6: Integrating the Normal – Java source code

```
import java.io.*;
import java.util.*;

public class IntegrateZTableOneTail {

        public static void main(String args[]) throws Exception {
                BufferedReader file = new BufferedReader(new FileReader(args[0]));
                double average = (new Double(args[1])).doubleValue();
                double stdev = (new Double(args[2])).doubleValue();
                double percentage = (new Double(args[3])).doubleValue();
                BufferedReader histogram = new BufferedReader(new FileReader(args[4]));

                String str = null;
                long count = 0; //number of points
                long inside = 0;

                do {
                        str = file.readLine();
                        if (str != null) {
                                count++;
                        }
                } while (str != null);

                double start = 0;
                double end = 0;
                int c = 0;

                do {
                        str = histogram.readLine();
                        String[] array = new String[3];
                        int p = 0;

                        StringTokenizer st = new StringTokenizer(str);
                        while (st.hasMoreTokens()) {
                                String s = st.nextToken();
                                if (!s.equals("-|") && !s.equals("->"))
                                        array[p++] = s;
                        }

                        if (str != null) {
                                start = (new Double(array[0])).doubleValue();
                                end = (new Double(array[1])).doubleValue();
                                c = (new Integer(array[2])).intValue();

                                if (((inside+c)/(count*1.0)) < percentage) {
                                        inside+=c;
                                } else {
                                        break;
                                }
                        }
                } while (str != null);

                System.out.println("percentage: "+percentage+" z_c: "+start);
        }
}
```

3.4.2 Two-tail test: distributions with known variance

Let the same treatment, presented in the last section, be considered by another researcher who wants to evaluate improvements and bad results. In such situation, the hypothesis would be different. The alternative would observe any change (to a greater or lower value) in the new average and not only changes lower than $\mu = 190$, as follows:

$$H_0 : \mu = 190 \tag{3.20}$$
$$H_a : \mu \neq 190 \tag{3.21}$$

In such situation, we consider two critical regions of type I error, one on each tail of the Normal distribution. Thus, the critical region is given by:

$$RC = \{x \in \mathbb{R} : x < x_{c_1} or x > x_{c_2}\} \tag{3.22}$$

As the researcher considers the treatment can give good or bad results, the operator $\neq$ is adopted in the hypothesis formalization. After defining the two critical points x_{c_1} and x_{c_2}, we need to define the area α in which we will reject H_0. Any observation value in the critical regions will not be considered in the same distribution assumed by H_0. Let's assume $\alpha = 0.05$ and the same situation before treatment, where $\mu = 190$ and $\sigma = 40$. The area $\alpha = 0.05$ is then divided by two (figure 3.16), where we will have 0.025 on each tail, what is obtained after integrating the Normal distribution. On the left tail we have $\alpha = 0.025$ and, therefore, $z_{c_1} = -1.96$. On the right tail we have $\alpha = 0.025$ and, consequently, $z_{c_2} = +1.96$.

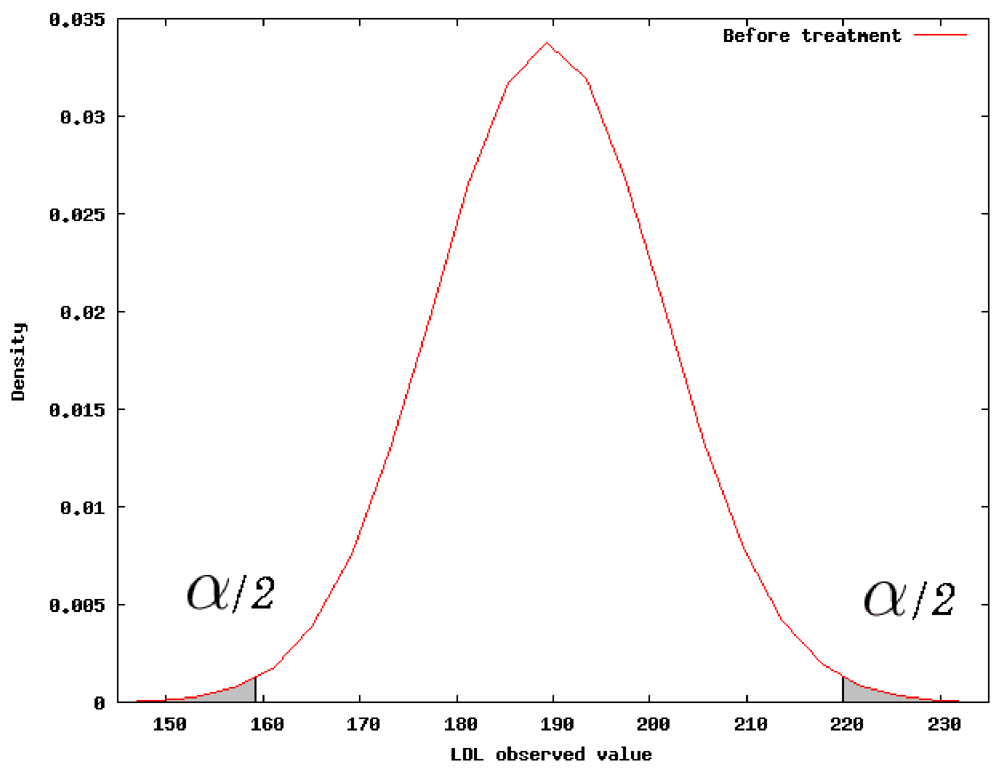


Figure 3.16: The area α for the two-tail test

Then we have:

$$P(\text{type I error}) = 0.05 \tag{3.23}$$
$$= P(\mu_{new} < x_{c_1} or \mu_{new} > x_{c_2}) \tag{3.24}$$
$$= P(Z < z_{c_1} or Z > z_{c_2}) \tag{3.25}$$

and, therefore $z_{c_j}(\frac{\alpha}{2}) = \frac{x_{c_j}-190}{40/\sqrt{30}}$ where $j = 1, 2$. After integrating $\frac{\alpha}{2}$ on each side we obtain $z_{c_1} = -1.96$ and $z_{c_2} = +1.96$. Consequently, we have:

$$x_{c_1} = 190 - 1.96 \cdot \frac{40}{\sqrt{30}} = 175.69 \tag{3.26}$$
$$x_{c_2} = 190 + 1.96 \cdot \frac{40}{\sqrt{30}} = 204.32 \tag{3.27}$$

In this way, the critical region is given by:

$$RC = \{x \in \mathbb{R} : x < 175.69 or x > 204.32\} \tag{3.28}$$

If we consider the new average $\mu = 140$, we observe it is lower than x_{c_1}, consequently, we reject H_0 and confirm the treatment efficiency. Any average lower than x_{c_1} or greater than x_{c_2} is not considered in the same distribution as the Normal $\mu = 190$.

3.4.3 Central limit theorem

Another important concept when dealing with the Normal distribution is the Central Limit Theorem, which assumes that large samples of data may behave as a Normal distribution. This is very useful when making experiments which are used to evaluate new techniques such as schedulers, optimizers, etc. Based on that, let's consider figures 3.17, 3.18, 3.19, 3.20, 3.21 and 3.22. They present the density histograms for different sample sizes. We may observe that, as we increase the dataset size, more they tend to Normal distributions. This assumption is valid for variables which behave as a Normal.

Such assumption supports the generalization of concepts. If we need, for instance, compute the standard deviation of a distribution, we might assume it follows a Normal and, based on the sample size, we approximate such measurement. This concept is considered in the next section.

3.4.4 Tests using distributions with unknown variances

We can conduct the hypothesis tests for samples in which we do not know the variance. In order to do that, we apply equation 3.4.5 ($\bar{x}$ is used to inform that the average considers only a data sample) which approximates the variance value by exploring the observations.

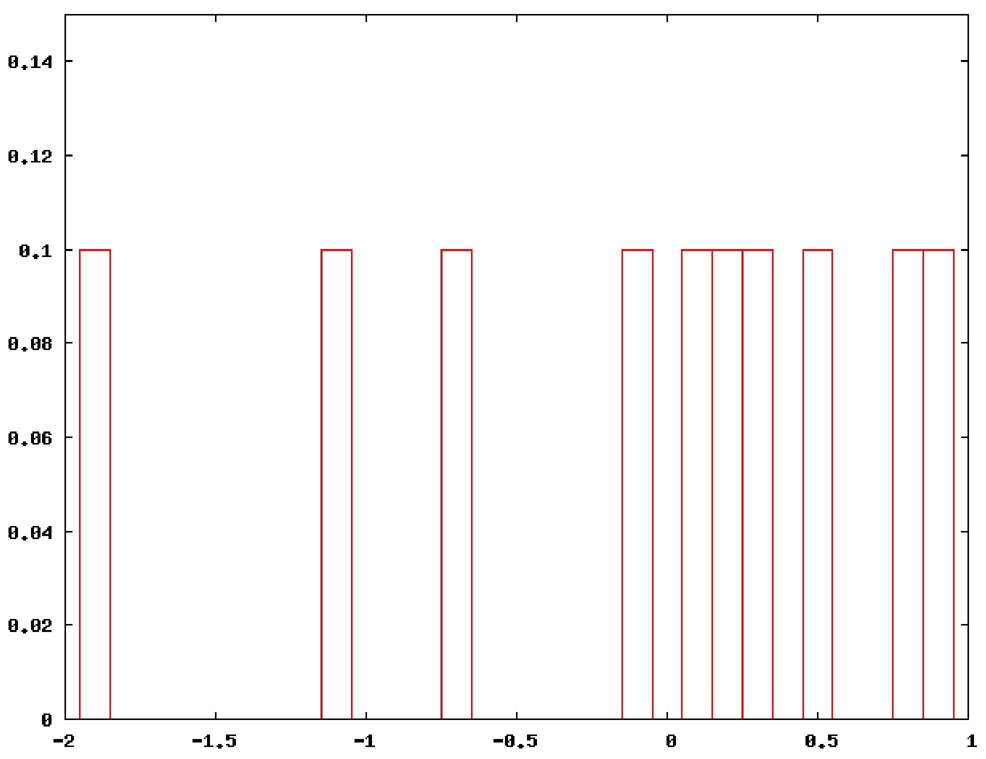


Figure 3.17: Data sample with 10 points

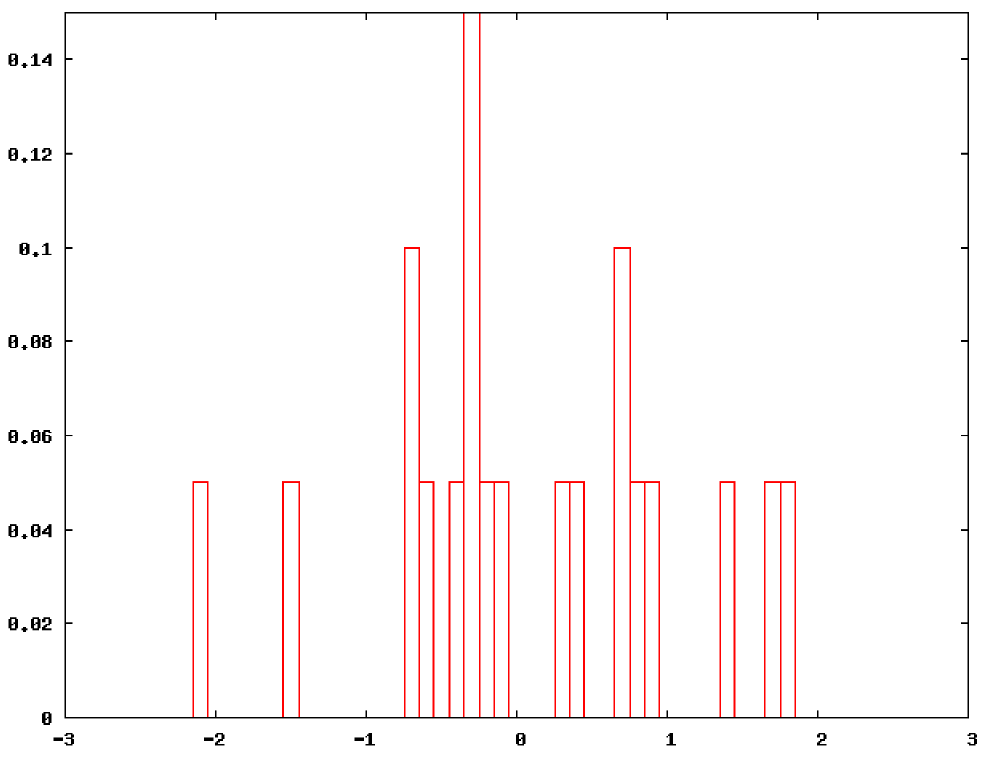


Figure 3.18: Data sample with 20 points

The term $n-1$ helps to approximate the variance by increasing it. As many observations we have, the more precise the test is.

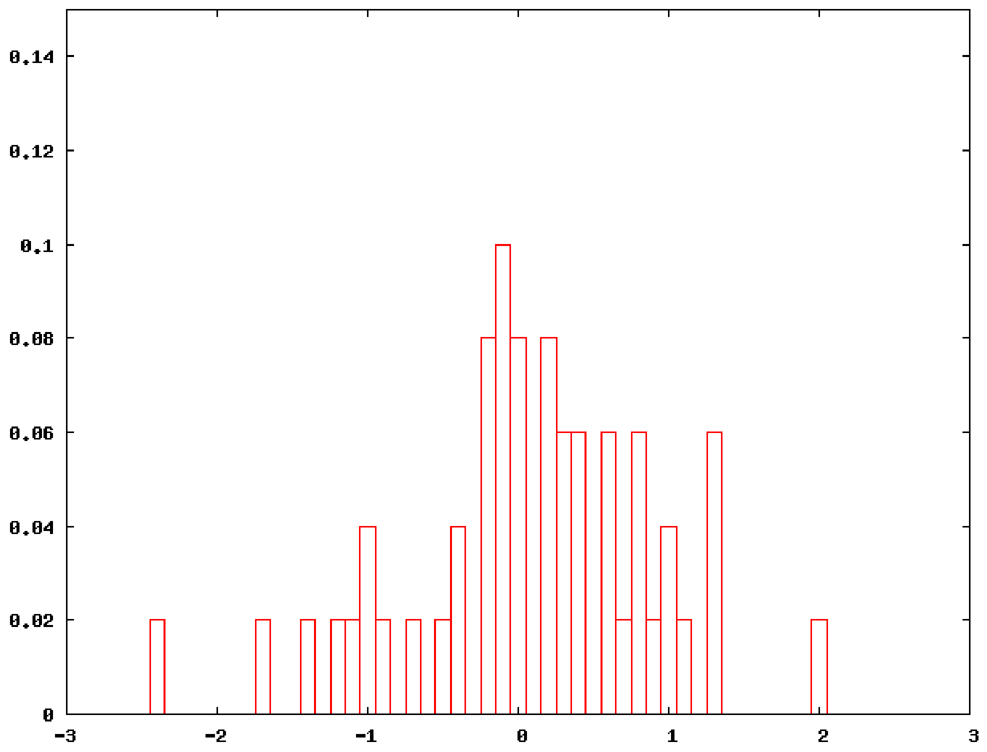


Figure 3.19: Data sample with 50 points

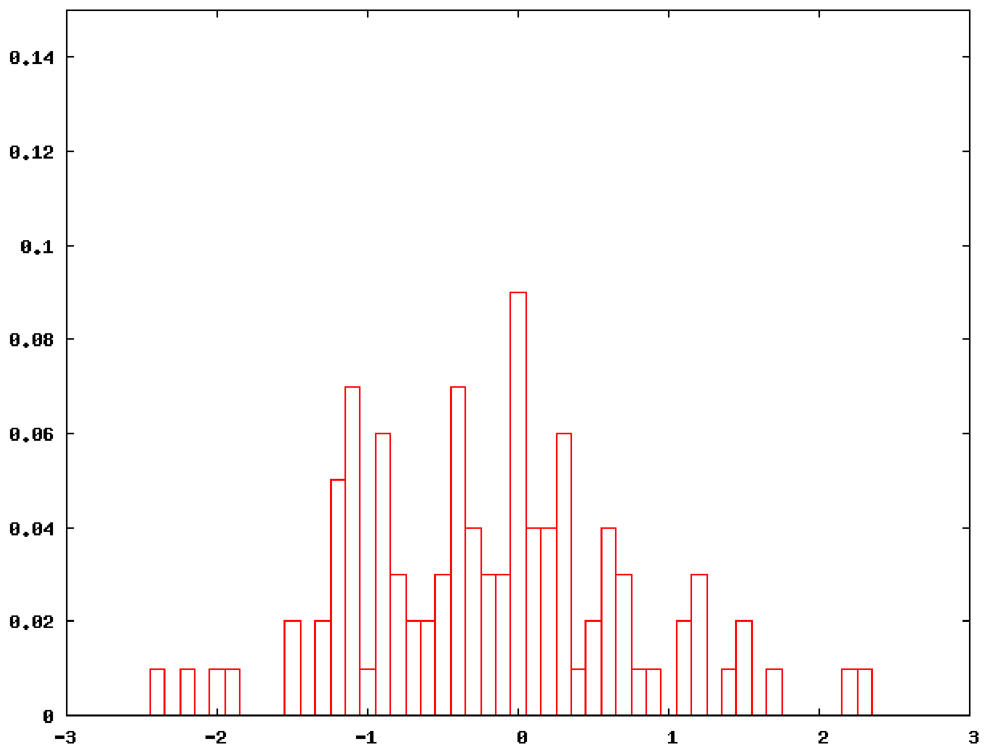


Figure 3.20: Data sample with 100 points

$$S^2 = \frac{\sum_{i=1}^{n} x_i^2 - n \cdot \bar{x}^2}{n-1} \tag{3.29}$$

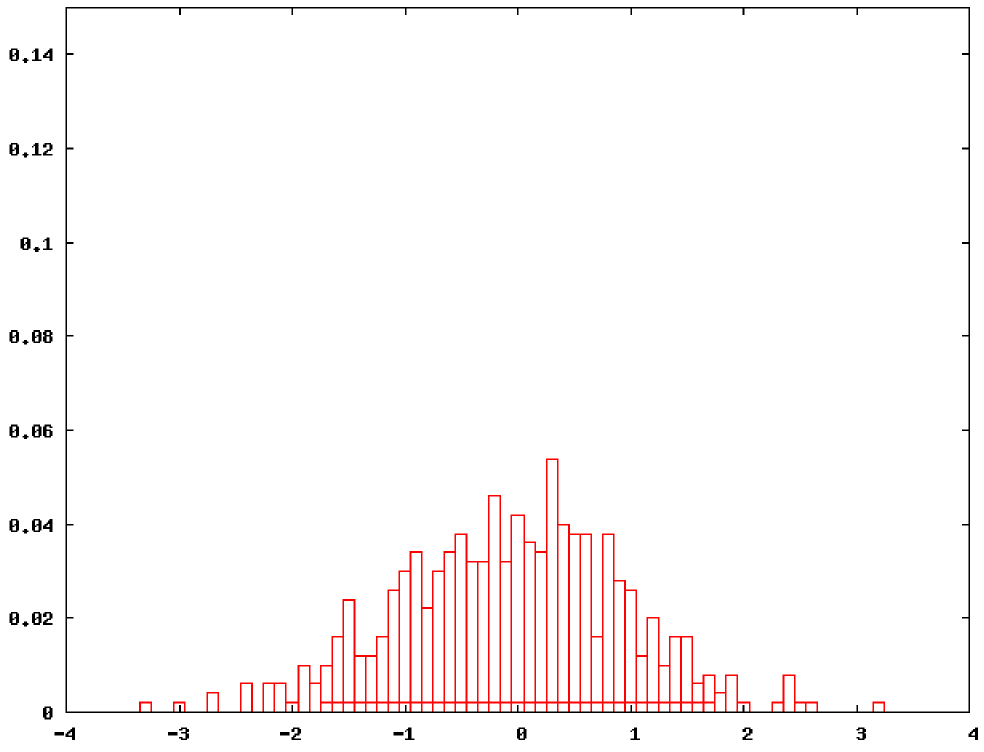


Figure 3.21: Data sample with 500 points

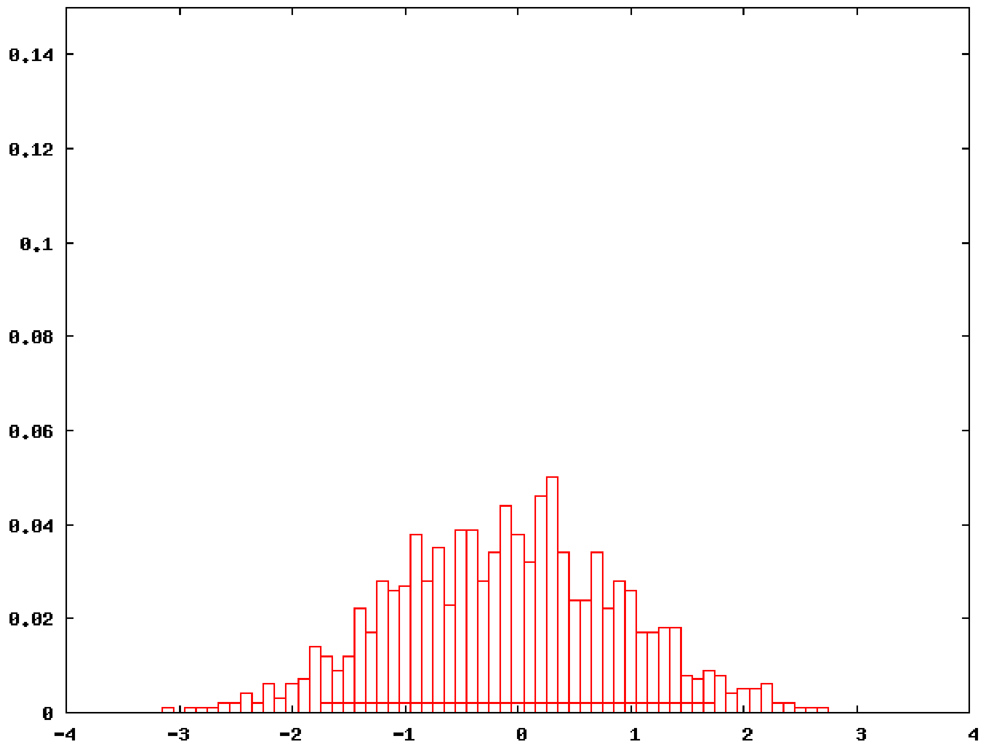


Figure 3.22: Data sample with 1000 points

After defining S^2, we find the number of standard deviations T, according to equation 3.30. T can be used in the same way z_c is (sections 3.4.1 and 3.4.2), however, instead of integrating the Normal distribution, we must consider the t-Student distribution. The latter considers different dataset sizes, which are called degrees of freedom. We can integrate areas α as we did for the Normal distribution, however, we need to compute it many times for each dataset size (sz), such as k times with $sz = \{2, 3, 4, \ldots, 121\}$. Those dataset sizes result, respectively, in the following degrees of freedom $df = \{1, 2, 3, \ldots, 120\}$.

Basically, degrees of freedom define different areas for the t-Student distribution. The more observations we have, the more t-Student tends to behave as the Normal distribution. From $df = 120$ on, we consider the integration of the Normal to define the areas α and β.

$$T = \frac{\bar{x} - \mu}{S/\sqrt{n}} \tag{3.30}$$

Let's compute T for a certain situation. Consider that the average workload of a system follows the Normal distribution where $\mu = 5.0$. The system was observed for some time after installing a new load balancing toolkit. Then, we obtained the following $n = 6$ observations: 1.5, 8.2, 4.3, 6.2, 5.2 and 4.5. In such situation, the test is given by:

$$H_0 = \text{there is no change after the toolkit installation} \tag{3.31}$$
$$H_a = \text{there is some change, and, therefore, the toolkit has improved the efficiency} \tag{3.32}$$

Then we have:

$$H_0 : \mu = 5.0 \tag{3.33}$$
$$H_a : \mu \neq 5.0 \tag{3.34}$$

The operator $\neq$ defines the two-tail test. As we only have a small sample and the workload follows a Normal, then we consider the t-Student distribution. Having 6 observations, we consider $df = 6 - 1 = 5$ degrees of freedom. The two-tail test is defined as follows:

$$RC = \{t \in \mathbb{R} : t < t_1 ort > t_2\} \tag{3.35}$$

The terms t_1 and t_2 are similar to z_{c_1} and z_{c_2}, where they define the critical regions for the t-Student. The first step is to compute T according to equation 3.30, however we need firstly to compute S^2 (equation 3.4.5). So, $S^2 = \frac{(1.5^2+8.2^2+4.3^2+6.2^2+5.2^2+4.5^2)-6\cdot 4.9833^2}{6-1}$, where the sample average is $\bar{x} = 4.9833$ and $S^2 = 4.9420$. Afterwards, we compute $T = \frac{\bar{x}-5.0}{S/\sqrt{n}}$, and $T = \frac{4.9833-5.0}{2.2230/\sqrt{6}}$, so, $T = -0.0184$.

However, now we need to define the area α to reject H_0. Let's assume $\alpha = 0.01$. Consulting the t-Student table, we find that in a two-tail test each distribution tail will contain $\frac{0.01}{2} = 0.005$, or, 0.5% of probability. Going to the table in $df = 5$ we obtain $t_1 = -6.869$ and $t_2 = 6.869$, then:

$$RC = \{t \in \mathbb{R} : t < -6.869 or t > 6.869\} \tag{3.36}$$

Consequently, we have obtained $T = -0.0184$ which does not belongs to RC, therefore, we cannot reject H_0, consequently, the toolkit was not proven to be efficient assuming $\alpha = 0.01$ (we can assume other areas for the t-Student distribution).

3.4.5 Confidence interval

Besides standard deviation and variance, there is another important estimator which is called confidence interval. It considers the same approach of a two-tail test, which is used to understand how data range varies. For example, let's consider the LDL blood measurements presented before where the LDL average is $\mu = 140$. Then, consider a 95% confidence interval. Suppose we have two results. The first presents a confidence interval equals to 140 ± 5 while a second group of people has 140 ± 10. This means that 95% of the distribution area in under the interval $[135, 145]$ for the first and $[130, 140]$ as presented in figure 3.23.

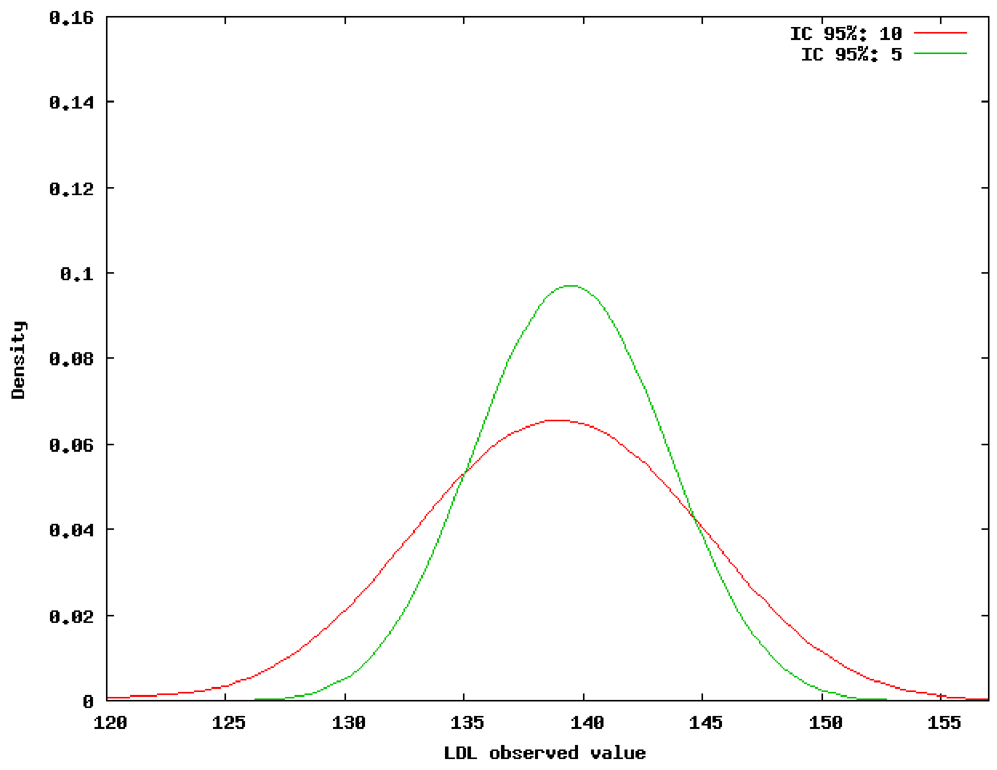


Figure 3.23: Two Normal distributions and their confidence intervals

We observe the values for the distribution 140 ± 5 are more concentrated than for 140 ± 10 what gives more precision or reliability on data. Let's consider you make experiments using two techniques, one of them has presented higher concentration, while the other has a very wide confidence interval. The first is more reliable, because you have a better idea about what to expect of the technique.

This is similar to what survey companies do in elections. They collect people opinion and create distributions. Right after, they define the confidence intervals, which help to understand how concentrated they are around the average.

Equation **??** presents how to compute the confidence interval for a distribution. It is basically the same one used to compute the critical region for a hypothesis test. Actually, it considers the same idea, however, only to understand the data distribution. If you do not know σ, you may compute it using the approximation provided by equation , which considers small samples.

$$Z = \frac{x_c - \mu}{\sigma/\sqrt{n}} \tag{3.37}$$

Then you must decide on the relevant area under the curve. Most of researchers consider 0.95 for the confidence interval, what means that they integrate 95% of the distribution area and find the critical points at the left and right sides. Consequently the interval is computed as follows:

$$IC(\mu, \gamma) = [\mu - z_1 \cdot \frac{\sigma}{\sqrt{n}}; \mu + z_2 \cdot \frac{\sigma}{\sqrt{n}}] \tag{3.38}$$

Let's go back to the LDL blood measurements. Consider we want to compute IC for two Normal distributions with 100 observations each: $\mu = 190$, $\sigma = 5$ and $\mu = 190$, $\sigma = 10$. Then we would obtain:

$$IC_{\sigma=5}(190, 0.95) = [190 - z_{\gamma/2} \cdot \frac{5}{\sqrt{100}}; 190 + z_{\gamma/2} \cdot \frac{5}{\sqrt{100}}] \tag{3.39}$$

$$IC_{\sigma=10}(190, 0.95) = [190 - z_{\gamma/2} \cdot \frac{10}{\sqrt{100}}; 190 + z_{\gamma/2} \cdot \frac{10}{\sqrt{100}}] \tag{3.40}$$

As we consider 0.95, we will reject 0.05 of distributions data, this is, $\frac{0.05}{2}$ on each side. By integrating the Normal distribution we may observe that the critical points z_j, where $j = 1, 2$, are $z_1 = -1.96$ and $z_2 = +1.96$. By substituting them at the equations, we obtain:

$$IC_{\sigma=5}(190, 0.95) = [190 - 1.96 \cdot \frac{5}{\sqrt{100}}; 190 + 1.96 \cdot \frac{5}{\sqrt{100}}] \tag{3.41}$$

$$IC_{\sigma=10}(190, 0.95) = [190 - 1.96 \cdot \frac{10}{\sqrt{100}}; 190 + 1.96 \cdot \frac{10}{\sqrt{100}}] \tag{3.42}$$

Consequently, we have:

Table 3.7: Occurence table of system faults

Category	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Observed Occurrences	1	5	10	20	11	4	2

Table 3.8: Occurrence table of system faults, including the Normal observations

Category	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Observed Occurrences	1	5	10	20	11	4	2
Normal Occurrences	1	7	5	18	13	5	4

$$IC_{\sigma=5}(190, 0.95) = [189.02, 190.98] \tag{3.43}$$

$$IC_{\sigma=10}(190, 0.95) = [188.04, 191.96] \tag{3.44}$$

We may conclude that the first one has lower variation on observations than the second. Lower variations here, implies in higher reliability. However, it is important to observe that, to be fair, we need to compute IC considering the same sample sizes, otherwise they are not too reliable. A good size to start with is 30 observations.

3.4.6 Chi-Square Test

Besides the Normal distribution, there are lots of other probability density functions (PDF). Sometimes we have data samples, but we have no idea how they behave, this means we are not sure that they would be correctly represented by a Normal. In such situations, we can use the Chi-Square test to evaluate how data fit to some of the known distributions: Normal, Exponential, Poisson, Bernoulli, etc.

In order to better understand, consider the occurrence table 3.7 obtained from 53 observations of time in between system faults (in hours). We want, then, decide how close it behaves to a Normal distribution $N(\mu = 20, \sigma = 10)$. Then, we generate data (the same 53 observations – code in table 1) for the Normal distribution and obtained the comparative table 3.8. The two samples are presented in figure 3.24.

After generating 53 observations for the Normal distribution, we need now to measure the difference in each interval of table 3.8. A basic statistical rule asks for joining intervals with less than 5 observations, what would result in table 3.9. We unified the intervals 1 and 2 and also the intervals 6 and 7.

Table 3.9: Occurrence table of system faults, including the Normal observations and joining intervals

Category	**1**	**2**	**3**	**4**	**5**
Observed Occurrences	6	10	20	11	6
Normal Occurrences	8	5	18	13	9

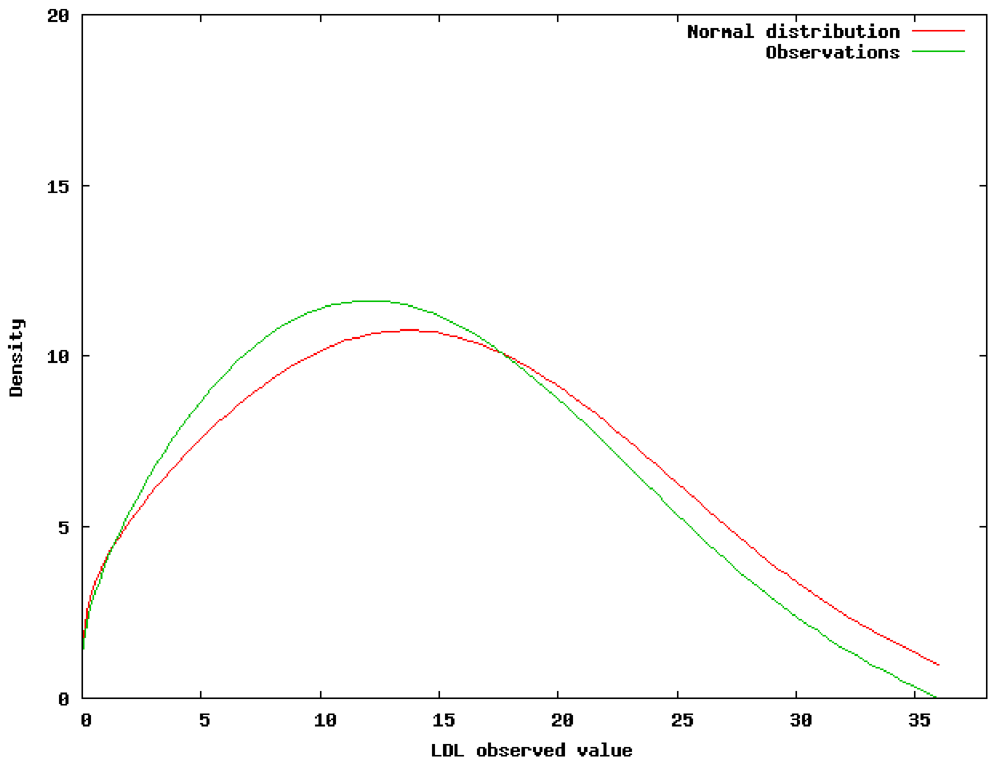


Figure 3.24: The Observed and Generated Occurrences

Now, we compute the Q^2 (equation 3.45) to evaluate how distant the observed value (o_i) is from the expected one (e_i) for each interval (being k intervals). The difference of the observed and expected values are powered by itself, as this avoids that negative and positive values influence in the computation. By dividing by e_i, we reduce the scale of Q^2. Then, we obtain $\frac{(6-8)^2}{8} + \frac{(10-5)^2}{5} + \frac{(20-18)^2}{18} + \frac{(11-13)^2}{13} + \frac{(6-9)^2}{9} = 7.03$.

$$Q^2 = \sum_{i-1}^{k} \frac{(o_i - e_i)^2}{e_i} \tag{3.45}$$

It is important to observe that the values of Q^2 also follow a distribution, in this case, the Chi-Square. Consequently, Q^2 defines a point in the Chi-Square distribution, which can be used to propose a hypothesis test:

$$H_0 = \text{the observations behave as a Normal } N(\mu = 20, \sigma = 10) \tag{3.46}$$

$$H_a = \text{the Normal } N(\mu = 20, \sigma = 10) \text{ is not adequate to represent the observed data} \tag{3.47}$$

Then:

$$H_0 : Q^2 \leq q_c H_a : Q^2 > q_c \tag{3.48}$$

Consequently, we need to define the critical area to be integrated (right side of q_c). It is defined according to the degree of freedom for the distribution. In the presented situation, the degree is given by the number of intervals minus one, this is, $5 - 1 = 4$. Then, we select the rejection area for this Chi-Square distribution and go to the Chi-Square table, finding $q_c = 9.49$. Figure 3.25 presents the integration of the critical region with 0.05, this is, 5% of the area, which is consider here. As $Q^2 = 7.03$ is lower than 9.49, what means that the Normal model is accepted to represent the 53 fault observations of the system.

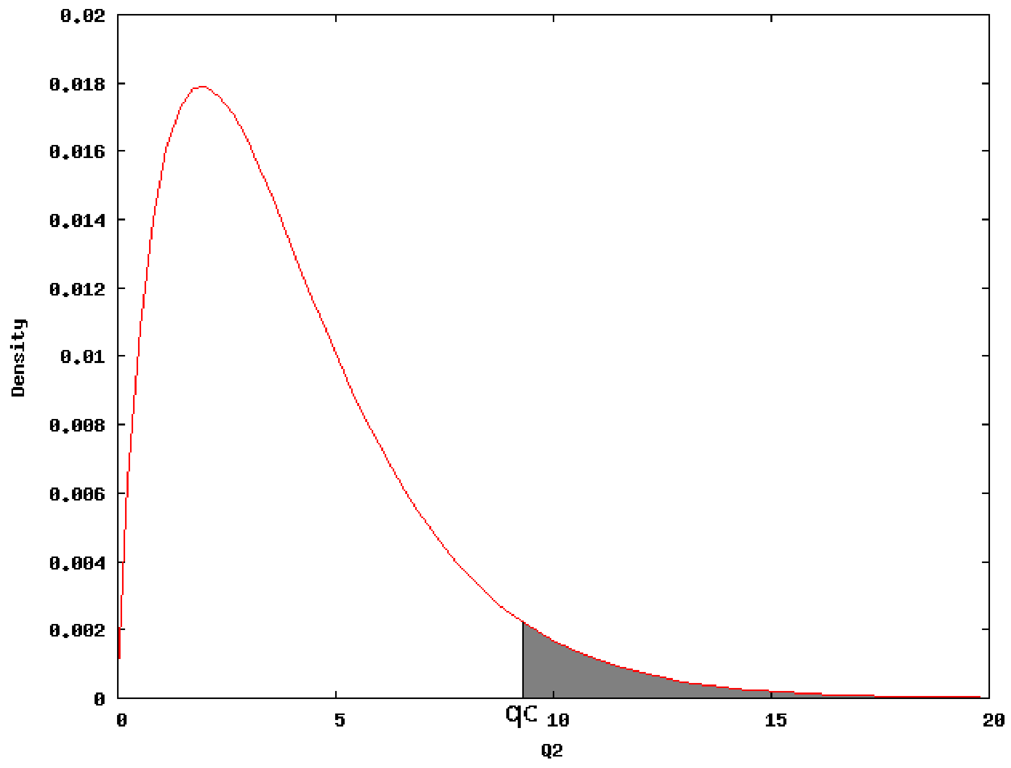


Figure 3.25: Chi-Square density distribution with 4 degrees of freedom

3.4.7 Receiver Operating Characteristic

Let's consider you have a problem to solve using any statistical, artificial intelligence or mathematical technique. To better instantiate the problem, consider a signature verifier. It may return that a valid signature is true, that the same is false, and also that a fake signature is true or false. When you conclude that a valid situation is true, we say that this is a true positive. When you find the answer that a valid situation is false, you have a false negative. Having a fake situation as true, it means we have a false positive. The fake situation classified as false means a true negative.

