9-2010

A Software Application For The Selection Of Temperature Measuring Sensors Using The Analytic Hierarchy Process (AHP)

Shadi Mohammad AL-B'ool
Binghamton University--SUNY, salbool1@binghamton.edu

A SOFTWARE APPLICATION FOR THE SELECTION OF TEMPERATURE MEASURING SENSORS USING THE ANALYTIC HIERARCHY PROCESS (AHP)

A SOFTWARE APPLICATION FOR THE SELECTION OF TEMPERATURE MEASURING SENSORS USING THE ANALYTIC HIERARCHY PROCESS (AHP)

by

Shadi Mohammad AL-B'ool

Advisor

Dr. Tarek H. Al-Hawari

Co-Advisor

Dr. Amer M. Momani

Thesis submitted in partial fulfillment of the requirements for the degree of

M.Sc. in Industrial Engineering

At

The Faculty of Graduate Studies

Jordan University of Science and Technology

September, 2010

A SOFTWARE APPLICATION FOR THE SELECTION OF TEMPERATURE MEASURING SENSORS USING THE ANALYTIC HIERARCHY PROCESS (AHP)

by

Shadi Mohammad AL-B'ool

Signature of Author ……

Committee Member	Signature and Date
Dr. Tarek H. Hawari (Chairman)	……
Dr. Amer M. Momani (Co-Advisor)	……..
Dr. Samer K. Khurais (Member)	.....
Dr. Rami H. Hadeethy (External Examiner)	.....

September, 2010

DEDICATION

This thesis is honorably dedicated to my precious mother, my dear father, and my brothers and sisters without whom I would have not achieved this success. It is also dedicated to my best friends Bassem Hassoun and Amer Momani for their continual assistance and encouragement without which this work has not come to the fore.

ACKNOWLEDGMENTS

It is a great pleasure for me to recognize the valuable help and assistance I have received from my supervisor, Dr. Tarek Al-Hawari and co-advisor Dr. Amer Momani. They really were my reliable tutors and mentors who led me through the way to fulfill this work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

A SOFTWARE APPLICATION FOR THE SELECTION OF TEMPERATURE MEASURING SENSORS USING THE ANALYTIC HIERARCHY PROCESS (AHP)

by

Shadi Mohammad AL-B'ool

This study presents a computer program that applies analytic hierarchy process (AHP) method to objectively select the best temperature sensors for various applications from multiple nominated alternatives. The underlying decision method based on AHP methodology, ranks temperature sensors with different features with a score resulting from the synthesis of relative preferences of each alternative to the others at different levels considering independent evaluation criteria. At each level, relative preferences of each candidate alternative with respect to the upper immediate level are calculated from pair-wise comparisons among the candidate alternative sensors based on the specifications of sensors with respect to a selected application. These pair-wise relative comparison weights are embedded in the computer software and are retrieved whenever the user specifies the application, the restrictions, and the available alternative sensors that meet these restrictions. AHP method proves to provide a quantitative and rational alternative performance evaluation method; it permits simpler, easier and more organized decision making process than subjective opinions that are subject to erroneous judgments. In this study, the application of AHP method in selecting the best temperature sensor for a particular application is embedded via the use of a computer program built using C# programming language to help perform the selection process in an easy graphical user interface GUI, ready-to-use, and computerized way and thus provides aid to those working in industry and in need of such a software tool.

The proposed computer program is versatile and applicable to multitude of temperature sensors selection situations. A case study from the automotive industry which is the catalytic convertor application is presented. This application demands the use of temperature sensors capable of monitoring high temperatures in the order of 500°C-750°C, with a maximum temperature of ~870°C [1]. The selection process is conducted from among three alternative sensor categories, these are: thermocouples, thermisters, and RTD thermometers. The computer program is

robust and applicable to a wider range of temperature sensors selection situations with a variety of applications and different arrays of candidate sensors.

Chapter One Introduction

1.1 Motivation for Process Measurement

The ultimate goal of any industrial company is to make profit. Companies that in the long run continue to provide an adequate return on stockholder investment tend to survive; those that fail to do so eventually disappear. This underscores the importance of profitable manufacturing operations, and it is ultimately the need to maximize profit that provides the motivation for a company to buy process measurement and control systems. The uniformity and quality of the product in any industrial process depend on the ability to maintain the correct operating conditions and parameters within a certain range. Process sensors are devices that measure these parameters, and the resulting data is used to control the process. In addition, such measurements enable better process understanding, which often drives process improvement. The connection between profit and process measurement is illustrated in Figure 1.1.

Product quality and uniformity have a large impact on market demand and share of any industrial company, especially if similar products are offered by competitors. Apart from machine malfunction or operator error, defects are usually caused by variability in the feed stock or excursions in the operating conditions. Process feedback gained from process sensors enables active process control, which can respond to such variations in feedstock and excursions in operating conditions as they arise. By automatically adapting the process to changing feedstock with a strong connection with the process measurement system, the process controller improves product uniformity and minimizes the amount of defective

product. The result is a top-quality product that will consistently meet customers' expectations.

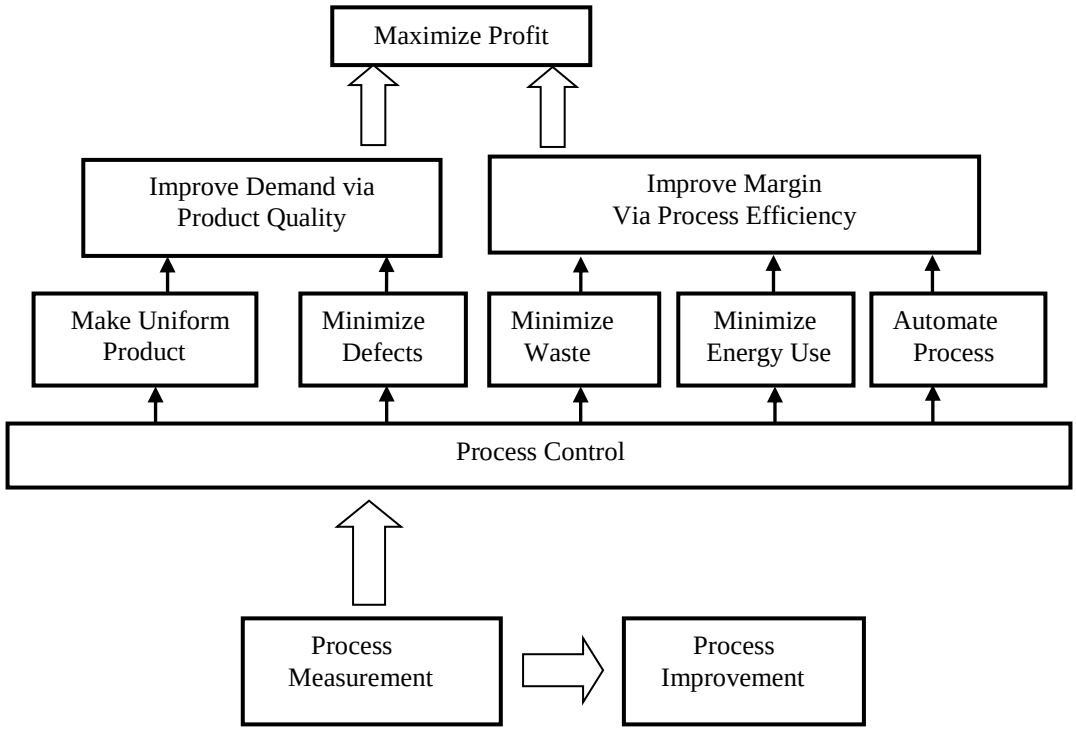

Figure 1.1: Process measurement is crucial to plant operation and profitability.

Process efficiency also has a major impact on profitability. By minimizing defective product, process control also minimizes the associated waste in raw material, effort, and energy. Automation of various operations within the process leads to lower labor costs. Finally, process measurement and control can reduce energy costs by running mills, mixers, and other energy-intensive devices. It is not uncommon to realize savings of more than 15% in energy and maintenance costs on such equipment. The resulting increase in asset productivity is also important when the plant is running near capacity. By running an efficient process, the company can maximize its profit margin on the product [2].

Since most processes operate entirely within closed metal vessels, the operator relies on sensor data for knowledge about the state of the process. It would therefore be impossible to run most plants without sensors.

1.2 Temperature Measurement

Temperature is one of the most frequently used process measurements. Almost all chemical processes and reactions are temperature-dependent. Frequently in chemical plant, temperature is the only indication of the progress of the process. A considerable loss of product may result from incorrect temperatures. In some cases, loss of control of temperature can result in catastrophic plant failure with the attendant damage and possibly loss of life. There are many areas of industry where temperature measurement is essential. Such applications include steam raising and electricity generation, metallurgical industries, typically steel and aluminum alloys, moulding and plastics manufacturing, food industries and many different others.

1.3 Definitions

For the understanding of temperature measurement it is essential to have an appreciation of the concepts of temperature and other heat-related phenomena.

1.3.1 Temperature

The temperature of a medium is an expression of its content of thermodynamic energy. The thermodynamic energy represents the average velocity of the unarranged molecular movement in the material. To measure the temperature by a temperature sensor, then the measurement medium and temperature sensor both must reach thermal equilibrium such that both assume the same temperature. To achieve this, the following 3 conditions must be fulfilled:

1- The bodies must not exchange heat with external or internal sources.

2- The bodies must be in mutual balance.

3- The bodies have had thermal contact through sufficiently long time [3].

Temperature is therefore related to the kinetic energy of the molecules at a localized region in a body; however, these kinetic energies cannot be measured directly and the temperature inferred. To circumvent this difficulty, the International Practical Temperature Scale (IPTS) has been defined in terms of the behavior of a number of materials at thermodynamic fixed points [4].

1.3.2 The International Practical Temperature Scale (IPTS)

The international Practical Temperature Scale (IPTS) is based on six primary fixed points that cover the temperature range from -183 °C(-297 °F) to 1065°C (1949°F) and other secondary fixed points each of which corresponds to an equilibrium state during a phase transformation of a particular material. Between the fixed points (both primary and secondary), the temperature is defined by the response of specified temperature sensors and interpolation equations. The scale is divided into four ranges with the sensors, fixed points, and temperature span as indicated in Table1-1 [4].