Table 3.10 summarizes all the possibilities for our signature classifier. People usually define the accuracy of a classifier by using equation 3.49. In order to better understand the application, let's consider two different signature classifier with the results presented in tables 3.11 and 3.12. They present the classification results for 100 signatures. Computing

Table 3.10: Types of classification

		Actual value		
		positive	**negative**	
Prediction	**positive'**	true positive	false positive	P'
Outcome	**negative'**	false negative	true negative	N'
		P	**N**	

Table 3.11: Signature classifier 1

		Actual value		
		positive	**negative**	
Prediction	**positive'**	true positive = 60	false positive = 30	P' = 90
Outcome	**negative'**	false negative = 40	true negative = 70	N' = 110
		P = 100	**N** = 100	

the accuracy we obtain $acc_1 = \frac{60+70}{100+100} = 0.65$ and $acc_2 = \frac{75+50}{100+100} = 0.625$. From the results we may conclude that the first classifier is better than the second. This is a very common measurement considered when comparing those binary classifiers.

$$ACC = \frac{TP + TN}{P + N} \tag{3.49}$$

Besides the accuracy, the classifiers can be compared using the precision measurement, which is presented in equation 3.50. In this situation, the precision would be $PREC_1 = \frac{60}{60+30} = 0.67$ for the first classifier and $PREC_2 = \frac{75}{75+50} = 0.60$ for the second. The first classifier is more precise than the second. This means how many correct positive classifications were done considering all the data predicted as positive.

$$PREC = TP/(TP + FP) \tag{3.50}$$

Besides the prediction, there is the recall, or true positive rate, which is defined in equation 3.51. For the two classifiers, the recall results are $REC_1 = \frac{60}{60+40} = 0.60$ and $REC_2 = \frac{75}{75+25} = 0.75$. This returns the fraction of the truly and correctly classified signatures over the ones that were correclt classified as true and the ones wrongly classified as false.

$$REC = TP/P = TP/(TP + FN) \tag{3.51}$$

Table 3.12: Signature classifier 2

		Actual value		
		positive	**negative**	
Prediction	**positive'**	true positive = 75	false positive = 50	P' = 125
Outcome	**negative'**	false negative = 25	true negative = 50	N' = 75
		P = 100	**N** = 200	

The greater the precision and recall, the better the results are. As the accuracy, they are very common measurements to compare classifiers. There is still another important measurement, this is the false positive rate, which is defined in equation 3.52. It summarizes the false positive divided by the false positive plus the true negative ones. In this situation, the classifiers would have $FPR_1 = \frac{30}{30+70} = 0.3$ and $FPR_2 = \frac{50}{50+50} = 0.5$.

$$FPR = FP/N = FP/(FP + TN) \tag{3.52}$$

A way of summarizing part of such results is the Receiver Operating Characteristic (ROC) space. It plots a two dimensional space using FPR in x-axis and REC in y-axis. Figure 3.26 presents the example considering the two previously defined classifiers.

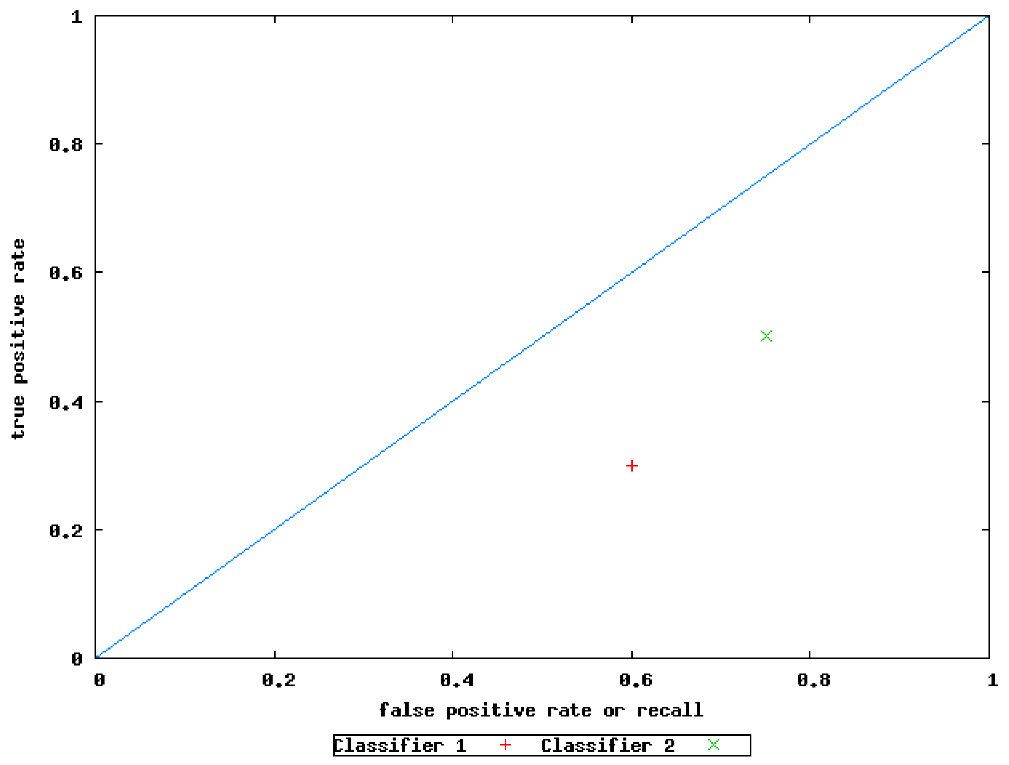


Figure 3.26: Receiver Operating Characteristic Space

Both classifiers are below the diagonal this means their results are not really good, because any random guess could present results on the diagonal. Now, consider a third classifier as presented in figure 3.27, which present better results than the random guess and the other techniques. The best result at all is also presented in the same figure, which is the false positive rate 0 and the true positive rate of 1. This means that all right signatures were classified as right and that all fake signatures were classified as false.

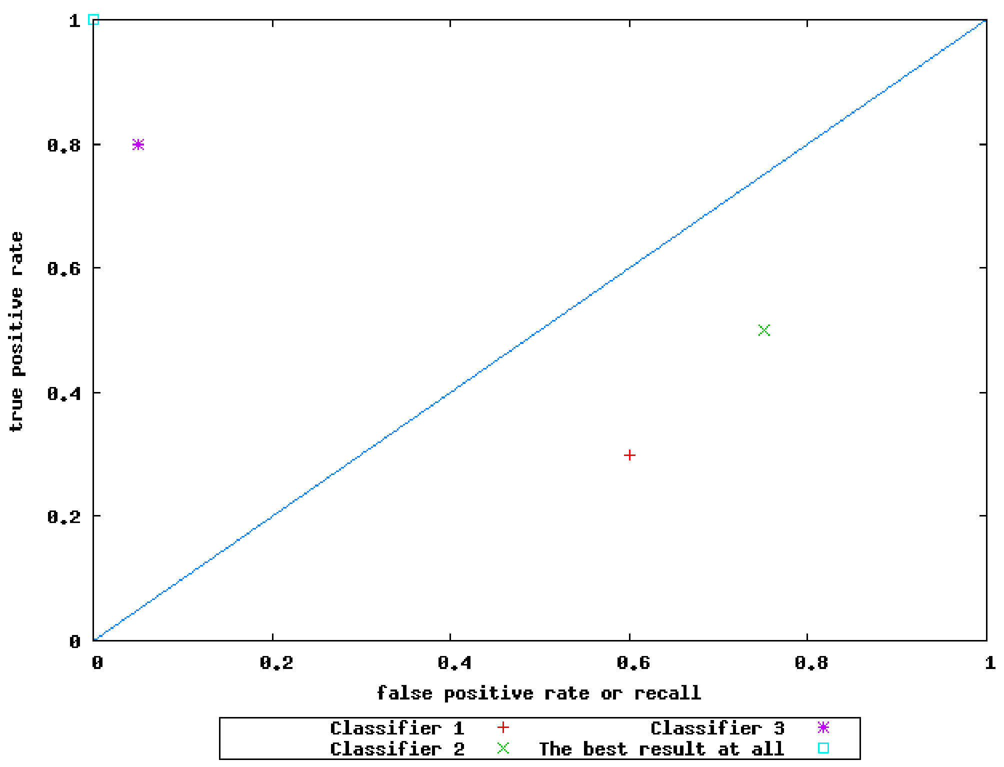


Figure 3.27: Receiver Operating Characteristic Space – including a third classifier

3.4.8 Other probability density functions

We have studied the Normal distribution, however there are many others which we may face while evaluating system information. Among those distributions are: Poisson, Exponential, Gamma, Weibull, Bernoulli, Uniform, etc. Part of those distributions are discrete, like Poisson. This is, they generate integer observations. Others, such as Exponential, are continuous. Their observations are in reals $\mathbb{R}$.

Now, we will start by exploring those distribution shapes. We already know that the Normal has a bell shape. Then, let's observe how the others are. Figure 3.28 presents the shapes for the Poisson distribution under different averages $\lambda \in \{1, 5, 10, 50, 100\}$. We may observe that is concentrates all data at the beginning when $\lambda = 1$. Increasing λ, its behavior tends to distribute data. For some k values such as 100, it looks like a Normal. The source code to generate data using the Poisson distribution is presented in table 3.13. You need to provide the distribution average $\lambda \in \mathbb{R}^+$ and the number of observations to be generated. The summary measurements for this distribution are presented in table 3.14.

Figure 3.29 presents the shapes for the Exponential density distribution. We observe that the higher the parameter λ is, the more the data is concentrated at the beginning of the distribution. However, differently of the Poisson, the Exponential distribution presents the same shape for different values. The Exponential generator is presented in table 3.15. The summary measurements for the Exponential distribution are presented in

Table 3.13: Source code: Poisson generator

```
public class Poisson {
        private double parameter;

        public Poisson(double parameter) {
                this.parameter = parameter;
        }

        public double simulate(){
                int arrivals = 0;
                double sum = -Math.log(1 - Math.random());
                while (sum <= parameter){
                        arrivals++;
                        sum = sum - Math.log(1 - Math.random());
                }
                return arrivals;
        }

        public static void main(String args[]) {
                double parameter = (new Double(args[0])).doubleValue();
                int points = (new Integer(args[1])).intValue();

                Poisson p = new Poisson(parameter);

                for (int i = 0; i < points; i++) {
                        double r = p.simulate();
                        System.out.println(r);
                }
        }
}
```

Table 3.14: Summary measurements: Poisson distribution

Measurement	Value
Average	λ
Median	$\approx \lfloor \lambda + 1/3 - 0.02/\lambda \rfloor$
Mode	$\lfloor \lambda \rfloor$ and $\lambda - 1$ if λ is an integer
Variance	λ

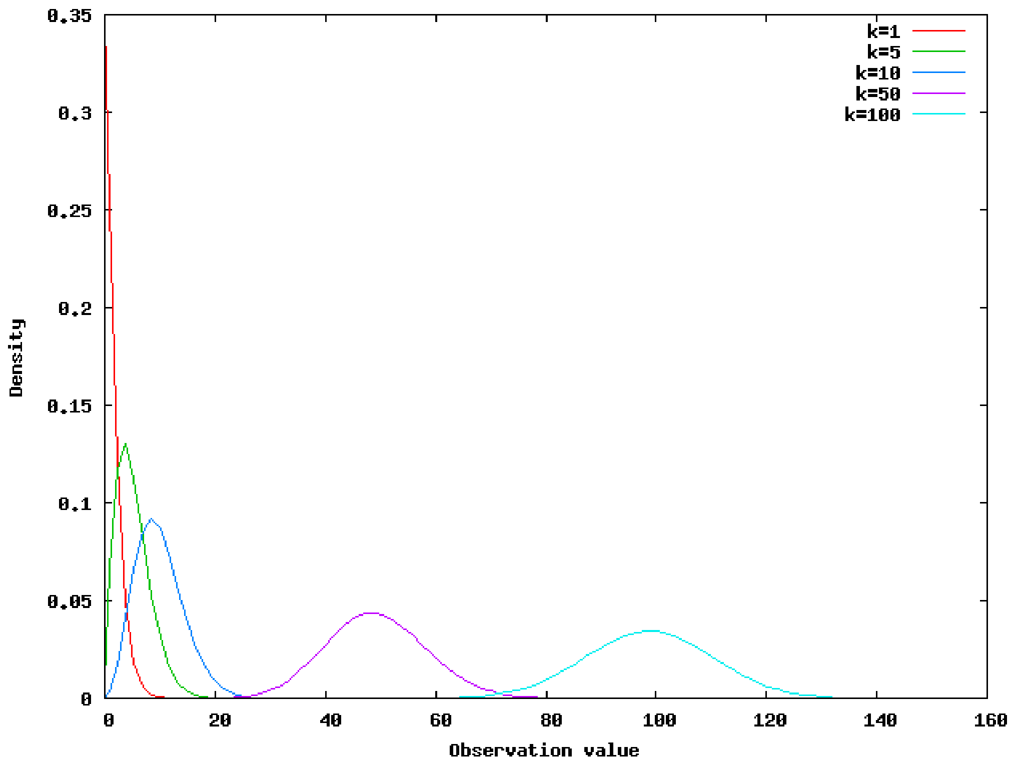


Figure 3.28: Shape: Poisson density distribution

table 3.16.

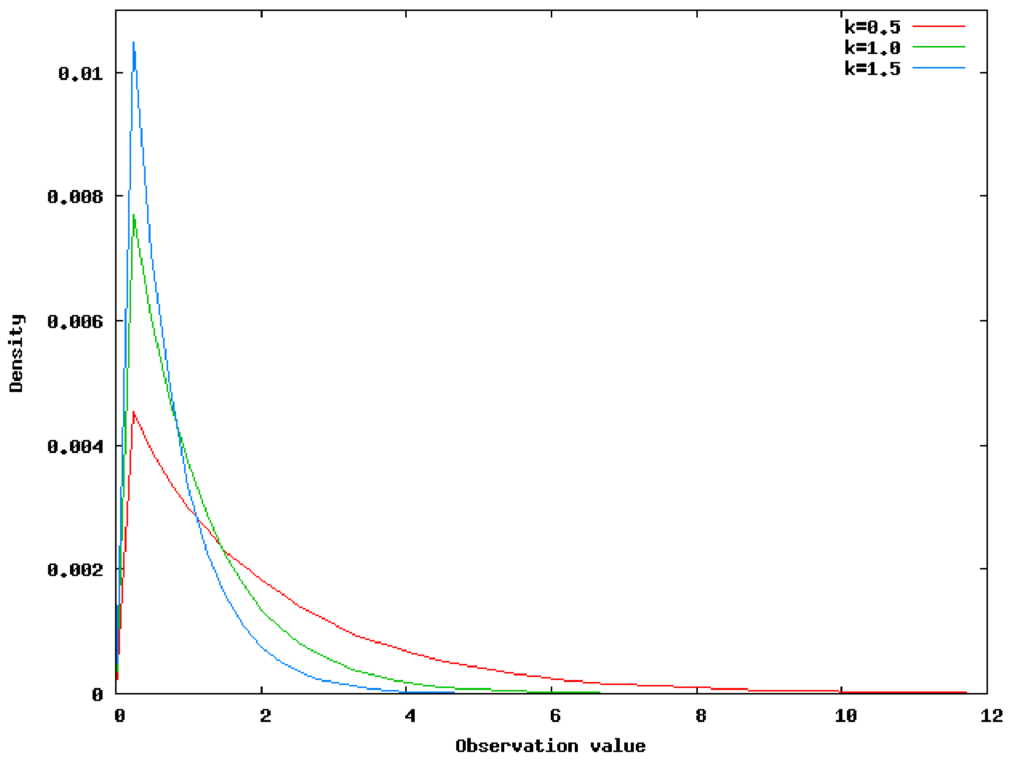


Figure 3.29: Shape: Exponential density distribution

Figure 3.30 presents shapes for the Uniform distribution which are the same. This

Table 3.15: Source code: Exponential generator

```
public class Exponential {
        private double parameter;

        public Exponential(double parameter) {
                this.parameter = parameter;
        }

        public double simulate(){
                return -(1.0 / parameter) * Math.log(Math.random());
        }

        public static void main(String args[]) {
                double parameter = (new Double(args[0])).doubleValue();
                int points = (new Integer(args[1])).intValue();

                Exponential p = new Exponential(parameter);

                for (int i = 0; i < points; i++) {
                        double r = p.simulate();
                        System.out.println(r);
                }
        }
}
```

Table 3.16: Summary measurements: Exponential distribution

Measurement	Value
Average	$\frac{1}{\lambda}$
Median	$\frac{\log 2}{\lambda}$
Mode	0
Variance	$\frac{1}{\lambda^2}$

distribution shares the occurrences among all the possible observation values. The source code to generate the Uniform distribution is presented in table 3.17. Table 3.18 presents the summary measurements for this distribution.

3.5 Some applications

3.5.1 Iris dataset: A Case Study

Firstly, consider the Iris dataset[2] is a multivariate dataset introduced by Fisher () to assess the flower features. It is composed of 150 samples from three species of Iris flowers (Iris setosa, Iris virginica and Iris versicolor). Each sample is composed of four characteristics: length and width of sepal and petal.

The statistics tools we have studied can be applied to distinguish the species. At the first step we may compute the histogram for each one of the 4 attributes according to the specie. Figures **??**, **??**, **??** and **??** present the histograms for the four variables. At first sight, we observe that the petal length and width are good indicatives of differences, once the range of observations make possible to distinguish species. The first and second

[2]Iris dataset – Available at: http://archive.ics.uci.edu/ml/datasets/Iris

Table 3.17: Source code: Uniform generator

```
public class Uniform {
        private double parameter;

        public Uniform(double parameter) {
                this.parameter = parameter;
        }

        public double simulate(){
                return this.parameter * Math.random();
        }

        public static void main(String args[]) {
                double parameter = (new Double(args[0])).doubleValue();
                int points = (new Integer(args[1])).intValue();

                Uniform p = new Uniform(parameter);

                for (int i = 0; i < points; i++) {
                        double r = p.simulate();
                        System.out.println(r);
                }
        }
}
```

Table 3.18: Summary measurements: Uniform distribution

Measurement	Value
Average	$\frac{a+b}{2}$
Median	$\frac{a+b}{2}$
Mode	any value in $[a, b]$
Variance	$\frac{(b-a)^2}{12}$

where a and b are, respectively, the lower and upper limits

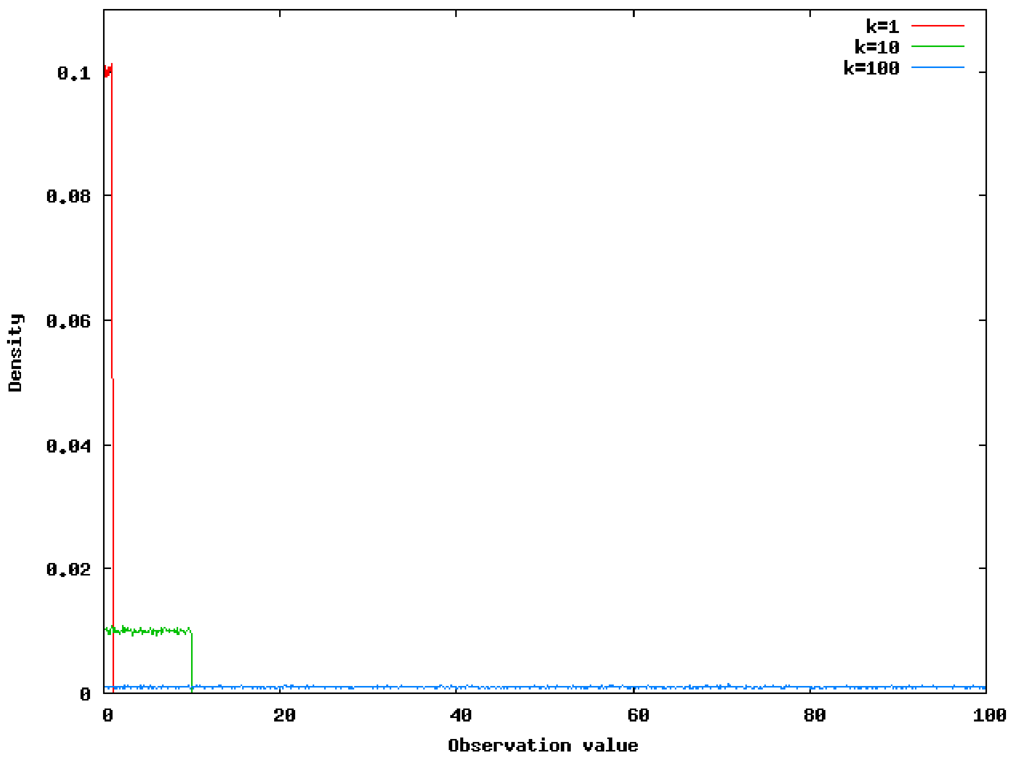


Figure 3.30: Shape: Uniform density distribution

attributes mix data a little, what makes difficult to apply, for instance, a hypothesis test to separate pairs of distributions.

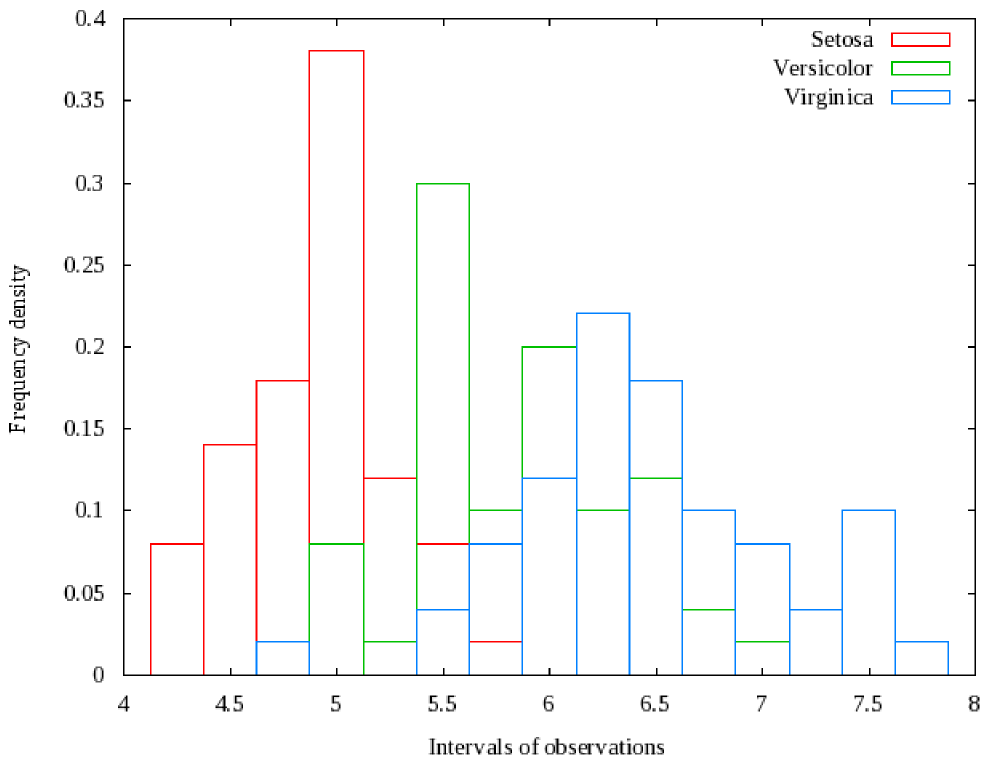


Figure 3.31: Iris – Sepal length in cm

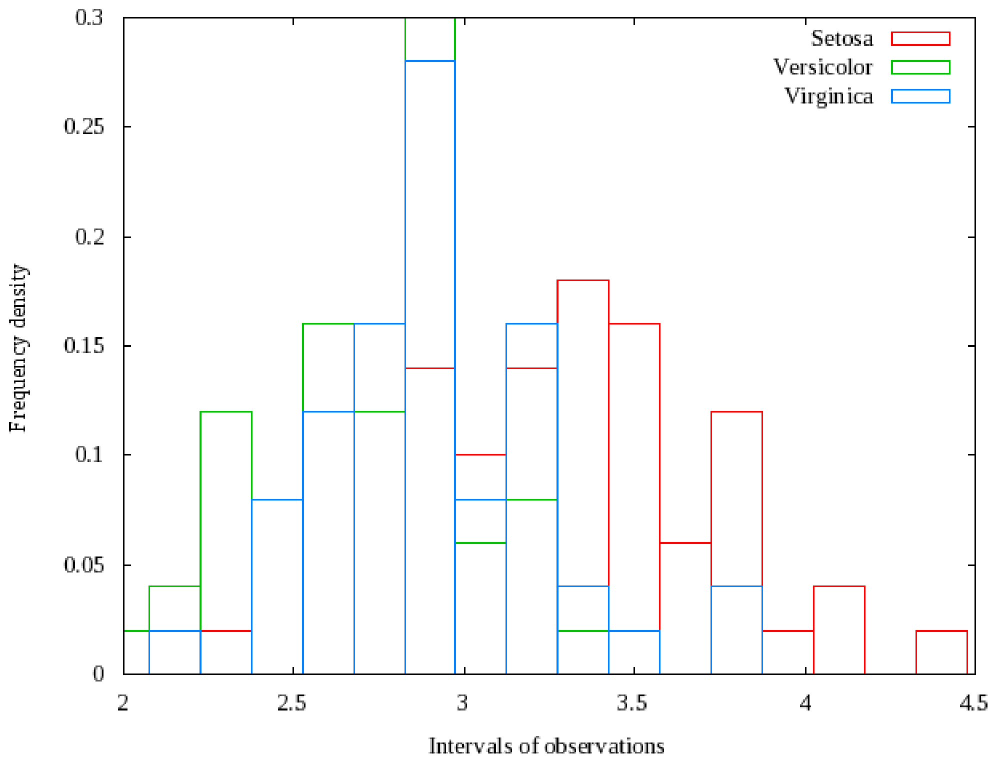


Figure 3.32: Iris – Sepal width in cm

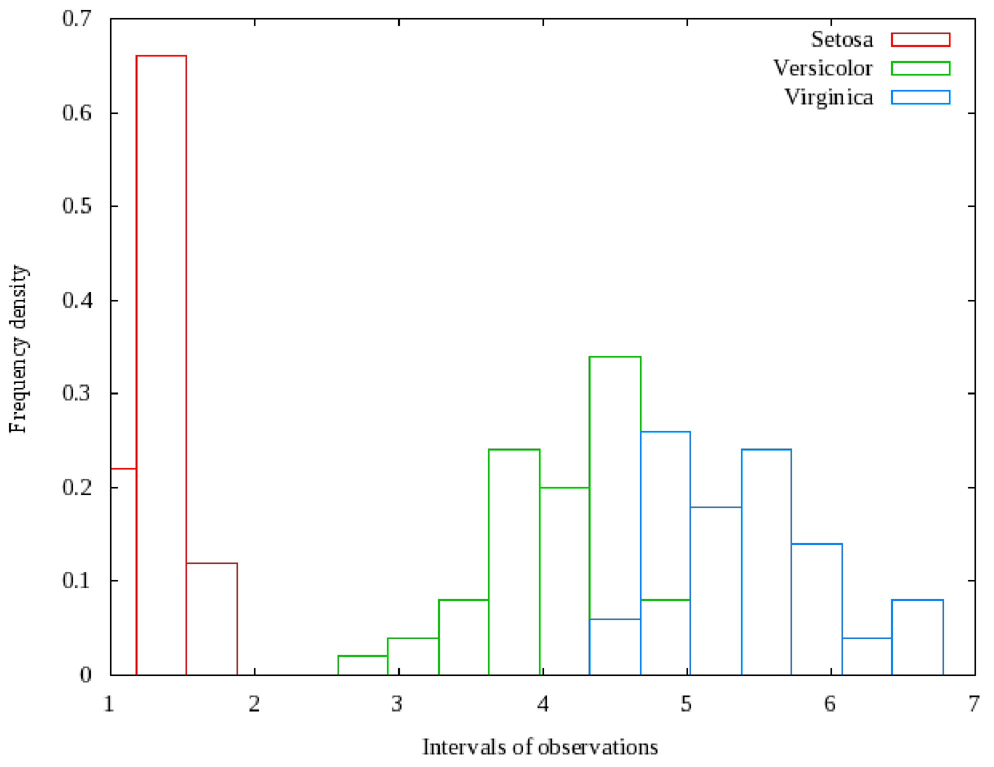


Figure 3.33: Iris – Petal length in cm

However, even having such overlaps in the first and second attributes, we might observe, for example, in Figure 3.31 that the bar centered in 5 has a higher probability

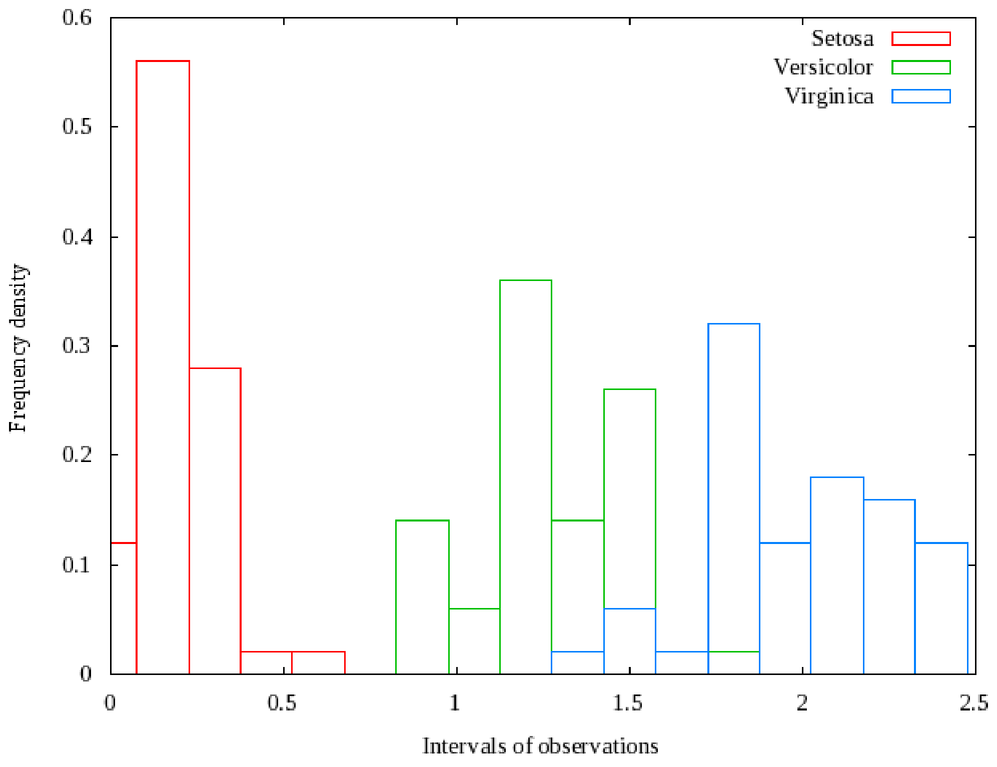


Figure 3.34: Iris – Petal width in cm

to pertain to the Setosa specie rather than the Versicolor. This means we can use such statistical indicatives to define the probability of pertaining to a particular class.

Besides that, we can even combine the attributes. To exemplify this, we roughly divided the intervals of observations for each attribute (we could also compute the type I and II errors when considering such division. A good point to select would be when the type I and II areas assume the same value). Consider that, if the Sepal length attribute is in the range $[4.00, 5.25[$ we consider the Iris as Setosa. When the same attribute is in $[5.25, 6.00[$, then it is versicolor. Otherwise, it is in $[6.00, 8.00]$ and it is considered Virginica.

Doing the same for the Sepal width, we obtained: $[2.00, 2.50[$ for Versicolor; $[2.50, 3.25[$ for Virginica; and $[3.25, 4.50]$ for Setosa. When considering the Petal length, we obtained: $[1.00, 2.00[$ for Setosa; $[2.00, 4.75[$ for Versicolor; and $[4.75, 7.00]$ for Virginica. When considering the Petal width, we obtained: $[0.00, 0.75[$ for Setosa; $[0.75, 1.75[$ for Versicolor; and $[1.75, 2.50]$ for Virginica. Now, we will classify the samples according to the ranges previously presented.

The algorithm in Tables 3.19 and 3.20 presents the source code of the statistical classifier. When it issues 0, it means Setosa, 1 means Versicolor and 2 means Virginica. The results when classifying the 150 samples are presented in table 3.23.

We now compute how good our classifier is using the right and wrong classifications. We obtained 142 right out of 150 and 8 wrongs, this means it rightly classifies in 94.66%

Table 3.19: Iris: Statistical classifier – Part 1

```
#include <stdlib.h>
#include <stdio.h>

double classes[3];

// 0 -> Setosa
// 1 -> Versicolor
// 2 -> Virginica

int attr1(double a1) {
        if (a1 >= 4.0 && a1 < 5.25) {
                return 0;
        } else if (a1 >= 5.25 && a1 < 6) {
                return 1;
        } else {
                return 2;
        }
}

int attr2(double a2) {
        if (a2 >= 2 && a2 < 2.5) {
                return 1;
        } else if (a2 >= 2.5 && a2 < 3.25) {
                return 2;
        } else {
                return 0;
        }
}

int attr3(double a3) {
        if (a3 >= 1 && a3 < 2) {
                return 0;
        } else if (a3 >= 2 && a3 < 4.75) {
                return 1;
        } else {
                return 2;
        }
}

int attr4(double a4) {
        if (a4 >= 0 && a4 < 0.75) {
                return 0;
        } else if (a4 >= 0.75 && a4 < 1.75) {
                return 1;
        } else {
                return 2;
        }
}

int test() {
        double max = 0;
        int index = -1, i;

        for (i = 0; i < 3; i++) {
                if (classes[i] > max) {
                        max = classes[i];
                        index = i;
                }
        }

        return index;
}
```

Table 3.20: Iris: Statistical classifier – Part 2

```
int main(int argc, char *argv[]) {
        FILE *fp;
        float a1, a2, a3, a4;
        int class, i;

        fp = fopen(argv[1], "r+");

        while (!feof(fp)) {
                fscanf(fp, "%f", &a1);
                fscanf(fp, "%f", &a2);
                fscanf(fp, "%f", &a3);
                fscanf(fp, "%f", &a4);
                fscanf(fp, "%d", &class);

                for (i = 0; i < 3; i++)
                        classes[i] = 0;

                if (!feof(fp)) {
                        classes[attr1(a1)]+=1;
                        classes[attr2(a2)]+=1;
                        classes[attr3(a3)]+=1;
                        classes[attr4(a4)]+=1;

                        // Defining weights
                        classes[attr1(a1)]*=1.00;
                        classes[attr2(a2)]*=1.00;
                        classes[attr3(a3)]*=1.00;
                        classes[attr4(a4)]*=1.00;

                        printf("%d%d\n", test(), class);
                }
        }
}
```

of the cases.

3.5.2 SMART dataset: A Case Study

In this second case study, we consider a SMART dataset[3] which presents the attributes values for failing and good hard drives. It is composed of 63 attributes and a additional column which describes if the hard disk has failed (1) or not (0).

To exemplify, we selected only the fourth attribute, which is one of the temperature descriptors (named Temperature 1). Figure 3.35 presents the histogram for such temperature. We may observe that ranges in x-axis do not allow to completely separate the good from the failed hard disks. However, we may observe that good hard disks present a uniform distribution for such temperature with a rough average of 0.06% per interval. The failed hard disks present a different distribution which might present exponential characteristics.

We could separate the good from the bad by evaluating the intervals. For instance, values in the interval centered in 500 have a higher probability to be good. On the other side, values in the first interval, close to 0, are most likely to have failed. We can compute such probabilities per interval and classify new occurrences according to them.

[3]SMART dataset – Available at: http://cmrr.ucsd.edu/people/hughes/smart/dataset/

Table 3.21: Iris: Classification results – part 1

Obtained	**Expected**
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0

Table 3.22: Iris: Classification results – part 1

Obtained	**Expected**
1	1
1	1
2	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
2	1
1	1
2	1
1	1
1	1
1	1
2	1
2	1
1	1
1	1
1	1
1	1
1	1
2	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1

Table 3.23: Iris: Classification results – part 1

Obtained	**Expected**
2	2
2	2
2	2
2	2
2	2
2	2
1	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
1	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2
2	2

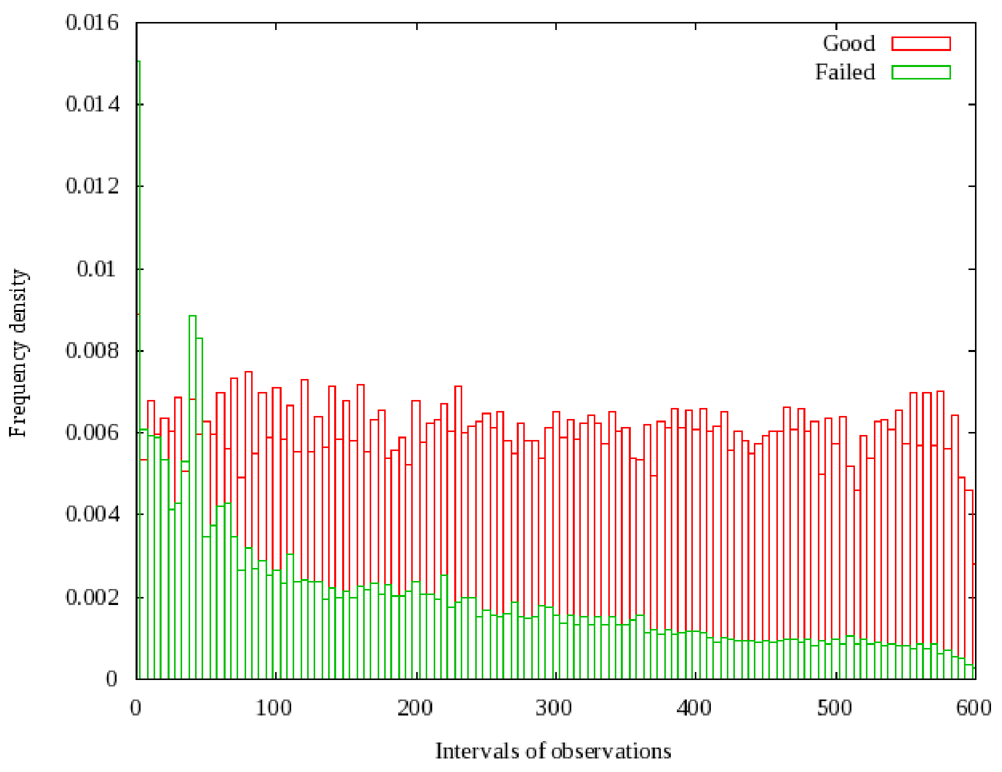


Figure 3.35: SMART – Histogram of attribute four

Table 3.24: SMART – classifier results

		Actual value		
		positive	**negative**	
Prediction	**positive’**	true positive = 2398	false positive = 1979	P’ = 4377
Outcome	**negative’**	false negative = 14663	true negative = 49371	N’ = 64034
		P = 17061	**N** = 51350	

Table 3.24 present the classification results aiming at identify failed hard disks. A true positive means that a failed disk was correctly classified as failed. A false positive means that a good disk was wrongly classified as failed. A true negative means that a good disk was correctly classified as good. A false negative means that a failed disk was wrongly classified as good.

Having such table, we can compute all the measurements previously presented in this chapter such as acurracy $\frac{2398+49371}{17061+51350} = 0.76$, precision $\frac{2398}{2398+1979} = 0.55$, recall $\frac{2398}{2398+14663} = 0.14$ and false positive rate $\frac{1979}{1979+49371} = 0.04$. Then, we can trace the ROC space and the situation of the temperature classifier.

Figure 3.36 presents the ROC curve for a random solution and the situation of the presented one-attribute classifier. It is slightly better than a Random classifier, once it is above the curve. However, it is not good enough. We could explore the other attributes and compose a multivariable classifier, what may present better results.

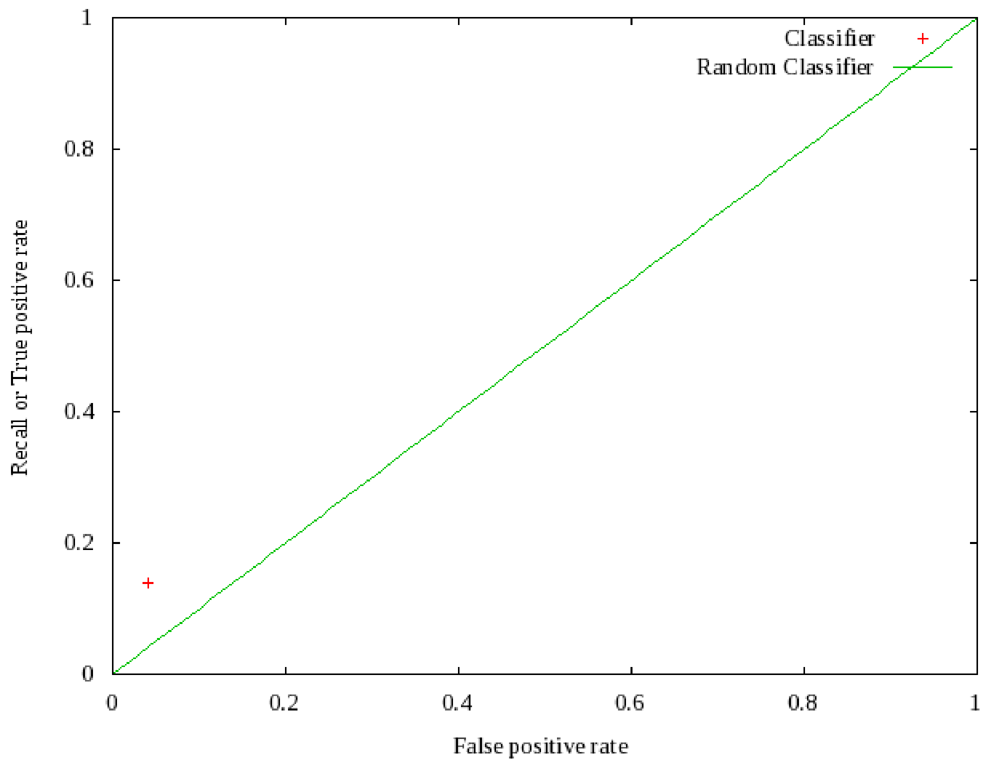


Figure 3.36: SMART – ROC curve for the one-attribute classifier

Exercises

1. Compute the average, median and mode for a dataset using vmstat or any other information extractor;

2. Propose and implement a guess-and-checking approach to approximate the α and β areas for a hypothesis test;

3. Generate the t-Student integration table from 1 degree of freedom up to 120;

4. Implement a guess-and-checking to propose regressions for a histogram considering Poisson, Uniform, Normal and Exponential (consider the Chi-Square test).

Chapter

4

Identifying Behavior

This chapter presents grouping and classifying techniques which help to identify the common behavior on time series. Grouping techniques join information to describe common observations. Classifiers label information based on previously trained data.

When we identify behavior, we figure out what is happening to a system and, therefore, which states it has been visiting. Based on such information we may estimate, predict and analyse the system events.

4.1 Instance-Based Learning

Aprendizado Baseado em Instâncias (`IBL`) é um paradigma de aprendizado que orienta a construção de algoritmos para encontrar instâncias similares em uma base de experiências por meio de funções de domínio real ou discreto (2; 86). O aprendizado consiste em armazenar dados de treinamento e utilizá-los para realizar buscas e aproximações.

Cada instância é composta por um conjunto de atributos, os quais podem ser classificados como entradas ou saídas: atributos de entrada descrevem as condições em que uma experiência foi observada; enquanto atributos de saída representam os resultados, de acordo com as condições descritas pelos atributos de entrada.

Os algoritmos `IBL` calculam a similaridade entre uma instância de consulta, x_q, e as instâncias da base de experiências, retornando um conjunto resposta. Nesses algoritmos, apenas as instâncias mais relevantes são utilizadas para estimar atributos de saída para o ponto de consulta.

O algoritmo `IBL` adotado no projeto `MidHPC` segue o modelo proposto por Senger *et al.* (100), o qual é baseado no aprendizado dos k-vizinhos mais próximos (k-*nearest neighbor*). O método assume que todas as instâncias em uma base de experiências correspondem a pontos em um espaço $\mathbb{R}^n$ e que uma função de distância, nesse caso Euclidiana, é utilizada para definir pontos próximos à consulta.

Seja uma instância x descrita por um vetor de atributos $(a_1(x), a_2(x), \ldots, a_n(x))$, em que $a_r(x)$ descreve o valor do r-ésimo atributo de um instância x. A distância entre duas instâncias x_i e x_j é dada por $E(x_i, x_j) = \sqrt{\sum_{r=1}^{n} d(a_r(x_i), a_r(x_j))^2}$, em que a função distância $d(.)$ é definida pelas equações 4.1 e 4.2.

$$d(x,y) = \begin{cases} 1 & \text{se em } x \text{ ou } y \text{ faltam atributos} \\ d_n(x,y) & \text{se os atributos são nominais} \\ \|x-y\| & \text{caso contrário} \end{cases} \tag{4.1}$$

$$d_n(x,y) = \begin{cases} 0 & \text{se } x = y \\ 1 & \text{caso contrário} \end{cases} \tag{4.2}$$

Um problema recorrente da distância Euclidiana ocorre quando um dos atributos apresenta intervalo mais amplo, nesse caso ele pode dominar as saídas da equação de distância, prejudicando os resultados do algoritmo (122). Por exemplo, se um atributo apresentar valores entre 1 e 1024 e o outro estiver entre 1 e 5, o segundo será dominado pelo primeiro. Pode-se solucionar essa questão por meio da normalização de atributos numéricos em uma mesma escala. Senger *et al.* (100) adotam uma função de normalização linear $h(x_i)$ que resulta em valores no intervalo $[0;1]$. Seja $\min_{x_i}$ o menor valor na base de experiências para o atributo x_i e $\max_{x_i}$ o maior valor para o mesmo atributo, a função $h(x_i)$ é definida na equação 4.3.

$$h(x_i) = \frac{x_i - \min_{x_i}}{\max_{x_i} - \min_{x_i}} \tag{4.3}$$

Assim, considera-se a equação 4.4 para aproximar uma função de domínio de valores reais $f : \mathbb{R}^n \rightarrow \mathbb{R}^+$, utilizando pesos w_i para cada um dos k vizinhos mais relevantes, de acordo com a distância ao ponto de consulta x_q.

$$g(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i} \tag{4.4}$$

Os pesos w_i são calculados segundo uma função, a qual gera resultados próximos a um valor constante, quando a distância tende a 0, e aproxima de 1, à medida que a distância tende ao infinito. Embora diferentes funções de ponderação possam ser empregadas, o `MidHPC` adota a proposta de Senger *et al.* (100), que aplica uma função Gaussiana centrada no ponto x_q com uma variância t, conforme a equação 4.5 (o termo $E(x_q, x_j)$ representa a distância Euclidiana entre o ponto de consulta x_q e outro ponto qualquer x_j).

$$w_i(x_j) = e^{\frac{-E(x_q,x_j)}{t^2}} \tag{4.5}$$

A figura 4.1 apresenta exemplos de valores para w gerados pela equação 4.5, considerando um intervalo de distâncias Euclidianas entre 0 e 10, além de $t^2 = \{0,125; 0,250; 0,500; 1,000; 2,000\}$.

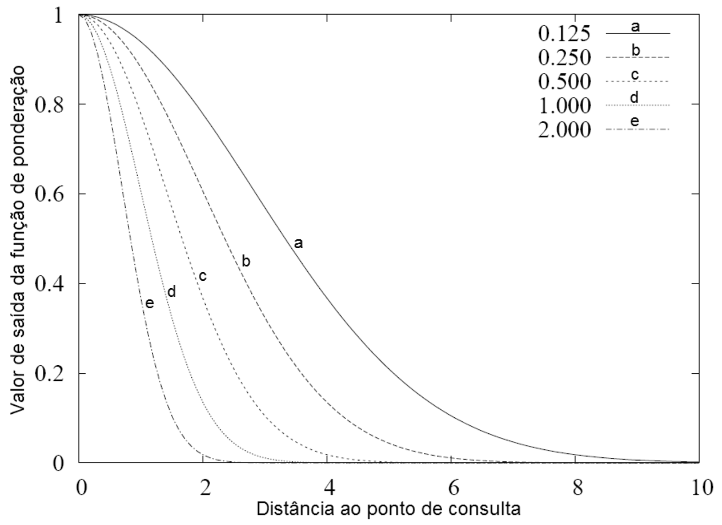


Figure 4.1: Comportamento da função de ponderação

A técnica `IBL` proposta em (100) é empregada pelo `MidHPC` para estimar a ocupação de recursos gerada por processos. Essa técnica é acionada antes de iniciar aplicações e tem sido empregada na parametrização de algoritmos de escalonamento de processos. Para exemplificar o uso do `IBL`, considere uma aplicação α composta por três processos (α_1, α_2 e α_3). Durante a execução da aplicação, os processos α_1, α_2 e α_3 são monitorados e seus dados de ocupação de recursos são salvos em uma base de dados. Em seguida, o `IBL` estima, com base nos históricos, a ocupação de recursos para novas aplicações. Essas estimativas são utilizadas para otimizar decisões de escalonamento.

4.2 K-Nearest Neighbors

K-Nearest Neighbors is a instance-based learning technique employed to classify objects according to the closest training samples of a database. Samples are stored in a database, when a new point is issued, it is compared to k points in a neighborhood. The majority of votes decides the type of the new point of the system.

For instance, consider figure 4.2, which presents a central point with its neighbors in inner circles. We may select the closest circle to classify the new point. In such situation we have more diamonds rather than rectangles. Therefore, we would classify the new point as a diamond.

New we have a better idea how KNN classify objects. It basically evaluates the class of the points in neighborhood. If we select $k = 1$, then, the classification will be conducted based on the nearest neighbor. It is better to consider an odd value for k, if even, we may

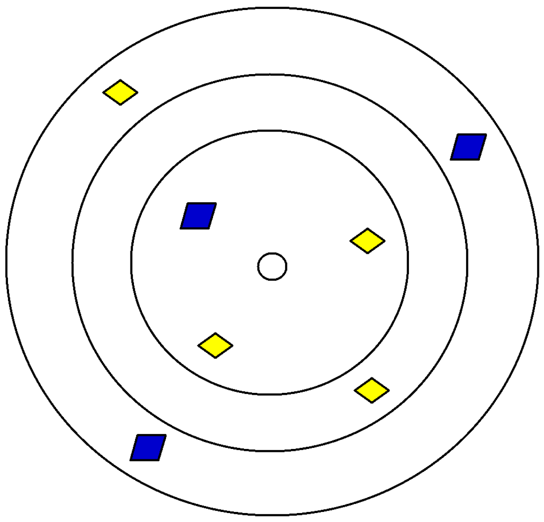


Figure 4.2: KNN example

be put in the situations where two neighbors are in neighborhood but each one pertains to a different class. In such circumstance, we can ramdonly select the class, what is not the best choice.

We can also define how far each neighbor is from the new point and, therefore, use such information as weight to define the class of the point. For instance, consider figure 4.3. we will give more relevance to the rectangle rather than the diamonds, besides there are more diamonds. Depeding on the weight function we choose, we may define the new point as a rectangle.

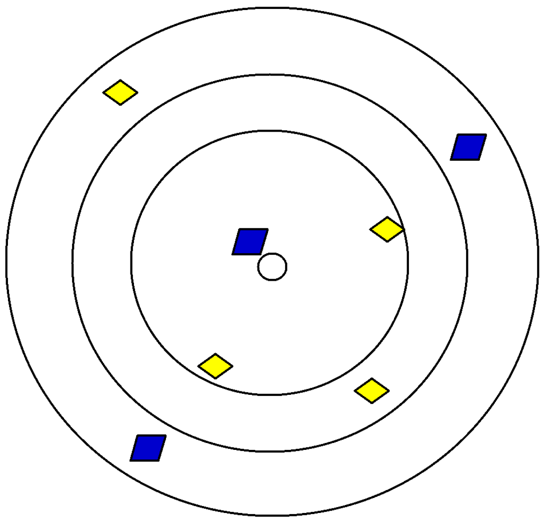


Figure 4.3: KNN example

Figure 4.4 presents the curves for the three weight functions in equations 4.6, 4.7 and 4.8. We observe that $f(x)$ gives much more relevance to the neareast neighbors rather than $g(x)$. The same is observed in relation of $g(x)$ to $h(x)$ Consequently, the far the neighbor is, the less relevance will be considered in the classification of the new point.

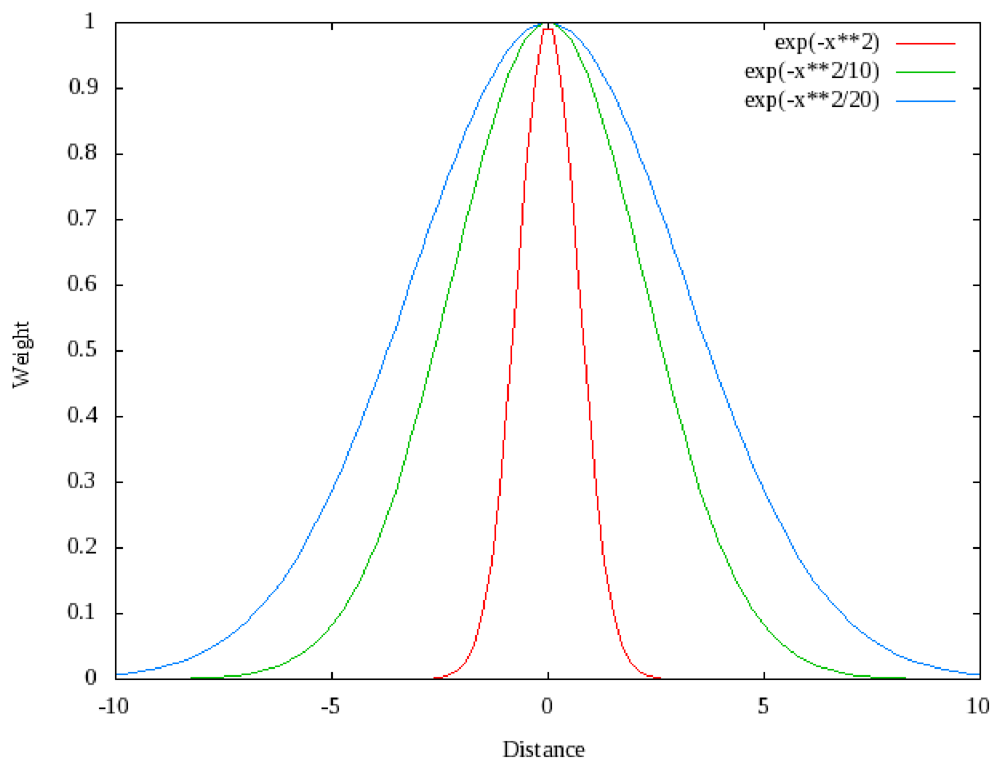


Figure 4.4: KNN example

$$f(x) = exp(-distance^2) \tag{4.6}$$

$$g(x) = exp(\frac{-distance^2}{10}) \tag{4.7}$$

$$h(x) = exp(\frac{-distance^2}{20}) \tag{4.8}$$

We have observed that the weight functions consider the distance. This is a representation of how far a neighbor is from the new point. People usually employ the Euclidean distance to compute it, however there are others to be considered (such as Manhattan, Hamming, etc.). Equation 4.9 presents the Euclidean distance, where p is the new point and n is the neighbor being evaluated.

$$E(p, n) = \sqrt{\sum_{i=0}^{|p|-1} (p_i - n_i)^2} \tag{4.9}$$

Any new point presented to KNN can be added into the database, consequently, this technique can be incremental. Table 4.1 presents the Java source code of KNN using a database (PostgreSQL in this case, however any Java-skilled person can modify it to support other databases). The code receives data points and generates an integer as outcome, which identifies the class for new points. It considers the Euclidean distance and the very simple weight function $q(x) = \frac{1.0}{|distance|}$, which behaves as presented in figure 4.5. It gives very high relevance to the closest neighbors and falls abruptly after some distance.

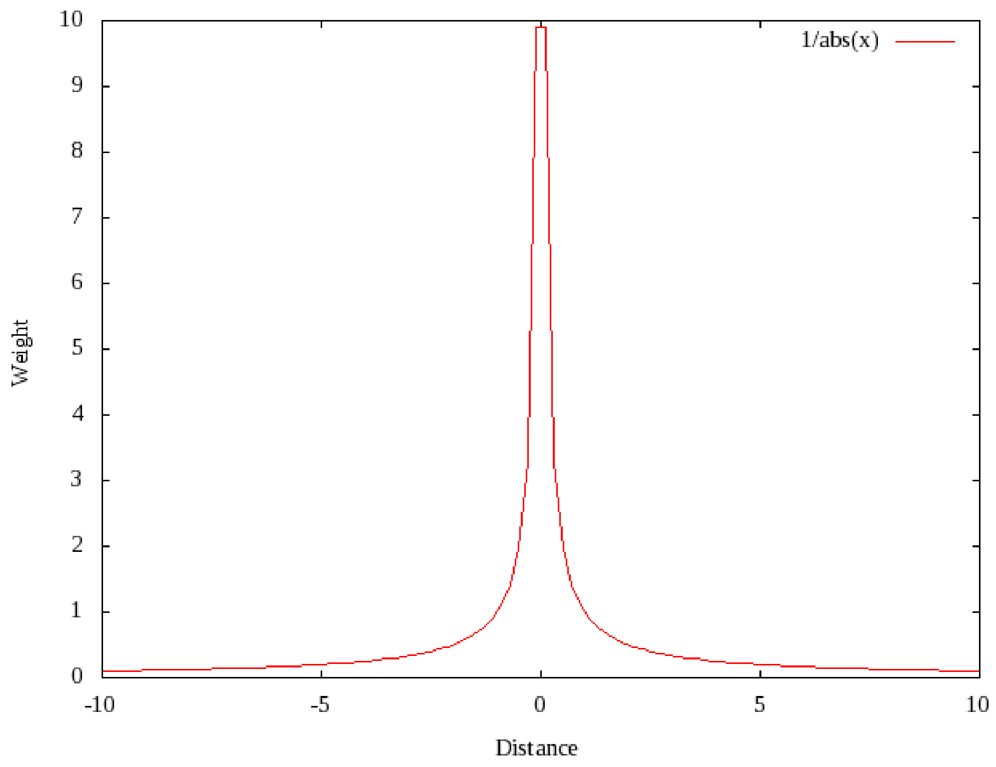


Figure 4.5: KNN weight function

4.3 Self-Organizing Maps

Self-Organizing Maps (SOM), also called Kohonen Maps (), builds a discretization of training samples, or Map, in a space. They consider a neighborhood function to group the closest information.

The architecture of SOM is composed of neurons, also called nodes. Each neuron has a weight vector wv associated which has the same dimension as the input data vectors. Those neurons rearrange themselves according to training samples what is based on how sensory information is handled by human brain (neurons specialize themselves to better respond to inputs).

In order to better understand SOM, consider figure 4.6, where the location of nine points were randomly initialized in $\mathbb{R}^2$. Every point is represented by a two-dimensional vector p_i. When we submit this system to several input data vectors, SOM moves such points in the direction of usual samples. For instance, consider 4.7, where several samples were presented to the network. Data points in the neighborhood of p_0 tend to move in its

Table 4.1: K-Nearest Neighbors – Java source code – part 1

```
import java.sql.*;
import java.util.*;

interface Database {
        public String url = "jdbc:postgresql://localhost:5432/knn";
        public String login = "knnuser";
        public String passwd = "knnpasswd";
        public String driver = "org.postgresql.Driver";
}

public class Knn implements Database {
        private Connection conn;
        private int dim;
        private int k;

        public Knn(int k, int dim) throws Exception {
                Class.forName(driver);
                this.conn = DriverManager.getConnection(url, login, passwd);
                this.k = k;
                this.dim = dim;
        }

        public void create() throws Exception {
                Statement stmt = this.conn.createStatement();
                stmt.executeUpdate("create sequence id_seq;");
                String str = "create table knowledge ( id integer not null primary key, ";

                for (int i = 0; i < dim; i++) { // +1 because of the group type
                        str += "field"+i+" numeric(16,8)";
                        str += ", ";
                }

                str += "gid integer, distance numeric(16,8));";

                stmt.executeUpdate(str);
                stmt.executeUpdate("create index distanceidx on knowledge(distance);");
                stmt.close();
        }

        public void destroy() throws Exception {
                Statement stmt = this.conn.createStatement();
                stmt.executeUpdate("drop sequence id_seq;");
                stmt.executeUpdate("drop index distanceidx;");
                stmt.executeUpdate("drop table knowledge;");
                stmt.close();
        }

        public void insert(double fields[], int group) throws Exception {
                String data = "";

                for (int i = 0; i < fields.length; i++) {
                        data += (new Double(fields[i])).toString();
                        data += ", ";
                }

                data += (new Integer(group)).toString();

                String str = "insert into knowledge values (nextval('id_seq'), "+data+");";
                Statement stmt = this.conn.createStatement();
                stmt.executeUpdate(str);
                stmt.close();
        }
```

direction.

Movements are conducted in neighborhood according to data samples. After some training, we represent a set of known groups in the system. The training sample must have all the information we need to represent. Unkown information is not repsented by

Table 4.2: K-Nearest Neighbors – Java source code – part 2

```
public double Euclidean(double fields[], double line[]) {
        double sum = 0;
        int i;

        for (i = 0; i < fields.length; i++) {
                sum += Math.pow(fields[i] - line[i], 2.0);
        }

        return Math.sqrt(sum);
}

public int getGroup(double fields[]) throws Exception {

        String data = "select id, ";

        for (int i = 0; i < fields.length; i++) {
                data += "field"+i;
                if (i < fields.length-1)
                        data += ", ";
        }

        data += " from knowledge;";
        Statement stmt = this.conn.createStatement();
        ResultSet rs = stmt.executeQuery(data);
        double line[] = new double[fields.length];

        while (rs.next()) {
                for (int i = 0; i < line.length; i++) {
                        line[i] = rs.getDouble("field"+i);
                }

                int id = rs.getInt("id");
                double distance = Euclidean(fields, line);

                Statement stmt2 = this.conn.createStatement();
                stmt2.executeUpdate("update knowledge set distance = "+distance+" where id = "+id);
                stmt2.close();
        }

        Hashtable hash = new Hashtable();
        rs = stmt.executeQuery("select id, gid, distance from knowledge order by distance asc limit "+k);
        double curr = 0;

        // Computing according to the distance
        while (rs.next()) {
                rs.getInt("id");

                if ((Double) hash.get(rs.getString("gid")) != null) {
                        curr = ((Double) hash.get(rs.getString("gid"))).doubleValue();
                } else {
                        curr = 0;
                }
```

the network.

SOM can be composed of two steps or only one. When two steps are considered, we may move data points to represent the training sample and, therefore, obtain groups. Afterwards, we can tell in which group a new pattern is, or the confidence level we have for it in each of the groups. In such situation, the training involves movements and the execution does not update data points.

When only one step is considered, we move data points as input patterns are received. Then, we can use metrics such as how much has varied with the addition of an input vector. We can also tell the confidence level of the input in relation of the existing groups.

Table 4.3: K-Nearest Neighbors – Java source code – part 3

```
                curr += 1.0 / rs.getDouble("distance");
                hash.put(rs.getString("gid"), new Double(curr));
            }

            Enumeration k = hash.keys();
            Enumeration e = hash.elements();

            double max = 0.0;
            int group = -1;

            for (; e.hasMoreElements() ;) {
                Object oe = e.nextElement();
                Object ok = k.nextElement();

                System.out.println(oe+" "+ok);

                if (max < ((Double) oe).doubleValue()) {
                    max = ((Double) oe).doubleValue();
                    group = (new Integer((String) ok)).intValue();
                }
            }

            stmt.close();

            return group;
        }

        public static void main(String args[]) throws Exception {
            Knn knn = new Knn(3, 3);
            knn.create();
            knn.insert(new double[]{0.0, 0.0, 0.0}, 0 );
            knn.insert(new double[]{1.0, 0.0, 0.0}, 1 );
            knn.insert(new double[]{0.0, 1.0, 0.0}, 2 );
            knn.insert(new double[]{0.0, 0.0, 1.0}, 3 );
            System.out.println(knn.getGroup(new double[]{0.0, 0.0, 0.0}));
            knn.destroy();
        }
    }
```

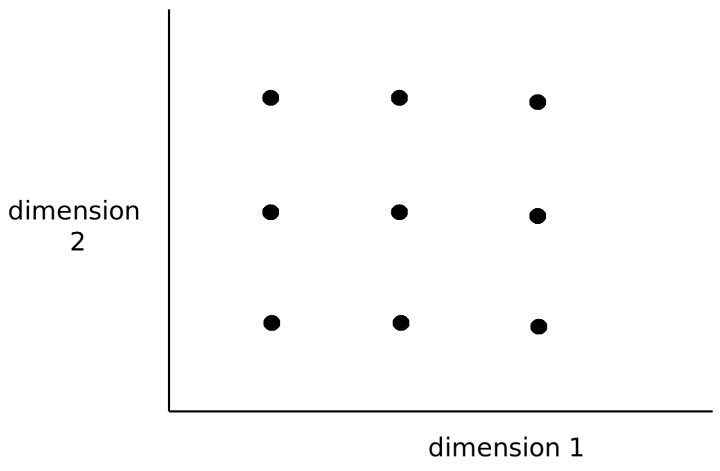


Figure 4.6: SOM – Step 1

However, in this case, the network starts behaving as an incremental memory, where new information modifies the existing groups.

Tables 4.4, 4.5 and 4.6 present a sample Java code for SOM. It considers the Euclidean distance to measure how far a input vector is from SOM data points. SOM data points move in direction of inputs according to equation 4.10, where $D(p, i)$ is the Euclidean

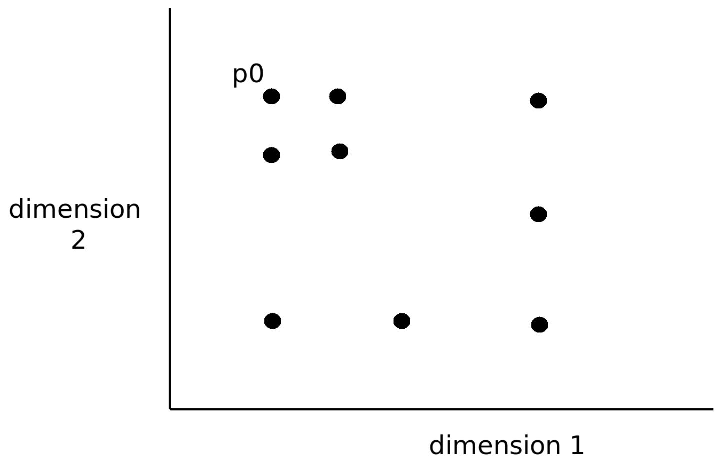


Figure 4.7: SOM – Step 2

distance from an input i to SOM data point p. This equation defines the weight that will be given to move data points in direction of the input vector.

$$w(p,i) = exp(-D(p,i)^2) \tag{4.10}$$

The weight is applied in equation 4.11, which defines the new weight vector for neuron p after an input i. The term η parametrizes the relevance of the new input and $dv(p,i)$ is the distance vector of the input to the data point being considered. The distance vector is the result of the difference of the input to the weight vector of neuron p (equation 4.12, where i is the input vector and wv is the weight vector associated to a neuron).

$$wv(p,i) = wv(p,i) + \eta \cdot w(p,i) \times dv(p,i) \tag{4.11}$$

$$\sum_{k=0}^{|p|-1} i_k - wv_k \tag{4.12}$$

4.4 K-Means

K-Means is a machine learning technique which attempts to partition n objects into k clusters (where $k < n$), associating each object with the closest group. Data points are initially read and allocated in a representation space. For instance, consider figure 4.8 where two-dimensional data points are organized. Then, consider we have defined $k = 3$, therefore, our goal is to partition the space into 3 regions.

To start the algorithm, we firstly select k random points in the space. Figure 4.9 presents the random points, also called centroids. Then, for each data point we compute the closest centroid and label the point as under such centroid. Afterwards, we recompute

Table 4.4: Self-Organizing Maps – Java source code – part 1

```
import java.io.*;
import java.util.*;

public class Som {
        private Vector<Node> nodeList;

        public Som(int neurons, int dim, Distance d, double eta, double min, double max) {
                this.nodeList = new Vector<Node>();

                for (int i = 0; i < neurons; i++) {
                        this.nodeList.add(new Node(dim, d, eta, min, max));
                }
        }

        public void process(double[] input) {
                for (int i = 0; i < this.nodeList.size(); i++)
                        this.nodeList.get(i).update(input);
        }

        public void printNodes() {
                for (int i = 0; i < this.nodeList.size(); i++)
                        this.nodeList.get(i).printWV();
        }

        public static void main(String args[]) throws Exception {
                if (args.length != 6) {
                        System.out.println("usage: java Som filename dimension neurons eta min max");
                        System.exit(0);
                }

                int dim = Integer.parseInt(args[1]);
                int neurons = Integer.parseInt(args[2]);
                double eta = Double.parseDouble(args[3]);
                double min = Double.parseDouble(args[4]);
                double max = Double.parseDouble(args[5]);
                Distance d = new Euclidean();

                FileInputStream fis = new FileInputStream(args[0]);
                BufferedInputStream bis = new BufferedInputStream(fis);
                DataInputStream dis = new DataInputStream(bis);
                String str = null;

                Som som = new Som(neurons, dim, d, eta, min, max);

                while ((str = dis.readLine()) != null) {
                        String[] strlist = str.split(" ");
                        double[] input = new double[strlist.length];

                        for (int i = 0; i < strlist.length; i++) {
                                String aux = strlist[i].trim();
                                if (aux.length() > 0) {
                                        input[i] = Double.parseDouble(aux);
                                }
                        }

                        som.process(input);
                }

                dis.close();
                bis.close();
                fis.close();

                som.printNodes();
        }
}
```

the centroid according to its members (the members are the data points associated to it).

In order to recompute each centroid c, we can consider the average distance of the

Table 4.5: Self-Organizing Maps – Java source code – part 2

```
class Node {
        private int id;
        private static int ids = 0;
        private int dim;
        private Random random;
        private double[] wv;
        private Distance d;
        private double eta;
        private double min;
        private double max;

        public Node(int dim, Distance d, double eta, double min, double max) {
                this.id = ids++;
                this.dim = dim;
                this.random = new Random();
                this.wv = new double[this.dim];
                this.d = d;
                this.eta = eta;
                this.min = min;
                this.max = max;
                this.randomize();
        }

        public void randomize() {
                for (int i = 0; i < this.dim; i++) {
                        double range = (this.max - this.min);
                        wv[i] = this.min + (range * this.random.nextDouble());
                }
        }

        public double getDistance(double[] input) {
                return this.d.getDistance(input, wv);
        }

        public double getNeighborhood(double[] input) {
                double a = Math.exp(-Math.pow(this.getDistance(input), 2.0));
                return a;
        }

        public void update(double input[]) {
                double[] nwv = new double[this.dim];
                double[] diff = this.d.getDistanceVector(input, wv);

                for (int i = 0; i < this.dim; i++) {
                        nwv[i] = wv[i] + this.getNeighborhood(input) * this.eta * diff[i];
                }

                for (int i = 0; i < this.dim; i++) {
                        wv[i] = nwv[i];
                }
        }

        public void printWV() {
                System.out.print(id+" -> ");
                for (int i = 0; i < this.dim; i++) {
                        System.out.print(wv[i]+" ");
                }
                System.out.println();
        }
}

interface Distance {
        public double getDistance(double[] input, double[] wv);
        public double[] getDistanceVector(double[] input, double[] wv);
}
```

members given by equation 4.13, where i_{b_j} is the input data member of class, this is associated to the centroid, b and p_b is the number of members of class b. At every step,

Table 4.6: Self-Organizing Maps – Java source code – part 3

```
class Euclidean implements Distance {
        public double getDistance(double[] input, double[] wv) {
                double sum = 0.0;

                for (int i = 0; i < input.length; i++) {
                        sum += Math.pow(input[i] - wv[i], 2.0);
                }

                return Math.sqrt(sum);
        }

        public double[] getDistanceVector(double[] input, double[] wv) {
                double[] difference = new double[input.length];

                for (int i = 0; i < input.length; i++) {
                        difference[i] = input[i] - wv[i];
                }

                return difference;
        }
}
```

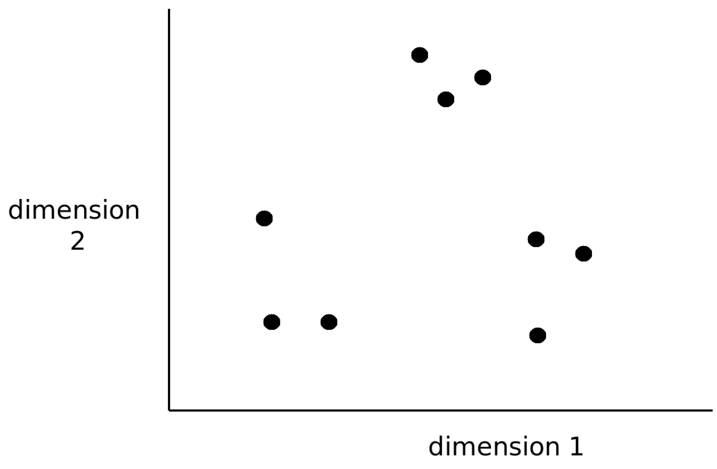


Figure 4.8: K-Means – Step 1

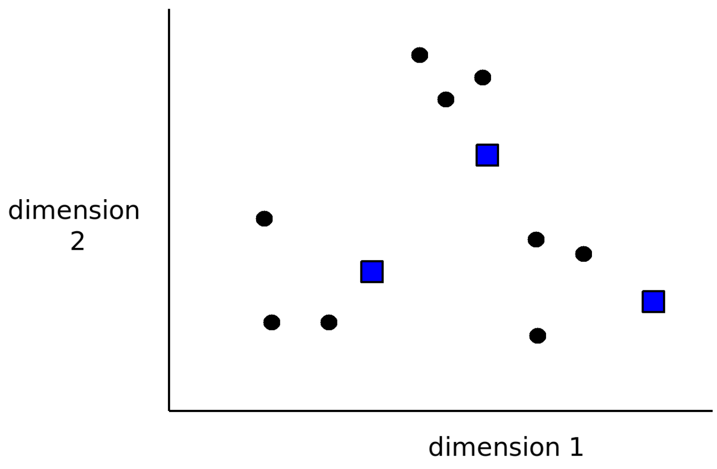


Figure 4.9: K-Means – Step 2

the centroids will move to the direction of its members. We may stop when there is no variation in centroids, this means there is no movement in direction of inputs, then,

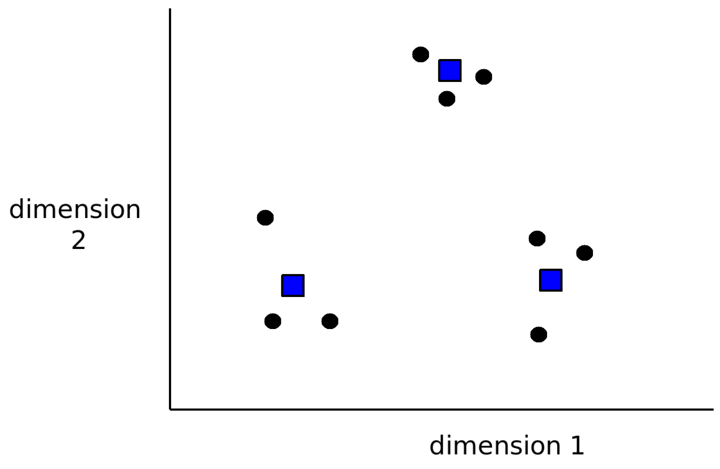


Figure 4.10: K-Means – Step 3

the learning phase can stop. At the end, we will obtain the coordinates of centroids to partition the input data such as the example in Figure 4.10.

$$c_c = \frac{\sum_{j=0}^{|i|} i_{bj}}{p_c} \tag{4.13}$$

Table 4.7 presents the C source code for the K-Means algorithm. It considers the previously presented steps.

4.5 Radial Basis Function Networks

Radial Basis Function Network is a type of artificial neural network commonly used to make function regression and prediction of time series. Its architecture is typically composed of three layers. The first is the input layer, where values are submitted to the architecture. The second is formed by neurons with radial activation functions. Each neuron is connected to the last layer. This connection has an associated weight. The last computes the neuron activations and generate the output.