Table 1-1: Temperature Range, Sensors, and Interpolation Equations for the International Practical Temperature Scale

Temperature range (°C)	**Sensor**	**Fixed Point**	**Equation**
-190 to 0	Platinum thermometer	Oxygen, ice, steam, sulfur	Reference equation
0 to 660	Platinum thermometer	Ice, steam, sulfur	Parabola
660 to 1063	10% rhodium platinum	Antimony, silver, gold	Parabola thermocouple
Above 1063	Optical pyrometer	羊	Planck's Law

1.3.3 Thermal Expansion

Themal expansion of solids is defined in terms of the coefficient of linear expansion α which is defined as the increase in length per unit length when the temperature is raised by 1 K:

$$l_t = l_0 + l_0.\alpha t \quad (1.1)$$

where l_0 = the initial length at temperature 0 °C

l_t = the length when the temperature is raised to t °C

1.3.4 Radiation

Radiation is the direct transfer of heat (or other form of energy) across space. Thermal radiation is electromagnetic radiation and comes within the infrared, visible and ultraviolet regions of the electromagnetic spectrum. In this thesis, the temperature measurement instrument that has to do with radiation heat transfer is the optical pyrometer.

1.3.5 Sensor Accuracy

A measure of how closely the sensor output approximates the true value of the measured variable, the temperature in this case [5].

1.3.6 Sensor Resolution

The smallest increment in the value of the measured variable (temperature) that results in detectable increment in the output [5].

1.3.7 Sensor Stability

Sensor stability is the ability of the sensor to maintain and reproduce its specific resistance-temperature characteristic for long periods of time within its specified temperature range of operation. The degree of thermometer stability is expressed in terms of drift, or more

simply sensor stability is the ability of the sensor to maintain R vs. T over time as a result of thermal exposure [6].

1.3.8 Sensor Repeatability Performance

Repeatability performance is closely related to stability. Repeatability is defined as the conformity of consecutive temperature measurements for an individual test thermometer at selected temperatures within its specified temperature range of operation, or more simply sensor repeatability is the ability of the sensor to maintain R vs. T under the same conditions after experiencing thermal cycling throughout a specified temperature range [6].

1.3.9 Sensor's Drift

It is an undesirable change in resistance over a period of time which is unrelated to the actual operating temperature. Usually, maximum drift is experienced by the sensor at high temperatures [6].

1.3.10 Sensor's Sensitivity

Is the incremental ratio of the sensor's output (y) to the input temperature (x):

$$S = \Delta y/\Delta x \tag{1.2}$$

1.3.11 Sensor Hysteresis

Difference in the output of the sensor for a given input value of the measured temperature when the measured temperature is reached from two opposite directions, i.e. during heating and during cooling you do not reach the same value of temperature.

1.3.12 Nonlinearity Behavior of Sensor

A measure of deviation from linearity of the sensor output.

1.3.13 Sensor's Operating Range

The range of input variable (temperature) that produces a meaningful sensor output.

1.3.14 Noise

Random fluctuation in the value of the measured temperature that causes random fluctuation in the output. Noise at the sensor output is due to either internal noise sources, such as resistors at finite temperatures, or externally generated mechanical and electromagnetic fluctuations. The external noise will become more important as the transducer size is made progressively smaller.

1.3.15 Sensor's Response Time

Response time for a sensor is normally measured by the thermal time constant τ. Thermal time constant is the 63.2% response to a step-function change in the sensor temperature when the power dissipated in the sensor is negligible [7]. A temperature sensor's response time is a function of the following characteristics:

1- Dimension of sensor or its size
2- Sensor's construction and encapsulation
3- Heat transfer ability between sensor and medium. Heat transfer between sensor and liquid medium is much easier than with gaseous medium and hence response time is faster.
4- Static or dynamic medium
5- Production method of sensor

1.3.16 Self-Heating of Sensor

The current that measures sensor resistance in RTD and thermister thermometers also heats the sensor. This is known as Joule heating effect. Because of this effect, the sensor's

indicated temperature is somewhat higher than the actual temperature. This inconsistency is called self-heating error. Self-heating errors, which are dependent on the application, can range from negligible values to 1°C. The greatest heating errors occur because of poor heat transfer between the sensing element and application, or excessive current used in measuring resistance.

1.3.17 Environmental Parameters

By environmental parameters we mean all the external variables such as pressure, humidity, and vibration that affect the performance of the sensor. It should be emphasized that a parameter is considered as an environmental parameter only if it is not the one to be sensed.

1.4 Thesis Objective

The objective of this thesis is to apply the AHP method in selecting the appropriate temperature sensor from among several alternative sensors for a particular industrial application. It provides the underlying mathematical work and a computerized tool for the selection process. This study can be considered a new addition to the multitude of AHP applications and opens the door to similar studies conducted in the field of sensors selection.

1.5 Significance of the Work

On one hand, no single study was found upon literature survey in the field of sensor selection using AHP methodology, and here comes to the fore the significance of this work of applying principles of AHP methodology in the sensor selection process. Furthermore, the computer software proposed by the thesis is laying a helping hand for those interested

in selecting between different temperature sensors. The software is versatile and contains seven sensor categories. It also comprises three industrial applications and numerous selection cases are possible by means of the software. Moreover, results easily obtained by the software can be utilized in further analysis such as conducting sensitivity analyses.

1.6 Organization of the Thesis

This thesis consists of six chapters. The first chapter is an introduction chapter, defining the abstract of the thesis in addition to the objective of the study and some definitions relevant to sensors science and essential for the well understanding of sensors features and characteristics. Chapter 2 tackles the different sensor categories that are used in the computer software in addition to listing main advantages and disadvantages of each category that make it more suitable or unsuitable for use in a particular industrial application . It should be noted that temperature sensors employed in industry and referenced in books are much more than the ones presented here. Nevertheless, the software is easily expandable to other sensors and applications. Chapter 3 narrates the basic Analytical Hierarchy Process (AHP) method theoretical background. Chapter 4 is a description of the proposed computer software and how the AHP method is implemented in the software. Chapter 5 is a practical implementation of a three sensors: the thermocouple, the thermister, and the RTD case study to the computer software in the automotive catalytic converter application in addition to Sensitivity Analysis employed to the software to test its solidity and ruggedness towards variations in system inputs. And finally, chapter 6 presents discussion of the results in chapters 4 and 5 in addition to concluding remarks and future work .

Chapter Two Temperature Measurement Sensors Used in Thesis

2.1 List of Different Sensor Categories Used in Thesis

Techniques for temperature measurement are very varied. Table 2-1[8] is a summary of the used measuring sensors in this thesis in the range quoted.

Table 2-1: General specifications of the sensors used in thesis for temperature Measurement.

Range (K)	Technique	Application	Resolution (K)
20-2700	Thermocouple	General-purpose	1.0
73-1123	K-type thermocouple (Nickel-Chromium versus constantan)	General-purpose	1.0
4-1300	Thermister	Laboratory Industrial	0.001 0.1
15-1150	Platinum resistance thermometer	Standard Industrial	0.00001 0.1
130-950	Liquid-in-glass	General-purpose	0.1
130-700	Bimetal	Industrial	1-2
950-3300	Optical (Disappearing filament) Pyrometer	Industrial	0.1
200-950	LCD thermometer	Industrial	0.001

2.1.1 Liquid-in-Glass Thermometers

Although several liquids are employed in the liquid-in-glass type thermometers, the one tackled in this thesis is mercury-in-glass thermometer.

2.1.1.1 Mercury-in-Glass Thermometers

This thermometer consists simply of stem of glass tubing having a very small uniform bore. At the bottom of the stem there is a thin-walled glass bulb. The bulb may be cylindrical or spherical in shape, and has a capacity many times larger than that of the bore of the stem. The bulb and bore are completely filled with mercury, and the open end of the

bore is sealed off either at high temperatures or under vacuum, so that no air is included in the system.

Drawbacks of mercury-in-glass thermometer

1- It has a fairly large thermal capacity, with glass being not a very good conductor of heat. Therefore, this class of temperature sensors have definite thermal lag, i.e., it will require a definite time to reach the temperature of its surroundings. This time should be allowed for before any reading is taken. In this regard, if the sensed temperature is varying rapidly, then the thermometer may never indicate the temperature accurately, particularly if the tested medium is gas.

2- Glass thermometers used in industry are usually protected by metal sheaths. These metal sheaths may conduct heat to or from the neighborhood of the bulb and cause the thermometer to read either higher or lower according to the actual conditions prevailing.

3- For particularly cheap mercury-in-glass thermometers, large errors may be introduced by changes in the size of the bulb due to aging. This occurs during manufacture of the thermometer when glass is heated to high temperatures, and upon cooling does not contract to its original size. Thus a long time after it has been made it contracts very slowly so that the original zero mark is too low on the stem, and thus the thermometer reads higher than the actual temperature. In order to minimize error introduced in this case, thermometers are annealed during manufacturing by baking them for several days at temperatures above that which they will be required to measure, and then cooled slowly over a period of several days [8].

2.1.1.2 Mercury-in-Glass Thermometers for Measuring High Temperatures

Mercury boils at 357 °C at atmospheric pressure. To extend this range, the top of the thermometer bore is enlarged into a bulb having a capacity 20 times that of the bore of the stem. This bulb, together with the bore above mercury, are then filled with nitrogen or carbon dioxide at sufficiently high pressure to prevent mercury boiling. A pressure of 20 bars is required to extend the range to 550 °C [8].

2.1.2 Bimetal Strip Thermometer

Bimetal strips are fabricated from two strips of different metals with different coefficients of thermal expansion bonded together to form, in the simplest case, a cantilever. Typical metals are brass and Invar. As the temperature rises the brass side of the strip expands more than the Invar side, resulting in the strip curling. The compound strip is formed by riveting or welding two layers of metals, chosen so as to have very different values of coefficient of linear expansion.