In our situation, we have considered Equation 4.14 to activate neurons in second layer for every input I, where C is the centroid of the neuron, σ is the standard deviation around it. $f(I, C)$ is given in Equation 4.15, where D is the number of dimensions of the input vector (which is the same as the centroid C).

$$a(I) = exp(-\frac{f(I,C)^2}{2.0 \cdot \sigma^2}) \tag{4.14}$$

$$f(I,C) = \sum_{i=1}^{D} (I_i - C_i)^2 \tag{4.15}$$

Then, every neuron executes the same activation function (Equation 4.14). After the

Table 4.7: K-Means – Java source code – part 1

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

#define PRECISION 0.0000001
#define MYMAXFLOAT 3.40282347e+38F

float *centroids_sum = NULL;
float *centroids_count = NULL;
float *centroids = NULL;
float *data = NULL;
float variation = 0;

float Euclidean(float *data, float *centroids, int dim) {
        float sum = 0;
        int i;

        for (i = 0; i < dim; i++) {
                sum += pow(data[i] - centroids[i], 2.0);
        }

        return sqrt(sum);
}

void sum(float *data, int centroid_id, int dim) {
        int i;

        for (i = 0; i < dim; i++) {
                centroids_sum[centroid_id*dim+i] += data[i];
                centroids_count[centroid_id]+=1;
        }
}

void average(int centroid_id, int lines, int dim) {
        int i;

        for (i = 0; i < dim; i++) {
                if (centroids_count[centroid_id] == 0)
                        centroids_sum[centroid_id*dim+i] = 0;
                else
                        centroids_sum[centroid_id*dim+i] /= centroids_count[centroid_id];
        }
}

float update(int centroid_id, int dim) {
        int i;
        float diff = 0;

        for (i = 0; i < dim; i++) {
                diff += pow(centroids[centroid_id*dim+i] - centroids_sum[centroid_id*dim+i], 2.0);
                centroids[centroid_id*dim+i] = centroids_sum[centroid_id*dim+i];
        }

        return diff / dim;
}
```

activations, the last layer computes the final result by considering Equation 4.16, where N represents the number of neurons in the second layer, $a_i(I)$ is the activation of neuron i for the input I, and w_i is the weight associated to the connection of neuron i and the last layer.

$$o(I) = \sum_{i=1}^{N} w_i \cdot a_i(I) \tag{4.16}$$

Table 4.8: K-Means – Java source code – part 2

```
void print_data(int lines, int dim) {
        int i, j;

        for (i = 0; i < lines; i++) {
                for (j = 0; j < dim; j++) {
                        printf("%f ", data[i*dim+j]);
                }
        }
        printf("\n");
}

void print(int centroid_id, int dim) {
        int i;

        printf("[");
        for (i = 0; i < dim; i++) {
                printf("%f", centroids[centroid_id*dim+i]);
                if (i < dim-1)
                        printf(", ");
        }
        printf("]\n");
}

int main(int argc, char *argv[]) {
        char *filename;
        int k, dim, count, lines, pos, i, j, w;
        FILE *fp;
        float *v, *centroids_aux, max, *data_aux, d, x;

        if (argc != 4) {
                printf("Usage: %s filename k dimensions\n", argv[0]);
                exit(0);
        }

        filename = (char *) malloc(sizeof(char) * (strlen(argv[1]) + 1));
        strcpy(filename, argv[1]);
        k = atoi(argv[2]);
        dim = atoi(argv[3]);

        v = (float *) malloc(sizeof(float) * dim);
        fp = fopen(filename, "r+");

        lines = 0;
        count = 0;

        // reading the file
        while (!feof(fp)) {
                fscanf(fp, "%f", &v[count]);
                count++;

                if (count == dim) {
                        data = (float *) realloc(data, sizeof(float) * dim * (lines + 1));

                        for (i = 0; i < dim; i++)
                                data[lines*dim+i] = v[i];

                        lines++;
                        count = 0;
                }
        }

        data_aux = (float *) malloc(sizeof(float) * lines);
        centroids = (float *) malloc(sizeof(float) * dim * k);
        centroids_sum = (float *) malloc(sizeof(float) * dim * k);
        centroids_count = (float *) malloc(sizeof(float) * k);

        printf("file has %d entries\n", lines);
```

Table 4.9: K-Means – Java source code – part 3

```
        srand(time(NULL));

        for (i = 0; i < k; i++) {
                printf("[");
                for (j = 0; j < dim; j++) {
                        centroids[i*dim+j] = random() / (RAND_MAX * 1.0);
                        printf("%f", centroids[i*dim+j]);
                        if (j < dim-1) {
                                printf(", ");
                        }
                }
                printf("]\n");
        }

        do {
                for (i = 0; i < dim * k; i++) {
                        centroids_sum[i] = 0.0;
                        centroids_count[i] = 0;
                }

                // defining the closest centroid
                for (i = 0; i < lines; i++) {
                        max = MYMAXFLOAT;
                        pos = -1;
                        for (j = 0; j < k; j++) {
                                d = Euclidean(&data[i*dim], &centroids[j*dim], dim);
                                if (d < max) {
                                        pos = j;
                                        max = d;
                                }
                        }

                        printf("Data entry %d is closer to cluster %d\n", i, pos);
                        data_aux[i] = pos;
                }

                // summing up the points
                for (i = 0; i < lines; i++) {
                        sum(&data[i*dim], data_aux[i], dim);
                }

                // points average
                for (i = 0; i < k; i++) {
                        average(i, lines, dim);
                }

                // updating the centroids
                for (i = 0; i < k; i++) {
                        variation += update(i, dim);
                }

                variation /= k;

                printf("variation %f\n", variation);

                // printing data
                print_data(lines, dim);

                // printing the new centroids
                for (i = 0; i < k; i++) {
                        print(i, dim);
                }

        } while (variation > PRECISION);

        return 0;
} height
```

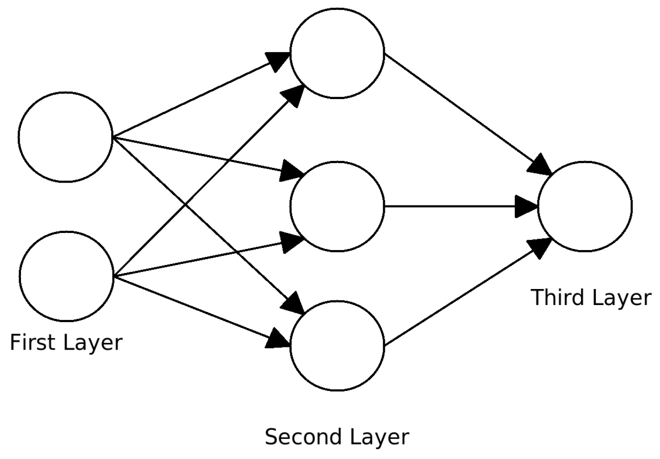


Figure 4.11: RBF Network Architecture

To exemplify how useful RBF Network is, we implemented it as presented in Tables 4.10, 4.11, 4.12 and 4.13. Then, we employ it to make the regression and prediction of the logistic function presented in Equation 4.17. We firstly generated $100,000$ data samples using the logistic function with $b = 3.8$ and $x_t = 0.5$.

$$x_{t+1} = b \cdot x_t \cdot (1.0 - x_t) \tag{4.17}$$

Before starting the execution, we had to prepare the information. We organized the training dataset in two columns. The first is the input value (one dimensional) and the second is the expected output for it. Table 4.14 presents a sample of this training dataset. It basically presents the value x_t to the network and expects, as output, the value of x_{t+1}.

During training, the RBF Network considers the gradient descent function presented in Equation 4.18 to update the associated weights, where the first index of w is the time instant and the second is the neuron identification, η is the learning parameter (which defines how fast the weights tend to the gradient, $e(I)$ is the expected output, $o(I)$ is the obtained output and $a(I)$ is the activation of the neuron i.

$$w_{t+1,i} = w_{t,i} + \eta \cdot (e(I) - o(I)) \cdot a_i(I) \tag{4.18}$$

During the training of the $99,900$ data points, the network makes the regression. The resulting regression is presented in Figures 4.12 and 4.13. The first shows how far RBF is from the expected values and the second confirms that the training made the network generate closer outputs. The prediction of 100 data points and its comparison to the expected observations are presented in Figure 4.14. Predictions were recursively done. This means that we predicted the $99,901$-th data point and used it to predict the $99,902$-th. Then, $99,902$-th was applied to predict the $99,903$-th and so on.

Then, we can consider the Mean Square Error (MSE) (equation 4.19, where N is the number of observations, o_i is the RBF output, and e_i is the expected output) to

Table 4.10: RBF Network – Source Code – Part 1

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

float *centers = NULL, *weights = NULL;
float *sigmas = NULL;
int count = 0, dim;
int numberofneurons = 0;
float eta = 0;

int dneurons = 0;
float *dynamic = NULL;
float weight = 30.0;          // predefined
float k = 0.05;               // predefined

float dynamic_activation(int dynamicid, float input) {
        float ret = 0.0;

        ret = (1.0 - exp(-k*(weight*dynamic[dynamicid]+input))) / (1.0 + exp(-k*(weight*dynamic[dynamicid]+input)));
        dynamic[dynamicid] = ret;

        return ret;
}

void compute_dynamic(float input[]) {
        int i;
        float in = 0.0;

        in = input[0];

        for (i = 0; i < dneurons; i++) {
                input[i+1] = dynamic_activation(i, in);
                in = input[i+1];
        }
}

float activation(int neuronid, float input[]) {
        int i;
        float sum = 0.0;

        for (i = 0; i < dim; i++) {
                sum += pow(input[i] - centers[neuronid*dim+i], 2);
        }

        return exp(-( pow(sqrt(sum), 2) / (2.0 * pow(sigmas[neuronid], 2) )));
}

float compute(float input[]) {
        int i;
        float sum = 0.0;

        for (i = 0; i < numberofneurons; i++) {
                sum += weights[i] * activation(i, input);
        }

        return sum;
}

void training(float input[], float output) {
        int i;
        float *ret = NULL;

        // Dynamic layer
        compute_dynamic(input);

        for (i = 0; i < numberofneurons; i++) {
                weights[i] = weights[i] + eta * (output - compute(input)) * activation(i, input);
        }
}
```

Table 4.11: RBF Network – Source Code – Part 2

```
void initialize_weights() {
        int i;
        float signal = 0;

        for (i = 0; i < numberofneurons; i++) {

                if (random() / (RAND_MAX * 1.0) < 0.5)
                        signal = -1;
                else
                        signal = 1;

                weights[i] = signal * (random() / (RAND_MAX * 1.0));
        }
}

void print_weights() {
        int i;

        printf("\nWeights\n");
        for (i = 0; i < numberofneurons; i++) {
                printf("%f\n", weights[i]);
        }
}

void slide(float input[], float ret) {
        int i;

        for (i = 0; i < dim-1; i++) {
                input[i] = input[i+1];
        }

        input[dim-1] = ret;
}

int main(int argc, char *argv[]) {
        FILE *fpcenters = NULL, *fpdata = NULL, *fpsigmas = NULL, *fpvalidation = NULL;
        float value = 0, sigma = 0;
        int i, j, predict;
        float *input = NULL;
        float ret;

        if (argc != 8) {
                printf("usage: %s centers_file dim data_file sigmas_file eta predict_points validation_file\n", argv[0]);
                exit(0);
        }

        fpcenters = fopen(argv[1], "r+");
        dim = atoi(argv[2]);
        fpdata = fopen(argv[3], "r+");
        fpsigmas = fopen(argv[4], "r+");
        eta = atof(argv[5]);
        predict = atoi(argv[6]);
        fpvalidation = fopen(argv[7], "r+");

        // reading the centroid values
        while (!feof(fpcenters)) {
                fscanf(fpcenters, "%f", &value);
                if (!feof(fpcenters)) {
                        centers = (float *) realloc(centers, sizeof(float) * (count + 1));
                        centers[count] = value;
                        count++;
                }
        }
```

measure how good is our regression and prediction. It is commonly considered when comparing artificial neural networks. Figures 4.15 and 4.16 present, respectively, the MSE for the training and prediction stages. We observe that during training the MSE

Table 4.12: RBF Network – Source Code – Part 3

```
// dynamic memory
dneurons = count - 1;
dynamic = (float *) malloc(sizeof(float) * dneurons); // one less neuron
for (i = 0; i < dneurons; i++) {
        dynamic[i] = 0;
}

// reading the sigma values
count = 0;
while (!feof(fpsigmas)) {
        fscanf(fpsigmas, "%f", &sigma);
        if (!feof(fpsigmas)) {
                sigmas = (float *) realloc(sigmas, sizeof(float) * (count + 1));
                sigmas[count] = sigma;
                count++;
        }
}

// printing out the centers
numberofneurons = count/dim;
for (i = 0; i < numberofneurons; i++) {
        for (j = 0; j < dim; j++) {
                printf("%f", centers[i*dim+j]);
        }
        printf("\n");
}

weights = (float *) malloc(sizeof(float) * numberofneurons);
srand(time(NULL));

initialize_weights();
print_weights();

// reading data points
input = (float *) malloc(sizeof(float) * dim);
i = 0;

while (!feof(fpdata)) {
        fscanf(fpdata, "%f", &value);
        if (!feof(fpdata)) {
                input[i] = value;
                i++;

                if (i == dim) {
                        ret = compute(input);

                        // reading the expected output
                        fscanf(fpdata, "%f", &value);

                        printf("\nOutput %f Expected %f\n", ret, value);

                        training(input, value);
                        print_weights();

                        i = 0;
                }
        }
}
```

decreases, because the gradient descent function helps the RBF network to approximate to the expected outcomes. However, the MSE is higher to predictions. This implies that additional error is introduced when predicting samples. This mainly happens because a small error in training propagates and influences all predictions.

$$MSE = \frac{\sum_{i=1}^{N}(o_i - e_i)^2}{N} \tag{4.19}$$

Table 4.13: RBF Network – Source Code – Part 4

```
        for (i = 0; i < predict; i++) {
                if (!feof(fpvalidation))
                        fscanf(fpvalidation, "%f", &value);
                else
                        value = 0;

                slide(input, ret);

                // Dynamic layer
                compute_dynamic(input);
                ret = compute(input);
                printf("Predicted %f Expected %f\n", ret, value);
        }

        return 0;
}
```

Table 4.14: RBF Network – Sample of the training dataset

Input (x_t)	**Output** (x_{t+1})
0.950000	0.180500
0.180500	0.562095
0.562095	0.935348
0.935348	0.229794
0.229794	0.672558
0.672558	0.836850

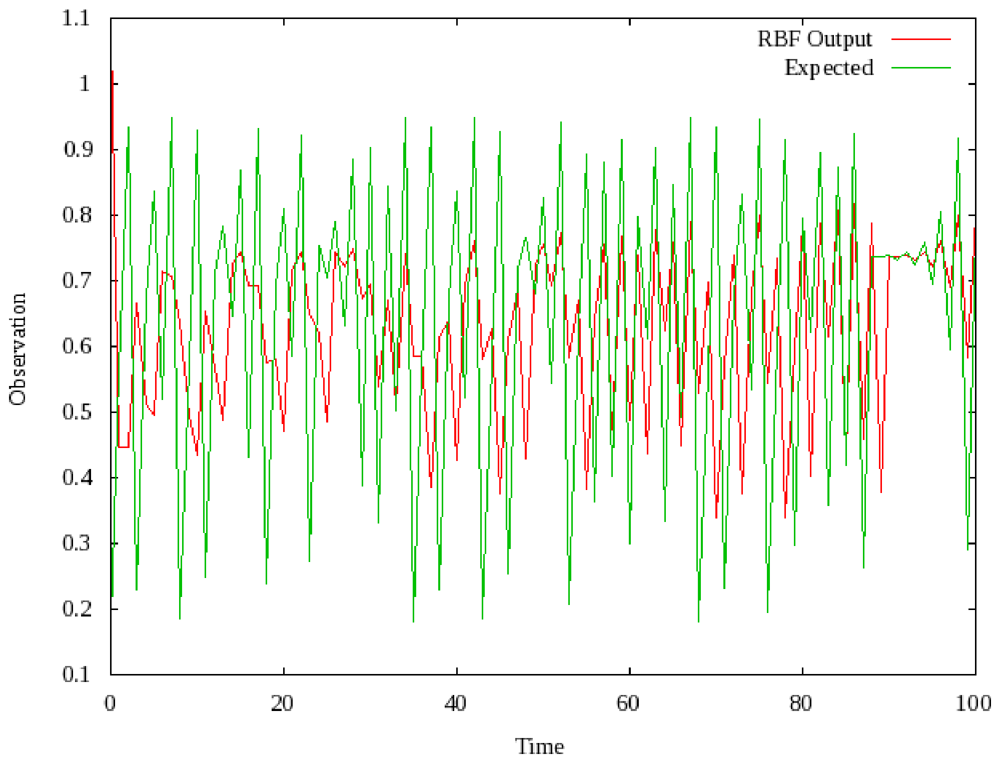


Figure 4.12: RBF – Regression at the beginning of training

4.6 Adaptive Resonance Theory 2A

4.6.1 Feature extraction

The first stage of the automatic classification model is composed of the pre-processing and feature extraction, which provide subsidies for starting up the classification. These

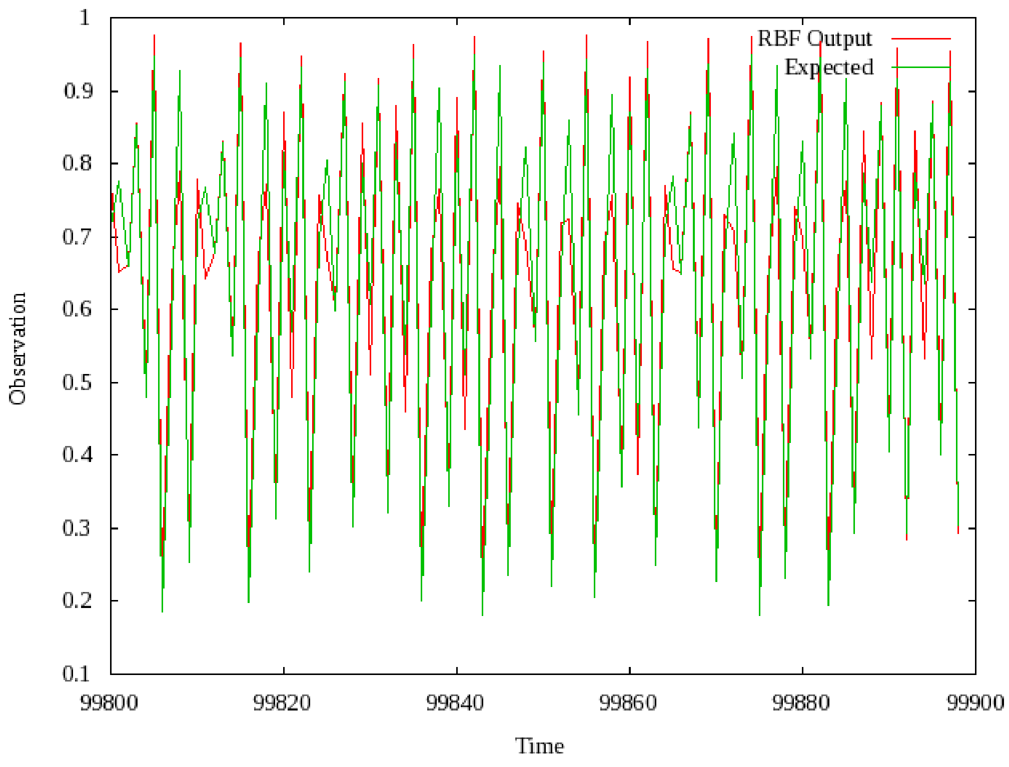


Figure 4.13: RBF – Regression at the end of training

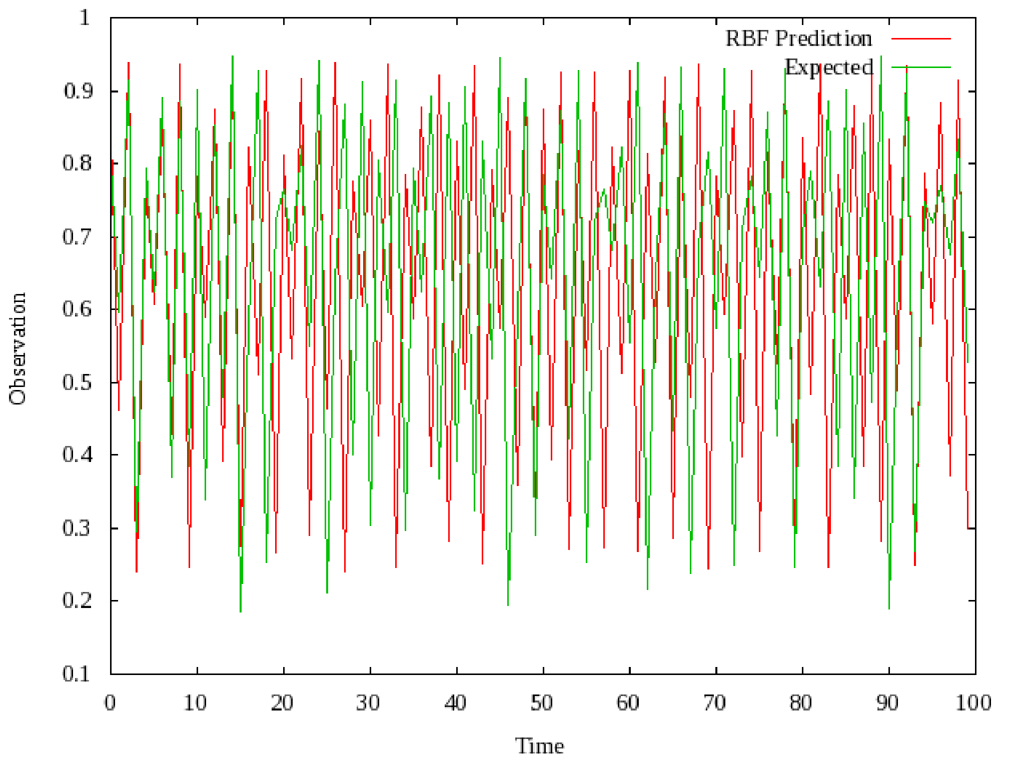


Figure 4.14: RBF – Prediction of 100 data points

are described as follows:

1. Conversion from document format to pure text - in this stage the documents are

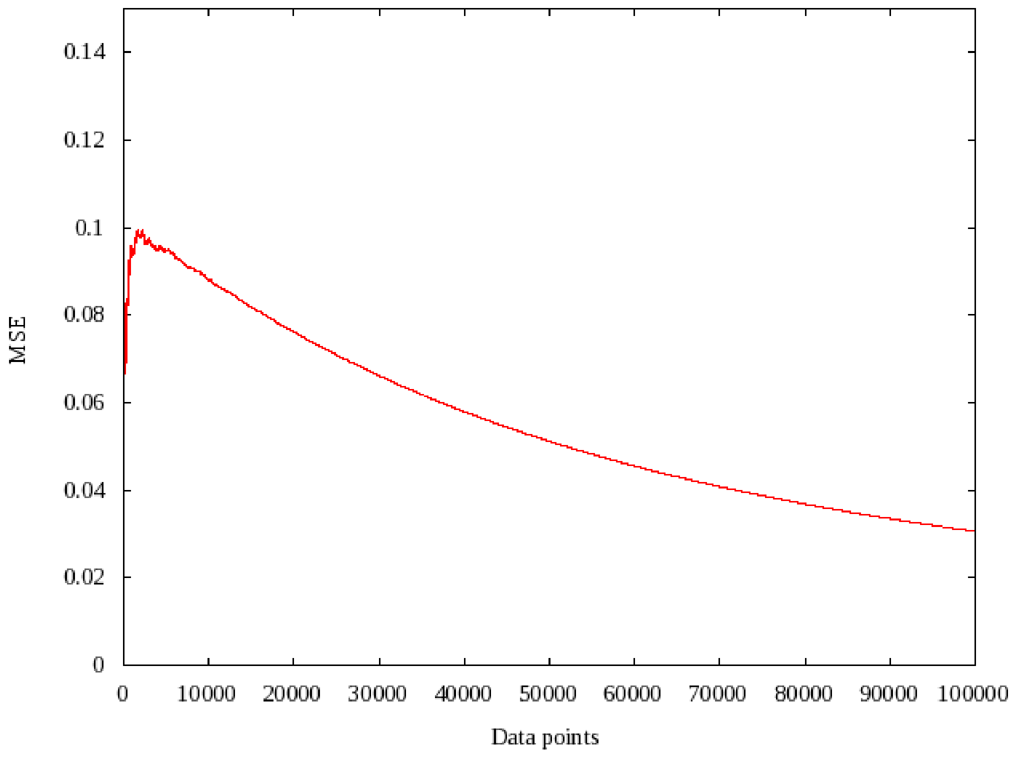


Figure 4.15: RBF – MSE for Training

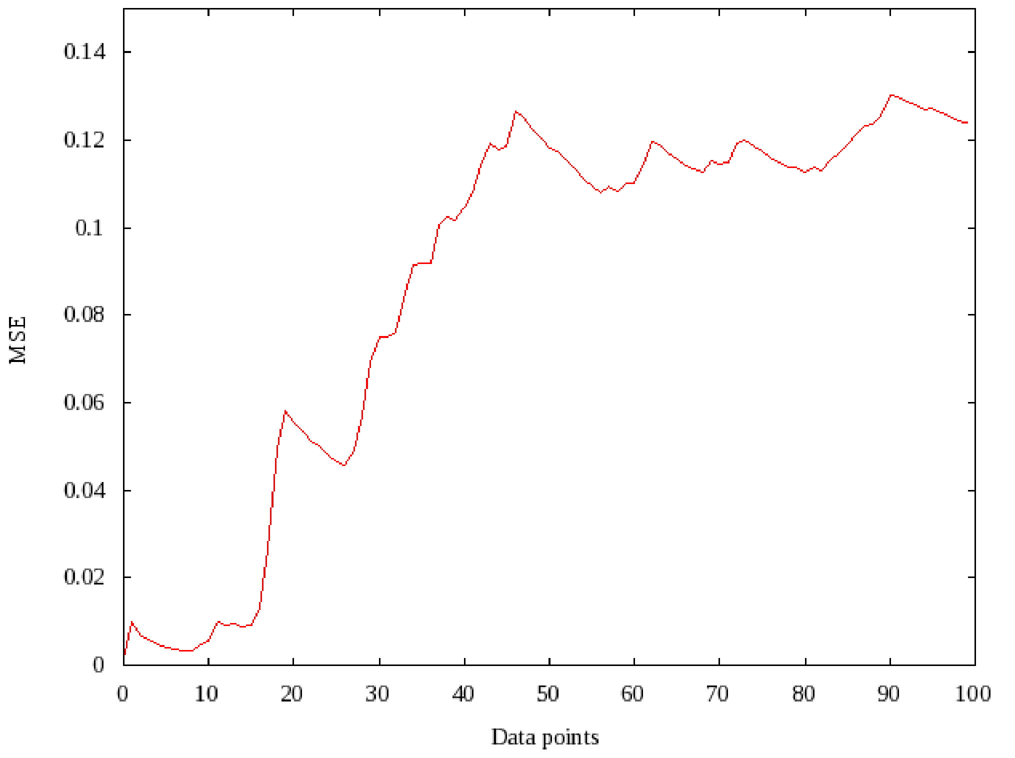


Figure 4.16: RBF – MSE for Prediction

converted into pure text to permit the counting of word occurrences of each document. The number of occurrences of a word in a certain document, named word frequency α in the document, is used as the classifier entry.

2. Stopwords Removal - after the document is converted, the words that do not show relevant significance are removed. Examples of such words are: *of*, *the*, *in*, *and*, *or* etc. Thus, only the most relevant words are left to represent the document.

3. Extraction of word roots contained in the text - with the main words of each document it is started the extraction of word roots. For instance, words such as *balancing*, *balance*, *balanced* are unified through the root *balanc*, giving the same meaning for them.

4. Root frequency counting - after the roots are extracted, they are counted and the number of occurrences is stored in a vector for each document. Such vectors contain the frequency of each word in a document. Each Vt_i vector (where $i = 0, 1, 2, ..., n$) shows the same number of elements, where $|Vt_1| = k, |Vt_2| = k, ..., |Vt_n| = k$. Elements of a same index quantify the frequency of the same word. Thus, a word w is at the same index j of the vectors $Vt_1, Vt_2, ..., Vt_n$.

4.6.2 Classification

The second stage is responsible for grouping and classifying the vector sets Vt_i (obtained in the feature extraction stage) by using an ART self-organizing artificial neural network architecture (23).

The ART (*Adaptive Resonance Theory*) neural network architectures are self-organizing and non-supervised which permit them to learn without any knowledge about the input patterns (21; 22). The ability of the ART network family differs from the other self-organizing architectures as it allows the user to control the similarity degree among the patterns grouped in a same representation unit (or cluster). This control allows the network to be sensitive to the differences existing in the input patterns and to be able to generate more or fewer classes in response to this control. Moreover, the learning in the networks ART is continuous: the network adapts itself to the incoming data, creating new processing units to learn the patterns, when required. The ART architecture allows the quick learning of input patterns represented by continuous values. This architecture has the following features: noise filtering, good computing and classification performance. The ART neural network family has been used in several domains, such as to recognize Chinese characters (41), interpretation of data from nuclear reactor sensors (120; 119; 64), image processing (115), detection of earth mines (38), treatment of satellite images (20) and robot control (9).

The ART network architecture is composed of two main components: the attention and orientation systems (figure 4.17). The attention system is provided with a pre-processing layer F_0, an input representation layer F_1 and a class representation layer F_2. The input and representation layers are interconnected through a set of adaptive weights called *bottom-up* ($F_1 \rightarrow F_2$) and *top-down* ($F_2 \rightarrow F_1$). The path from the neuron ith of the layer F_1 to the neuron jth of the layer F_2 is represented by w_{ij}. Likewise, the neuron jth of the layer F_2 is connected to the ith of the layer F_1 through the adaptive weight w_{ji}. These weights multiply the signals that are sent among the neuron layers and are responsible for storing the knowledge obtained by the network. The interactions between the layers F_1 and F_2 are controlled by the orientation system, which uses a vigilance parameter ρ, and the way through which the weights are updated, to obtain knowledge from the input patterns, is defined by the training algorithm.

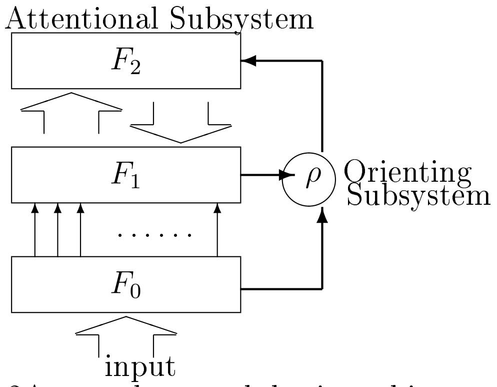


Figure 4.17: ART 2A neural network basic architecture

Training algorithm

The ART dynamics is defined by the vigilance parameter $\rho \in [0, 1]$ and the learning rate $\beta \in [0, 1]$. Initially, the output layer F_2 does not have any class. In our classification model, the input pattern I^0 is composed of a set of attributes associated to the text to be classified. The ART training algorithm itself is composed of the following stages: pre-processing, activation, search, resonance or reset and adaptation.

- **Pre-processing:** this phase performs input normalization operations I^0:

$$I = \aleph(F_0(\aleph(Vt^0))) \tag{4.20}$$

where $\aleph$ and F_0 describe the following operations:

$$\aleph(x) \equiv \frac{x}{\|x\|} \equiv \frac{x}{\sum_{i=0}^{n} x_i^2}, \; F_0(x) = \begin{cases} x & \text{if } x > \theta \\ 0 & \text{otherwise} \end{cases} \tag{4.21}$$

Such operations perform the Euclidean normalization and the noise filtering. The noise filtering, through the parameter θ, only makes sense if the main features of the input patterns, which lead to the creation of different classes, are exclusively represented by the highest values of the input components.

- **Activation:** this phase is responsible for sending out the incoming signals to the neurons of the representation layer F_2:

$$T_j = \begin{cases} Iw_{ij} & \text{if } j \text{ indexes a committed prototype} \\ \alpha \sum_j I_j & \text{otherwise} \end{cases} \tag{4.22}$$

  where T_j corresponds to the activation of the neuron j of the layer F_2. Initially, all the neurons are marked as uncommitted and become committed when their weights are adapted to learn a certain input pattern. The choice parameter α defines the maximum depth of search for a fitting cluster. With $\alpha = 0$, value used in this work, all committed prototypes are checked before an uncommitted prototype is chosen as a winner.

- **Search:** This phase is responsible for finding the candidate neuron to store the current pattern. The network competitive learning indicates the most activated neuron which is chosen as a candidate to represent the input pattern:

$$T_J \ = \ max\ \{T_j : \text{ for all } F_2 \text{ nodes}\} \tag{4.23}$$

- **Resonance or reset:** after selecting the most activated neuron, the *reset* condition is tested:

$$y_J > \rho \tag{4.24}$$

  If the inequality is real, the candidate neuron is chosen to store the pattern and the adaptation stage is initiated (resonance). If not, the winning neuron is inhibited and the searching stage is repeated (reset).

- **Adaptation:** this stage describes how the pattern will be learned by the network. Such stage comprehends the updating of the network weights for the winning neuron J, which, then, becomes committed:

$$w_{Ji}^{new} = \begin{cases} \aleph(\beta\aleph\Psi + (1-\beta)w_{Ji}^{old}) & \text{if } j \text{ indexes a committed prototype} \\ I & \text{otherwise} \end{cases} \tag{4.25}$$

$$\Psi_i \equiv \begin{cases} I_i & \text{if } w_{Ji}^{old} > \theta \\ 0 & \text{otherwise} \end{cases} \tag{4.26}$$

The table 4.15 illustrates the examples of values for the `ART` network parameters. The vigilance value ρ defines the number of classes that will be created by the network. The

Table 4.15: ART main parameters

Parameter	Description	Value example
m	number of input units	7
n	maximum number of representation units	15
θ	noise suppression parameter	$\theta = \frac{1}{\sqrt{m}}$
β	learning rate	$\beta = 0.7$
ρ	vigilance parameter	$\rho = 0.9$

value ρ forms a circular decision boundary with a radius of $\sqrt{2(1-\rho)}$ around the weight vector of each category (48). With $\rho = 0$, all input patterns are grouped into the same class. With $\rho = 1$, it is created a class for each input pattern presented to the network. The learning rate β defines the adaptation speed of the prototypes in response to the input patterns. The ART should not be used with $\beta \cong 1$, as the prototypes tend to jump among the input patterns associated to a class, instead of converging to the pattern average.

Each committed neuron of the layer F_2 defines a similar pattern group. The committed neuron set defines the classification generated by the network for the submitted values. As the input patterns over the process behavior are not previously labeled, it is required an algorithm to define a label to represent each class created by the network.

The ART learning algorithm is incremental. Thus, the neural network adapts to new inputs indefinitely, meaning that an ART network has both plasticity and stability (99). New clusters can be formed when the environment does not match any of the stored patterns, but the environment cannot change stored patterns unless they are sufficiently similar.

Categorization Algorithm

The categorization algorithm is built in accordance with the idea that the ART network weights resemble the input patterns that have been learned by a certain neuron of the layer F_2 (99). The ART network weights are also called prototypes because they define the direction for the data grouping. The data normalization operations performed by the ART network allows all the vectors to be canonically normalized. Thus, only the angle formed among the prototypes is preserved. If the process monitoring values are initially normalized (see equation 4.21), such values do not differ too much in their magnitude. In addition, according to the rule for updating the ART network weights (see equation 4.26), the prototypes are also normalized. Thus, each attribute contribution can be obtained, based on its value in the weight vector. Such value represents the attribute significance for the local grouping chosen by the network.

After the data grouping is performed by the ART network, a label is added to each neuron of the layer F_2. For this purpose, a significance matrix is defined, which is composed of a significance value set SV_{ij} (112). The significance matrix supports the decision about which components of the input vector are significant to label each committed neuron of the layer F_2. The significance values are directly obtained from the ART network weights, where: the number of columns of the significance matrix is equal to the number of committed neurons of the layer F_2, which represent the obtained classes; and the number of lines is equal to the number of components of the input vector. For instance, a significance matrix $SM = (SV_{ij})^{7\times4}$ is obtained through a network that has 4 classes to represent a document described by an input vector composed of $m = 7$ components.

The categorization stage is illustrated by the Algorithm 2. In order to detect the most important attributes to describe a class, the significance values of the attributes are normalized in relation to the sum of the total significance values of a certain class, that is, the sum of the elements of the column. After such normalization is done, the column values are arranged in a decreasing order and the accumulated frequency of the significance values is calculated. For labeling a class, the set of the most significant attributes is selected until the accumulated frequency sum does not exceed a certain threshold χ. At the end of the algorithm execution there will be a set of the most relevant attributes C for each category created by the network. These attributes label a class.

Algoritmo 2 Labeling of the Classes obtained by the ART network

```
1: defines the threshold value $\chi$ (p.e. $\chi = 55\%$) e
   $m$ (input vector dimension)
2: create the significance matrix, one column for each class created by the ART network
3: for each column created in the significance matrix do
4:    sum the significance values of each attribute
5:    normalize the significance values based on the sum
6:    calculate the distribution of the accumulated frequency
7:    arrange in a decreasing order the column attributes
8:    $sum := 0$
9:    $i := 1$
10:   $C := \emptyset$
11:   while ($sum \leq \chi$) and ($i \leq m$) do
12:      add the $attribute_i$ to the set $C$
13:      add the $attribute_i$ accumulated frequency to the $sum$ variable
14:      $i := i + 1$
15:   end while
16:   label the class based on the attributes contained in set $C$
17: end for
```

The categorization stage obtains a set of attributes C to represent each class. The elements of this set are afterwards used to index documents.

4.7 Self-Organizing Novelty Detection Neural Network

Time series may present unknown dynamics and behavior fluctuation that requires continuously updating knowledge structures to improve novelty detection. The updating process should integrate and accommodate novelty events into a normal behavior model, possibly incurring the revaluation of long-term memories. This has been addressed by the proposal of incremental techniques which are able to continuously restructure the behavior model without human supervision, such as self-organizing neural networks (78; 54) and regression techniques (76). However, such techniques have limitations when detecting novelties in unstable time series because they do not update normality/novelty thresholds and do not keep track of old knowledge (54).

These limitations motivated the proposal of an incremental learning neural network architecture, named Self-Organizing Novelty Detection (SONDE) (3), which represents unknown dynamics and fluctuation of established behavior patterns by updating knowledge structures and normality/novelty thresholds. The knowledge accumulated by SONDE is employed to detect temporal and non-temporal novelties as well as measure the level of novelties.

The basic knowledge representation unit of SONDE is the neuron which accumulates historical input patterns. The neuron continuously adapt itself according to input pattern variations. The SONDE architecture is composed of three layers (figure 4.18): a first input and pre-processing layer – in which patterns are optionally normalized what defines a unitary space for multidimensional input patterns, simplifying the SONDE parametrization ; a second layer with neurons – this layer represents the input pattern knowledge; and the last, named competitive layer – where occurs the selection of the most representative neuron (*Best-matching unit* – BMU) for a certain input pattern.

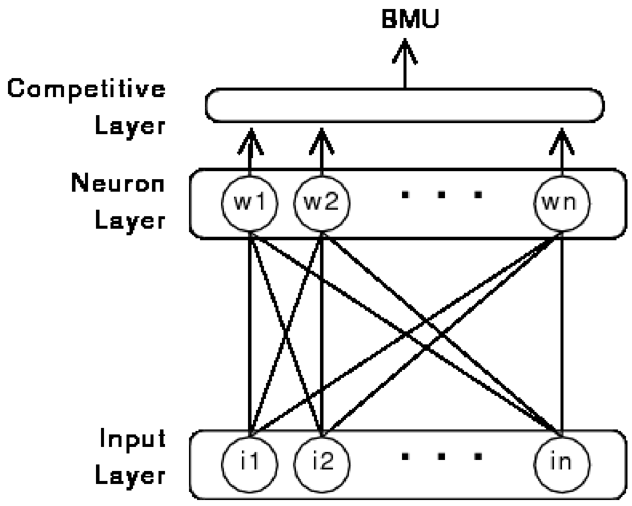


Figure 4.18: Self-Organizing Novelty Detection architecture

Let a multidimensional input pattern be defined as $\vec{I_t} \in \Re^n$, where t is the time instant.

The first SONDE layer may optionally normalize each input pattern $\vec{I_t}$, when $n > 1$, as follows $\vec{I_t} = \frac{\vec{I_t}}{\|\vec{I_t}\|}$. The second layer computes the distance of the pattern $\vec{I_t}$ to each neuron c and inform the last layer about it. Such computation considers three neuron components which are responsible by the individual knowledge representation:

1. the prototype $\vec{w}_c$ – this represents the tendency of the input patterns classified in the neuron c;
2. the average radius rad_c – this quantifies the dispersion of input patterns around the prototype $\vec{w}_c$; and
3. the minimum similarity degree α_c – it represents a minimum activation threshold to accept new input patterns in the neuron c.

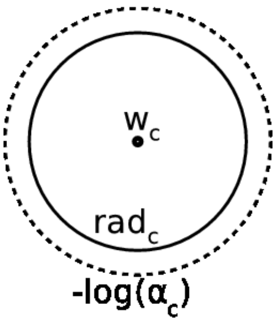


Figure 4.19: SONDE architecture: illustration of the neuron c and its components

These components are adapted when the neuron is elected as the Best-Matching Unit (BMU) by the competitive layer. This adaptation improves the representation of the most recent knowledge according to the equations 4.27 and 4.28, which are responsible for updating the prototype $\vec{w}_{BMU}$ and the average radius rad_{BMU}, respectively. The adaptation equations follow exponentially weighted moving averages (EWMA) (107), which allows the incremental update according to the inputs. These equations consider the learning parameters $\gamma \in [0, 1]$ and $\Omega \in [0, 1]$. The higher the parameter value is, the more the network forgets on previous patterns. The neuron adaptation causes an exponential network forgetfulness (as illustrated in section 4.7), allowing the architecture to follow behavior and tendency drifts.

$$\vec{w}_{BMU_t} = \vec{w}_{BMU_{t-1}} * (1 - \gamma) + \vec{I_t} * \gamma \tag{4.27}$$

$$rad_{BMU_t} = rad_{BMU_{t-1}} * (1 - \Omega) + \|\vec{I_t} - \vec{w}_{BMU_{t-1}}\| * \Omega \tag{4.28}$$

The last neuron component, the minimum similarity degree α_{BMU}, is adapted according to the equation 4.30, where p is the relative modification rate of the average radius rad_{BMU} (equation 4.29).

$$p = \frac{\|rad_{BMU_t} - rad_{BMU_{t-1}}\|}{\|\max(rad_{BMU_t}, rad_{BMU_{t-1}})\|} \tag{4.29}$$

$$\alpha_{BMU_t} = \min\left((1+p) * \alpha_{BMU_{t-1}}, \exp^{-(1+p)*rad_{BMU_t}}\right) \tag{4.30}$$

This adaptation mechanism for α_{BMU} ensures two learning tendencies, which consider the distance between the average radius (this defines an inner boundary of neuron specialization) and the minimum acceptance threshold for neuron representation (defined as $-\ln(\alpha_{BMU})$), according to equation 4.31. These tendencies are considered when computing the distance D (equation 4.31) applied by the adaptation mechanism as follows: 1) the greater is the distance D, the faster is the specialization to represent input patterns; 2) when patterns are uniformly classified within the neuron average radius, the similarity threshold $-\ln(\alpha_{BMU})$ tends to enclose towards the average radius, that characterizes the neuron adaptation to input patterns that it represents.

$$D = \mid rad_{BMU} - \ln(\alpha_{BMU}) \mid \tag{4.31}$$

In the third architecture layer the neuron competition occurs to represent each input pattern $\vec{I_t}$. The neuron with the highest activation a_c for the current input (equation 4.32) and that respects the minimum similarity degree α_c is selected as the Best-matching Unit. Afterwards, the neuron components are updated to represent the new information of $\vec{I_t}$.

$$a_c = \exp(-\|\vec{I_t} - \vec{w_c}\|) \tag{4.32}$$

If no neuron is capable of representing the vector $\vec{I_t}$, a new is created, indicating novelty. The new neuron prototype $\vec{w}_{new}$ is initially set with the value of the input pattern responsible for the creation. The minimum similarity degree α_{new} is defined according to a constant α_0 and the initial average radius rad_{new} equals to $-\ln(\alpha_0)$.

Implementation and behavior learning of SONDE

Simulations were conducted to evaluate the operations of the Self-Organizing Novelty Detection neural network and describe the influence of the learning parameters. Such parameters influence the exponentially weighted moving average (EWMA) equations of the network architecture. This motivated the first simulation to compute the forgetfulness effect of EWMA and a second one to evaluate how a neuron adapts and learns considering an uniform distribution $U(0, 1)$ of the input patterns.

In the first simulation, an evaluation of neuron adaptation parameters (EWMA weights defined in equations 4.27 and 4.28) was conducted aiming at illustrating the learning and forgetting behavior of the neural network architecture. For such evaluation, an equivalent formulation, shown in equation 4.33 (with weighting parameter $\psi \in [0, 1]$), is adopted, because it characterizes the behavior of an hypothetical EWMA.

$$average_t = average_{t-1} * (1 - \psi) + input_t * \psi \tag{4.33}$$

Using this formulation, we want to analyze the forgetfulness of the EWMA equation when considering different forgetting factors (P) according to the equation 4.34. This equation represents the forgetfulness caused by n new input patterns applied to the equation 4.33. The figure 4.20 shows the forgetfulness when applying a number of the input patterns as previously mentioned. This figure considers different forgetting factors P. Each figure curve shows the number of patterns necessary to forget the historical behavior (vertical axis) versus the weighting parameter ψ (horizontal axis) defined for different forgetting factors P.

$$P \leq \psi * (1 - \psi)^{n-1} \, n \in \mathbb{Z}, n \geq 1 \tag{4.34}$$

Considering the figure, we may estimate a value for the weighting parameter which defines the SONDE neuron forgetfulness. There are some ways to set such parameter, one of them is by adopting the highest value of the auto-correlation function for the input pattern series (such as the curve of forgetting factor $P = 0.1$). This helps the neuron memory to preserve the knowledge on the most important time slice (this is, within the auto-correlation period).

The next simulation has involved the learning evaluation of the EWMA equation 4.33 when classifying input patterns that follows an uniform distribution $U(0, 1)$. The objective is to illustrate the neuron adaptation capacity. In the first step we created a dataset using samples of the $U(0, 1)$. Each sample represents the distance of an input pattern to the prototype. In that sense, a value 0 represents an input pattern equals to the prototype and 1, the farthest input pattern that a neuron could accept. The results of this simulation are presented in figure 4.21 where the weighting parameter varied according to $psi \in \{0.0025, 0.01, 0.04, 0.08\}$. Considering such results, the lower the learning parameter ψ is, the lower is the learning speed and the sensitivity to the pattern noise.

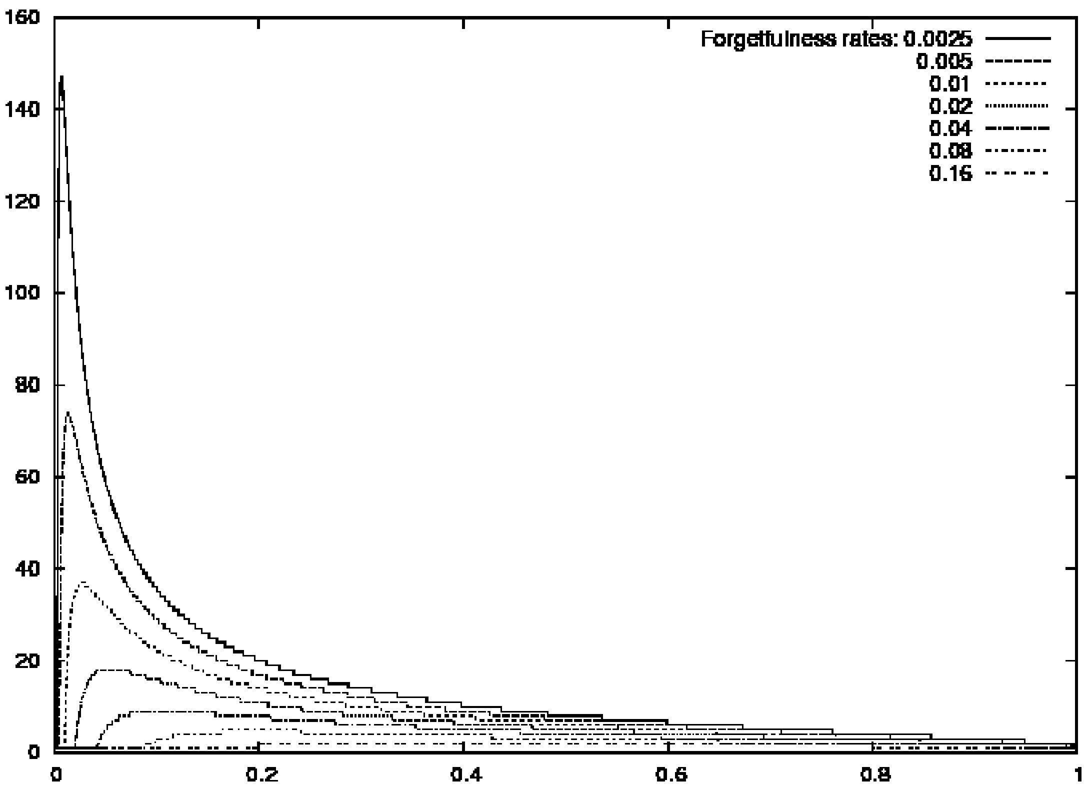


Figure 4.20: Number of patterns (n) necessary (axis y) to apply a forgetting factor of at least $P \in \{0.0025, 0.005, 0.01, 0.02, 0.04, 0.08, 0.16\}$ in a past pattern x_{t-n}, according to the parameter ψ

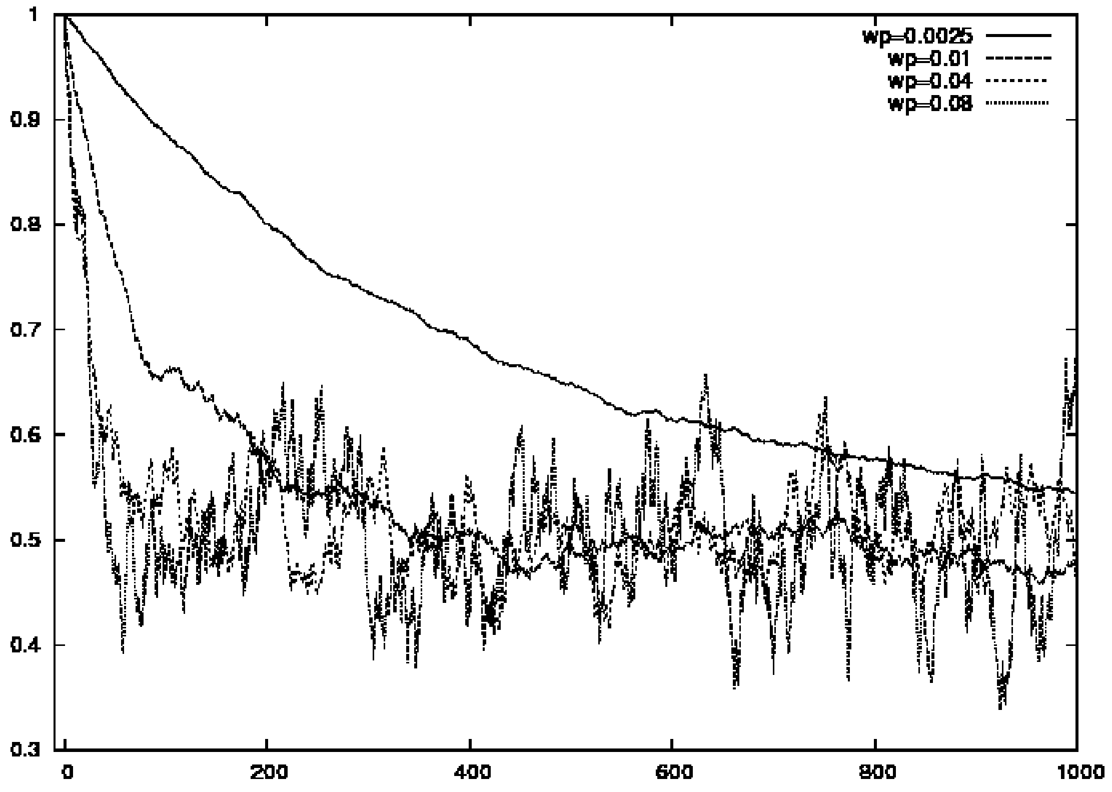


Figure 4.21: Moving average convergence to the mean of an uniform distribution $U(0, 1)$

Chapter

5

Metaheuristics

5.1 Simplex

5.2 Hill Climbing

5.3 Greedy Search

5.4 Tabu Search

5.5 Simulated Annealing

The simulated annealing meta-heuristic is inspired in the thermal treatment, named annealing, which aims at improving molecular organization of metals, by regulating or removing internal tensions and, consequently, finding a microstructure at less internal energy. The annealing process is organized in three steps: the controlled heating up to a certain temperature, which is below the metal fusion point, what permits the atomic reorganization (aiming failure removal); afterwards, the metal is kept in high temperature what gives kinetic energy to the atoms which can freely move over the structure; then the temperature is slowly decreased what smoothes the material and improves its ductability [1]. If the temperature decreases too fast, the final microstructure has higher internal energy, because of the internal tensions caused by the irregularity in the atomic spatial organization. In such situation, the thermal energy is so fast removed that the atoms

[1]The ductability is a physical property of materials of supporting the plastic deformation, under pressure, without fracturing.

do not have enough time and mobility to reorganize themselves in low energy spatial configurations.

Metropolis *et al.*(85) propose a simple algorithm to describe the thermal equilibrium of a solid when immersed at a fixed temperature. This algorithm uses the Monte Carlo method to obtain a micro-state sequence, where each next state is generated by adding a perturbation. This transforms the current state through a small distortion. The state transition results in an system internal energy variation and the new micro-state is accepted as the current. The basic simulated annealing algorithm uses a sequence of evaluations of the Metropolis' algorithm to different temperature values (which must slowly and gradually decrease). This algorithm motivated works on optimization, where researchers look for a function minimum which corresponds to the the lowest energy value after the thermal annealing treatment. Such idea was proposed by Kirkpatrick (66), who demonstrated that the Metropolis' algorithm could be employed in generic optimization problems by establishing a relation between the physical annealing and the optimization problem elements.

When optimizing a problem, we can define the current solution as s and a neighbor solution as s', which are evaluated using an objective function $f()$, also known as cost function. In the SA algorithm, the acceptance criterion, which determines if s' will substitute the current solution s, is probabilistic and similar to the criterion used in the Metropolis' algorithm. In this way, better neighbor solutions are always accepted and bad solutions may be accepted according to the probabilistic criterion. This ensures a way to get out of local minima of the energy function. The acceptance level for the bad solutions depends on the iteration number k of the algorithm and it is related to the temperature control parameter T. When the temperature T is high, the transitions, which degrade the cost function, are easily accepted; as the temperature decreases, the lower is the probability to accept bad solutions. Then, the basic SA algorithm is characterized by two operations:

- the parameter T is decreased along the iterations k, based on a annealing profile $\phi(k)$;
- cycle, of size $nrep$, of generation and acceptance of probabilistic solutions, where the temperature T is the probabilistic adjust parameter.

The basic SA algorithm is presented in 3. At the beginning an initial solution is defined, which is the starting point to new solutions. An new solution is generated adding a perturbation in the current one, afterwards it is evaluated using the cost function $f()$. A better solution is always accepted, otherwise the solution may be accepted with a certain probability, which depends on the current temperature. The algorithm starts at a high temperature, then the probability to accept a bad solution is higher. The stop criterion is when the temperature reaches a predefined threshold.

The main algorithm parameters are the initial temperature T, the number of evaluations *nrep* at a fixed temperature and the learning rate α. An initial high temperature allows a better exploration of the search space, helping to get out of local minima. The annealing mode is associated to this parameter, expressed through the function $\phi(k)$. The number of evaluation *nrep* defines the search size for neighbor solutions. The learning rate α, which assumes values between $[0, 1]$, defines how much the temperature has to decrease per iteration. The closer α is to one, the slower is the learning rate, what consequently increases the number of iterations. When α tends to zero, the number of iterations tends to decrease and, consecutively, the algorithm runtime, although this may produce bad results because it reduces the search space.

Algoritmo 3 Basic Algorithm of Simulated Annealing

```
1: define the initial solution $s$, the initial temperature $T_0$ , the learning rate $alpha$, the
   cooling schedule function $\phi(k)$ and the minimal number of iterations $k_{min}$;
2: $k = 0$, $T = T_0$;
3: while system is warm do
4:    define a number of evaluations $nrep$ for the current $T$;
5:    for each nrep do
6:       generate a new solution $s'$;
7:       compute $\Delta f = f(s') - f(s)$;
8:       if $\Delta f < 0$ then
9:          $s = s'$;
10:      else
11:         generate a random value $\mu$ in the range [0,1]
12:         if $exp(-\frac{f(s')-f(s)}{T}) > \mu$ then
13:            $s = s'$
14:         end if
15:      end if
16:   end for
17:   decrease the temperature ($T = \phi(k)$)
18:   $k = k + 1$;
19: end while
```

This work considers improvements to the basic SA algorithm. The first improvement consists in adapting the number of evaluations *nrep* to the temperature. As the temperature decreases, the higher is the number of evaluations, what better explores the solution neighborhood. The number of iterations *nrep* to the current temperature corresponds to the equation 5.1, where k is the current iteration and k_{min} is the minimum number of evaluations at a fixed temperature.

$$nrep = k + k_{min} \tag{5.1}$$

In this way, the number of evaluations *nrep* linearly increases in relation to the current iteration, the search becomes ampler at lower temperatures. Another improvement is that we save the best solution obtained during the iterations. Then, if the algorithm converges to a local minimum and it cannot get out of it, a better solution can be defined.

The annealing profile corresponds to the equation 5.2, which proposes an exponential temperature decrease, considering the learning rate α and the current iteration. The curves to three different learning rates are presented in the figure 5.1 (they were obtained by using the equation 5.2; in the following graphs, $\alpha = \frac{alpha}{100}$). We observed that as the learning rate increases, smoother is the decrease applied to the current temperature. The advantage of an α close to 1 is the quality of the final solution, although it is more processing consuming.

$$T_{k+1} = T_0 \times \alpha^k \tag{5.2}$$

Figure 5.1: Temperature cooling scheme for different α values

5.6 Genetic algorithms

Genetic Algorithms (`GA`) are being applied as search and optimization techniques in several domains. These algorithms are based on nature select mechanisms focusing at survival of the most capable individuals. GA does not always give the best possible solution, however provides good local solutions for NP-complete problems.

The problem solution using genetic algorithms involves two different aspects: solution encoding into the form of chromosomes, where each chromosome represents a possible solution, and a fitness function applied to find the best solution.

Different encoding techniques can be used for different kind of problems, such as binary strings, bitmaps, real numbers, and so on. The fitness function is responsible for the evaluation of possible solutions. This function receives a chromosome as parameter and returns a real number, informing the quality of the obtained solution, e.g., how adequate is the solution for the currently studied problem.

The most adequate chromosomes are identified and stored during the evolution process. The weakest ones, on the other side, are eliminated. Different techniques can be applied for the identification of the best chromosomes, such as the proportional selection, ranking selection and tournament-based selection (10; 11).

In the proportional selection, individuals are transfered to the next generation according to their fitness value probability proportion. One of the possible implementations of this technique consists in the usage of a roulette, divided into N parts, N being the number of individuals (chromosomes) in the current population. The size of each part is proportional to the fitness value of each individual. The roulette is rotated N times afterwards, and at each turn the appointed individual is selected and inserted into the new population.

Ranking-based selection can be subdivided into two steps. During the first one, the solutions are ordered according to their fitness function values. Once the list is ordered, each individual receives a new fitness function value equivalent to its position in the ranking. After that, a procedure that selects the individuals, according to their ranking position, is applied. Thus, the individuals with better ranking position have more chances to be selected.

Finally, a tournament-based selection does not automatically attribute probabilities to individuals. A tournament size (k) is defined, with $k \geq 2$ individuals. Then, k individuals are chosen randomly from the current population, and their fitness functions are compared. The individual with best fitness value is selected for reproduction. The k value is defined by the user, representing the selection pressure – e.g., the speed with the strongest individuals will dominate the population, generating the extermination of the weakest ones.

Once selected the individuals for the reproduction, it is necessary to modify their genetic characteristics using reproduction techniques known as genetic operators. The most common operators are crossover and mutation. The single-point crossover operator is the most used one. In order to apply it, two individuals (parents) are selected and two new individuals are created from them (children). A single random splitting point is selected in parents chromosomes, and the new chromosomes are created from the combination of the parents, as shown on figure 5.2. The figure 5.2 (a) shows the parent individuals and the splitting point marked by | symbol. New individuals created from the combination of the parent chromosomes are shown on figure 5.2 (b), illustrating the crossover operator.

$$\begin{array}{c} X_1X_2|X_3X_4X_5X_6 \\ Y_1Y_2|Y_3Y_4Y_5Y_6 \end{array} \qquad \begin{array}{c} X_1X_2|Y_3Y_4Y_5Y_6 \\ Y_1Y_2|X_3X_4X_5X_6 \end{array}$$

(a) Before the crossover **(b) After the crossover**

Figure 5.2: Crossover operator

The mutation operator is used for changing a single gene value for a new random one. When an individual is represented by a bitmap, this operation consists of a random choice of a chromosome gene and the swapping of its value from 1 to 0 (or from 0 to 1, correspondently). The goal of the mutation operator is to maintain the diversity of a population, always allowing a chromosome to cover a significantly large result space (51). It is usually applied at a low rate, as at high ones the results tends to be random.

Another technique called niching consists in division of the population into species (that group individuals with similar characteristics), reducing their competition for resources and creating stable sub-populations with each one of them concentrated on a single niche of the results space. Niching techniques are known for their capacity of creation and maintainance of diverse population. It is possible to cite two most common niching techniques: sharing and crowding (103).

The sharing mechanism consists in the alteration of the fitness function attribution for an individual. The fitness value for each individual is modified according to the number of individuals similar to currently selected one (97). The fitness sharing function of an individual, called F', equals to its fitness function value F divided by its niching counter. The niching counter represents the sum of the values of sharing function (sh) among currently selected individual and all individuals of the population (including the currently selected one). The formula 5.3 defines the fitness sharing function of an individual i, with n being the total number of individuals in the population.

$$F'(i) = \frac{F(i)}{\sum_{j=1}^{n} sh(d(i,j))} \tag{5.3}$$

The sharing function is defined according to a distance d between two elements in the population, and returns 1 when both elements have the same value, 0 when their difference is bigger than the *dissimilarity threshold* and an intermediary value in range [0..1] according to their dissimilarity levels. A dissimilarity threshold is defined by a constant value σ_{share}. If a distance between two elements of the population is bigger or equals to the σ_{share} values, they don't affect the fitness function of each other.

Thus,

$$sh(d) = \begin{cases} 1 - (d - \sigma_{share})^{\alpha} & \text{, if } d < \sigma_{share} \\ 0 & \text{, otherwise.} \end{cases} \tag{5.4}$$

with α being a fitness function regulation constant, with usual value of 1.

Sharing can be used considering both phenotype and genotype distances.

Crowding technique, on its turn, inserts new individuals into population in place of existent similar individuals, thus maintaining population diversity. In the same way as the sharing technique, it uses distance measures among individuals (genotypic or phenotypic) in order to find similar individuals (10).

5.7 Hopfield Artificial Neural Network

Artificial neural networks were developed based on human neuron behavior (47). These networks are usually composed of layers, where each layer contains a set of neurons or *units*. A neuron i receives inputs which are computed according to a function 5.5, where net_i is the input values of the neuron i, x_j is each input value j to the neuron i and w_{ij} is the weight associated to the input signal x_j which will be forwarded to the neuron i. The net_i value is applied into a function which returns the neuron activation value $a_i(net_i)$. This value is submitted to the output function of a neuron 5.6 which is propagated to neurons in the next layer.

$$net_i = \sum_j x_j \dot{w}_{ij} \tag{5.5}$$

$$o_i = a_i(net_i) \tag{5.6}$$

In order to solve a classification problem, the information storage (or memory), it is used a serie of neurons organized in layers. Usually there is one input layer that receives data, hidden layers to receive output values from input layer. The hidden layer also calculates neuron activations for the output layer. The serie of layers used for solving a problem is called the neural network architecture.

The Hopfield artificial neural network (52) was introduced as an associative memory between input and output data. This network realizes association of inputs and outputs using an energy function which is based on a Lyapunov energy function. This function is applied in dynamic systems theory in order to evaluate the existence of stable states. The goal of such systems is to prove that modifications in the system state results in an energy function decreasing, up to a minimum value. The usage of this energy function motivated the adoption of the Hopfield network for solving optimization problems. In such problems the goal is to obtain the global minimum of the energy function, what represents the best solution for a certain situation. The energy function surface is stored in the network weight values.

The output function adopted for Hopfield networks (40) is defined in the equation 5.7, with γ being the constant gain parameter and u_i is the network input (net_i) for the neuron i.

$$v_i = g_i(\gamma u_i) = \frac{1}{2}(1 + tanh(\gamma u_i)) \tag{5.7}$$

In order to solve an optimization problem using the Hopfield network it is necessary to define an energy function to be applied on the matrix which represents a solution for the system. As an example, it is possible to consider the traveling salesman example (40) where it is necessary to visit a serie of cities with the following restrictions: the best route, visiting all the cities and each cite only once. Each solution for the problem is represented in form of a matrix where the lines are cities and the columns the visiting order. Thus, the line r_0 represents city A, the line r_1 represents city B and so on. The column c_0 represents the first city to be visited, c_1 the second and so on.

The figure 5.8 gives a sample solution for the traveling salesman problem. The value 1 means the visit to a city. It is possible to observe that each city is not visited more than once and that all cities are visited at lest once. The first visit occurs in city r_2, next on in city r_0 and, finally, in city r_1.

$$\begin{matrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{matrix} \tag{5.8}$$

The energy function is applied to the solution matrix. This function must not favor undesired states. For example, in order to unfavor more than one visit to a city, it is necessary to perform the sum of products of matrix values for each city according to equation 5.9. In this equation, X represents a city (matrix line), i and j represents the column indexes and v_{Xi} and v_{Xj} the values stored at line X and columns i and j.

In a stable state and locating the global minimum, the equation 5.9 should return zero, as each city was visited once. If a city was visited more than once, this equation returns positive values, what is used to decrease the output values of the neural network neurons. When neurons offer an output (matrix like the one shown in figure 5.8) that satisfies all the energy function equations, the system stops executing and the neural network generates the sequence of visits to cities and the elapsed distance.

$$\sum_{X=1}^{n}\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} v_{Xi}\cdot v_{Xj} \tag{5.9}$$

The equation 5.9 analyzes only one of the restrictions for the solution. Other terms of the function must be formulated. The equation 5.10 presents the second term which ensures the solution will only be considered valid if one city was visited at a time. Two cities cannot be visited simultaneously by the same traveler. In the same way, if a column of the matrix stores the sum of products and it is equal to zero, the proposed solution satisfies this term of the energy function. If this value is greater than zero, two or more cities may have been visited simultaneously. As this is not possible according to the problem definition, this solution must be marked as an invalid one.

$$\sum_{i=1}^{n}\sum_{X=1}^{n}\sum_{Y=1,Y\neq X}^{n} v_{Xi}\cdot v_{Yi} \tag{5.10}$$

The third term of the energy function is shown in equation 5.11. This term guarantees that the number of visits to cities must be equal to the number of cities (n). If this value differs, new matrix values must be computed in order to decrease the neurons input values, providing lower output values.

$$(\sum_{X=1}^{n}\sum_{i=1}^{n} v_{Xi} - n)^2 \tag{5.11}$$

The last term, shown in equation 5.12, aims to minimize the distance among cities, where d_{XY} is the distance between cities X and Y, these are stored in another matrix. This term is used to compute the total cost for a path passing through all the cities and returning to the first one.

$$\sum_{X=1}^{n}\sum_{Y=1,Y\neq X}^{n}\sum_{i=1}^{n} d_{XY} v_{Xi}(v_{Y,i+1} + v_{Y,i-1}) \tag{5.12}$$

The terms shown in the equation 5.9, 5.10, 5.11 and 5.12 compose the energy function 5.13 which must be minimized, with A, B, C and D representing the relevance of each term.

$$\begin{aligned} E = A \cdot \sum_{X=1}^{n}\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} v_{Xi} \cdot v_{Xj} \\ +B \cdot \sum_{i=1}^{n}\sum_{X=1}^{n}\sum_{Y=1,Y\neq X}^{n} v_{Xi} \cdot v_{Yi} \\ +C \cdot (\sum_{X=1}^{n}\sum_{i=1}^{n} v_{Xi} - n)^2 \\ +D \cdot \sum_{X=1}^{n}\sum_{Y=1,Y\neq X}^{n}\sum_{i=1}^{n} d_{XY} v_{Xi}(v_{Y,i+1} + v_{Y,i-1}) \end{aligned} \tag{5.13}$$

The Hopfield artificial neural network applies the energy function on a possible solution, evaluates it and decreases the input values of the neurons. After that, it proposes a new solution to be analyzed. This cycle is executed until a solution, which satisfies all the energy function terms, is obtained.

5.8 Boltzmann Machine

5.9 Ant Colony Optimization

Chapter

6

Dynamical Systems

6.1 Considerações iniciais

O protótipo do `MidHPC` permitiu comprovar as contribuições da utilização de conhecimentos sobre aplicações e recursos computacionais na otimização de escalonamento de processos. Esses conhecimentos são atualmente obtidos por meio de ferramentas de *benchmark*, monitores de sistema e de processos (83; 100).

Os *benchmarks* extraem capacidades máximas de recursos tais como processador, leitura e escrita em memória primária, meio de armazenamento secundário e redes. O monitor de sistemas avalia a carga instantânea de recursos. O monitor de processos intercepta eventos de utilização de recursos (tais como volume de processamento, acesso à memória principal, leitura e escrita em rede e disco rígido) e os armazena em uma base de dados. Sobre essa base é aplicado um algoritmo de aprendizado baseado em instâncias, a fim de estimar comportamentos de novas aplicações submetidas ao sistema. Esses comportamentos são representados por médias; conseqüentemente, não se observa, avalia ou emprega suas variações no tempo.

A compreensão dessas variações motivou o estudo e aplicação de conceitos sobre sistemas dinâmicos com o objetivo de melhorar os atuais suportes de otimização e demais aspectos de autonomia em grades computacionais. Conceitos sobre sistemas dinâmicos são apresentados neste capítulo.

6.2 Conceitos

Um sistema dinâmico é composto por um conjunto de estados possíveis e uma regra que determina seu estado atual em função do passado. Para exemplificar, considere a equação $x_{n+1} = 2x_n$ como representação da taxa de crescimento de uma população de indivíduos no tempo, onde n indica o instante de tempo e x_n define, precisamente, o tamanho da população em n. Nesse caso a regra, ou equação, define o próximo estado, ou tamanho da população no próximo instante, em função do passado. Sistemas com tais tipos de regras são denominados determinísticos. Há, contudo, outra classe de sistemas dinâmicos sujeita a outros efeitos além do passado. Considere um modelo de predição de preços de ações do mercado em função do tempo. O preço atual é composto por uma equação que considera preços anteriores. Contudo, essa equação conta, também, com uma variável aleatória, que modifica seus resultados. Esse tipo de sistema é denominado dinâmico estocástico ou aleatório (6).

Para melhor compreender um sistema dinâmico considere uma seqüência de observações de uma variável aleatória X no tempo, ou série temporal, $\{x_0, x_1, ..., x_{n-1}\}$. Esses eventos podem ser monitorados de forma discreta ou contínua, oriundos de regras determinísticas ou de processos estocásticos. A fim de compreender os próximos estados desse sistema, deve-se investigar modelos matemáticos capazes de representá-lo. Seus comportamentos são simples de serem estudados, caso se conheça sua regra (conjunto de equações que define o próximo estado com base no passado) e ele seja determinístico. Conhecendo a regra e sendo ele definido por processo estocástico, pode-se, também, compreender suas tendências comportamentais. Desconhecendo a regra, deve-se encontrar um conjunto de equações capaz de representá-lo, o que se torna mais complexo para sistemas estocásticos, pois há termos aleatórios.

Uma das técnicas mais antigas e estudadas para obter as regras que governam um sistema dinâmico é o modelo auto-regressivo (AR) (92). Esse modelo visa encontrar uma equação na forma $x_k = c + \sum_{n=0}^{T-1} a_n \cdot x_{k-n} + \epsilon_k$ que permite obter o próximo estado x_k em função de uma soma de termos passados (onde c é uma constante, $a_0, a_1, \cdots, a_{T-1}$ são parâmetros do modelo e ϵ_k é um ruído branco, sinal ou processo aleatório). Esse modelo apresenta erros médios de predição ótimos para séries lineares (15), contudo gera resultados insatisfatórios para séries mais complexas, as quais motivaram aproximações locais (24; 123) (que também apresentam limitações para séries caóticas), e o estudo de regularidades internas de sistemas dinâmicos (6) (pontos fixos, órbitas, etc.).

6.3 Estudo de Órbitas

Ao modelar o conhecimento embutido em um sistema dinâmico compreende-se a repetição de seus padrões a qual permite, por exemplo, conhecer suas tendências, realizar predições e classificar suas operações. Essas possibilidades motivam diferentes áreas da ciência, dentre elas a voltada para o estudo de populações (6). Considere, por exemplo, que o tamanho de uma população dobra a cada hora de observação, segundo a equação $f(x) = 2x$. A evolução temporal da população é definida por $f^k(x)$, onde k representa o número de instantes de tempo futuros que será feita a próxima observação. Dessa forma, caso a população inicial seja igual a 10, após uma hora ter-se-á 20 e após $k = 3$, onde $f^3(x) = f(f(f(x)))$, 80. De acordo com a equação dada, a população terá fator de crescimento exponencial igual a 2.

Table 6.1: Tamanho da população aplicando a função $f^k(x)$

k	$f^k(x)$
0	$0,01$
1	$0,02$
2	$0,04$
3	$0,08$
4	$0,16$
5	$0,32$
6	$0,64$
7	$1,28$
8	$2,56$
9	$5,12$
10	$10,24$

A tabela 6.1 apresenta os resultados para uma população inicial igual a $0,01$ milhão. Um modelo denominado Cobweb plot (6) foi adotado a fim de estudar a variação dos valores produzidos, ou órbita, por esse sistema (considerando que uma saída no instante t torna-se entrada em $t + 1$). Segundo esse modelo, traça-se uma curva com os valores das saídas produzidas em relação às entradas e uma reta diagonal $g(x) = x$. A órbita do sistema em relação a uma condição inicial é dada como segue.

Considere uma entrada inicial, seja $x = 0,01$, calcule a saída $f(0,01) = 0,02$. Em seguida, considere $0,02$ como entrada e calcule $f(0,02)$. O Cobweb plot auxilia na representação desses cálculos por meio de segmentos de reta que tocam $f(x)$ e $g(x)$. Por exemplo, inicialmente traça-se um segmento de reta vertical a partir da entrada $0,01$ até tocar a função $f(x)$. Esse segmento indica que a saída para $0,01$ é igual a $f(0,01) = 0,02$. Em seguida, traça-se um segmento horizontal do padrão entrada-saída $(0,01; 0,02)$ até tocar na reta diagonal $g(x)$. Em seguida, traça-se outro segmento de reta vertical com origem no ponto que toca em $g(x)$ até chegar em $f(x)$, tal como apresentado na figura 6.1. Os pares de valores entrada-saída obtidos, ou seja, os pares de pontos da curva $f(x)$, representam as saídas produzidas pelo sistema, sua órbita ou, também, variação no tempo.

No exemplo anterior, $f(x)$ apresenta a órbita de um sistema com características lineares e que tende a gerar saídas infinitas. Contudo, a fim de melhor caracterizar o tamanho de populações, deve-se considerar funções que impõem limites máximos de crescimento, respeitando os recursos escassos do ambiente. Para isso, seja, por exemplo, a função $h(x) = 2x(1 - x)$ onde x representa a população de entrada em milhões. Nesse caso, a população não é simplesmente resultante de x, mas sim do produto de x pelo termo

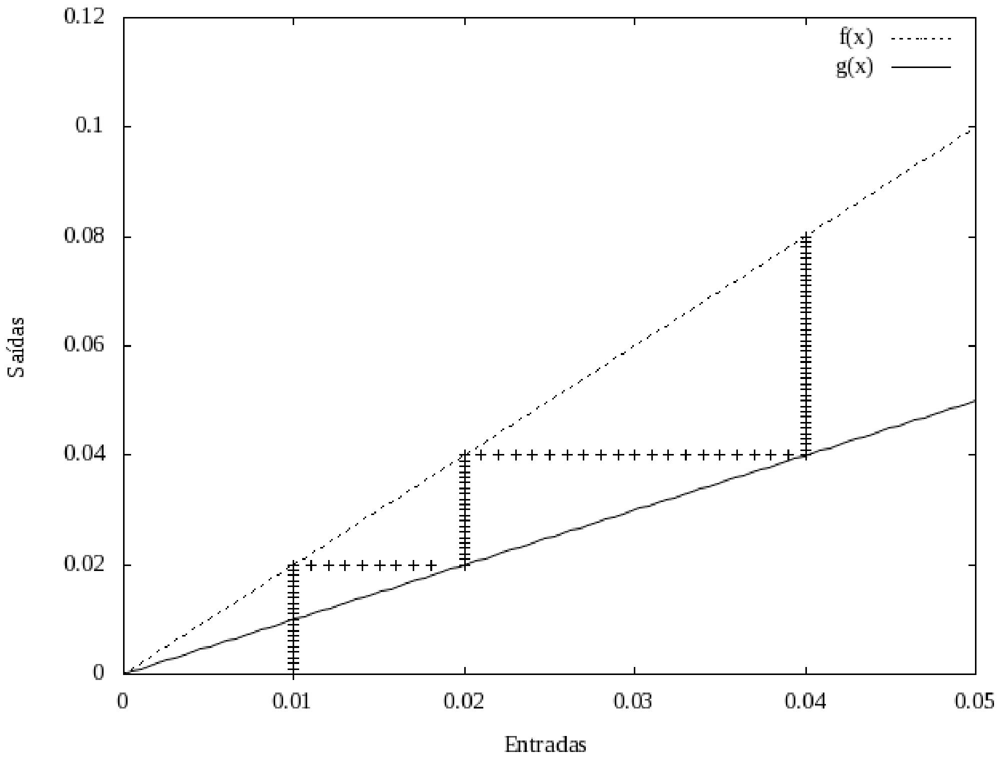


Figure 6.1: Órbita associada à função $f(x)$

$(1-x)$, o que gera um efeito não linear no crescimento. Nessa circunstância, $h(x)$ define um crescimento logístico para a população, o que é mais próximo de situações reais (6).

Table 6.2: Tamanho da população aplicando a função $h^k(x)$

k	$f^k(x)$
0	$0,010$
1	$0,019$
2	$0,038$
3	$0,074$
4	$0,138$
5	$0,238$
6	$0,362$
7	$0,462$
8	$0,497$
9	$0,499$
10	$0,499$
11	$0,500$
12	$0,500$

A tabela 6.2 apresenta resultados dessa função para uma população inicial de $0,01$ milhão. Para estudar a órbita desse sistema aplica-se o mesmo método, Cobweb plot, anteriormente abordado. A figura 6.2 permite observar que, em contraste com o mapeamento linear anteriormente considerado, para quaisquer valores entre $[0,0;0,5]$, a função

$h(x)$ resulta em valores próximos de $0,5$.

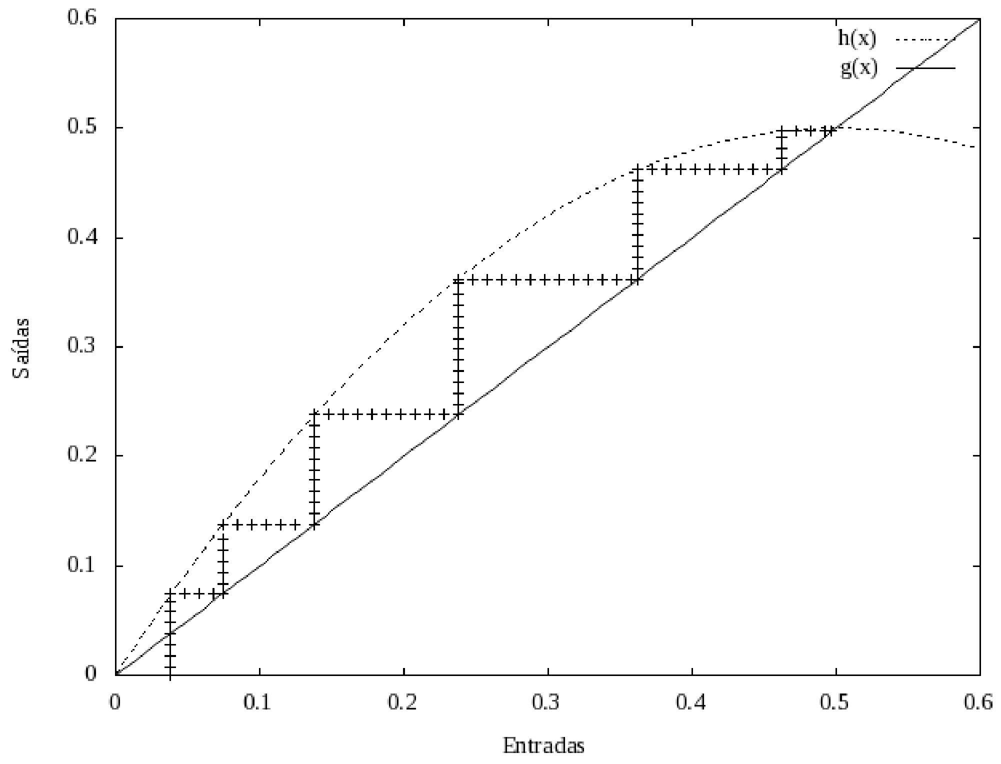


Figure 6.2: Órbita associada à função $h(x)$

Considere agora outro exemplo que permite melhor exemplificar a órbita de uma função em relação a condições iniciais. Seja um sistema definido pela função $q(x) = \frac{3x-x^3}{2}$ com valores iniciais $x = 1,6$ e $x = 1,8$. Traça-se o Cobweb plot para as duas condições iniciais, conforme a figura 6.3. Cabe ressaltar que, quando a curva estiver abaixo de $g(x)$, deve-se traçar linhas horizontais da órbita à esquerda e quando estiver acima, à direita, conforme realizado nos exemplos anteriores. Observa-se, nesse caso, que para a primeira condição inicial, tende-se ao ponto $x = 1$, enquanto para a segunda, a $x = -1$. Valores próximos ao ponto $x = 0$, contudo diferentes, fazem a órbita mover-se em direção à -1 ou 1. Esses pontos de atração, tais como $x = 1$ e $x = -1$, ou de repulsão, tais como regiões vizinhas de $x = 0$, são tratados por pesquisas adicionais em estabilidade de sistemas dinâmicos.