2.1.3 Platinum Resistance Thermometers (PRTDs)

One common way to measure temperature is by using Resistive Temperature Detectors (RTDs). These electrical temperature instruments provide highly accurate temperature readings: simple industrial RTDs used within a manufacturing process are accurate to ± 0.1°C, while Standard Platinum Resistance Thermometers (SPRTs) are accurate to ± 0.0001°C. [7]

All metals are electrical conductors that show resistance to the passage of electric current. The proportional relationship of electrical current and potential difference is given by Ohm's law:

$$R = E / I \tag{2.1}$$

Where R = the electrical resistance in ohms

E = potential difference in volts

I = current in amperes

The resistance of a conductor is proportional to its length and inversely proportional to its cross-sectional area, i.e.

$$R = \rho . L/A \quad (2.2)$$

Where R = the resistance of the conductor

ρ = resistivity of the material

L = the length of the conductor

A = cross-sectional area of the conductor

Units of ρ are ohms. meter. The resistivity of a conductor is temperature dependent and changes in a known and predictable manner, depending on the rise or fall in temperature. As temperature rises, the electric resistance of the metal increases. As temperature drops, electric resistance decreases. RTDs use this characteristic as a basis for measuring temperature. The sensitive portion of an RTD, called an element, is a coil of small-diameter, high-purity wire, usually constructed of platinum, copper, or nickel. This type of configuration is called a wire-wound element. Another configuration; thin-film element, with thin-film of platinum deposited on a ceramic substrate is also present [7].

The temperature coefficient of resistivity α is positive for metals, that is, resistance increases with temperature, and for semiconductors the temperature coefficient is negative. The temperature coefficient α is a measure of the sensitivity of the resistance thermometer. It is also an expression of the mean value for the relative change in resistance per °C between 0 and 100°C. As a general guide at normal ambient temperatures the coefficient of resistivity of moat elemental metals lies in the region of 0.35 per cent to 0.7 per cent per Kelvin [8]. Table 2.5 shows the resistivity and temperature coefficients for a number of common metals: both elements and alloys.

This study was concerned with Platinum resistance thermometer. Platinum, a noble metal, is the best metal for RTD elements for three reasons:

1- It follows a very linear-to-temperature relationship
2- It follows its resistance-to-temperature relationship in a highly repeatable manner over its temperature range.
3- It has the most stable resistance-to-temperature relationship over the largest temperature range -184.44 °C to 648.88°C, although Platinum thermometers can be used for temperatures up to 800 °C and down to -253 °C. Platinum is not the most sensitive metal; however, it is the metal that offers the best long term stability and repeatability, RTDs can be removed from service and recalibrated for verifiable accuracy and checked for any possible drift. The accuracy of an RTD is significantly better than that of a thermocouple within RTD's normal temperature range of -184.44 °C to 648.88°C [7].

In operation, the measuring instrument applies a constant current through the RTD. As the temperature changes, the resistance changes and the corresponding change in voltage is measured. There are three main classes of Platinum Resistance Thermometers (PRTs):

1- Standard Platinum Resistance Thermometers (SPRTs).
2- Secondary Standard Platinum Resistance Thermometers (Secondary SPRTs).
3- Industrial Platinum Resistance Thermometers (IPRTs). Table 2-2 [7] presents information about each.

Table 2-2: Characteristics of different classes RTDs.

Probe	Basic application	Temperature (°C)	Cost	Probe style [a]	Handling
SPRT	calibration of secondary SPRT	-200 to 1000	$ 5000	I	Very fragile
Secondary SPRT	Lab use	-200 to 500	$ 700	I, A	Fragile
Wirewound IPRT	Industrial field use	-200 to 648	$ 60-$ 180	I, S, A	Rugged
Thin-film IPRT	Industrial field use	-50 to 260	$ 40-$ 140	I, S, A	Rugged

[a] I = immersion; A = air; S = surface.

2.1.4 Thermister Thermometers

A themister is a thermally sensitive resistor whose primary function is to exhibit a change in electric resistance with a change in body temperature. a thermister is a ceramic semiconductor. Depending on the type of material used, a thermister can have either a large positive temperature coefficient of resistance (PTC device) or a large negative temperature coefficient of resistance (NTC device). The focus in this thesis is on NTC thermistors.

NTC thermisters consist of metal oxides such as the oxides of chromium, cobalt, copper, iron, manganese, nickel, and titanium. Such units exhibit a decrease in electric resistance with an increase in temperature. The resistance–temperature characteristics of NTC thermisters are nonlinear.

Because of its nonlinear resistance–temperature characteristic, the temperature coefficient of resistance of an NTC thermister changes with temperature. Depending on the material used, the temperature coefficient at 25°C typically is in the range of –3 to –5% $°C^{-1}$ [7].

The thermal time constant τ is the 63.2 % response to a step-function change in the thermister temperature when the power dissipated in the thermister is negligible.

Two major categories of NTC thermisters exist. Bead-type thermisters have platinum alloy lead wires sintered into the body of the ceramic. Chip, disk, surface-mount, flake, and rod-

type thermisters have metallized surface electrodes. Table 2-3 shows thermal properties for some thermister types [7].

Table 2-3: Thermal Properties of Thermisters with Metallized Surface Electrodes.

Style	Diameter (mm)	Dissipation constant (mW °C^{-1})	Time constant (s)
Chip or disk in glass diode package	2	2-3	7-8
Interchangeable epoxy coated chip or disk	2.4	1	10
Disk with radial or axial leads	2.5	3-4	8-15
Disk with radial or axial leads	5.1	4-8	13-50
Disk with radial or axial leads	7.6	7-9	35-85
Rod with radial or axial leads	1.8	4-10	35-90
Rod with radial or axial leads	4.4	8-24	80-125

It should be noticed, however, that these values of thermal properties of these units depend on the environment and the measurement medium in which the sensor will be used.

2.1.5 Thermocouple Thermometers

Thermocouple thermometers are self-generating sensors, i.e. they do not need external source of power to drive them. They remain the most generally used sensors for thermometry because of their versatility, simplicity, and ease of use. Any pair of electrically conducting and thermoelectrically dissimilar materials coupled at an interface is a thermocouple. The legs are thermoelements. The Seebeck effect produces a voltage in all such thermoelements when they are not at a uniform temperature. Any electric interface between dissimilar electric conductors is a thermoelectric junction.

The Seebeck effect happens when a closed circuit is formed of two metals, and the two junctions of the metals are at different temperatures, then an electric current will flow round the circuit, as shown in Figure 2.1, which shows a wire of two different metals such

as iron and copper. If one junction remains at room temperature, while the other is heated to a higher temperature, a current is produced which flows from copper to iron at the hot junction, and from iron to copper at the cold one.

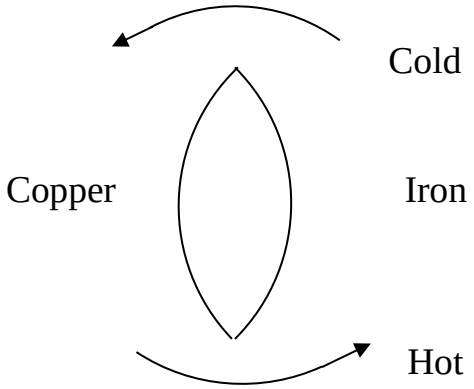

Figure 2.1: Simple thermocouple.

The most commonly used industrial thermocouples are identified for convenience by type letters. The main types, together with the relevant British Standard specification and permitted tolerance on accuracy, are shown in Appendix I [8]. Also shown are their output e.m.f.s with the cold junction at 0 °C.

2.1.5.1 Thermocouple Materials

Broadly, thermocouple materials divide into two arbitrary groups based upon cost of the materials, namely, base metal thermocouples and precious metal thermocouples.

Base Metal Thermocouples

The commonly used base metal thermocouples are types E, J, K, and T. Of these J and K are probably the most usual ones. They have a high e.m.f. output and type K is reasonably resistant to corrosion. Type T has a slight advantage, where the temperature measurement points are very remote from the instrumentation, because one conductor is copper the overall resistance of the circuit can be lower than for other types. A comprehensive list of the industrially available thermocouples alongside their designations and color codes can be found in Appendix II [3].

Precious Metal Thermocouples

Thermocouples types B, R, and S clearly carry a considerable cost penalty and normally are only used when essential for their temperature range or their relatively high resistance to chemical attack. Their temperature top limit is 1500 °C for continuous use or 1650 °C for intermittent, spot reading, applications. This compares with 1100 °C continuous and 1300 °C intermittent for type K.

Errors in type R and S thermocouple readouts result from strain, contamination and rhodium drift. The effect of strain is to reduce the e.m.f. resulting in low readings.

Noble metal thermocouples may be used for measuring cryogenic temperatures. Iron-gold/nickel-chromium may be used for temperatures from 1 K to 300 K. For high temperature work special thermocouples have been developed, tungsten 5 per cent rhenium/tungsten 20 per cent rhenium for use in hydrogen, vacuum and inert gas atmospheres up to 2320 °C [8].

2.1.6 CD Semiconductor Thermometer

This thermometer chosen in this thesis is basically a wire wound platinum resistance temperature sensor, but with higher level of accuracy than normal RTDs due to the use of a calibration algorithm embedded in the instrument. It is an LCD display temperature sensor a resolution of 0.001°C. It is perfect for metrology applications where extreme accuracy is required and as a calibration reference thermometer.

2.1.7 Optical Disappearing Filament Thermometers (Pyrometers)

Optical pyrometers are one type of radiation thermometers and are considered to be the most accurate radiation thermometers for temperature range 700°C to 3000°C [7]. The operating principle of optical pyrometer is based on Planck's law which states that

intensity and color of a surface changes with temperature. This type of thermometers is suitable for non-contact measurement of medium temperature. The idea behind the design of the pyrometer instrument is to balance the radiation from an object (tungsten filament) having a known temperature against unknown temperature from a target. The pyrometer has a lens through which the operator views the target when an image of a tungsten filament is superimposed on the image of the target. The filament is warmed up by an electric current to glow. The operator views the target through the eyepiece and manually adjusts the heating current to the level when an image of the glowing filament visible in the foreground disappears — that is, when both the target and the filament have the same brightness and color.

The measurement accuracy of an optical pyrometer is typically ±5K between 800°C and 1300°C and ±10K between 1300°C and 2000°C [8]. Modern industrial pyrometers are accurate to ± 0.1°C and this figure was adopted in the computer software proposed by the thesis.