6.4 Estabilidade de Pontos Fixos

A seção anterior considerou três exemplos para estudar órbitas de sistemas dinâmicos. O primeiro, $f(x) = 2x$, apresenta comportamento linear. O segundo, $h(x) = 2x(1-x)$, e terceiro, $q(x) = \frac{3x-x^3}{2}$, contêm componentes não lineares. Esses exemplos utilizam uma reta $g(x)$ para auxiliar na observação gráfica das órbitas (simplifica a geração de próximas entradas e cálculo de saídas). Além de auxiliar na visualização de órbitas, $g(x)$ permite

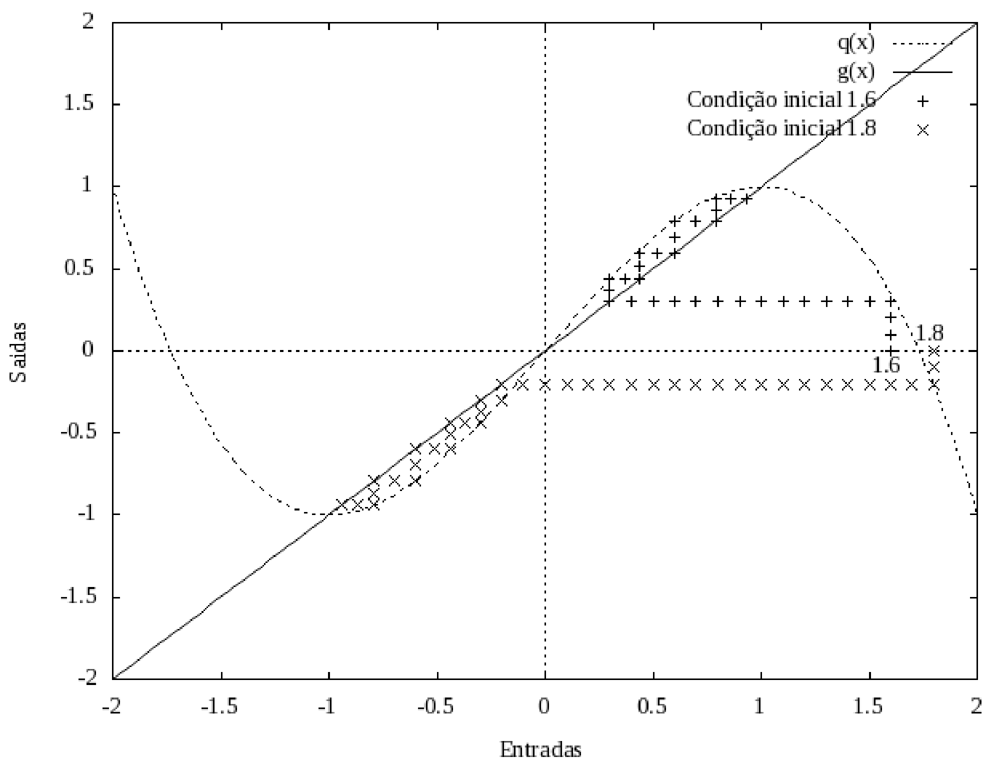


Figure 6.3: Órbita associada à função $q(x)$

definir alguns pontos relevantes para os sistemas em estudo.

Percebe-se que todos os sistemas exemplificados, $f(x)$, $h(x)$ e $q(x)$ cruzam a função diagonal $g(x)$, ou seja, há pares de pontos, denominados pontos fixos, em que a abscissa e a ordenada são iguais. No caso de $f(x)$ o único ponto fixo é $x = 0$; para $h(x)$ há dois pontos fixos $x = 0$ e $x = 0,5$; finalmente, em $q(x)$ observa-se três pontos fixos $x = -1$, $x = 0$ e $x = 1$. Em $f(x)$, para qualquer população de entrada positiva diferente de zero, diverge-se do ponto fixo $x = 0$ e tende-se ao infinito. Para $h(x)$, a tendência é divergir de 0 e convergir para uma população máxima de $0,5$ milhão. Para $q(x)$, o sistema diverge de 0 e tende a -1 ou 1, dependendo das condições iniciais (tal como observado na seção anterior, onde duas condições são apresentadas: $x = 1,6$ e $x = 1,8$).

Os pontos fixos de convergência são denominados estáveis, enquanto os de divergência, instáveis. Por exemplo, uma esfera no topo de uma montanha está sobre um ponto instável, qualquer pequena perturbação faz com que ela se mova até encontrar um vale, ou ponto estável, onde o movimento cessa. Assim, pontos estáveis atraem a órbita de um sistema, enquanto os instáveis a repelem.

Para melhor definir estabilidade e instabilidade de pontos fixos, considere que o comprimento Euclidiano de um vetor $v = \{x_1, \ldots, x_m\} \in \mathbb{R}^m$ é dado por $|v| = \sqrt{x_1{}^2 + \ldots + x_m{}^2}$. Seja $p = \{p_1, \ldots, p_m\} \in \mathbb{R}^m$ um ponto em um plano e ϵ um número positivo. A vizinhança ϵ, $N_\epsilon(p)$, é dada pelo conjunto $\{v \in \mathbb{R}^m : |v - p| < \epsilon\}$ de pontos dentro da distância Euclidiana ϵ de p. Seja f uma função em $\mathbb{R}^m$ e p, também em $\mathbb{R}^m$, um ponto fixo, onde

$f(p) = p$. Se existe um $\epsilon > 0$ tal que, para todo v em $N_\epsilon(p)$, $\lim_{k\to\infty} f^k(v) = p$, então p é um ponto fixo estável do tipo sorvedouro ou atrator. Se existe uma vizinhança ϵ, $N_\epsilon(p)$, tal que para cada v em $N_\epsilon(p)$, exceto o próprio p, mapeiam-se pontos mais distantes e, conseqüentemente, fora de $N_\epsilon(p)$, então p é um ponto fixo instável do tipo fonte ou repulsor. Além de pontos fixos estáveis do tipo sorvedouro e fixos instáveis do tipo fonte, há um terceiro tipo, que ocorre somente em espaços multidimensionais, denominado sela. Esses pontos têm ao menos uma direção de atração e uma de repulsão (6).

Conhecer os pontos fixos de um sistema permite inferir suas tendências. Por exemplo, sejam as temperaturas globais da terra definidas como uma série temporal. Esses valores definem a órbita do sistema dinâmico em questão. Essa órbita pode ser utilizada para encontrar sua regra de origem. Ao conhecer essa regra pode-se definir seus pontos fixos e estudar, por exemplo, regiões estáveis de temperatura. Essas regiões definiriam máximos ou mínimos de temperatura para o planeta. Além disso, pode-se encontrar regiões instáveis, onde temperaturas tendem, sempre, a divergir. Ao compreender a função geradora dessa órbita e seus pontos fixos, pode-se, por exemplo, avaliar a influência do aquecimento global em outros sistemas, tais como fauna e flora (25; 93; 65).

Os pontos fixos estáveis e instáveis são úteis, também, para estudar o comportamento de preços de ações em bolsa de valores. Considere os preços de uma ação, os quais definem a órbita do sistema. Deve-se obter a regra origem dessa órbita e encontrar seus pontos fixos. Esses pontos permitem avaliar, por exemplo, quando ações mudam suas tendências de preços, as quais permitem estimar os melhores momentos de compra e venda. Percebe-se, portanto, que, ao compreender a órbita e os pontos fixos de um sistema, pode-se predizer seu comportamento.

6.5 Expoente de Lyapunov

Formas de compreender o comportamento de sistemas dinâmicos motivaram diversos trabalhos (24; 102), dentre eles, estudos sobre variações em relação a condições iniciais, utilizando o expoente de Lyapunov (34). Esse expoente mede a taxa de variação de trajetórias vizinhas considerando uma separação inicial $\delta\mathbf{Z}_0$. Essa divergência dá-se por $|\delta\mathbf{Z}(t)| \approx e^{\lambda t}|\delta\mathbf{Z}_0|$, onde t é o instante de tempo e λ é o expoente de Lyapunov.

$$\lambda = \lim_{t\to\infty} \frac{1}{t} \ln \frac{|\delta\mathbf{Z}(t)|}{|\delta\mathbf{Z}_0|} \tag{6.1}$$

Considera-se o expoente de Lyapunov como maior valor de divergência entre trajetórias definido na equação 6.1. O valor desse expoente permite caracterizar um sistema dinâmico em (35; 96):

1. $\lambda < 0$ – a série temporal é atraída para um ponto fixo estável ou para uma órbita periódica (onde pontos da órbita se repetem no tempo). Esses expoentes são

característicos de sistemas dissipativos, ou não conservativos, tais como osciladores harmônicos. Quanto menor o valor de λ, mais rapidamente o sistema tende ao ponto de equilíbrio;

2. $\lambda = 0$ – a órbita do sistema tende a um ponto fixo neutro. Esses sistemas são conhecidos como conservativos. Por exemplo, considere dois osciladores harmônicos idênticos com diferentes amplitudes. Como a freqüência é independente da amplitude, a fase de ambos seria similar a dois círculos concêntricos, ou seja, as órbitas manteriam uma separação constante. Esse valor para o expoente indica que o sistema está no modo conhecido como estado fixo;

3. $\lambda > 0$ – o sistema apresenta órbita caótica e instável. Isso significa que a distância entre pontos da trajetória irá sempre divergir, em média, à uma taxa exponencial definida pelo expoente de Lyapunov (96; 33).

Os valores do expoente máximo de Lyapunov permitem compreender tendências de sistemas dinâmicos. Eles podem convergir para pontos fixos ($\lambda < 0$), apresentar comportamento conservativo ($\lambda = 0$) ou divergir ($\lambda > 0$). Dado que esse expoente expressa a tendência de um sistema, pode-se utilizá-lo para compreender o horizonte máximo (número máximo de pontos) de predição de sistemas dinâmicos. Por exemplo, considere um modelo qualquer (e.g. equações que representam o comportamento do sistema) para um sistema com divergência dada pelo expoente de Lyapunov. Quanto maior a divergência, menos eventos podem ser previstos sem alterar ou incrementar o modelo inicial. Caso haja nenhuma divergência, pode-se predizer inúmeros eventos futuros com base no modelo inicial.

Até mesmo estimativas qualitativas são impossíveis para intervalos além desse horizonte (36). O horizonte é definido por $-\frac{\ln \epsilon}{\lambda}$ onde ϵ é o erro medido no estado inicial, isto é, o ponto inicial de predições. Por exemplo, considere que qualquer técnica foi utilizada para modelar uma série temporal. Tal técnica foi treinada com um conjunto de dados e o erro de predição de um próximo valor é $\epsilon = 0,001$, o Lyapunov da mesma série é $0,692$, conseqüentemente seu horizonte de predição é igual a $-\frac{\ln 0{,}001}{0{,}692} = 9,98$. Isso significa que é possível predizer, no máximo, o comportamento dos próximos $9,98$ valores futuros.

A figura 6.4 apresenta o comportamento do horizonte de predição de acordo com o erro ϵ. Erros abaixo de $0,001$ foram considerados nessa circunstância. Observa-se que o horizonte é maior para pequenos valores de erro inicial.

Esta tese considera o método de Kantz (59) para calcular o expoente de Lyapunov de séries. Esse é um dos métodos implementados pelos autores do pacote *Nonlinear Time Series Analysis* (`TISEAN`) (50), que o recomendam em relação aos demais. O comando do pacote `TISEAN` que executa esse método é o `lyap_k`, o qual gera como saída logaritmos do fator de extensão (do inglês *stretching factor*), conforme proposto em (59). Para computar

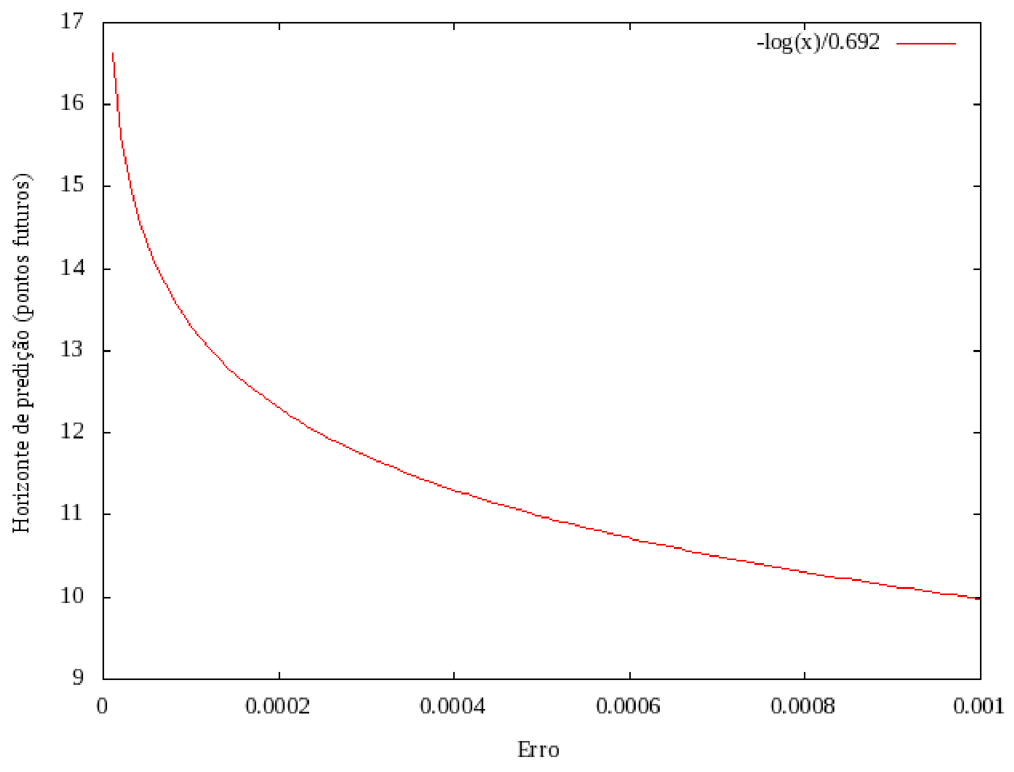


Figure 6.4: Horizonte de predição utilizando um expoente de Lyapunov igual a $0,692$

o expoente de Lyapunov, deve-se traçar a regressão linear desses logaritmos. O ângulo da curva de regressão resultante define o valor do expoente.

6.6 Expoente de Hurst

Além do expoente máximo de Lyapunov, pode-se estimar o expoente de Hurst de um sistema dinâmico, o qual mede a aleatoriedade de um conjunto de dados. Esse expoente pode ser estimado por diferentes técnicas, a mais adotada é a *Rescaled Range* (`R/S`) (60). Para exemplificá-la, considere a série temporal $X(n) = \{x_0, x_1, ..., x_{n-1}\}$. O primeiro passo consiste em computar a média $AVG_{X(k)}$ para todos os valores de $X(k)$, onde $k = n$.

Em seguida, encontra-se o valor mínimo (min) e o máximo (max), em relação à média, do conjunto de dados, conforme o algoritmo 4. Então, o desvio padrão $STDEV_{X(k)}$ é calculado para os termos $X(k)$ e o primeiro valor de `R/S` (denominado RS_0) para o subconjunto $k \in [0; n-1]$ é encontrado, definido por $RS_0 = \frac{|\max - \min|}{stdev_{X(k)}}$. O índice 0 em RS_0 faz referência à primeira iteração do cálculo.

A próxima iteração é iniciada pela divisão da série temporal original em dois subconjuntos. O primeiro com $k \in [0; \frac{n}{2} - 1]$ e o segundo com $k \in [\frac{n}{2}; n-1]$. Os mesmos passos são conduzidos para calcular `R/S` em cada subconjunto da série temporal e, conseqüentemente, obter-se-á dois valores (um para o subconjunto $[0, \frac{n}{2} - 1]$, o qual é definido por $RS_{1,[0;\frac{n}{2}-1]}$, e outro para o segundo subconjunto, $[\frac{n}{2}; n-1]$, representado por $RS_{1,[\frac{n}{2};n-1]}$).

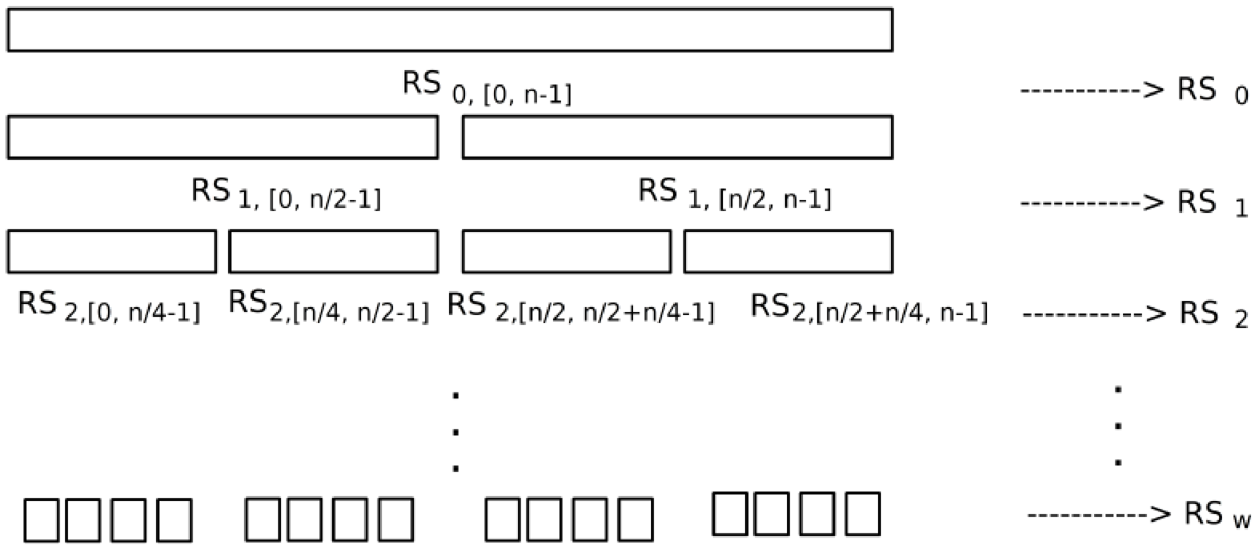


Figure 6.5: Cálculo de *Rescaled Range*

Algoritmo 4 *Rescaled Range*: Localizando os valores mínimo e máximo

```
float min = MAX_FLOAT;
float max = MIN_FLOAT;
float $v$ = 0;
for $i = first$; $i < k$; $i$++ do
  $v = v + (x_i$ - $Avg_{X(k)})$;
  if $v <$ min then
    min = $v$;
  end if
  if $v >$ max then
    max = $v$;
  end if
end for
```

Calcula-se, então, a média dos dois valores e obtém-se RS_1, o qual representa o valor de R/S para a segunda iteração. A série temporal é recursivamente particionada a fim de calcular $RS_2, RS_3, ..., RS_w$, onde w é a última iteração. De acordo com Kaplan (60), o particionamento pode ser feito, recursivamente, até que o menor subconjunto formado tenha um número mínimo de elementos no passo w (esse processo de particionamento é apresentado na figura 6.5). O mesmo autor conclui que há uma boa aproximação considerando cerca de 8 elementos na menor partição (valor adotado no exemplo da tabela 6.3).

Depois de calcular todos os valores $RS = \{RS_0, RS_1, ..., RS_w\}$, obtém-se o expoente de Hurst por meio de uma regressão linear dos pontos $(\log_2(|X(\frac{n}{2^y})|), \log_2(RS_y))$, onde $y = 0, 1, 2, ..., w$ é a iteração e $|X(\frac{n}{2^y})|$ é o número de elementos por partição da série na iteração y. A tabela 6.3 apresenta um exemplo. Fazendo a regressão linear, onde a coluna 4 representa o eixo x e a coluna 5 o y, obtém-se a seguinte equação $y = 0,727024x-$

$0,745555$. O ângulo da curva, definido nesse caso por $H = 0,727024$, é o expoente de Hurst[1].

Table 6.3: *Rescaled Range*: estimativa do expoente de Hurst

iteração	**tamanho da partição**	RS	$\log_2$**(tamanho da partição)**	$\log_2(RS)$
0	1024	$RS_0 = 96,4451$	$10,0$	$6,5916$
1	512	$RS_1 = 55,7367$	$9,0$	$5,8006$
2	256	$RS_2 = 30,2581$	$8,0$	$4,9193$
3	128	$RS_3 = 20,9820$	$7,0$	$4,3911$
4	64	$RS_4 = 12,6513$	$6,0$	$3,6612$
5	32	$RS_5 = 7,2883$	$5,0$	$2,8656$
6	16	$RS_6 = 4,4608$	$4,0$	$2,1573$
7	8	$RS_7 = 2,7399$	$3,0$	$1,4541$

O intervalo de valores do expoente de Hurst é $H \in [0;1]$. Um expoente próximo de 1 indica comportamento persistente, isso significa que há correlação entre um evento e a ocorrência de outro no futuro. Caso o valor seja próximo de zero, a série temporal apresenta comportamento anti-persistente, ou seja, há uma auto-correlação negativa entre pares de eventos. Nesse caso, um acréscimo em valor passado da série gera o decréscimo de outro elemento futuro e vice-versa. Valores próximos de $0,5$ indicam que a série temporal é uma caminhada aleatória (*random walk*), dessa forma, valores futuros não dependem do histórico. Observa-se, portanto, que o expoente de Hurst é uma ferramenta importante para avaliar dados históricos.

6.7 Dimensão embutida e de separação

Além do auxílio dos expoentes de Lyapunov e Hurst na interpretação de órbitas, é necessário encontrar formas de estimar as regras, ou funções, que definem os comportamentos de sistemas dinâmicos. Para exemplificar a obtenção dessas funções, considere, inicialmente, a equação logística 6.2 com condições iniciais $t \in [0;4000]$, $b = 3,8$ e $x_0 = 0,5$. A figura 6.6 apresenta as saídas, ou órbita percorrida no tempo, para essa regra, a qual tem expoente de Hurst $H = 0,40535$ e Lyapunov $\lambda = 0,447039$[2]. O expoente de Hurst evidencia pequeno grau de anti-correlação, contudo a série apresenta, em geral, comportamento aleatório. O expoente de Lyapunov aponta seu comportamento caótico e instável (a distância entre pontos da trajetória tende sempre a divergir, o que dificulta a modelagem). Esses expoentes permitem concluir que ao aplicar, diretamente, uma técnica de predição sobre tal série, tende-se a obter resultados ruins. Pode-se, contudo, aplicar uma forma

[1]O *dataset* utilizado foi obtido do livro *Chaos and Order in the Capital Markets*, segunda edição, Edgar Peters.

[2]Esse expoente de Lyapunov foi calculado utilizando 10 iterações para o programa `lyap_k` do pacote TISEAN (50). O número de iterações é informado por meio do argumento `-s`.

alternativa de reconstrução dessa órbita, a qual permite observar regularidades internas e simplificar a compreensão do sistema em estudo.

$$x_{t+1} = b \cdot x_t \cdot (1,0 - x_t) \tag{6.2}$$

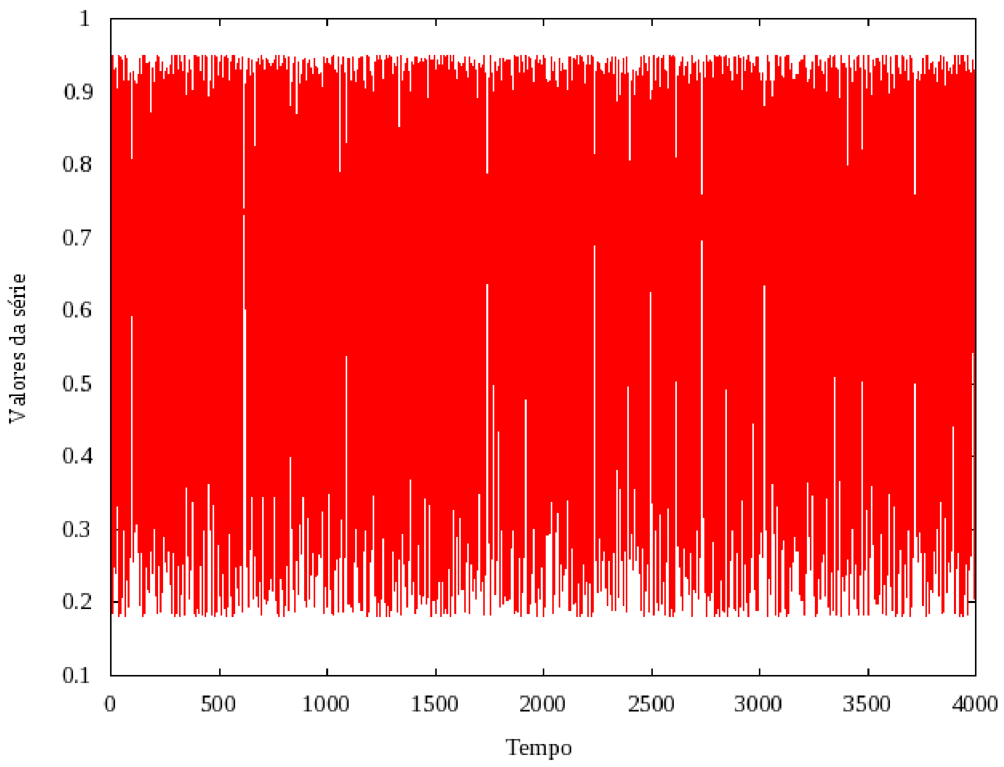


Figure 6.6: Saídas da função logística

Whitney (121) aplicou variedades diferenciáveis como forma de reconstruir funções utilizando transformações para espaços Euclidianos multidimensionais. Matematicamente, diz-se que $M \subset \mathbb{R}^k$ é uma variedade diferenciável de dimensão m se, para cada ponto $p \in M$, existem uma vizinhança $U \subset M$ de p e um homeomorfismo $x : U \to U_0$, U_0 um aberto de $\mathbb{R}^m$, tais que o homeomorfismo inverso $x^{-1} : U_0 \to U \subset \mathbb{R}^k$ é uma imersão de classe C^{inf}. Isto é, para cada $u \in U_0$, a derivada $dx^{-1}(u) : \mathbb{R}^m \to \mathbb{R}^k$ é biunívoca. Diz-se, nesse caso, que (x, U) é uma carta local em torno de p e U é uma vizinhança coordenada de p (88).

A figura 6.7 apresenta um exemplo de parametrização de um plano $\mathbb{R}^{m-1} \times \mathbb{R}_+$ para outro $\mathbb{R}^k$. Dado um ponto q' pode-se, por meio de $\phi_q : H_0 \to H \cap M$, encontrar um ponto q em M correspondente. O mesmo ocorre com p', contudo, nesse caso, obtém-se uma região na borda de M. Esse exemplo ilustra o mapeamento de um ponto e sua vizinhança em um plano com maior número de dimensões.

Whitney (121) observou que esse mapeamento permitiria a compreensão de comportamentos não observáveis ou pouco representativos quando descritos sob número reduzido de dimensões. A partir disso, ele propôs seu teorema de imersão, segundo o qual qualquer

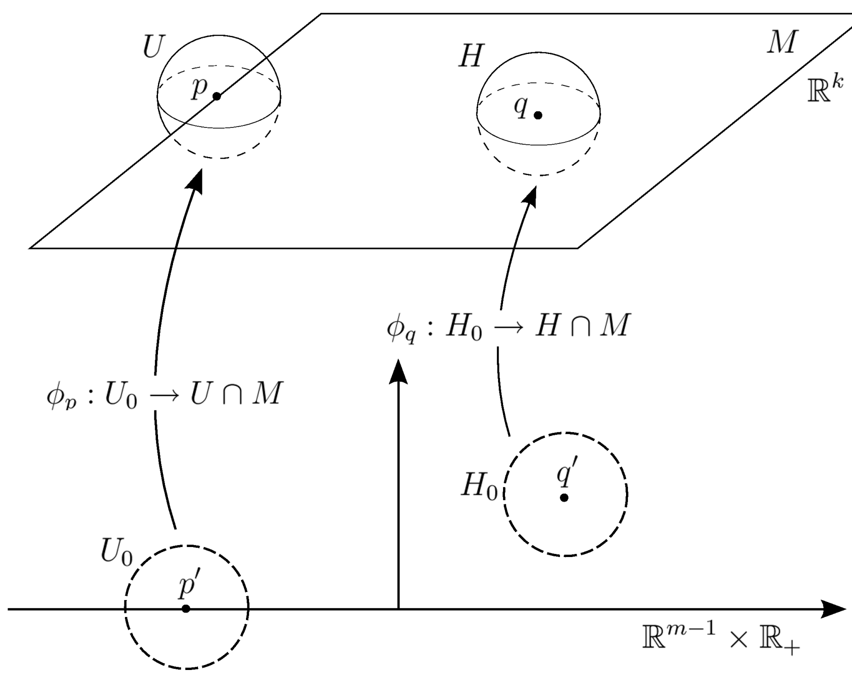

U
p
H
q
M
$\mathbb{R}^k$
$\phi_q : H_0 \rightarrow H \cap M$
$\phi_p : U_0 \rightarrow U \cap M$
H_0
q'
U_0
p'
$\mathbb{R}^{m-1} \times \mathbb{R}_+$

Figure 6.7: Exemplo de variedade

variedade em n dimensões pode ser mapeada em espaço Euclidiano de $2n + 1$ dimensões.

Baseado nos estudos de Whitney (121), Takens (111) prova que, ao invés de mapear os estados de um sistema dinâmico em espaço Euclidiano de $2n + 1$ dimensões, pode-se reconstruí-lo considerando deslocamentos no tempo. Segundo o teorema de imersão de Takens (111), uma série temporal $x_0, x_1, ..., x_{n-1}$ pode ser reconstruída em espaço multidimensional $x_n(m, \tau) = (x_n, x_{n+\tau}, ..., x_{n+(m-1)\tau})$, ou de coordenadas de atraso, onde m é a dimensão embutida e τ representa um *time delay* ou dimensão de separação. Essa técnica de mapeamento ou reconstrução permite transformar as saídas produzidas por sistemas dinâmicos, representadas por séries temporais unidimensionais, em um conjunto de pontos em espaço Euclidiano de m dimensões. Essa reconstrução foi, posteriormente, empregada para auxiliar na obtenção de regras de sistemas dinâmicos, simplificando, conseqüentemente, o estudo de comportamentos e sua aplicação para diferentes fins, tais como estudo de órbitas, tendências e predição (6).

Para melhor compreender as dimensões embutida e de separação, considere as saídas da função logística, anteriormente abordada, reconstruídas em um espaço multidimensional onde $m = 2$ e $\tau = 1$, a qual resulta em pares de pontos (x_t, x_{t+1}) (figura 6.8). Após a reconstrução, o comportamento da função logística, que era aparentemente uma caminhada aleatória (figura 6.6), pode ser estudado, compreendido e modelado de forma mais simples. Ao realizar uma regressão dos pontos resultantes, pode-se obter a regra do sistema dinâmico e determinar seus comportamentos futuros. Tendo essa regra e um x_t, pode-se, por exemplo, definir o próximo valor da série, x_{t+1}, o qual pode ser retroalimentado para gerar x_{t+2}, e assim sucessivamente.

A dimensão embutida define, basicamente, o número de eixos, do espaço de coorde-

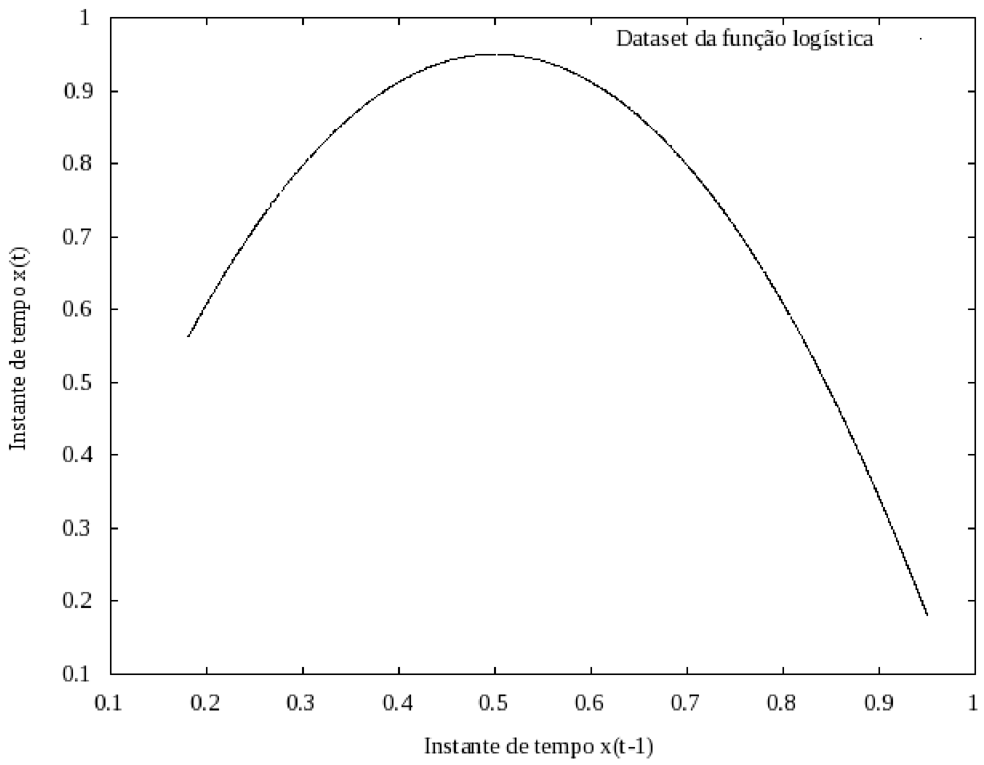


Figure 6.8: Função logística reconstruída em dimensão embutida 2 e de separação 1

nadas de atraso, necessários para plotar o comportamento reconstruído da série. Nesse caso a série necessitou de duas dimensões, outras podem requerer espaços com mais eixos. Esse comportamento é, por exemplo, observado no atrator de Lorenz cujas saídas são apresentadas na figura 6.9.

Ao considerar a reconstrução da série utilizando dimensão embutida 2, obtém-se espaço de coordenadas de atraso similar ao da figura 6.10. Contudo, ao adicionar uma nova dimensão e reconstruí-la, portanto, com $m = 3$, desdobra-se todo o comportamento da série, simplificando sua compreensão (figura 6.11). Pára-se de adicionar dimensões caso não haja desdobramentos de novos comportamentos; nesse caso, cessa-se com $m = 3$, que representa a melhor dimensão para o atrator de Lorenz.

Além da dimensão embutida há ainda a de separação, que auxilia na extração de comportamentos periódicos de séries. A dimensão de separação informa o deslocamento de valores históricos que devem ser avaliados a fim de predizer comportamento futuro (ela permite encontrar a sazonabilidade da série). Por exemplo, a fim de predizer temperaturas de uma região do mundo na data de 12 de dezembro de 2007, pode-se observar medidas do ano anterior (em 12 de dezembro de 2006). Espera-se, portanto, que o próximo ano tenha temperaturas próximas aos dos anteriores. Nesse exemplo, a dimensão de separação seria igual a 365 dias.

Por outro lado, caso seja considerado o período de um mês para o problema abordado, observar-se-ia que o comportamento das temperaturas não se repetiria da maneira esper-

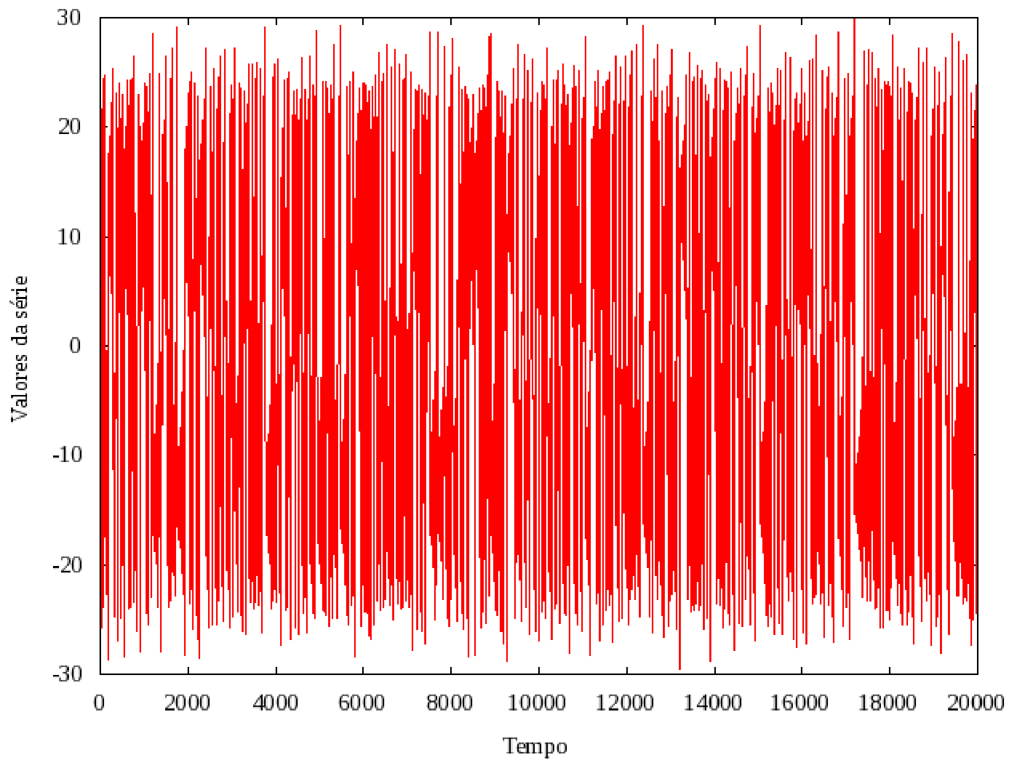


Figure 6.9: Saídas do atrator de Lorenz

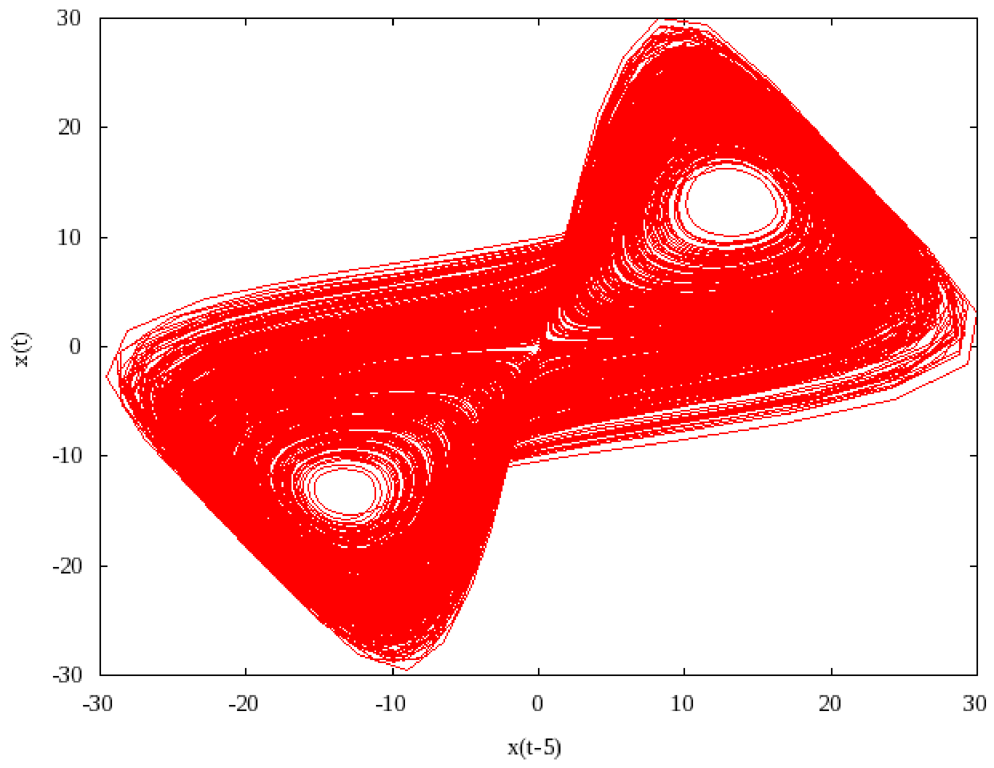


Figure 6.10: Atrator de Lorenz reconstruído em espaço de coordenadas de atraso com dimensão embutida 2 e de separação 5

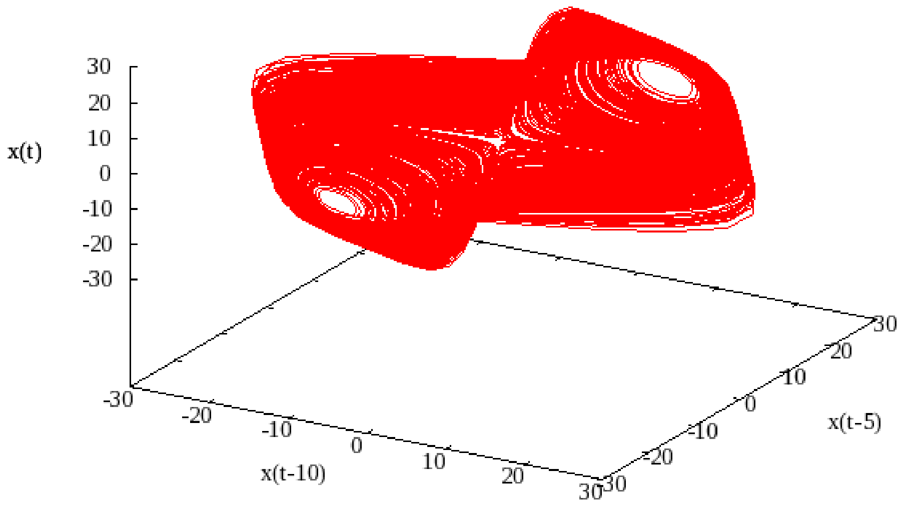


Figure 6.11: Atrator de Lorenz reconstruído em espaço de coordenadas de atraso com dimensão embutida 3 e de separação 5

ada[3]. Nesse caso seriam consideradas, por exemplo, as temperaturas de 12 de novembro de 2007, 12 de outubro de 2007, 12 de setembro de 2007 e 12 de agosto de 2007 que, muito provavelmente, não auxiliariam na predição das temperaturas de 12 de dezembro de 2007.

Exemplos anteriores evidenciam como as dimensões embutida e de separação auxiliam no estudo de algumas séries específicas. Contudo, é necessário determinar essas dimensões para quaisquer séries oriundas de dados experimentais. De acordo com Abarbanel *et al.* (1), uma função de auto-correlação (equação 6.3, onde $E[.]$ é o valor esperado, μ é a média, k é o deslocamento no tempo e σ^2, a variância) auxilia na determinação da dimensão de separação de séries. A auto-correlação mensura a repetição de comportamento de um trecho da série em relação a seu histórico. Contudo, essa técnica é formulada para séries lineares, e, conseqüentemente, não é adequada para outros tipos de séries.

$$ACF(k) = \frac{E[(X_i - \mu)(X_{i+k} - \mu)]}{\sigma^2} \tag{6.3}$$

Fraser & Swinney (39) estudaram e confirmaram que a técnica de auto-informação mútua (*Auto Mutual Information* – `AMI`) apresenta melhores resultados na estimativa de dimensões de separação. Essa técnica não depende de séries lineares. Para obter a separação de uma série, aplica-se essa técnica considerando diferentes deslocamentos no

[3]No último exemplo, envolvendo o atrator de Lorenz, considerou-se $t = 5$, o qual é indicado como dimensão de separação segundo Lorenz (74); Kennel *et al.* (63).

tempo. Em seguida, traça-se uma curva em função dos deslocamentos (iniciando em 1 e incrementando) e adota-se seu primeiro mínimo como dimensão de separação.

A informação mútua média é definida pela equação 6.4, onde X e Y seguem, respectivamente, as funções de distribuição de probabilidades P_X e P_Y, e X e Y ocorrem em pares com distribuição conjunta P_{XY} (61). Aplicando essa técnica sobre o conjunto de dados de Lorenz, previamente estudado, obtém-se a figura 6.12, a partir da qual se encontra o primeiro mínimo, igual a 5, confirmando os resultados apresentados por Lorenz (74); Kennel *et al.* (63).

$$I(X;Y) = \int P_{XY}(x,y) \log_2 \frac{P_{XY}(x,y)}{P_X(x)P_Y(y)} dxdy \tag{6.4}$$

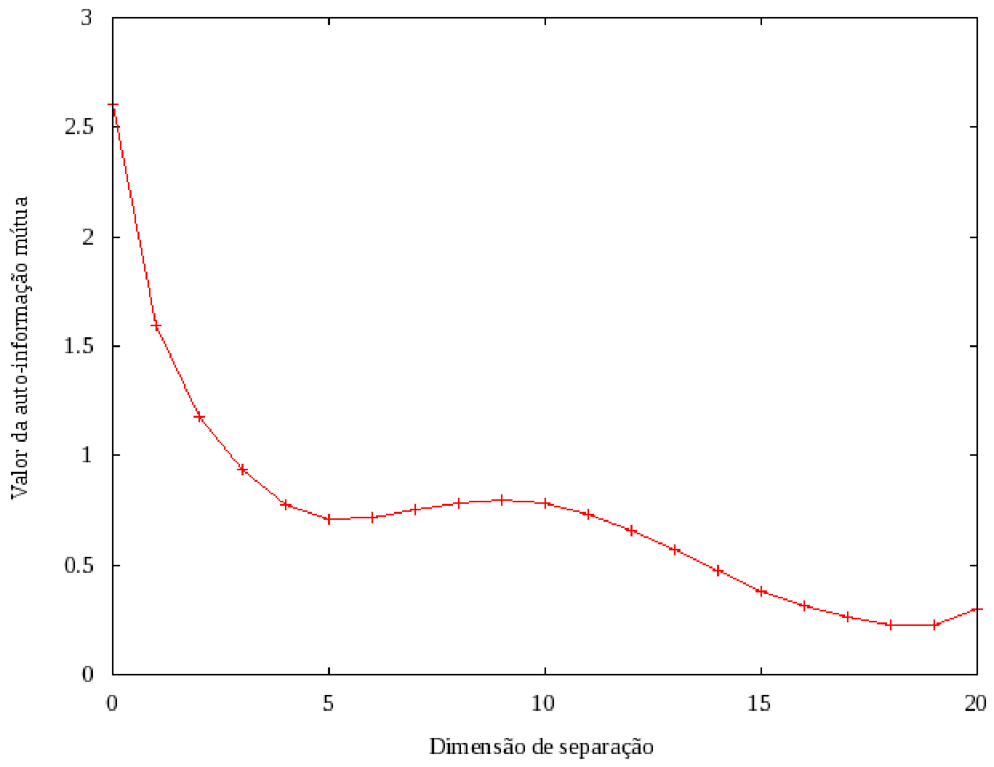


Figure 6.12: Lorenz – auto-informação mútua

Após definir a dimensão de separação de uma série, deve-se encontrar a dimensão embutida. Takens (111) e Mañé (79) estudaram e confirmaram que o limite superior da dimensão embutida D_e (valor inteiro) pode ser estimado utilizando a dimensão fractal D_f, de acordo com a equação $D_e > 2,0 \cdot D_f$. Contudo, a dimensão resultante dessa equação é, em geral, maior que o necessário. Por exemplo, a dimensão fractal do atrator de Lorenz é $2,06$ (82), conseqüentemente, o limite superior da dimensão embutida seria $D_e > 2,0 \cdot 2,06$, que resulta em 5. Contudo, de acordo com (63), o atrator pode ser representado em $D_e = 3$. Do ponto de vista matemático (62; 63), pode-se modelar esse sistema utilizando 3 ou 5 dimensões, pois, uma vez que todos os possíveis estados foram encontrados, pode-se conduzir a análise comportamental. Contudo, ao trabalhar, desnecessariamente, com mais dimensões, adiciona-se complexidade e tempo de processamento à modelagem e à análise de resultados (63).

Uma forma alternativa para obter a dimensão embutida mínima é por meio do cálculo de invariantes do sistema (tais como o expoente de Lyapunov (6)) para diferentes valores de dimensão, observando a saturação dos resultados. A complexidade dessa abordagem motivou Kennel *et al.* (63) a propor o método *False Nearest Neighbors* (`FNN`), que calcula os vizinhos mais próximos de cada ponto, no espaço de coordenadas de atraso (iniciando com dimensão embutida igual a 1). Em seguida, uma nova dimensão é adicionada e a distância entre vizinhos mais próximos é novamente calculada. Caso haja acréscimo nessa distância, os pontos são considerados falsos vizinhos, o que evidencia a necessidade de mais dimensões para reconstruir o comportamento da série.

Kennel *et al.* (63) consideram uma dimensão embutida d onde o r-ésimo vizinho mais próximo de $y(n)$ é dado por $y^{(r)}(n)$. A distância Euclidiana entre o ponto $y(n)$ e seu r-ésimo vizinho mais próximo é dada pela equação 6.5. Ao adicionar uma nova dimensão, reconstrói-se a série em $d+1$ e adiciona-se a coordenada $(d+1)$ em cada vetor $y(n)$, a qual é incluída na equação de distância Euclidiana (termo $x(n+dT)$ da equação 6.6). Dessa forma, o critério mede a variação de distância ao adicionar uma nova dimensão, conforme descrito pela equação 6.7.

$$R_d^2(n,r) = \sum_{k=0}^{d-1} \left(x(n+kT) - x^{(r)}(n+kT)\right)^2 \tag{6.5}$$

$$R_{d+1}^2(n,r) = R_d^2(n,r) + \left(x(n+dT) - x^{(r)}(n+dT)\right)^2 \tag{6.6}$$

$$V_{n,r} = \sqrt{\frac{R_{d+1}^2(n,r) - R_d^2(n,r)}{R_d^2(n,r)}} = \frac{|x(n+dT) - x^{(n)}(n+dT)|}{R_d^2(n,r)} \tag{6.7}$$

Segundo os autores, se $V_{n,r} > R_{tol}$ então os pontos são considerados falsos vizinhos, onde R_{tol} é um limiar. Eles ainda concluem, empiricamente, que $R_{tol} \geq 10,0$ é um bom limite para a geração de resultados.

Aplicando o método `FNN` sobre o conjunto de dados do atrator de Lorenz (utilizando a dimensão de separação 5 previamente obtida), obtém-se os resultados apresentados na figura 6.13. Essa figura traça a fração dos falsos vizinhos versus a dimensão embutida considerada. Quando a fração é igual a zero, encontra-se a melhor dimensão embutida. Nesse caso, a dimensão embutida encontrada é 3, o que confirma os resultados obtidos por Kennel *et al.* (63).

Após definir as duas dimensões, aplica-se a teoria de imersão de Takens (111), conforme apresentado anteriormente, onde a série temporal $x_0, x_1, ..., x_{n-1}$ é reconstruída no espaço multidimensional, ou de coordenadas de atraso, $x_n(m,\tau) = (x_n, x_{n+\tau}, ..., x_{n+(m-1)\tau})$ (O termo m representa a dimensão embutida e τ, a separação). A reconstrução desdobra,

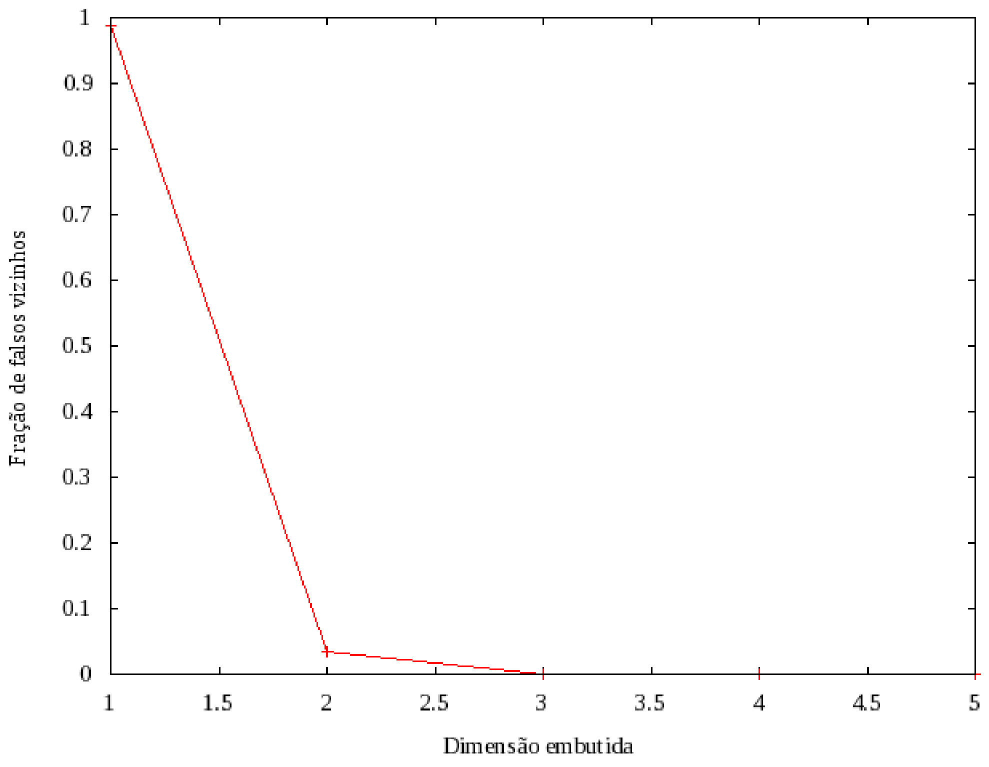


Figure 6.13: Atrator de Lorenz – estudos para encontrar a dimensão embutida

completamente, o comportamento da regra desse sistema dinâmico. Para exemplificar esse desdobramento, considere o conjunto de dados de Lorenz apresentado na tabela 6.4. Depois de reconstruir esse conjunto com dimensão embutida 3 e separação 5, obtém-se a tabela 6.5 (a curva resultante dessa reconstrução é apresentada na figura 6.11)[4].

Essa reconstrução permite desdobrar o comportamento da série e obter sua regra, ou seja, a função que define sua órbita no tempo. Obtendo tal função, pode-se estudar seus pontos fixos estáveis e instáveis, compreender suas tendências e, também, predizer seu comportamento futuro.

6.8 Aproximação de funções

Após reconstruir uma órbita em espaço de coordenadas de atraso, obtém-se um conjunto de pontos que define a regra do sistema dinâmico em estudo. Por exemplo, considere a órbita da função logística da figura 6.14, após calcular a dimensão de separação pelo método de auto-informação mútua (obtendo, neste caso, $\tau = 1$) e a dimensão embutida pelo método dos falsos vizinhos (obtendo $m = 2$), reconstrói-se sua órbita no espaço de coordenadas de atraso tal como apresentado na figura 6.15.

A órbita reconstruída permite encontrar a regra origem do sistema dinâmico, o que

[4]Essa tabela apresenta mais valores que a tabela 6.4. Esse é apenas um exemplo de desdobramento dos dados originais a fim de obter a função geradora do sistema dinâmico em questão.

Table 6.4: Conjunto de dados original do atrator de Lorenz

Dimensão 1
$-9,6559617$
$-6,9902085$
$-4,9834927$
$-3,5773619$
$-2,6589215$
$-2,1120568$
$-1,8411753$
$-1,7784935$
$-1,8834828$
$-2,1397586$
$-2,5521791$
$-3,1453527$
$-3,9638112$
$-5,0733551$
$-6,5619076$
$-8,5356685$
$-11,100864$
$-14,311700$
$-18,056232$
$-21,873802$
$-24,819411$

Table 6.5: Conjunto de dados do atrator de Lorenz reconstruído segundo a dimensão embutida e de separação encontradas ($m = 3$ e $\tau = 5$)

Dimensão 1	**Dimensão 2**	**Dimensão 3**
$-9,655962$	$-2,112057$	$-2,552179$
$-6,990209$	$-1,841175$	$-3,145353$
$-4,983493$	$-1,778494$	$-3,963811$
$-3,577362$	$-1,883483$	$-5,073355$
$-2,658921$	$-2,139759$	$-6,561908$
$-2,112057$	$-2,552179$	$-8,535668$
$-1,841175$	$-3,145353$	$-11,100864$
$-1,778494$	$-3,963811$	$-14,311700$
$-1,883483$	$-5,073355$	$-18,056232$
$-2,139759$	$-6,561908$	$-21,873802$
$-2,552179$	$-8,535668$	$-24,819410$

auxilia no estudo de suas tendências. Conhecendo as tendências pode-se, por exemplo, classificar, predizer e determinar esse comportamento e influências em demais sistemas interdependentes. Para obter a regra origem, deve-se realizar a regressão dos pontos do espaço de coordenadas de atraso.

A figura 6.16 exemplifica os pontos obtidos em espaço de coordenadas de atraso e sua regra origem, a qual é definida pela equação $f(x) = -3,8x^2 + 3,8x - 1,49236 \cdot 10^{-07}$

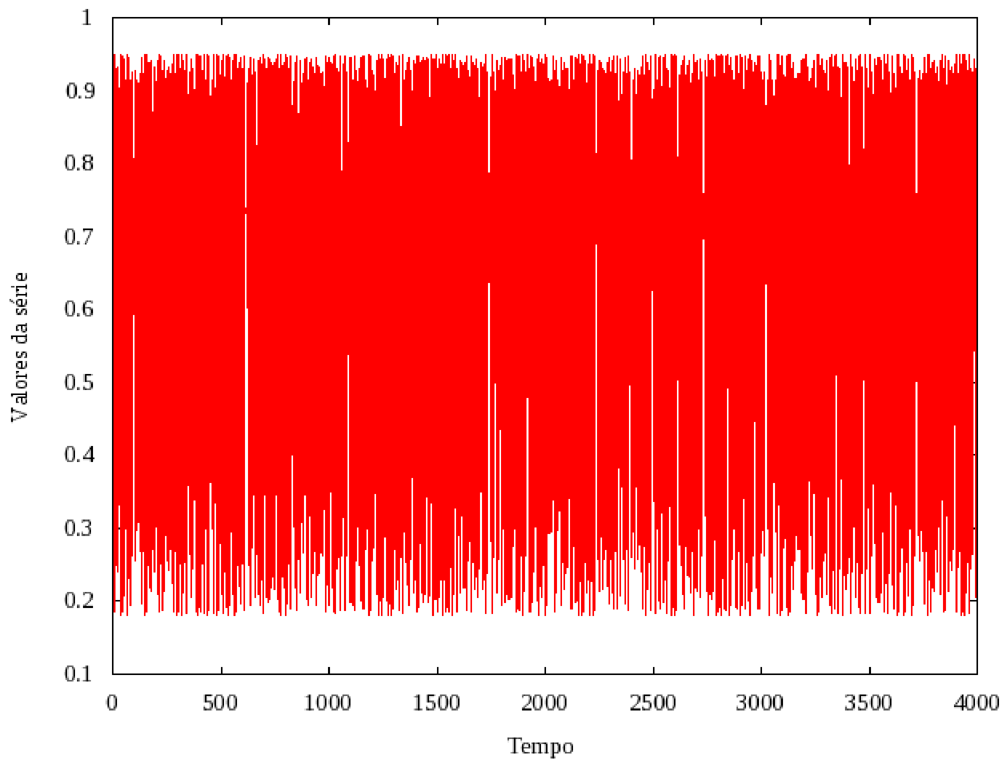


Figure 6.14: Órbita da função logística

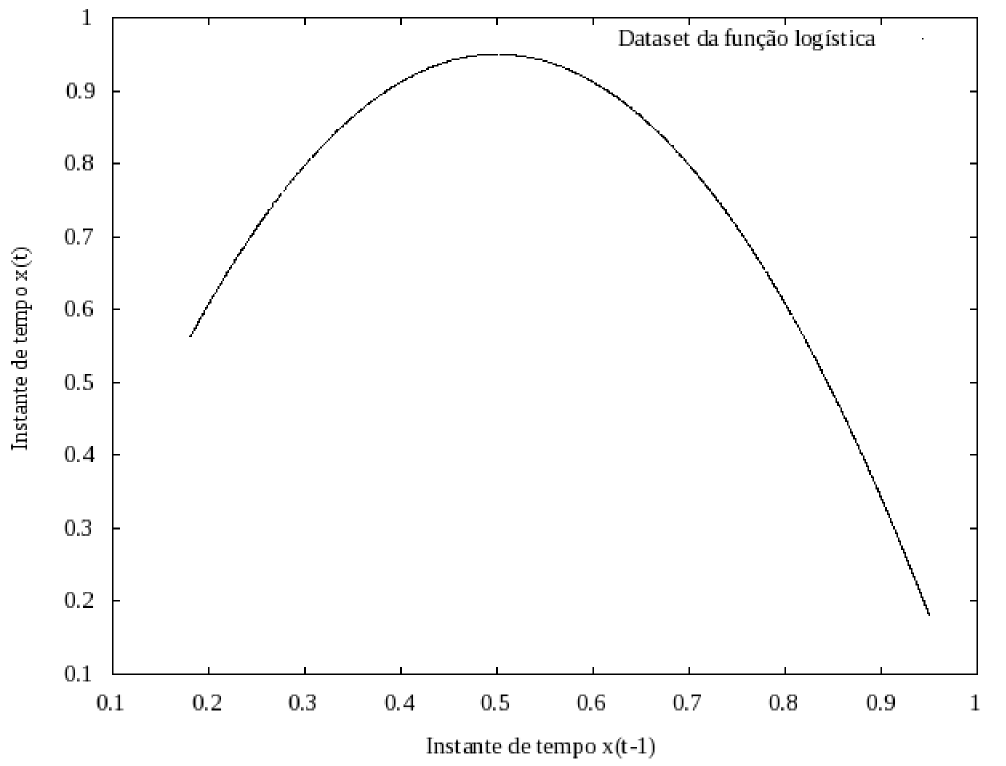


Figure 6.15: Função logística reconstruída em dimensão embutida 2 e de separação 1

(regressão realizada pelo método dos mínimos quadrados (16)). Pode-se aplicar uma condição inicial, ou seja, x_t, à essa regra e estimar saídas para k instantes de tempo futuros. Pode-se, também, empregar mapas lineares e não lineares (6), como matrizes Jacobianas, a fim de estudar pontos fixos, períodos de órbitas e tendências globais do sistema.

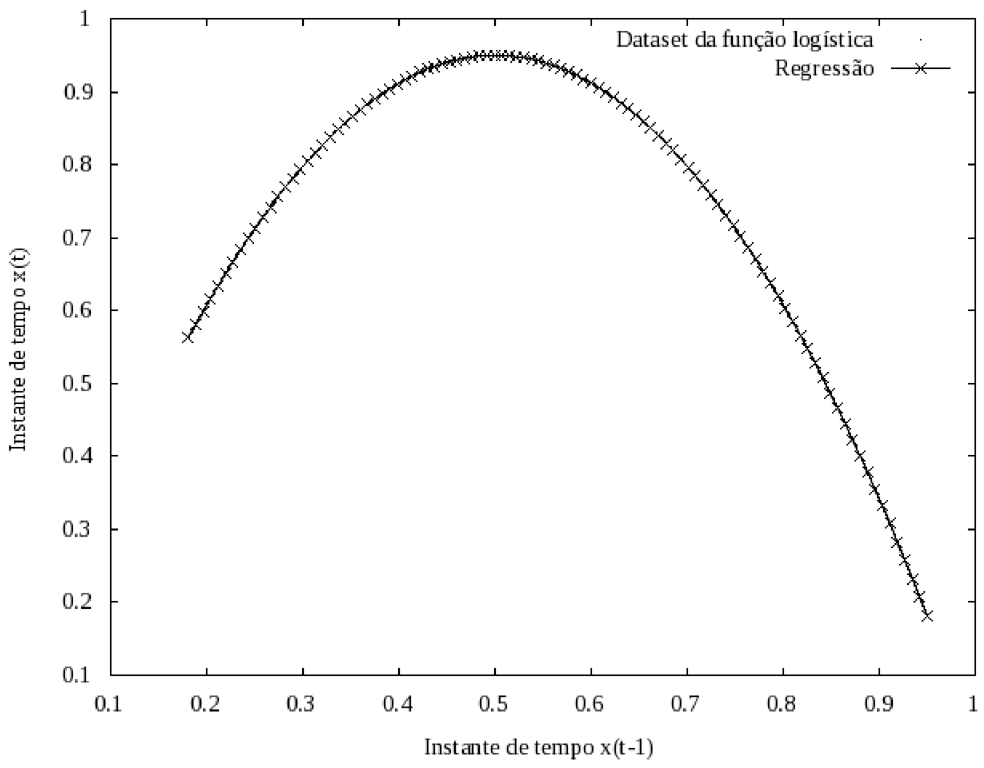


Figure 6.16: Regressão dos pontos no espaço de coordenadas de atraso

No exemplo anterior foi adotado o método dos mínimos quadrados para a regressão dos pontos em espaço multidimensional. No entanto, outras abordagens poderiam adotadas, tais como métodos auto-regressivos (15), funções radiais (19), filtros de Kalman (116; 14; 69; 30; 94), cadeias de Markov (84), redes neurais artificiais `TDNN`, `ATNN`, *Long Short-Term Memory* (`LSTM`), *Radial Basis Function* (`RBF`) e `RRBF` (31). Essas abordagens aproximam, de forma distinta, funções sobre o conjunto de pontos em plano multidimensional.

Chapter
7

Biological Immunity Concepts

7.1 Introduction

Before we present some of the artificial immune system's concepts, we need a clear understanding on how the biological immune systems works. This chapter will first present the basic concepts of classic immunology, and then review some of the latest works this area.

The classic immunology discussion will be based on Abbas and Lichtman's textbook(?). The next section will present the different cell types (especially lymphocytes and antigen-presenting cells), the distinction between innate and adaptive immunity, the interactions between different cells and the life-cycle of immune cells. We do not intend to present a complete overview of all immunology concepts because we intend to focus on the concepts that are more easily applied when designing artificial immune systems for computer intrusion detection.

Some of the most important recent ideas used by this work are Matzinger's danger model(?) and Jerne's idiotypic network(?). These works are controversial and there is no consensus between immunologists regarding their completeness or correctness. Nevertheless, they may fill some gaps left open by classical immunology when we want to create more efficient and autonomous intrusion detection systems based on the immune system.

7.2 Classic Immunology

According to classic immunology, the function of the immune system on the body of mamals is to prevent infections and respond to established infections. The human immune system is composed of two mechanisms: innate immunity and adaptive immunity.

Innate immunity is the body's first protection mechanism to deal with a microbe. It is composed of epithelial barriers, phagocytes, proteins from the complement system and natural-killer cells. Responses from the innate immunity are fast and are not directed to specific microbes, but usually to a whole class of microorganisms and substances.

Adaptive immunity is a mechanism developed by the body to deal with specific pathogenic microbes. It is only stimulated after the body has a strong and durable contact with the microorganisms. It is composed of lymphocytes (B-cells and T-cells) and by antibody molecules. The first response from the adaptive immunity is usually slow on the first encounter with an antigen (it takes around a week until it's effector mechanisms start to work), but subsequent infections are handled much faster due to it's memory mechanisms.

Although the adaptive immunity and innate immunity are composed by different kinds of cells and have distinct functions, elements from these mechanisms communicate with each other, and the development of the cells and their responses are always mediated by these interactions.

7.2.1 Innate Immunity

All multicellular organisms have innate immunity mechanisms. These are germline-encoded, intrinsic processes that protect the body against common infections.

The first innate defense againts microbes is the epitelial barrier. This barrier is not limited to the phisical isolation of enviroments since there are natural antibiotics and specialized cells on epitelial tissues preventing microbes from entering the body. Innate immunity is the first line of defense against infections, providing a powerful early defense mechanism by triggering reactions before the adaptive immunity becomes active.

When a microorganism or a toxic substance manages to breach through the epithelial barriers they are first attacked by phagocytes or proteins from the complement system. When a phagocyte ingest a microbe, it will release cytokines, and these molecules will attract other cells from the innate and adaptive system to the infection site. Innate mechanisms trigger responses against microbes, but not against non-infectious foreign substances, because their molecular pattern recognitions mechanism is based on patterns commonly found on pathogens. These patterns are called PAMP's, for "Pathogen Associated Molecular Patterns".

The receptors of the innate immunity (TLR's, for tool-like receptors) are germline-encoded and are not produced by somatic hipermutation, as happens on the adaptive immunity. Thus the innate immune mechanisms evolve as the populations evolve, like other genetic characteristics. Since the innate immunity does not change during the life of an individual, the innate immune system responds the same way to repeated exposures to the same antigen.

Some microorganisms have developed the ability to resist the responses of the innate immunity, for instance by being able to escape from phagocytic vesicles or by producing substances that inhibit the defense system's mechanisms. Defense against this kind of microbes depends on the actions from the adaptive immune system.

7.2.2 Adaptive Immunity

The adaptive immune system is composed of lymphocytes (B-cells and T-cells) and the antibody proteins that B-cells produce. Instead of recognizing classes of pathogen-associated molecules, these elements recognize specific microbes' structures called antigens. Adaptive immune responses have a slow activation process and demand strong signaling to be triggered.

Although each lymphocyte is able to match only a single antigen, the adaptive immune system has the potential for distinguishing at least a billion different antigen or portions of antigens, so we may say that the collection of antigen recognizers (the lymphocyte repertoire) is extremely diverse. All PAMP receptors of the innate immunity recognize less than a thousand microbial patterns.

The adaptive immune responses are performed on sequencial phases. The first matching to an unknown antigen is performed by a lymphocyte that was never exposed to it's antigen (called naive lymphocite). This event triggers the cells' activation process, where the effector mechanisms against the microbes takes place. The lymphocyte will then start a clonal expansion and cellular differentiation processes. Clonal expansion is a process where activated lymphocytes proliferate in an asexual, mitotic process. Most clones express exactly the same genes from the progenitor cell, but on some of them a process of somatic mutation occurs, generating some lymphocytes with slightly different receptors.

Lymphocytes will also differentiate either into effector cells or memory cells. Effector cells are short-lived cells that take part on the immune system task of removing the pathogen from the organism, while memory cells are long-lived cells that perform no action, but instead wait for a subsequent intrusion of the same pathogen. When a memory cell come across it's antigen, it begins a quick clonal expansion and differentiation process, so that the effector mechanism may take place much faster than the first response.

Adaptive and innate immunity processes are very dependent on each other, as we will see on the next sections. The concept of second-signaling as confirmation of a pattern recognitions depends on elements from both systems. Innate immune responses release molecules that provide second signals to T-cells and B-cells, activating these cells and triggering the adaptive response. This mechanism protects the body from activation of cells that would react against harmless substances, since the innate system will only trigger it's reactions against cells and molecules with pathogen-associated patterns.

Newborn infants have no prior exposure to microbes, so they are born only with innate immune mechanisms ready (since these mechanisms are germline-encoded). It's adaptive immunity repertoire will be developed throughout it's life, and will not be passed on to it's offspring.

Some of the main concepts of immunity are listed below, and will be discussed on the next sections.

- Second signaling
- Cell maturation
- Cell activation
- Negative selection
- Inflammation

7.2.3 Concepts on the Immune System

Antigen-Presenting Cells (APC's)

Professional Antigen-Presenting Cells are cells whose main function is to capture, process and present pathogens to the adaptive immune system. Examples of antigen presenting cells are dendritic cells, macrophages and some B-lymphocytes. APC's express receptors on their membrane, and these receptors are responsible for matching the antigens. When a match occurs, the APC captures the pathogen through phagocitosys, perform processing of the captured element and then move to a peripheral lymph node where the immune response is initiated.

APC's express on their membrane specialized peptide molecules called Major Histocompatibility Complex (MHC). After processing, the antigen fragments are bound to the MHC so that they may be presented to T-lymphocytes.

The epithelia contains a population of professional APC's called dendritic cells. Dendritic cells are cells from the immune system that at certain development stages grow branched projections called dendrites (hence the name. These cells have no relation to neurons). They capture the antigens that breach through the epithelial barriers and move to the peripheral lymphoid organs to perform antigen presentation to the T-cells. The result of this process is that when a microbe manages to breach through the skin at any place on the body, it will be captured by a cell from the innate immune system and be sent to evaluation from the adaptive immune system.

Macrophages are also an important type of APC. They are present on all tissues, their function is to phagocyte microbes and present the antigens to T-cells. When a T-cell provides the second-signaling to the macrophage it will be activated to kill the microbes. B-cells also are able to phagocyte microbes and perform the presentation of the antibodies.

B-lymphocytes and T-lymphocytes

B-lymphocytes are the only cells that produce antibodies. They have membrane-bound antibodies that are able to match antigens dissolved on the environment, can phagocyte pathogens and may also secrete antibodies as an effector mechanism against a pathogen. T-lymphocytes, on the other hand, can only recognize antigens presented by APC's. Also, they do not have direct action against pathogens: they do not produce antibodies and do not perform phagocitosis. Their function is to activate APC's so that they destroy the ingested microbes and stimulate B-cells so that they secrete antibodies.

Some of the most important types of T-lymphocytes are T-helper cells and Cytotoxic T-cells. T-helper cells coordinate responses from B-lymphocytes and macrophages by providing a second-signaling as confirmation of a captured pattern. On the other hand, cytotoxic T-cells induce death on cells that were attacked by microbes, so that these released pathogens can be processed by leukocytes.

All lymphocytes are born from stem cells in the bone marrow. They are born as immature cells: they have antigen receptors, but are unable to express any response against pathogens. They pass through a maturation process to avoid auto-immunity (immunological responses against self-antigens, i.e. cells and molecules that compose the body) and also to eliminate cells that do not express valid receptors (i.e. cells that could never match any antigen). B-cells undergo maturation on the bone marrow, and T-cells move to the thymus for this process, hence their name. During this maturation, a process named negative selection takes place. The bone marrow and the thymus are protected environments on the body, where we usually find only self-antigens, so when an immature cell can match an antigen it also means that it is capable of matching the host's antigens. These cells undergo apoptosis to avoid auto-immunity reactions.

Apoptosis is a process of programmed cell death. It involves a series of biochemical events that lead to a variety of morphological changes like nuclear condensation and chromosomal DNA fragmentation. The released cellular debris after apoptosis do not damage the organism. On the other hand, necrosis involve unregulated cell death, resulting in cell lysis and inflammation due to the released cell debris.

After maturation, the cells leave their generative organs and move to the circulation and to the peripheral lymphoid organs. At this moment, they are called naive cells. They will survive for several days or months, waiting to find it's antigen. When an antigen is taken via an APC to a naive cell and this APC presents a strong co-stimulation signal, the cell will be activated.

A co-stimulator is a membrane-bound molecule on the surface of an antigen presenting cells, that provide additional stimulus required for T-cell activations. These molecules bind to molecules on T-cells surface and increase the affinity between the cells, which increases the chance of T-cell activation.

After activation, these lymphocyte will proliferate and differentiate into effector or memory cells. Most cells will perform their effector functions and die as the antigen is eliminated, while a small number will survive as memory cells.

The activation of lymphocytes is a regulated process, that demands time and depends on strong signaling. The adaptive immune repertoire will depend on what kind of antigen the body was exposed to.

It is important to point out that the lymphocytes not necessarily recognize the whole antigen, as sometimes only small portions of these molecules are necessary to trigger the matching. These fragments are called epitopes. Also, it may happen that the same antigen is matched by different epitopes when checked by B-cells and T-cells. So we may have different cells checking different aspects of the same molecule.

Natural Killer Cells

Natural killer cells are lymphocytes that belong to the innate immune system. They play an important role defending the organism against cells that were attacked by microbes and from tumoral cells. They do not express T or B-cell receptors on their surface, and their function is to recognize cells that were modified by the microbe invasion and not foreign antigens. If a cell from the host does not express NK-Cell inhibitory molecules on it's surface (which is a signal of cell damage) it will then be killed by the NK-cell.

These cells act by releasing cytotoxic substances that will induce cell lysis and death. They are named "natural killer" because they do not need a second activation signal to begin their effector mechanisms.

Tissues

The organs where the immune cells develop are the generative lymphoid organs and the peripheral lymphoid organs. Generative lymphoid organs are organs where the lymphocytes mature, i.e. the thymus for T-cells and the bone marrow for B-cells. Peripheral lymphoid organs are the organs where most of the antigen-presentation takes place. These organs are the lymph nodes, the spleen, the mucosal and the cutaneous immune system.

After maturation, the lymphocytes migrate from the generative to the peripheral lymphoid organs. Meanwhile, antigen presenting cells migrate through the body looking for antigens that match their receptor. When an APC finds a microorganism, it will phagocyse it, process it and move to a peripheral lymphoid organ. There, the APC will present the antigen (or parts of it) through it's MHC, expecting to find a T-cell that matches this antigen. If this T-cell is found and there is a strong binding between both cells, the T-cell is activated and an immune response takes place. This way, the chance of a T-cell to find it's matching antigen is increased at the peripheral lymphoid organs because the pathogenic antigens are transported to and concentrated in this region by APC's.

This is a very important aspect of the immune system considering the specifity of the lymphocytes receptors.

Lymphocytes at different stages of their life migrate to different places according to their function. After activation, T-cells leave the peripheral lymphoid organs while B-cells stay on them. This is because B-cells take part on the humoral (extra-cellular) immunity while T-cells are important on cell-mediated immunity. When stimulated, B-cells are able to secrete antibodies, that will be transported by the blood stream to the inflammation sites. On the other hand, T-cells are important on the activation of macrophages, so they need to move to the inflammation sites to perform signaling to effector cells.

Maturation of Lymphocytes

The process of lymphocyte maturation is based on three processes: proliferation of immature cells, expression of antigen receptor genes and selection.

The proliferation phase takes place before the immature cells are able to express receptors. On this phase a large pool of cells is generated, which will allow a diverse repertoire of antigen receptors to be created. These receptors are generated from several gene segments that undergo somatic hyper-mutation. Receptors are created from gene fragments through somatic recombination and not through a completely random process.

The lymphocytes will be selected through the maturation phase according to their usefulness and the risk they present to the body. In the process of creating diversity, some of the recombined genes will not express proteins and are, therefore, useless. If a cell is unable to express a receptor it will die by apoptosis. On the other hand, if a lymphocyte express a functional receptor, but this receptor has high affinity to an antigen during it's maturation, then it will also die by apoptosis because this cell would potentially trigger a reaction against the host, considering that most antigens present on the generative lymphoid organs are self-antigens.

There are some differences on the maturation process of B-cells and T-cells, for instance T-cells pass through a positive selection to check if they are able to bind to MHC molecules, but the basic processes decribed here apply to both kinds of cells and are sufficient to be used on this work.

7.2.4 Inflammation

Inflammation is a reaction of the innate immune system in vascularized tissues where leukocytes are attracted to an infection site. When activated, T-cells and macrophages secrete proteins called cytokines. These proteins, like antigens and PAMP's, can provide signaling to the attraction and the proliferation of cells on the infection site. As more cells are able to respond to the same stimuli, they also begin to secrete cytokines, providing a positive feedback that amplifies and localize the responses.

The process of cell clonal expansion and differentiation takes place in the peripheral lymphoid organs, and, after activation, the effector T-cells leave these tissues and migrate to the infection sites, attracted by cytokines. As the infection is cleared, the stimulating molecules concentration decrease and so most lymphocytes are deprived from survival factors. They start then a process of apoptosis, and the body return to it's balanced state.

The attraction of T-cells to the site of infection is independent on what antigen was recognized, i.e. all T-cells are attracted independently on which receptors they express. On the other hand, T-cells that are able to bind to APC's have stimuli to be retained on the infection site, while the other cells will return to the circulation. With this process, we have a concentration of infection-specific T-cells on the infection site.

It is important that we have all these confirmation mechanisms and second signalings so that we do not have immune reactions to non-pathogenic substances, because the immune reactions may also injure normal tissues.

An inflammation is an excellent example on how the innate and adaptive immune systems depend on each other: when a macrophage finds an antigen, it will release cytokines, which attract T-cells. If there is a binding between the antigen presented by the macropage and the T-cells, this T-cell is activated and this process also releases cytokines, activating and attracting more macrophages. This way, the immune response is localized and amplified by the responses of both systems.

Affinity Maturation

After repeated exposure to an antigen, the antibodies start to express a stronger affinity for it. This process is termed affinity maturation. This happens due to the fact that B-cells have mutations on some of it's genes variable regions during the proliferation phase, right after the antigen is recognized.

After the immune response is triggered, the amount of B-cells increases and some genetic recombination occurs with this expansion. These B-cells will start competing for the antigen, and the cells with the higher affinity are more capable of binding the antigen. If a cell is not able to bind, it will die by apoptosis.

Although this process presents the benefit of triggering more specific responses to the most common antigens, it also has it's drawbacks (**?**). When faced with a slightly different variation of an antigen, the body is less prepared to deal with it, due to the greater specificity of the immune system.

Tolerance

Even after maturation, peripheral tolerance may be induced on T-cells when some dendritic cells present self-antigens to lymphocytes on the peripheral lymphatic system

(?). This presentation without co-stimulation yields either apoptosis of the T-cell, anaergy (lack of expression) or differentiation into a regulatory cell. This can work as a back-up mechanism to prevent auto-immune reactions, or to signal that a new foreign antigen is not harmful and should not be attacked.

Also, when mature B-cells encounter long-lasting concentrations of their antigen they may become anergic as a tolerance mechanism. In fact, the effector mechanism is valid as long as the infection is being eliminated. If the concentration is not decreasing after a long time, either the antigen is part of the self or the effector mechanisms of this cell are not being efficient, so this cell is no longer be necessary.

7.3 Other Works on Immunology

This section presents some works on immunology that are not widely accepted by the immunologists. Although some of these theories are incomplete, and some of them might not really be presenting new concepts, they are useful when creating artificial immune systems because they may fill some gaps left open by classical immunology, or by providing more efficient ways to implement some mechanisms.

7.3.1 Danger Theory

The danger model (?) is a model proposed by Matzinger in 1994. In this model, little attention is given to self / nonself discrimination, and it considers that there are danger signals triggering immune responses.

On her work, Matzinger states that classical immunity views the immune response being triggered when a strange antigen is found on the body. On the other hand, the danger theory proposes the hypothesis that other confirmation signals would take part on the response. Residual particles of necrotic cell deaths released on the body would provide danger signals while the ausence of these particles together with the presence of apoptotic death residues would mean regulatory signals(?). It considers that the immune system does not need to attack everything that is foreign, but only foreign elements that have caused some harmful reactions(?). Self/nonself differentiation is considered when defining the antibodies, but the presence of stange elements is not what triggers the immune response.