2.2 Selection of Temperature Sensors

2.2.1 Important Criteria for the Selection of Temperature Sensors

Selection of sensors in practice should be based on a total evaluation, in which the following parameters are considered (see Appendix II):

1- Temperature range

2- Accuracy

3- Response time

4- Sensitivity

5- Corrosion conditions and resistance

6- Breaking down due to wear and tear

7- Interchangeability

8- Variations in temperature – temperature shock

9- Pressure conditions

2.2.2 Relative Merits for Each Category

1- Thermistors provide high resolution, have the widest range of applications, are the most sensitive, and are low cost, but are nonlinear and have limited temperature range.

2- Thermocouples have the highest temperature range and are durable for high vibration and high shock applications, but require special extension wire of the same material as the thermocouple itself.

3- RTDs are the most linear and are highly accurate and stable, but they are large size and expensive.

4- Advantages and Disadvantages of Bimetal Strip thermometers

Advantages

a- Direct interface with application for fast response

b- No additional circuitry/components required

c- High current carrying capacity

d- Wide operating temperature range: 130 K to 700 K

Disadvantages

a- Less accurate than most electronic-based systems

b- Larger size than electronic-based systems

c- Creepage-type device cannot interface with electronic components

d- Can fail “closed” at end of life

e- Advantages and disadvantages of optical pyrometer thermometer

Advantages

a- Allows for non-contact measurement of moving objects or hazardous materials.

b- Can be used in conjunction with fiber optics for remote sensing

c- Typical temperature range 270 K to 5000 K

d- Accuracy is typically ±5K between 800°C and 1300°C and ±10K between 1300°C and 2000°C.

Disadvantages

a- Accuracy can be affected by surface finish

b- Field of view must be matched to target size

c- Ambient temperature can affect readings

d- Wavelength filter must be matched to the application

e- Higher cost ($200+) can be even higher if control circuitry is required

f- Calibration can be difficult and costly

g- Dust, gas, or other vapors in the environment can affect the accuracy of the system.

2.2.3 Application-Related Issues

The following are application-related issues that have to be taken into mind in selecting temperature sensors.

2.2.3.1 Contact or Non-contact Sensing?

Does the application need contact or non-contact sensing? If the application is moving or if physical contact is not practical due to contamination or hazardous material issues, Optical Pyrometry is the technology of choice.

2.2.3.2 What is the Temperature Range?

What is the temperature range the sensor is required to measure or control? Thermocouples have the broadest temperature range, 20 K to 2700 K. Depending on design and material, thermsisters have a usable range of 4 K to 500 K and this range can be extended to 800 K. Bi-metal thermostats can handle temperatures from 130 K to 700 K. For cryogenic temperatures, RTDs are capable of approaching absolute zero (0K). Maximum temperature is 1000 K. For non-contact Pyrometers, temperatures above 973K (700°C) and up to 5000 K are attainable.

2.2.3.3 What is the Rate of Temperature Change?

What is the rate of temperature change of the application? For applications where the rate of temperature change is rapid (>1.0°C/minute), the mass of the sensor may become an issue. For extremely rapid changes, sensor mass should be kept to a minimum to allow it to more accurately track the change of the application.

2.2.3.4 Tolerance

How tightly do you need to control or monitor the temperature? For certain processes applications involving chemical reactions, tolerances of ±0.1°C or less may be required. For any application requiring tolerances of less than ±1.7°C, an electronic system will be required. Silicon, RTD, thermocouple or thermister-based systems can all be designed to maintain these extremely tight tolerances. Typically RTDs will provide the greatest overall accuracy.

2.2.3.5 Cost

How important is total system cost in the selection of the sensor? In high-end applications costing thousands of dollars like, say, automotives and chemical reactions applications the cost of the temperature sensor is typically insignificant. For this reason, the selection is

based on required system accuracy. An accuracy of ±0.1°C or less will require a sophisticated (and expensive) alternative.

Chapter Three Theoretical Background for the Analytic Hierarchy Process (AHP) Method

3.1 Introduction

The analytic hierarchy process is a multi-attribute decision-making tool mostly used when a decision maker is faced with a problem involving multiple objectives. It can also be used to handle problems involving uncertainty. The method, which was developed by Thomas Saaty has been very widely applied to decision making problems in areas like: planning, economics, energy policies, material handling and purchasing, trading strategies, project and businesses selection, budget allocations and forecasting. Alongside the analytical procedure of the method, a user-friendly computer package, EXPERT CHOICE has been developed to support the method.

In a typical decision making problem, the decision maker will have an objective or multiple objectives that he wants to fulfill and a group of candidate alternatives that are to be assessed in terms of the best alternative that meets that objective. These alternatives will have certain attributes and sub-attributes (criteria and sub-criteria) qualifying these alternatives. The alternatives, criteria and sub-criteria, and the objective are linked in a hierarchal structure and each forms a hierarchal level. Each component at a particular level is relatively pair-wise compared with its sister components with respect to the immediate upper level and weights of all components are determined and aggregated for upper levels. The final outcome of the method is a fractional score for each alternative representing its relative preference towards the objective.

The method widespread use may be considered as an evidence of the method power and reliability among decision makers in dealing with different decision problems. However,

critics have questioned its axiomatic basis and the degree it can lead to a reliable and true representation of the decision maker's preferences [9].

3.2 Description of the Method

Once the decision maker has identified the objective of his problem, the alternatives that have to be compared towards the goal and the criteria and sub-criteria governing the comparison process between the alternatives, then the application of the method becomes easy and can be described in terms of these steps:

Step 1: Set up the decision hierarchy. The decision hierarchy will be made of the objective level, the criteria level, the sub-criteria level, and finally the alternatives level.

Step 2: Perform the pair-wise comparisons of the alternatives, sub-criteria, and criteria. This is done to determine the relative importance of the criteria and sub-criteria and also to determine how well the alternatives score on each sub-criteria and criteria. This is done starting from the alternative level and hierarchically through sub-criteria level and criteria level up to reach the objective level.

Starting from the alternative level, the relative importance of one alternative over the other with respect to the same sub-criteria in the decision hierarchy can be determined using Saaty's scale [10] (Table 3-1). According to Saaty, the relative weight of alternative i compared to alternative j with respect to the same sub-criteria can be obtained from a 9-point scale and assigned to the (i , j)th position of the pair-wise comparison matrix or judgment matrix.

Table 3-1: The pair-wise comparison scale (Saaty, 1980)

Intensity of importance	Definition
1	Equally important
3	Weakly more important
5	strongly more important
7	very strongly more important
9	extremely more important
2, 4, 6, 8	Intermediate values between two adjacent judgments

In a more general form, let $\boldsymbol{A_1}, \boldsymbol{A_2}, \dots, \boldsymbol{A_n}$ be the set of stimuli (these stimuli can be the alternatives, the sub-criteria , or he criteria), were $\boldsymbol{n}$ refers to the number of these stimuli. The quantified judgments on pairs of stimuli $\boldsymbol{A_i}, \boldsymbol{A_j}$ are represented by the judgment matrix $\boldsymbol{A}$:

$$\boldsymbol{A} = [\boldsymbol{a_{ij}}], \qquad \boldsymbol{i}, \boldsymbol{j} = 1,2, \dots, \boldsymbol{n}. \tag{3.1}$$

The comparison of any two criteria *Ci* and *Cj* is made using the questions of the type: of the two criteria *Ci* and *Cj* which is more important and by how much. Saaty's scale is used to transform verbal judgments of relative preference of one alternative to the other into numerical quantities representing the values of $\boldsymbol{a_{ij}}$. The entries $\boldsymbol{a_{ij}}$ are governed by the following rules:

$$\boldsymbol{a_{ij}} > 0 , \qquad \boldsymbol{a_{ji}} = 1/\boldsymbol{a_{ij}} , \qquad \boldsymbol{a_{ii}} = 1 \text{ for all } \boldsymbol{i}. \tag{3.2}$$

Thus, the reciprocal of the assigned value is automatically assigned to the (*j, i*)th position in the judgment matrix.

Step 3: Transform the comparisons set in the previous step into weights corresponding to the different criteria, sub-criteria and alternatives and check the consistency in decision maker's comparisons in terms of consistency index and consistency ratio.

After the pair-wise comparison matrices (***A***s) have been numerically established , the next step is to recover the weights of the different alternatives from these comparison matrices. This can be done by solving for the eigenvectors of the pair-wise comparison matrices. Let ($W_1, W_2, \dots, W_n$) be weights of the alternatives relating to a certain sub-criteria. Consider the following equation:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ . & . & \dots & . \\ . & . & \dots & . \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} W_1 / W_1 & W_1 / W_2 & \dots & W_1 / W_n \\ W_2 / W_1 & W_2 / W_2 & \dots & W_2 / W_n \\ . & . & \dots & . \\ . & . & \dots & . \\ W_n / W_1 & W_n / W_2 & \dots & W_n / W_n \end{pmatrix} \quad (3.3)$$

for a perfectly consistent decision maker, and the following equation:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ . & . & \dots & . \\ . & . & \dots & . \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \approx \begin{pmatrix} W_1 / W_1 & W_1 / W_2 & \dots & W_1 / W_n \\ W_2 / W_1 & W_2 / W_2 & \dots & W_2 / W_n \\ . & . & \dots & . \\ . & . & \dots & . \\ W_n / W_1 & W_n / W_2 & \dots & W_n / W_n \end{pmatrix} \quad (3.4)$$

for not perfectly consistent decision maker. Let us multiply both sides of the equation (3) with the weights vector $W = (W_1, W_2, \dots, W_n)$, then we have:

$$AW^{\mathbf{T}} = \Delta W^{\mathbf{T}} \quad (3.5)$$

This is a system of homogenous linear equations, where Δ is an unknown number and $W^{\mathbf{T}}$ is an unknown n-dimensional column vector [11], for any number Δ, equation (5) always has the trivial solution W= (0, 0, ..., 0). It can be shown that if A is the pair-wise comparison matrix of a perfectly consistent decision maker, i.e. equation (3) applies, and we do not allow Δ = 0, then the only nontrivial solution to (5) is $\Delta = n$ and $W = (W_1, W_2,$

… , W_n). However, if the decision maker is not perfectly consistent, i.e. equation (4) applies in this case, then let Δ_{max} be the largest number for which (5) has a nontrivial solution (call it W_{max}) . Saaty verified that if the decision maker's comparisons do not deviate very much from perfect consistency, then Δ_{max} is close to n and W_{max} is close to W. Saaty's method computes W as the principal right eigenvector of the matrix A. Saaty also proposed measuring the decision maker's consistency by looking how close Δ_{max} is to n. Δ_{max} is called the maximum eigenvalue of matrix A.

Consistency index (CI) is a measure of how consistent and rational the decision maker is in his pair-wise comparisons. According to Saaty CI is defined as:

$$CI = (\Delta_{max} - n) / (n - 1) \quad (3.6)$$

Consistency ratio (CR) is defined in terms of consistency index (CI) and random index (RI) as:

$$CR = CI / RI \quad (3.7)$$

Where the random index (RI) represents the average consistency index value taken over numerous random entries of the same order matrices as matrix A. Values of RI for the appropriate value of n are given in Table 3-2 [11].

Table 3-2: Random Index Values

n	RI	n	RI
2	0.0	7	1.32
3	0.58	8	1.41
4	0.90	9	1.45
5	1.12	10	1.51
6	1.24		

An index of zero value refers to perfect consistency, while an index of more than zero refers to inconsistency in decision maker's comparisons. For example, suppose the decision maker's response imply that alternative 1 is twice as important as alternative 2, while alternative 2 is judged to be three times as important as alternative 3. To be perfectly consistent the decision maker should judge that alternative 1 is six times more important than alternative 3. Any other response will lead to an index greater than zero. Saaty recommends that inconsistency should only be a concern if the index exceeds 0.1, in which case the comparisons should be reexamined. But he stresses that minimizing inconsistency should not be the main goal of the analysis since a set of erroneous judgments about importance and preference may be perfectly consistent, but they will not lead to the 'best' decision.

A simple method [11] is used to approximate Δ_{max} , W_{max} , CI and CR which comprises the following steps:

1- Find the normalized matrix A_{norm}. This can be done by dividing each entry for each of A's columns by the sum of all entries in the same column.

2- An approximate estimate of the weights vector W is by estimating each entry W_i as the average of the entries in row i of A_{norm}.

3- An approximate estimate of Δ_{max} is:

$$\Delta max = \sum_{i=1}^{i=n} \frac{ith\,entry\,in\,AW^T}{ith\,entry\,in\,W^T} \qquad (3.8)$$

4- The consistency index,

CI = (result in step 3 - n) / (n - 1) (3.9)

5- The consistency ratio, CR = CI / RI (3.10)

The consistency ratio will enable the decision maker to examine how consistent he is with his pair-wise comparisons and to examine how robust the provisional decision in the previous step is to changes in the ratings of importance and preference.

Step 4: aggregate these weights up to obtain scores for the different alternatives towards the final objective and make a provisional decision.

Step 5: Perform the sensitivity analysis. Sensitivity analysis is important to examine how sensitive the preferred alternative is to changes in the judgments made by the decision maker. The decision maker may provide rough judgments or he may be unsure exactly what judgments to provide. Sensitivity analysis can provide for the extent of change that can be made to the criteria or sub-criteria weights before the preferred alternative changes in favor of another alternative. EXPERT CHOICE is fitted with such a tool and it will be utilized for this purpose as will be shown in the next chapter.

3.3 Strengths and Weaknesses of AHP

3.3.1 Strengths [9]

1- Formal structuring of the problem. This allows complex decision problems to be broken down into sets of simpler judgments that can be more easily assessed.

2- The pair-wise comparisons are simple, which simplifies the task for the decision maker by concentrating on each small part of the problem when comparing two entities at a time. Also the method, aided with Saaty scale of relative importance fits well for verbal comparisons most likely preferred by decision makers who might have difficulty in expressing their preferences numerically.

3- Redundant assessment of the relative importance of multiple criteria allows the decision maker to check consistency in his judgments. for example, if a decision maker states that criteria A is as twice important as B, and B, in turn, is as three times as important as C, then it can be deduced that A is six times more important than C. however, by also asking the decision maker separately to compare A to C it is possible to check consistency of the judgments.

4- Versatility. The wide range of applications of the AHP is evidence of its versatility.

3.3.2 Weaknesses [9]

1- Conversion from verbal to numeric scale. The correspondence between the verbal and numeric scales is based on untested assumptions.

2- The 1 to 9 scale imposes inconsistencies. In some cases of comparisons the scale is bound in such away it enforces inconsistencies in comparisons. For example, if A is 6 times more important than B, and B is 5 important than C. then the consistent comparison of A to C is that A 30 times more important than C which is impossible.

3- New alternatives introduced into the problem can reverse the rank of existing alternatives. For example, suppose that AHP is used to choose a site for a new company, and suppose the site alternatives are: X, Y, and Z and after the use of AHP the order of preference is: 1-Y, 2-X, 3-Z. However, it was discovered that a new site W is worth considering, when introduced into the problem and AHP reused on the basis of four alternatives the order of preference can very likely become: 1-X, 2-Y, 3-W, 4-Z showing reversal between Y, X alternatives though relative importance of criteria is left unchanged.

4- Number of comparisons required becomes large to handle as number of alternatives increases.

5- The axioms of the method are not founded on testable descriptions of rational behavior, as argues Dyer [12].

Chapter Four A Software Application for the Selection of Temperature Sensors Using the Analytic Hierarchy Process (AHP)

4.1 Literature Review

Previous literature indicates the massive use of AHP methodology as a multi-criteria decision making tool in selecting from among nominated alternatives in many industrial fields. However the literature survey has not revealed any research conducted specifically on the selection of temperature sensors using AHP method, and here comes to the fore the importance of this study. Vaidya and Kumar [13] conducted a research that overviewed different applications of Analytic Hierarchy Process method. In their paper, they presented a literature review of various applications of Analytic Hierarchy Process (AHP), they referred to a total of 150 application papers, of which 27 were critically analyzed. In their work, they analyzed the applications papers according to three main groups: (a) applications based on a theme, (b) specific applications, and (c) applications combined with some other methodology, with all application papers in a specific group given distribution in the form of a pie-chart. Some theme-specific applications which were mentioned in the paper used AHP in: selection, evaluation, benefit-cost analysis, resource allocation, decision making, forecasting, medicine, and QFD. Some application area-specific papers were in: social, political, manufacturing, engineering, education, industry, government, and others. And finally, the distribution of reviewed papers over the years was investigated in the form of a pie-chart. Yurdakul [14] applied AHP method as a strategic decision-making tool to justify machine tool, namely machining centers, selection. He tested AHP approach in his research based on a three-machining centre case study for Dizayn Machinery

Manufacturing and Engineering Inc., in which case the company opted to purchase new machine tools in order to reduce lead times without compromising quality and cost of its products. Analytic Hierarchy Process (AHP) method was used to combine different types of evaluation criteria in a multi-level decision structure to obtain a single score for each alternative machine tool to rank the alternatives. Analytic Network Process (ANP) method was used in the same paper to account for the calculation of the weights of criteria due to interdependencies and interrelationships that exist among the evaluation criteria. Yurdakul stated that the company management found the application and results satisfactory and implementable in their machine tool selection decisions. Pi-Fang et al [15] presented an AHP method in objectively selecting medical waste disposal firms in Taiwan based on the results of interviews with experts in the field. In their study, an appropriate weight criterion based on AHP was derived to assess the effectiveness of medical waste disposal firms. The proposed AHP-based method in the paper offered a more efficient and precise means of selecting medical waste firms than subjective assessment methods did, thus reducing the potential risks for hospitals. Che-Wei et al [16] studied and developed a manufacturing quality yield model for forecasting 12 in. silicon wafer slicing machine based on AHP framework. In their work, Exponentially weighted Moving Average EWMA control chart was presented to demonstrate and verify the feasibility and effectiveness of the proposed AHP-based algorithm, and selective analysis was performed to test the stability of the priority ranking. Okada et al [17] applied AHP to irrigation project improvement. In his study, Okada divided his work into two parts. In the first part, a questionnaire survey was distributed among irrigation professionals to determine the evaluation factors they see the most important in evaluating an irrigation project, the answers to the survey were processed by the AHP method. A hierarchy was formed with the objective of Quality of the internal process of the irrigation project and criteria of: 1- serviceability of water delivery, 2-

suitability of hardware, and 3- managing entities, and alternatives divided into two groups. Group 1 comprises: actual water delivery services at the most downstream point, actual water delivery services at the point to the individual ownership units and actual water delivery services at the point from the primary canals to the secondary canals. Group 2 comprises: primary canals and secondary canals. After applying the AHP to the answers, local weights of evaluation factors were obtained. These local weights were statistically analyzed and modeled by probability density functions. Results of the study indicated that professionals give the first priority to water delivery services and that they consider the irrigation infrastructure (hardware) of primary canals is more important than that for secondary canals.

Despite the fact that the literature survey reveals a wide array of papers applied in AHP for different applications, the survey does not reveal its use in evaluating temperature sensors alternatives, rather, research on temperature sensors was primarily concerned about proposing new temperature sensors designs that satisfy certain special demands and requirements. Vavra et al [18] proposed the use of Fe/Cr magnetoresisitive sensors at temperatures below 2 K in the milliKelvin temperature range. Hoa et al [19] studied electrical resistance drift of molybdenum disilicide ($MoSi_2$) thin film temperature sensors to study their thermoresistance, i.e. resistance vs. temperature (R-T) characteristics. Bianchi et al [20] discussed the properties, characteristics, applications and sensing principles of most of present-day integrated smart temperature sensors. A CMOS process-compatible temperature sensor developed for low-cost high-volume integrated Microsystems for a wide range of fields (such as automotive, space, oil prospecting, and biomedical applications) was also described. Han & Kim [21] developed a diode temperature sensor array (DTSA) for measuring the temperature distribution on a small surface with high resolution. The DTSA consisted of an array of 32x32 diodes (1024) for

temperature detection in an 8mmx8mm surface area and was fabricated using the very large scale integration (VLSI) technique.

This study presents a computer program built using C# programming language to perform the selection process of the best temperature sensor for a particular application from among available alternative sensors that meet the restrictions set by the program and chosen by the user, this is done by embedding the AHP method in a ready-to-use and in an easy graphical-user-interface computerized way. The proposed computer program is versatile and applicable to a multitude of temperature sensors selection situations, but it should be noted that as an example of the proposed program, the work in this paper presents a single case study in which an application is considered in the automotives industry in which three temperature sensors are being assessed and compared, these are: thermocouple, thermister and RTD thermometer. Nonetheless, the computer program is expandable and applicable to a wider range of temperature sensors selection situations with different applications and different arrays of candidate sensors.