We find experimental evidence supporting the hypothesis that danger signals may trigger immunological responses on several works. In (?) Piccioli et. al. observe that NK-cells respond to dangerously strong immunological responses by killing dendritic cells (i.e. we have a response to danger even if it this danger is presented by self-cells). This can be considered as mild evidence of the abstract concept of danger. In (?) Gavin et. al. also observe that PAMP's are not required for strong immunological responses, which indicate that other mechanisms may be involved on the risk characterization.

In (?) Kool et. al. provide a stronger evidence of the danger mechanism. This work is based on vaccine adjuvants. Vaccine adjuvants are substances added to vaccines to increase it's effectiveness (?). Some vaccines (especially those that are not based on whole attenutated viruses, but on DNA or protein fragments) do not trigger an immunological response, because they may not have pathogen associated molecular patters and do not present any risk to the body, so these antigens must be blended with other substances to create effective vaccines.

Aluminum hydroxide (alum) is a widely used vaccice adjuvant, but it's action mechanism is not well defined(?). *In vitro* experiments show that alum has no direct effect on antigen presenting cells activation, while *in vivo* injections result in strong and persistent T-cell activation. It is also observed on mice experiments (?) that alum injections result in the production of uric acid, which may be considered as danger signal. This way we have evidence of the danger theory when we consider that the injected alum induced cell necrosis, and that the body has associated the antigen injected together with the alum as the origin of the necrosis, so this is what triggered the strong immunological response.

Even providing useful and valid ideas for the characterization of some immunological responses, the danger theory attracted strong criticism, mostly because it is presented as a copernican revolution that intends to change everithing that is known about immunology with a new and revolutionary model (?), and this has caused a lot on unjustified fanrare around this work (?). Matzinger's ideas (?) are based on the naive assumption that the classical immunology is based solely on the characterization of self and non-self, and that every immunological response must be triggered by some kind of danger (although what exactly characterizes danger is obscure). These ideas ignore the fact that we have several elements triggering innate and adaptive immune responses, like antigens and toxins. PAMP's also induce immunological responses without presenting any danger. This can be observed when virosomes are used as vaccine adjuvants. Virossomes are vesicles containing viral membrane proteins (usually from the influenza virus) blended with antigens (?). This approach triggers an effective immune response without any danger element, since the virosome is a harmless PAMP.

To sum up, the whole immune system should be thought today as a multi-level data-fusion system(?), where the adaptive immunity handles antigens, the innate immune system detects PAMP's, toxins and handles the debris of necrotic cells, and both systems work together to control the strenght that needs to be used for each kind of situation. The self-nonself differentiation is an oversimplification of the the whole immune system, hence the danger model should be thought as a new process on a greater set of processes that compose the whole immune system. We should not sacrifice the model accuracy just for the sake of having a "grand unified theory"(?).

What is important to be taken from this theory when building AIS's is that we might observe other aspects of an environment when looking for an intrusion, instead of only watching the structure of the elements. It is important also to see that the immune system will distinguish some self from some non self, so there must be some leniency when the analisys are performed: no response need to be triggered against harmless non-self but responses need to be performed against harmful self. The danger model also depends a lot on the compartimentalization aspects of the immune system, because the proximity of the antigen to the danger signal is the aspect that merges both signals.

7.3.2 Idiotypic Network

In (**?**), Jerne proposed a mathematical framework for the populations of cells on the immune system. This hypothesis is based on the assumption that the b-cells are not independent and isolated cells, but instead form a network through interactions between receptors (i.e. by matching antibodies with other antibodies, not only with antigens). According to this theory, contact with an antigen would not only activate the antibodies that bind to that antigen (**?**), but also the antibodies that bind to this newly activated antibody, cascading the activation of other antibodies:

$$Ag \rightarrow Ab1 \rightarrow Ab2 \rightarrow Ab3 \rightarrow \ldots \tag{7.1}$$

The net result is that contact with an antigen activates a whole chain of antibodies, and this chain would be responsible to activate or suppress the immune responses, and would also help with memory mechanisms (**?**). This network might also be useful to protect the diversity of the repertoire, since some variances on the receptors might be protected, while the excess of redundancy might be supressed.

TODO: Describe criticisms (Idiotipic network as absurd)

7.4 Conclusion

In this chapter, we have presented some of the most important concepts of immunology when we want to build artificial immune systems. Not all concepts were discussed, since we do not intend to present a complete description of the human immune system. Only the concepts that make more sense when building artificial immune systems were presented. We did not limit this discussion to the widely accepted theories because some works that are controversial to immunologists might present useful and efficient analogies when building artificial immune systems.

It should also be clear that we do not intend, on this work, to mimic all the mechanisms present on natural systems. For instance, if lymphocytes are mapped as agents, a clonal expansion process could imply that the agents might need be duplicated. This does not need to be implemented literally, as there are other ways to increase the strength of an activated agent without the overhead of multiplying it's instances. Also, some aspects of migration might not be required as we can send as many agents to as many nodes are necessary in more efficient ways.

Chapter

8

Complex Networks

Redes complexas constituem uma teoria de representação de conhecimento que empregam estruturas topológicas (grafos) para armazenamento de informações. Elas permitem que diversos tipos de conexões sejam estabelecidas entre entidades, possibilitando que os sistemas modelados reflitam o sistema real. Dentro do contexto deste plano de pesquisa, redes complexas serão utilizadas para modelar *Grids* computacionais. Tais modelos serão adotados para avaliação das técnicas de otimização de acesso a dados propostas.

Adotando modelos de redes complexas pode-se avaliar sistemas computacionais distribuídos de diferentes escalas, eliminando a necessidade de um ambiente real totalmente controlado para experimentos. Ambientes sem controle não permitem obter resultados conclusivos, uma vez que outras aplicações podem influenciar nas medições (57). Além disso os modelos adotados devem permitir a extração de parâmetros do ambiente computacional tais como: grau de interconexão, distância e latência entre computadores. Esses parâmetros, em conjunto com o conhecimento de aplicações paralelas (seção **??**), tais como o acesso para leitura e escrita em arquivos distribuídos, serão adotados por técnicas de otimização a fim de reduzir o tempo consumido no acesso a dados, diminuindo, conseqüentemente, o tempo de execução de aplicações.

Esta seção apresenta conceitos básicos de redes complexas, principais modelos de redes complexas e aplicações relacionadas ao plano de pesquisa.

8.0.1 Conceitos básicos

Uma rede complexa pode ser representada por um grafo $G = (V, E)$ no qual existe um conjunto de vértices V e um conjunto de arestas E que os conectam. As arestas estabelecem uma relação entre dois vértices v_1 e v_2 na forma $\{v_1, v_2\} \in E$. Tais relação são especificamente criadas para melhor representa o problema a ser modelado. É possível, por exemplo, modelar toda a estrutura física de uma grande rede de computadores tal como a Internet. Nesse caso, os computadores conectados à Internet podem ser representados por vértices enquanto os canais de comunicação por arestas. Outras analogias podem também ser utilizadas, tais como o conteúdo de páginas WEB, relações sociais entre grupos de pessoas, redes organizacionais ou de negócios entre companhias, redes neurais, redes metabólicas, cadeia alimentar, entre outras.

O grafo de uma rede complexa pode ou não ser direcionado. Em um grafo direcionado (dígrafo), cada aresta tem um sentido (direção) que conecta um vértice origem a um destino. Exemplos de dígrafos são aqueles usados para representar chamadas telefônicas e mensagens de *e-mails*, nos quais as mensagens são direcionadas de uma pessoa para outra. Os dígrafos pode ser cíclicos, quando existe um caminho de um vértice para ele mesmo, ou acíclicos quando não existe esse caminho.

É importante destacar que nem todo grafo pode ser considerado uma rede complexa, pois essa classificação só é possível se o grafo apresentar as seguintes propriedades topológicas: vértices com alto grau, homogêneos ou heterogêneos, arestas com propriedades específicas, e finalmente, redes complexas podem evoluir ao longo do tempo.

Vértices com alto grau em relação aos demais são denominados *hubs* e sua presença tem grande influência na estrutura de uma rede complexa. Em uma rede pequena, faz sentido buscar o vértice cuja remoção seja determinante para a quebra de conectividade. Em uma rede com milhares de vértices, há pouca probabilidade da eliminação de um único vértice afetar significativamente sua funcionalidade. É importante determinar propriedades estatísticas dessas redes para modelagem de aplicações específicas, bem como formas de construção, configurações e evolução temporal (27; 26).

As redes complexas podem ser formadas por vértices homogêneos ou heterogêneos. Os componentes de uma rede que apresentam diferentes características podem pertencer a classes distintas tais como terminais e roteadores, pontos de geração e de consumo de um dado recurso, animais de sexos opostos, pessoas de diferentes comunidades, etc. Da mesma maneira, as arestas podem assumir propriedades específicas: valores de distância em uma malha rodoviária, velocidade máxima em uma ferrovia, largura de banda em uma rede de computadores, etc. À medida que são fornecidos mais detalhes dos fenômenos modelados, mais difícil torna-se a compreensão da rede complexa. A análise passa então a considerar métricas específicas para classes de vértices ou tipos de ligações.

Além disso, deve-se considerar que as redes complexas podem evoluir ao longo do tempo, com a inclusão ou remoção de vértices e arestas. Isso permite flexibilidade na modelagem e na análise comportamental, porém implica em um modelo mais complexo.

Outros termos associados às redes complexas são apresentados a seguir:

1. Coeficiente de aglomeração – usado para indicar transitividade e estrutura de comunidade;

2. Caminho – seqüência de vértices em que há arestas ligando cada um ao seguinte até que se alcance o último dos vértices do conjunto;

3. Caminho geodésico – menor caminho através da rede entre dois vértices. Não é necessariamente único;

4. Componente – conjunto de vértices que tem um caminho entre quaisquer dois vértices;

5. Correlação de graus – considera-se o grau dos vértices na investigação da probabilidade de conexão (vértices de mesmo grau compõe a mesma classe ou tipo para efeitos de medição);

6. Diâmetro – tamanho do mais longo caminho geodésico presente na rede;

7. Distribuição dos graus – é a função de distribuição de probabilidade de um dado vértice;

8. Navegação na rede – caminhamento na rede, de um vértice para outro ou em direção a um determinado destino;

9. Resiliência da rede – capacidade da rede em resistir à remoção de alguns de seus vértices sem a perda de sua funcionalidade;

10. Transitividade (*clustering*) – probabilidade média de dois vértices vizinhos de um outro vértice serem adjacentes. Indica a presença de triângulos na rede, ou seja, conjuntos de três vértices em que cada um se conecta aos outros dois.

8.0.2 Modelos de redes complexas

Embora a origem das redes complexas reportam a trabalhos pioneiros a respeito de grafos aleatórios (37), pesquisas nessa área tornaram-se, recentemente, foco da atenção de pesquisadores (26). O motivo para isso foi a descoberta de características de conectividade não aleatória em redes reais (Internet, WWW, etc.). Ao invés disso, redes derivadas de informações reais apresentam estruturas de comunidades, leis de potência para distribuição de grau e a presença de *hubs*, entre outros aspectos estruturais. Dois trabalhos, em particular, tiveram grande contribuição para esse avanço: Watts & Strogatz (117) na investigação de redes mundo pequeno e Barabasi & Albert (12) na caracterização de modelos livre de escala.

Os dois principais fatores para a popularidade das redes complexas são a flexibilidade e a generalidade na representação de qualquer estrutura natural, incluindo aquelas que modificam dinamicamente a topologia da rede. Atualmente, essas estruturas tornaram-se atrativas não apenas na aplicação de conceitos relacionados a comportamentos e configurações de redes reais, mas também em estudar a evolução dinâmica da topologia da rede.

As redes complexas podem ser analisadas segundo suas características quantitativas relacionadas à topologia. A descrição quantitativa de suas propriedades também fornecem subsídios fundamentais para classificação das redes. Os principais modelos de redes complexas são descritos a seguir.

Modelo aleatório

O modelo de redes complexas aleatório, definido por Erdös e Rény (37), pode ser considerado como o mais básico, o qual gera grafos aleatórios com n vértices desconectados e E arestas (figura 8.1). O processo de geração da rede consiste em conectar aleatoriamente pares de vértices segundo uma probabilidade. Dessa premissa, Erdös e Rényi concluíram que todos os vértices, em uma determinada rede, têm aproximadamente a mesma quantidade de conexões, ou igualdade nas chances de receber novas ligações. O número médio de conexões k nessa rede é definido pela equação 8.1, onde, p é a probabilidade e n o número total de vértices.

$$k = p(n-1) \tag{8.1}$$

Se p é obtido como uma função de n para manter k fixo então: $p = k/(n-1)$. Para esse modelo, $P(k)$, denominada distribuição de grau é dada por uma função de Poisson, definida na equação 8.2.

$$P(k) = \frac{e^{-k}k^k}{k!} \tag{8.2}$$

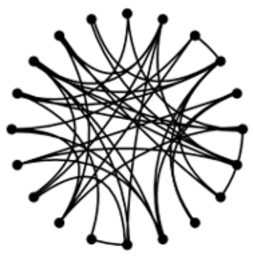


Figure 8.1: Exemplo de uma rede complexa aleatória

Modelo mundo pequeno

Watts & Strogatz (117) descobriram que sistemas reais organizados como redes apresentam padrões altamente conectados, tendendo a formar pequenos aglomerados (*clus-*

ters) de vértices. Eles propuseram um modelo semelhante ao de Erdös e Rényi, no qual as ligações são estabelecidas entre os vértices adjacentes e algumas estabelecidas de modo aleatório. Esse modelo demonstra que a distância média entre dois vértices de uma grande rede não ultrapassa um número pequeno de saltos (18).

No modelo de Watts e Strogatz cada vértice conhece a localização de vários outros, que por sua vez têm outras localizações conhecidas. Em larga escala, essas conexões demonstram a existência de poucos graus de separação entre vértices. Além disso, bastam poucas ligações entre vários aglomerados para transformar uma rede em um grande aglomerado (18). Um exemplo desse tipo de rede é apresentado na figura 8.2.

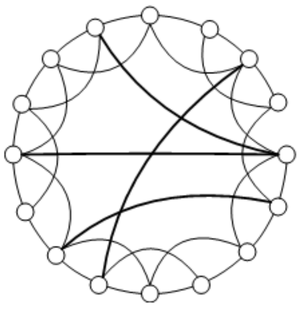


Figure 8.2: Exemplo de uma rede complexa mundo pequeno

Nesse modelo, a função de distribuição de grau é definida na equação 8.3, onde n representa o número inicial de vizinhos de cada vértice na rede (na figura 8.2 $n = 4$) e p a probabilidade. Dessa maneira, o número médio de conexões, k, esperadas é dado por: $k = 2n$.

$$P(k) = \sum_{i=1}^{min(k-n,n)} \binom{k}{i} (1-p)^i p^{k-1} \frac{(pk)^{k-n-i}}{(k-n-i)!} e^{-pk} \tag{8.3}$$

Modelo livre de escala

Barabasi & Albert (12) demonstram que algumas redes não são formadas de modo aleatório. Acredita-se na existência de uma ordem na dinâmica em sua estruturação, a qual segue características específicas. Uma dessas características é denominada conexão preferencial (*preferential attachment*), onde um novo vértice tende a se conectar com outro pré-existente, porém, com maior probabilidade de se interligar ao vértice que apresenta maior número de conexões. Isso implica em outra premissa fundamental: as redes não seriam constituídas de vértices igualitários, isto é, com mesma probabilidade no número de conexões.

Ao contrário, tais redes apresentam poucos vértices que seriam altamente conectados (*hubs*) e uma grande maioria de vértices com poucas conexões. As redes com essas características (exemplo na figura 8.3) foram denominadas livre de escala (*scale free*).

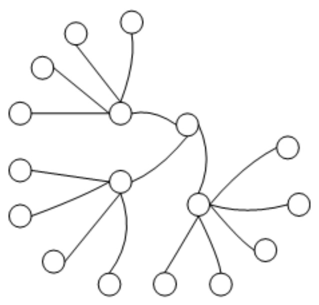


Figure 8.3: Exemplo de uma rede complexa livre de escala

O nome **livre de escala** advém de características da representação matemática do grau de conexão dos vértices da rede, o qual segue uma função denominada *power law*. Essa função decresce abruptamente, construindo uma longa **cauda**. Essa distribuição implica em muitos vértices com apenas algumas ligações e uma minoria, mas significativa, com grande quantidade de ligações.

Uma lei de potência entre dois vértices escalares x e y é formalmente definida por $y = ax^z$, onde, a denomina-se constante de proporcionalidade e z o expoente, a e z são constantes. A lei de potência pode ser interpretada como uma linha reta em um gráfico $log - log$ cuja equação anterior pode ser descrita pela equação 8.4.

$$log(y) = zlog(x) + log(a) \tag{8.4}$$

Na definição de uma rede complexa livre de escala, a distribuição de grau segue uma lei de potência para um dado k, equação 8.5, onde, k é o número médio de conexões e γ é o expoente do grau, definido entre os valores 2 e 3 (13). O número médio de conexões, k, é dado por: $k = 2m$, onde m é o número de arestas que são inseridas na rede no instante que um novo vértice é adicionado por meio do mecanismo conexão preferencial.

$$P(k) \sim k^{-\gamma} \tag{8.5}$$

A tabela 8.1 relaciona cada modelo de rede complexa apresentando suas duas características principais: número médio de conexões em cada vértice e a função de distribuição de grau (26).

Table 8.1: Relação entre os modelos de redes

Métrica	Erdõs-Rény	Watts-Strogatz	Barabási-Albert
Distribuição de Grau	$P(k) = \frac{e^{-k}k^k}{k!}$	$P(k) = \sum_{i=1}^{min(k-n,n)} \binom{k}{i}(1-p)^i p^{k-1} \frac{(pk)^{k-n-i}}{(k-n-i)!} e^{-pk}$	$P(k) \sim k^{-\gamma}$
Número médio de conexões	$k = p(n-1)$	$k = 2n$	$k = 2m$

Bibliography

[1] Abarbanel, H. D. I.; Brown, R.; Sidorowich, J. J.; Tsimring, L. S. (1993). The analysis of observed chaotic data in physical systems. *Reviews of Modern Physics*, v.65, p.1331–1392.

[2] Aha, D. W.; Kibler, D.; Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, v.6, n.1, p.37–66.

[3] Albertini, M. K.; Mello, R. F. (2007). A self-organizing neural network for detecting novelties. *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, p. 462–466, Nova Iorque, EUA. ACM.

[4] Allen, B. (2004). Monitoring hard disks with smart – available at: http://www.linuxjournal.com/article/6983.

[5] Allen, B. (2009). Smartctl man pages – available at linux systems, calling the man command for smartctl.

[6] Alligood, K. T.; Sauer, T. D.; Yorke, J. A. (1997). *Chaos: An Introduction to Dynamical Systems*. Springer.

[7] Attachment, A. (2009). At attachment – available at: http://en.wikipedia.org/wiki/advanced_technology_attachment.

[8] Axelsson, S. (2000). Intrusion detection systems: A survey and taxonomy. Relatório Técnico Technical report 99-15, Department of Computer Engineering, Chalmers University.

[9] Bachelder, I.; Waxman, A.; Seibert, M. (1993). A neural system for mobile robot visual place learning and recognition. *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching*, v. LNCS 807, p. 198–212, Berlin.

[10] Back, T.; Fogel, D. B.; Michalewicz, Z., editores (1999a). *Basic Algorithms and Operators*. IOP Publishing Ltd., Bristol, Reino Unido.

[11] Back, T.; Fogel, D. B.; Michalewicz, Z., editores (1999b). *Basic Algorithms and Operators*. IOP Publishing Ltd., Bristol, Reino Unido.

[12] Barabasi, A.-L.; Albert, R. (1999). Emergence of scaling in random networks. *Science*, v.286, n.5439, p.509–512.

[13] Barabasi, A.-L.; Ravasz, E.; Vicsek, T. (2001). Deterministic scale-free networks. *PHYSICA A*, v.299, p.3.

[14] Bianchi, G.; Tinnirello, I. (2003). Kalman filter estimation of the number of competing terminals in an ieee 802.11 network. *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, v. 2, p. 844–852 vol.2.

[15] Box, G.; Jenkins, G. M.; Reinsel, G. (1994). *Time Series Analysis: Forecasting & Control (3rd Edition)*. Prentice Hall.

[16] Bretscher, O. (2004). *Linear Algebra with Applications*. Prentice Hall.

[17] Brown, D. J.; Suckow, B.; Wang, T. (2001). A survey of intrusion detection systems. Relatório Técnico Graduate Operating System Project – CSE 221, Department of Computer Science, University of California, San Diego.

[18] Buchanan, M. (2002). *Nexus - Small Worlds and the Groundbreaking Science of Networks*. W. W. Norton & Company.

[19] Buhmann, M. D. (2003). *Radial Basis Functions: Theory and Implementations*. Cambridge University Press.

[20] Carpenter, G. A.; Gjaja, M. N.; Gopal, S.; Woodcock, C. E. (1997). ART neural networks for remote sensing: Vegetation classification from lansat TM and terrain data. *IEEE Transactions on Geoscience and Remote Sensing*, v.35, n.2.

[21] Carpenter, G. A.; Grossberg, S. (1988). The ART of adaptive pattern recognition by a self-organizing neural network. *Computer*, v.21, n.3, p.77–88.

[22] Carpenter, G. A.; Grossberg, S. (1989). ART 2: Self-organization of Stable Category Recognition Codes for Analog Input Patterns. *Applied Optics*, v.26, n.23, p.4919–4930.

[23] Carpenter, G. A.; Grossberg, S.; Rosen, D. B. (1991). ART 2-A: An Adaptive Resonance Algorithm for Rapid Category Learning and Recognition. *Neural Networks*, v.4, p.494–504.

[24] Casdagli, M. (1989). Nonlinear prediction of chaotic time series. *Physica D Nonlinear Phenomena*, v.35, p.335–356.

[25] Clark, J. D. (1990). Modeling and simulating complex spatial dynamic systems: a framework for application in environmental analysis. *SIGSIM Simul. Dig.*, v.21, n.2, p.9–19.

[26] Costa, L. F.; Rodrigues, F. A.; Travieso, G.; Boas, P. R. V. (2007). Characterization of complex networks: A survey of measurements. *Advances In Physics*, v.56, p.167–242.

[27] Costa, L. F.; Travieso, G.; Ruggiero, C. A. (2005). Complex grid computing. *The European Physical Journal B - Condensed Matter*, v.44, n.1, p.119–128.

[28] DEBAR, H.; BECKER, M.; SIBONI, D. (1992). A neural network component for an intrusion detection system. *SP '92: Proceedings of the 1992 IEEE Symposium on Security and Privacy*, p. 240, Washington, DC, USA. IEEE Computer Society.

[29] Debra Anderson, Thane Frivold, A. V. (1994). Next generation intrusion detection expert system (nides). Relatório Técnico A008, SRI International, Menlo Park, CA 94025-3493.

[30] Doblinger, G. (1998). An adaptive kalman filter for the enhancement of noisy ar signals. *Circuits and Systems, 1998. ISCAS '98. Proceedings of the 1998 IEEE International Symposium on*, v.5, p.305–308.

[31] Dodonov, E.; Mello, R. F. (2008). Estudo sobre abordagens de extração, classificação e predição de comportamento de processos. Relatório Técnico 322, Instituto de Ciências Matemáticas e de Computação, USP, São Carlos, SP, Brazil.

[32] Dowel, C.; Ramstedt, P. (1990). The computer watch data reduction tool. *Proceedings of the 13th National Computer Security Conference*, p. 99–108, Washington DC, USA. NIST, National Institute of Standards and Technology/National Computer Security Center.

[33] Eckmann, J.-P.; Ruelle, D. (1985). Ergodic theory of chaos and strange attractors. *Reviews of Modern Physics*, v.57, p.617–656.

[34] Edmonds, A. N. (1996). *Time Series Prediction Using Supervised Learning and Tools from Chaos Theory.* Tese (Doutorado), University of Luton.

[35] Elert, G. (2005). *The Chaos Hypertextbook: Measuring Chaos – Disponível em: http://hypertextbook.com/chaos/. Acesso em 10 de setembro de 2007.*

[36] Elmer, F.-J. (1998). The lyapunov exponent – disponível em: http://monet.unibas.ch/ elmer/pendulum/lyapexp.htm. acesso em 10 de setembro de 2007.

[37] Erdos, P.; Renyi, A. (1959). On random graphs. *Publicationes Mathematicae*, v.6, p.290–297.

[38] Filippidis, A.; Jain, L. C.; Lozo, P. (1999). Degree of familiarity ART2 in knowledge-based landmine detection. *IEEE Transactions on Neural Networks*, v.10, n.1.

[39] Fraser, A. M.; Swinney, H. L. (1986). Independent coordinates for strange attractors from mutual information. *Phys. Rev. A*, v.33, n.2, p.1134–1140.

[40] Freeman, J. A.; Skapura, D. M. (1991). *Neural networks: algorithms, applications, and programming techniques.* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, EUA.

[41] Gan, K.; Lua, K. (1992). Chinese character classification using adaptive resonance network. *Pattern Recognition*, v.25, n.8, p.877–888.

[42] Garey, M. R.; Johnson, D. S. (1979a). *Computers and Intractability : A Guide to the Theory of NP-Completeness.* Series of Books in the Mathematical Sciences. W. H. Freeman.

[43] Garey, M. R.; Johnson, D. S. (1979b). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., Nova Iorque, NY, EUA.

[44] Goldberg, I.; Wagner, D.; Thomas, R.; Brewer, E. A. (1996). A secure environment for untrusted helper applications confining the wily hacker. *SSYM'96: Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography*, p. 1–1, Berkeley, CA, USA. USENIX Association.

[45] Gärtner, F. C. (1999). Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys*, v.31.

[46] Habra, N.; Charlier, B. L.; Mounji, A.; Mathieu, I. (1992). Asax: Software architecture and rule-based language for universal audit trail analysis. *Proceedings of ESORICS'92, European Symposium on Research in Computer Security.*

[47] Haykin, S. (2008). *Neural Networks and Learning Machines.* Prentice-Hall, 3 edição.

[48] He, J.; Tan, A.-H.; Tan, C.-L. (2003). Modified art 2a growing network capable of generating a fixed number of nodes. *IEEE Transactions on Neural Networks*, v.3, n.15, p.728–737.

[49] Heberlein, L.; Dias, G.; Levitt, K.; Mukherjee, B.; Wood, J.; Wolber, D. (1990). A network security monitor. *Research in Security and Privacy, 1990. Proceedings., 1990 IEEE Computer Society Symposium on*, v., p.296–304.

[50] Hegger, R.; Kantz, H.; Schreiber, T. (2009). Tisean 3.0.1: Nonlinear time series analysis. Disponível em: http://www.mpipks-dresden.mpg.de/~tisean/Tisean_3.0.1/index.html. Acesso em: 26 de janeiro de 2009.

[51] Hinterding, R. (2000). Representation, mutation and crossover issues in evolutionary computation. *Proc. of the 2000 Congress on Evolutionary Computation*, p. 916–923, Piscataway, NJ. IEEE Service Center.

[52] Hopfield, J. J. (1988). Neural networks and physical systems with emergent collective computational abilities. *Neurocomputing: foundations of research*, v., p.457–464.

[53] Ilgun, K. (1993). Ustat: a real-time intrusion detection system for unix. *Research in Security and Privacy, 1993. Proceedings., 1993 IEEE Computer Society Symposium on*, v., p.16–28.

[54] Itti, L.; Baldi, P. (2005). A principled approach to detecting surprising events in video. *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1*, p. 631–637, Washington, DC, USA. IEEE Computer Society.

[55] Jackson, K. A.; DuBois, D. H.; Stallings, C. A. (1991). An expert system application for network intrusion detection. *Proceedings of the 14th National Computer Security Conference*, p. 215–225, Washington, D.C. National Institute of Standards and Technology/National Computer Security Center.

[56] Jacobson, V.; Leres, C.; McCanne, S. (2009). Tcpdump man pages – available at linux systems, calling the man command for tcpdump.

[57] Jain, R. (1991). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurements, Simulation and Modeling.* John Wiley & Sons.

[58] Jou, Y. F.; Gong, F.; Sargor, C.; Wu, S. F.; Rance, C. W. (1997). Architecture design of a scalable intrusion detection system for the emerging network infrastructure. Relatório Técnico CDRL A005, Dept. of Computer Science, North Carolina State University, Releigh, N.C, USA.

[59] Kantz, H. (1994). A robust method to estimate the maximal Lyapunov exponent of a time series. *Physics Letters A*, v.185, p.77–87.

[60] Kaplan, I. (2003). Estimating the hurst exponent – disponível em http://www.bearcave.com/misl/misl_tech/wavelets/hurst/index.html. acesso em 10 de setembro de 2007.

[61] Kennel, M. (2002). The multiple-dimensions mutual information program – disponível em: http://www-ncsl.postech.ac.kr/en/softwares/archives/mmi.tar.z. acesso em: 10 de setembro de 2007.

[62] Kennel, M.; Brown, R.; Abarbanel, H. (1992a). Determining embedding dimension for phase space reconstruction using the method of false nearest neighbors. Relatório técnico, Institute for Nonlinear Science and Department of Physics, University of California, San Diego, Mail Code R-002, La Jolla, CA, EUA.

[63] Kennel, M. B.; Brown, R.; Abarbanel, H. D. I. (1992b). Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Phys. Rev. A*, v.45, n.6, p.3403–3411.

[64] Keyvan, S.; Rabelo, L. C. (1992). Sensor signal analysis by neural networks for surveillance in nuclear reactors. *IEEE Transactions on nuclear science*, v.39, n.2.

[65] King, R. L.; Birk, R. J. (2004). Developing earth system science knowledge to manage earth's natural resources. *Computing in Science and Engineering*, v.6, n.1, p.45–51.

[66] Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, Number*, v.220, n.4598, p.671–680.

[67] Klein, A. (2005). Optimizing enterprise it: Six strategies for reducing costs and ensuring business value – available at: http://www.informationweek.com/news/showarticle.jhtml?articleid=60404936. *Information Week*, v.

[68] Ko, C.; Fink, G.; Levitt, K. (1994). Automated detection of vulnerabilities in privileged programs by execution monitoring. *In Proceedings of the 10th Annual Computer Security Applications Conference*, p. 134–144.

[69] Kohler, M. (1997). Using the kalman filter to track human interactive motion - modelling and initialization of the kalman filter for translational motion. Relatório Técnico 629/1997, Fachbereich Informatik, Universität Dortmund, Fachbereich Informatik, Universität Dortmund, 44221 Dortmund, Alemanha.

[70] Kumar, S.; Spafford, E. H. (1994). A Pattern Matching Model for Misuse Intrusion Detection. *Proceedings of the 17th National Computer Security Conference*, p. 11–21.

[71] Lageweg, B. J.; Lenstra, J. K. (1977). Private communication.

[72] Laprie, J.-C.; Randell, B. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, v.1, n.1, p.11–33. Fellow-Avizienis,, Algirdas and Senior Member-Landwehr,, Carl.

[73] Lm-Sensors (2009). Hardware monitoring by lm-sensors – available at: http://www.lm-sensors.org.

[74] Lorenz, E. (1963). Deterministic nonperiodic flow. *Journal of Atmospheric Science*, v.20, p.130–141.

[75] Lunt, T. F.; Jagannathan, R.; Lee, R.; Listgarten, S.; Edwards, D. L.; Neumann, P. G.; Javitz, H. S.; Valdes, A. (1988). Ides: The enhanced prototype, a real-time intrusion detection system. Relatório Técnico Technical Report SRI Project 4185-010, SRI-CSL-88-12, CSL SRI International, Computer Science Laboratory, STI Intl. 333 Ravenswood Ave.,Menlo Park, CA 94925-3493, USA.

[76] Ma, J.; Perkins, S. (2003). Online novelty detection on temporal sequences. *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, p. 613–618, New York, NY, USA. ACM Press.

[77] Magalhães, M. N.; de Lima, A. C. P. (2004). *Noções de Probabilidade e Estatística.* Edusp, 6 edição.

[78] Marsland, S. (2002). *On-line Novelty Detection Through Self-Organisation, With Application to Inspection Robotics.* Tese (Doutorado), University of Manchester.

[79] Mañé, R. (1980). *On the dimension of the compact invariant sets of certain nonlinear maps.* Springer.

[80] McGraw, G. (2004). Software security. *Security & Privacy, IEEE*, v.2, n.2, p.80–83.

[81] Mckinley, P. K.; Sadjadi, S. M.; Kasten, E. P.; Cheng, B. H. C. (2004). A taxonomy of compositional adaptation. Relatório técnico, Technical Report MSU-CSE-04-17, Software Engineering and Network Systems Laboratory, Michigan State University, East Lansing, Michigan.

[82] Medio, A.; Gallo, G. (1993). *Chaotic Dynamics: Theory and Applications to Economics.* Cambridge University Press.

[83] Mello, R. F.; Senger, L. J. (2006). Model for simulation of heterogeneous high-performance computing environments. *7th International Conference on High Performance Computing in Computational Sciences – VECPAR 2006*, p. 1–11.

[84] Meng, A. (2003). An introduction to markov and hidden markov models. http://www2.imm.dtu.dk/pubdb/p.php?3313.

[85] Metropolis, N.; Ulam, S. (1953). A property of randomness of an arithmetical function. *The American Mathematical Monthly*, v.60, n.4, p.252–253.

[86] Mitchell, T. M. (1997). *Machine Learning.* McGraw-Hill.

[87] Murch, R. (2004). *Autonomic Computing.* IBM Press.

[88] Palis Jr., J.; Melo, W. (1978). *Introdução aos Sistemas Dinâmicos.* Edgard Blücher, 1 edição.

[89] Papadimitriou, C. M. (1994). *Computational complexity.* Addison-Wesley, Reading, Massachusetts.

[90] Parashar, M.; Hariri, S. (2006). *Autonomic Computing: Concepts, Infrastructure, and Applications.* CRC Press.

[91] Paxson, V. (1999). Bro: A system for detecting network intruders in real-time. *Computer Networks*, p. 2435–2463.

[92] Penny, W.; Harrison, L. (2006). Multivariate autoregressive models. Friston, K.; Ashburner, J.; Kiebel, S.; Nichols, T.; Penny, W., editores, *Statistical Parametric Mapping: The analysis of functional brain images.* Elsevier, Londres.

[93] Perry, G. L. W.; Enright, N. J.; Jaffre, T. (2001). Spatial modelling of landscape-scale vegetation dynamics, mont do, new caledonia. *S. Afr. J. Sci.*, v.97, n.11-12, p.501–509+.

[94] Piovoso, M.; Laplante, P. A. (2003). Kalman filter recipes for real-time image processing. *Real-Time Imaging*, v.9, n.6, p.433–439.

[95] Porras, P. A.; Neumann, P. G. (1997). Emerald: Event monitoring enabling responses to anomalous live disturbances. *Proc. 20th NIST-NCSC National Information Systems Security Conference*, p. 353–365.

[R & Mathur] R, J. K.; Mathur, A. P. Software engineering for secure software - state of the art: A survey. Relatório técnico, Purdue University.

[96] Rosenstein, M. T.; Collins, J. J.; Luca, C. J. D. (1993). A practical method for calculating largest lyapunov exponents from small data sets. *Physica D*, v.65, p.117–134.

[97] Sastry, K.; Abbass, H. A.; Goldberg, D. E.; Johnson, D. D. (2005). Sub-structural niching in estimation of distribution algorithms. *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, p. 671–678, Nova Iorque, NY, EUA. ACM Press.

[98] Sebring, M. M.; Shellhouse, E.; Hanna, M. E.; Whitehurst, R. A. (1988). Expert systems in intrusion detection: A case study. *In Proceedings of the 11th National Computer Security Conference*, p. 74–81, Baltimore, Maryland.

[99] Senger, L. J.; Mello, R. F.; Santana, M. J.; Santana, R. H. C. (2005). An on-line approach for classifying and extracting application behavior on linux. *High Performance Computing: Paradigm and Infrastructure*, p. 381–402. John Wiley and Sons.

[100] Senger, L. J.; Mello, R. F.; Santana, M. J.; Santana, R. H. C. (2007). Aprendizado baseado em instâncias aplicado à predição de características de execução de aplicações paralelas. *Revista de Informática Teórica e Aplicada*, v.14, p.44–68.

[101] Shefler, W. C. (1988). *Statistics: Concepts and Applications.* The Benjamin/Cummings.

[102] Shenshi, G.; Zhiqian, W.; Jitai, C. (1999). The fractal research and predicating on the times series of sunspot relative number. *Applied Mathematics and Mechanics*, v.20, n.1, p.84–89.

[103] Shir, O. M.; Back, T. (2005). Niching in evolution strategies. *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, p. 915–916, Nova Iorque, NY, EUA. ACM Press.

[104] Sickle, L. V.; Chandy, K. M. (1977). Computational complexity of network design algorithms. *IFIP Congress*, p. 235–239.

[105] Smaha, S. (1988). Haystack: an intrusion detection system. *Aerospace Computer Security Applications Conference, 1988., Fourth*, v., p.37–44.

[106] SMART (2009). Self-monitoring, analysis, and reporting technology – available at: http://en.wikipedia.org/wiki/s.m.a.r.t.

[107] Smith, T.; Boning, D. (1997). A self-tuning ewma controller utilizing artificial neural network function approximation techniques. *Components, Packaging, and Manufacturing Technology, Part C, IEEE Transactions on*, v.20, n.2, p.121–132.

[108] Snapp, S. R.; Brentano, J.; Dias, G. V.; Goan, T. L.; Heberlein, L. T.; Ho, C.-L.; Levitt, K. N.; Mukherjee, B.; Smaha, S. E.; Grance, T.; Teal, D. M.; Mansur, D. (1998). Dids (distributed intrusion detection system)—motivation, architecture, and an early prototype. v., p.211–227.

[109] Staniford-chen, S.; Cheung, S.; Crawford, R.; Dilger, M.; Frank, J.; Hoagl, J.; Levitt, K.; Wee, C.; Yip, R.; Zerkle, D. (1996). Grids - a graph based intrusion detection system for large networks. *In Proceedings of the 19th National Information Systems Security Conference*, p. 361–370.

[110] Sweitzer, J. W.; Draper, C. (2006). Architecture overview for autonomic computing. *Autonomic Computing: Concepts, Infrastructure, and Applications.* CRC Press.

[111] Takens, F. (1980). Detecting strange attractors in turbulence. *Dynamical Systems and Turbulence*, p. 366–381. Springer.

[112] Ultsch, A. (1993). Self-organising neural networks for monitoring and knowledge acquisition of a chemical process. *Proceedings of ICANN-93*, p. 864–867.

[113] Vaccaro, H. S.; Liepins, G. E. (1989). Detection of anomalous computer session activity. p. 280–289.

[114] van Renesse, R.; Birman, K. P. (2006). Autonomic computing: A system-wide perspective. *Autonomic Computing: Concepts, Infrastructure, and Applications.* CRC Press.

[115] Vlajic, N.; Card, H. C. (2001). Vector quantization of images using modified adaptive resonance algorithm for hierarchical clustering. *IEEE Transactions on Neural Network*, v.12, n.5, p.1147–1162.

[116] Wan, E. A.; Van Der Merwe, R. (2000). The unscented kalman filter for nonlinear estimation. *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, p. 153–158.

[117] Watts, D. J.; Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, v.393, p.440–442.

[118] White, G. B.; Pooch, U. W. (1996). Cooperating security managers: Distributed intrusion detection systems. *Computers & Security*, v.15, n.5, p.441–450.

[119] Whiteley, J. R.; Davis, J. F. (1993). Qualitative interpretation of sensor patterns. *IEEE Expert*, v.8, p.54–63.

[120] Whiteley, J. R.; Davis, J. F. (1996). Observations and problems applying ART2 for dynamic sensor pattern interpretation. *IEEE Transactions on Systems, Man and Cybernetics-Part A: Systems and Humans*, v.26, n.4, p.423–437.

[121] Whitney, H. (1936). Differentiable manifolds. *The Annals of Mathematics*, v.37, n.3, p.645–680.

[122] Wilson, D. R.; Martinez, T. R. (1997). Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, v.6, p.1–34.

[123] Zeevi, A. J.; Meir, R.; Adler, R. J. (1998). Non-linear models for time series using mixtures of autoregressive models. Relatório técnico.