4.2 Computer Software Description

In this thesis, the computer program that is used for the selection process of the best sensor from among different alternatives was built using Microsoft Visual Studio.NET. Starting from a C# Windows application template, a base -code project was created in which a two-page form was designed to show sensor selection based on AHP principles.

The first page in the form is used to select the application from three predefined applications: HVAC, Automotives, and Chemical Processes. In the first page also lie restrictions applicable to the mentioned applications that the user should specify such as: Temperature Range, Resolution, and Response time. Upon user's selection of the application required and restrictions pertaining to the application in the first page, the

second page tab can be pressed to list the available alternative candidate sensors which can be used in the selected application and that the user can further choose from. These available alternative sensors would appear in activated checkboxes, while those sensors that do not conform to the restrictions set and chosen by the user in the first page will automatically be shown by the system in an inactivated- checkbox mode in the second page, and thus the user cannot choose from.

In practice, the user selects the application in page one and depending on the restrictions selected some sensors will be enabled while others will be disabled on page two. The user then presses the selection button in the second page which would initiate the selection process. The results of the calculations that are automatically based on AHP method will be displayed and the final scores of the checked-in sensors will be shown from top to bottom in the same arrangement and number that the checked sensors appear on the second tab of the program. Of course the best sensor will be the one with the largest final weight while the worst choice for the application will be the one with the smallest weight. Relevant calculations of weights of sub-criteria, weights of criteria, consistency ratio, consistency index and final scores of the alternative sensors are all shown on the console provided on the second page.

4.3 AHP Application and Results from the Computer Software

In order to select the best temperature sensor from among different candidate sensors using the proposed software, five steps are performed by the user in order to achieve this. First, the user has to start up the computer program. Second, the user specifies the application on the first tab in which the sensors are to be used. Third, he or she specifies the restrictions pertaining to the application on the first tab in terms of temperature range, resolution, and the response time. Fourth, the user checks in the available candidate

sensors that the software suggests on the second tab. Fifth, the user presses the Select button on the second tab and then views the results.

The software uses the AHP method to assess the preferences of the checked sensors. The different weights in the hierarchal structure that are needed to perform the necessary calculations in the AHP method are assessed and built into the software. It is worth noting here; however, that the user can access the program and change the weights values according to his assessment of the weights or according to new expert opinions. The weights values are then used and being aggregated by the software to obtain the weight of the components in the immediate upper levels. The software then calculates the weights for the all components in the hierarchal structure, synthesizes the contribution of the components for the whole hierarchy and for all levels and displays the overall ranking scores for the different alternatives on the software console.

The software performs the consistency test in terms of consistency index and consistency ratio which can be regarded as a measure of consistency in the decision maker's comparisons and displays these indices on the same console.

4.3.1 Starting the Program

When the user opens the application, a window will open up as shown in Figure 4.1.

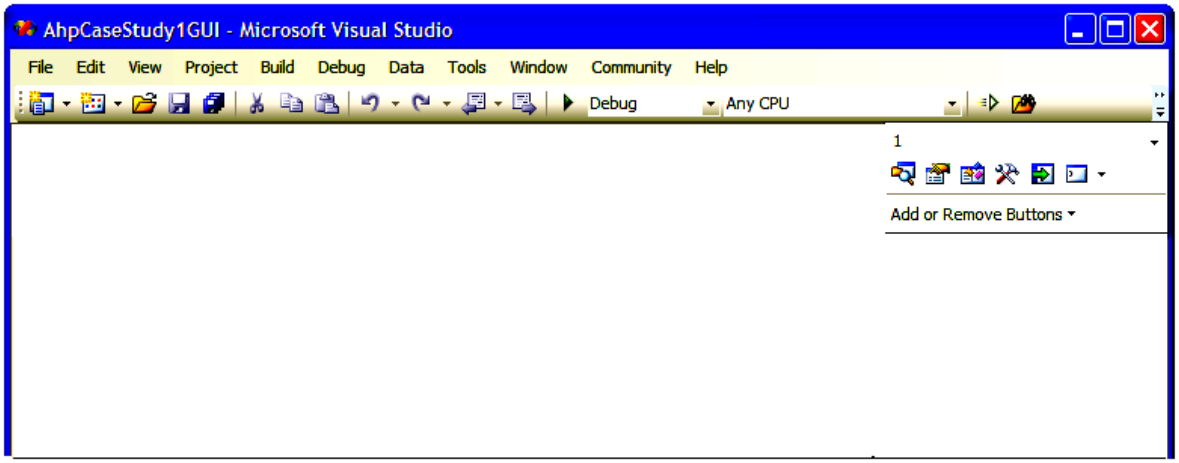

Figure 4.1: The main application window.

The Solution Explorer tab may not be visible. If so, the user presses the Solution Explorer icon at the top right of the main window to make it visible as shown in Figure 4.2

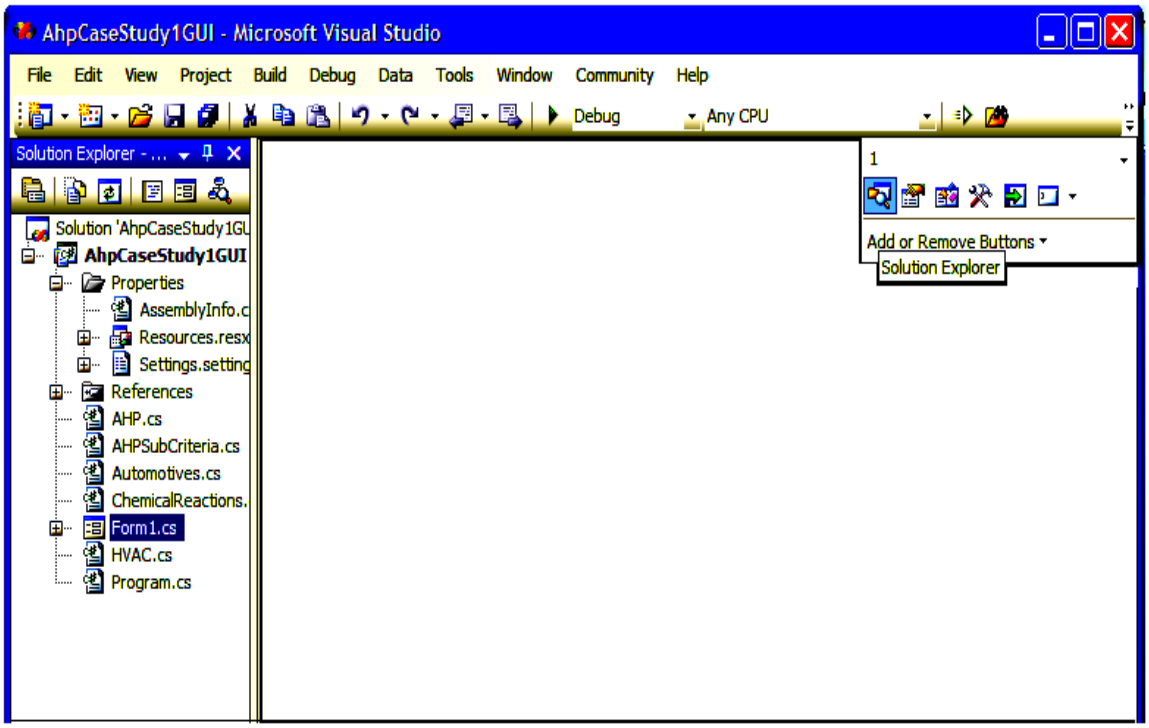

Figure 4.2: Visualizing Solution Explorer tab.

For visualizing the GUI main window from which the selection process of the best sensor will be launched, the user will first need to double click the C# file named Form1.cs in the Solution Explorer tab at the far left side of the application window and the design form of the application window will appear as shown in Figure 4.3. The user then runs the program to start it.

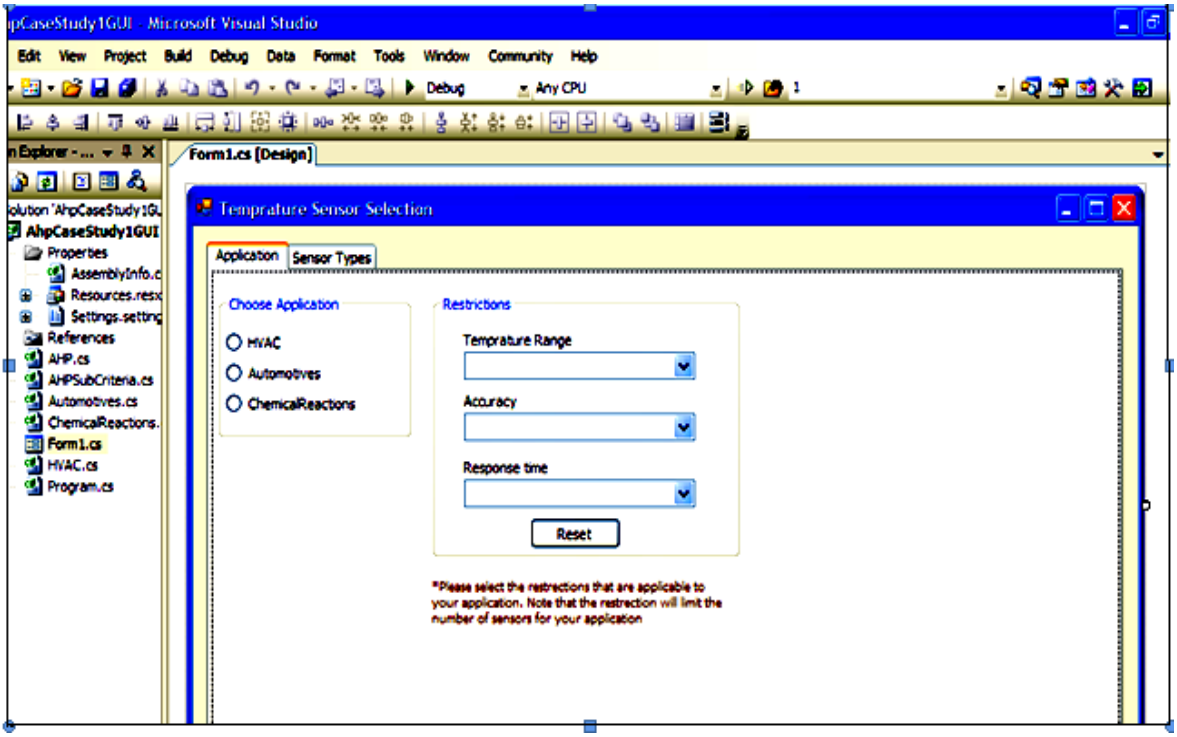

Figure 4.3: The two-tab page GUI main window.

In the GUI main window appears the two-tab page from which the user can choose the application under concern as well as the restrictions pertaining to that application in terms of Temperature Range, Resolution, and Response Time. All this can be done from the first tab. Upon completion of the first tab, the user can proceed to the next tab where the available candidate alternative sensors that meet the restrictions set in the first tab for the application under concern are listed in an activated checkbox mode, and those alternative sensors that do not conform to restrictions are disabled and shown in an inactivated mode. Figure 4.4 shows the components of the second tab.

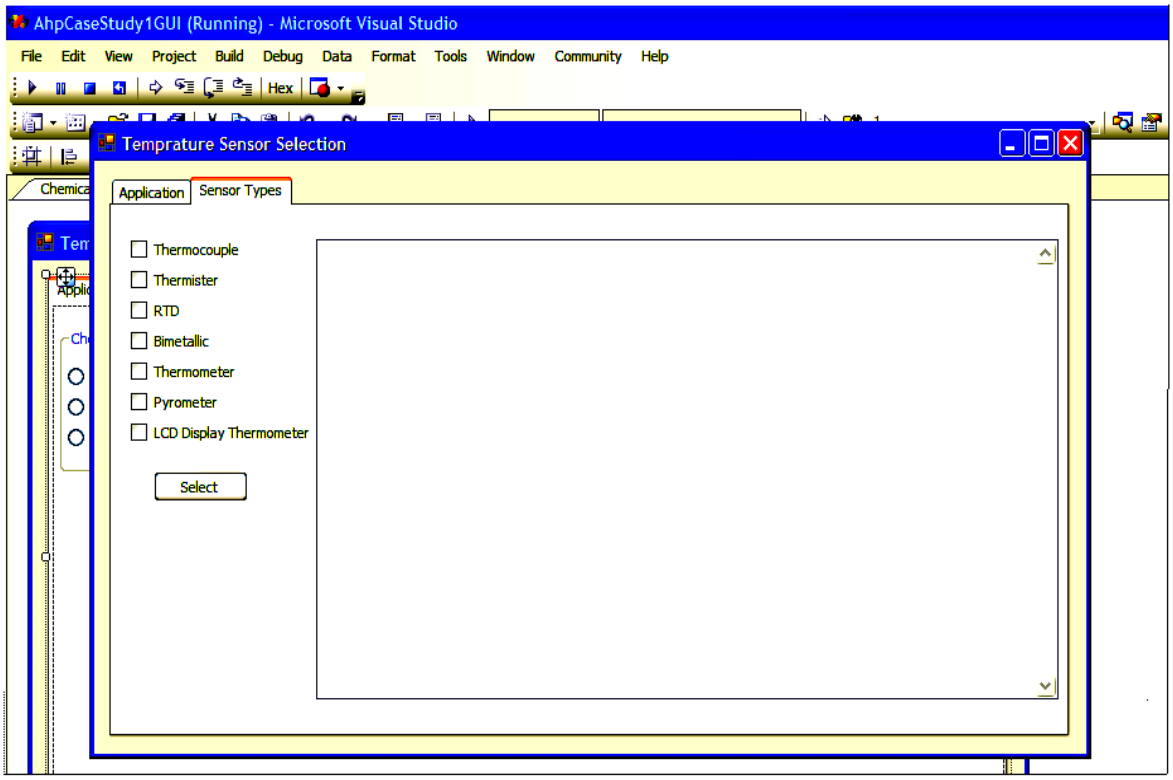

Figure 4.4: Alternative sensors in the second tab in the application main window.

4.3.2 The Evaluative Criteria and Sub-criteria

Upon literature survey in the field of sensors and sensors selection, four broad criteria were settled on, within each criterion lie multiple sub-criteria. These parent criteria and sub-criteria form the basis for the comparison between alternative sensors. Table 4-1 shows these criteria and sub-criteria which are incorporated inside the software.

Table 4-1: Criteria and sub-criteria factors used as basis for comparison between alternative sensors.

Criteria	Sub-Criteria
Static Criteria (C1)	Maximum Operating Temperature (CS1) Minimum Operating Temperature (CS2) Temperature Curve (CS3) Sensitivity (CS4) Self-Heating Issues (CS5) Long Term Stability and Accuracy (CS6) Typical Temperature Coefficient (CS7) Extension Wires (CS8) Long Wire runs from Sensor (CS9) Measurement Parameter (CS10) Temperature Measurement (CS11)
Dynamic Characteristics (C2)	Stimulation Electronics required (CS12) Existence of Maximum Sensitivity Region (CS13) Typical Fast Thermal Time Constant (CS14)
Environmental Parameters (C3)	Typical Small Size (CS15) Noise Immunity (CS16) Fragility-Durability Characteristics (CS17) High Thermal Gradient Environment (CS18) Corrosion Resistance (CS19)
Other Criteria (or Simply Others) (C4)	Point or Area Measurement (CS20) Manufacturing Variances (CS21) NIST Standards exist (CS22) Cost (CS23)

Static criterion category refers to the inherent technical characteristics or qualities of the sensor that are not time related. Dynamic category, on the other hand, refers mainly to the transient time-related characteristics of the sensor. Environmental category refers to characteristics of the sensor that are environment-related. Others category refers to other miscellaneous characteristics.

4.3.3 The Hierarchal Structure

The best temperature sensor can then be selected and evaluated by the software based on four evaluation criteria, twenty –three evaluation sub-criteria. Figure 4.5 shows the hierarchal structure for the temperature sensor selection problem. The software is

programmed to automatically perform calculations based on the hierarchal structure shown in Figure 4.5.

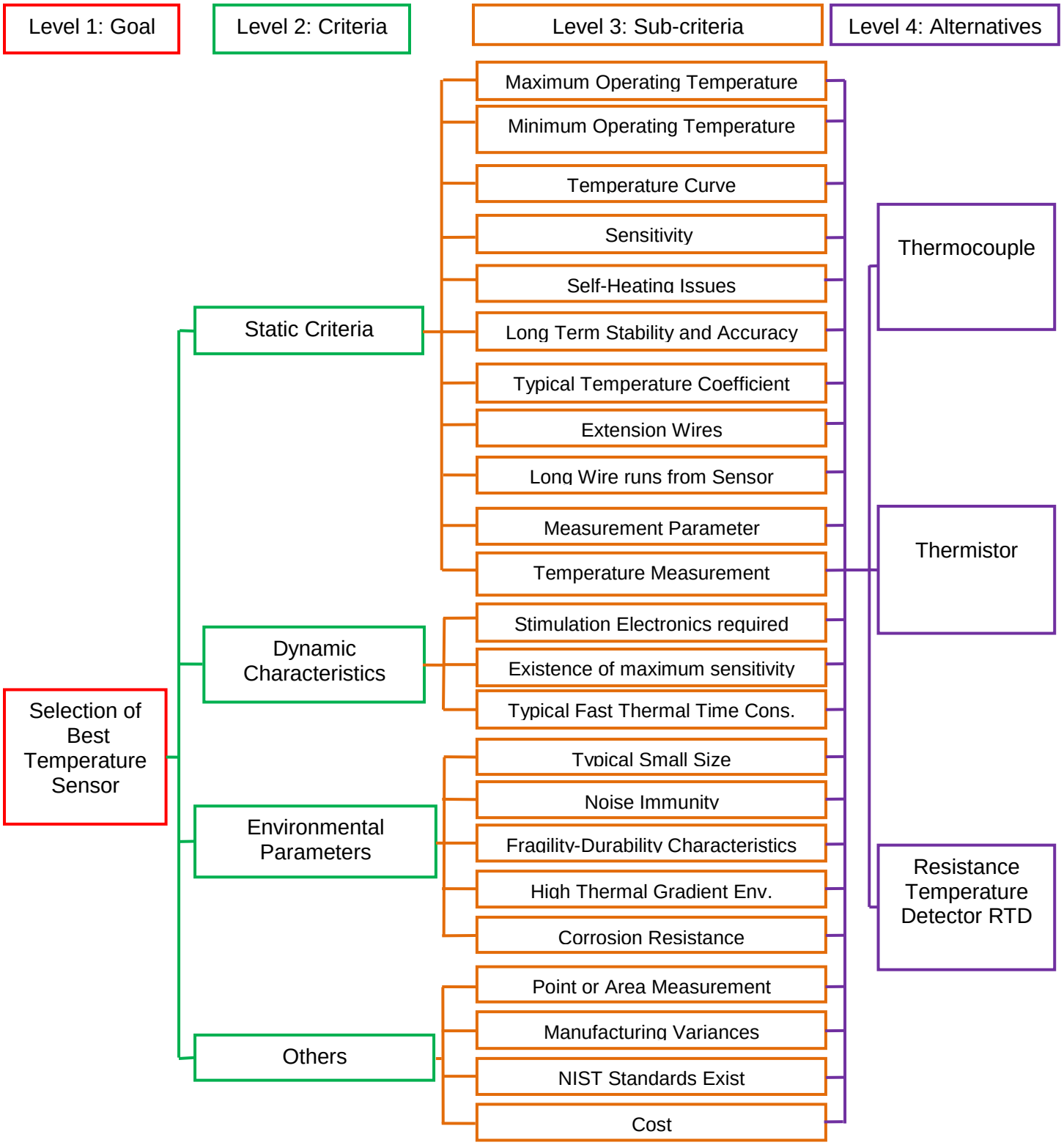

Figure 4.5: AHP Structure for Sensors Selection Problem

4.3.4 Components Weights

Appendix III shows a complete list of all component weights including weights of alternatives with respect to the 23 sub-criteria, the weights of the 23 sub-criteria with respect to each criterion, and the weight of the 4 criteria with respect to the goal for the three applications: HVAC, Automotives, and Chemical Process applications. Note that these weights are listed in the Alternatives weight vectors, Sub-criteria weight vectors, and Criteria weight vectors rows in the Appendix, respectively.

4.3.5 Components Weights Interpretation (Automotives)

The relative preference of one alternative sensor with respect to another alternative sensor against certain sub-criterion can be assessed by asking a question of the type: of the two alternative sensors, which scores more on a certain sub-criterion and by how much? Saaty scale is used to give a numerical value for the comparison as in Table 3.1. For the 7 sensors alternatives, the 23 sub-criteria, and the 4 criteria, entries for 23 matrices of the dimension 7X7 representing relative weights of the 7 sensors against the 23 sub-criteria , and entries for an 11X 11 matrix, 3X3 matrix, 5X5 matrix, and 4X4 matrix representing relative weights of the Static, Dynamic, Environmental and Others sub-criteria towards their corresponding parent criteria, respectively, and entries for a 4X4 matrix representing the relative weights of the four criteria towards the final goal are introduced and incorporated in the software. This work is separately repeated for the three applications: the HVAC, the Automotives, and the Chemical Process. These relative weights for the three different applications are listed in Appendix III. These weights were based on the view of experts in the sensor field including professors from the Mechanical Engineering Department in Jordan University of Science and Technology and from external experts working for National Paints, Inc., Amman, Jordan. Also, these weights were based on

sensors literature taken from sensors text books and from catalogues from the Web containing different sensor products from different universal manufacturing companies.

The Interpretation of the components relative weights presented in the coming sections applies to the Automotives application.

4.3.5.1 Alternatives Weights Interpretation for Selected Sub-criteria

Maximum Operating Temperature Sub-criterion: according to experts views and based on the review of sensors literature and technical data material, the fitness of a particular sensor with respect to the temperature range sub-criterion, and hence to the maximum and minimum operating temperatures is based on the closeness of the operating temperature range for the sensor to the operating temperature range of the requested application, in other words the suitability, and hence the preference of a proposed sensor towards the maximum and minimum operating temperatures of the application is governed by how close the maximum and minimum operating temperatures of the proposed sensor are to the maximum and minimum operating temperatures of the application respectively. The maximum operating temperature for the catalytic convertor application case study is, for example, 1023 Kelvin, while the minimum operating temperature for the application is 773 Kelvin. Moreover, the proposed sensor should ideally be able to measure the maximum malfunctioning temperature condition upon which melting of the packing material (substrate material of the catalytic converter) occurs, this is a temperature of 1143 K. To facilitate the evaluation of one alternative sensor with respect to another against the maximum operating temperature sub-criterion, each sensor was relatively ranked with respect to the rest sensors in terms of its preference towards the sub-criterion, then the relative weights were determined by taking all combinations of relative preferences (ranks) of one alternative sensor to another for all

seven sensors. The following are considerations of the characteristics for the seven sensors that relate to the maximum operating temperature sub-criterion:

Thermocouple: according to Table 2.1, K-type thermocouple (which has Nickel-Chromium as the positive conductor and Constantan as the negative conductor) has a maximum operating temperature of 1123 Kelvin. By comparing the closeness of the maximum operating temperature for all seven sensors with the maximum operating temperature for the application, and putting in mind that the compared sensors should be capable of measuring the melting point of the packing, thermocouple was found the closest and thus relatively ranked among the seven sensors the first, i.e. has rank (1).

Thermister: has a maximum operating temperature of 1300 Kelvin, with a difference from the value 1123 K by almost 170 K, though it satisfies measurement of the melting point, ranked among the seven sensors (2).

RTD: has a maximum operating temperature of 1150 K, satisfies measurement of the melting point, ranked (1).

Bimetallic: has a maximum operating temperature of 700 K, doesn't measure the maximum operating temperature of the application at all, theoretically, has a rank (5), practically in the software, it is excluded on the first tab, when the user chooses the temperature range for the application.

Note: in the cases where the alternative does not match the minimum required constraints of an application, or subsequently, the minimum requirements of the sub-criteria related to that application, the pair-wise comparisons (weights) for the two alternatives are theoretically evaluated and set into the 23 comparison matrices even though the system takes care of the problem in advance in the filtering stage (constraint stage) on tab one of the application program when the user chooses the nearest temperature range to the application from the list in the Temperature Range list box which is in this case 700-1150

K. Those alternatives on the second tab that fail to meet temperature range constraint on the first tab are rejected (excluded) on the second tab and appear in an inactivated-check box mode. Moreover, if a certain alternative passes the temperature range constraint for the application, but fails to satisfy the next constraint, the resolution, or the third constraint for the application, the response time, it will be filtered out on either cases, and will not be further considered for comparison on the second tab, yet its relative weight is fictitiously set in all the 7X7 sub-criteria matrices but it is not actually taken into account in the comparison process.

Thermometer: has a maximum operating temperature of 950 K. Upon consulting the experts and upon literature review, its maximum temperature span can be customized through special advanced manufacturing techniques and enlarged to entail the maximum operating temperature of the application (1023). It is ranked (3).

Pyrometer: has a maximum operating temperature of 3300 K, well above the required operating temperature for the application, so minimizing its relative preference among other sensors, ranked (4).

LCD: has a maximum operating temperature of 950 K. The same analogy of the thermometer applies to the LCD semiconductor thermometer, so it is ranked (3).

According to above characteristics, the following 7X7 judgment matrix representing the relative weights for the seven alternatives against the first sub-criterion, the Maximum Operating Temperature, as it appears in Appendix III, , is set in the program:

$$\boldsymbol{A}_{max\ temp} = [\boldsymbol{a}_{ij}] = \begin{pmatrix} 1.0 & 3.0 & 1.0 & 9.0 & 4.0 & 6.0 & 4.0 \\ 0.3333 & 1.0 & 0.3333 & 6.0 & 2.0 & 5.0 & 2.0 \\ 1.0 & 3.0 & 1.0 & 9.0 & 4.0 & 6.0 & 4.0 \\ 0.1111 & 0.1667 & 0.1111 & 1.0 & 0.1667 & 0.3333 & 0.1667 \\ 0.25 & 0.5 & 0.25 & 6.0 & 1.0 & 4.0 & 1.0 \\ 0.1667 & 0.2 & 0.1667 & 3.0 & 0.25 & 1.0 & 0.25 \\ 0.25 & 0.5 & 0.25 & 6.0 & 1.0 & 4.0 & 1.0 \end{pmatrix}$$

Based on the aforementioned descriptions of the maximum operating temperature characteristics for each sensor alternative, the evaluation of the relative weights of each alternative sensor with respect to the other sensors against the maximum operating temperature sub-criterion, i.e. entries in the above matrix, is an easy task now. Let's interpret, for example, the evaluation of the first row of the above matrix. Since the relative rank of the thermocouple among the seven sensor is (1), i.e. thermocouple is the best, and since the thermister is the second preferred choice and has a relative rank (2), then the thermocouple can be evaluated as being weakly more important than the thermister and thus given a relative weight of 3.0 according to Saaty scale. This value corresponds to entry $\boldsymbol{a_{12}}$ in the above matrix. Because the RTD has the same preference as the thermocouple (rank 1), so there is no relative preference of the thermocouple with respect to the RTD, i.e. both alternatives are equally important. Thus entry $\boldsymbol{a_{13}}$ is given a value 1.0. And since the bimetallic strip thermometer comes fifth rank when relatively compared to the rest of the seven sensors (the worst choice among the other choices with respect to the maximum operating temperature sub-criterion), then the thermocouple is extremely more important than the bimetal owing to a relative weight (entry $\boldsymbol{a_{14}}$) of 9.0. And because the mercury-in-glass thermometer comes rank (3) in the above designation, then the relative importance (entry $\boldsymbol{a_{15}}$) of the thermocouple with respect to the thermometer comes according to Saaty scale mid way between weakly more important and strongly more important with a value set to 4.0. And as the optical disappearing

filament pyrometer comes rank (4) in the above designation, then the relative importance (entry a_{16})of the thermocouple with respect to the pyrometer according to Saaty scale is mid way between strongly more important and very strongly more important, and thus given a value of 6.0. And finally, since the LCD semiconductor thermometer is rank (3), just the same as the thermometer's, then the relative importance (entry a_{17}) of the thermocouple with respect to the pyrometer according to Saaty scale is mid way between weakly more important and strongly more important, and thus given a value of 4.0, just, the same value for the relative weight between thermocouple and thermometer. The same discussion can be elaborated to interpret all the remaining rows of the above matrix.

After running the software, and if we assume that the user checks in the boxes of only thermocouple, thermister, and RTD sensors, then the software extracts automatically a 3X3 matrix from the 7X7 matrix presented above representing values that correspond to these three sensors, and the software calculates the weights of the three sensors with respect to the maximum operating temperature sub-criteria, in addition to this, the software calculates the consistency index and consistency ratio for this 3X3 matrix. The software results in terms of the three sensors: the thermocouple, the thermister, and the RTD weights against the Maximum Operating Temperature sub-criterion in the case study are shown in Table 4-2.

Table 4-2: Weights of the three alternatives against the Maximum Operating Temperature sub-criterion as they appear in the software results

Maximum Operating Temperature matrix:			
1	3	1	
0.3333	1	0.3333	
1	3	1	
	Thermocouple	Thermister	RTD
Alternatives Weight Vector =	0.42858	0.14285	0.42858
Consistency Index = 0			
Consistency Ratio = 0			

As can be seen from Table 4.2 the weights of the three sensors: thermocouple, thermister, and RTD against maximum operating temperature sub-criterion are: 0.42858, 0.14285, and 0.42858 respectively, indicating that the RTD and the Thermocouple score equally the best against maximum operating temperature sub-criterion while the thermister scores the worst. Table 4.2 also shows the consistency index for this 3X3 case study matrix to be 0 and consistency ratio to be 0 indicating that decision maker's judgments are completely consistent.

Temperature Curve Sub-criterion

Appendix IV [6, 23] lists descriptions for the performance of the thermocouple, the thermister, and the RTD alternative sensors against the 23 sub-criteria. Information in this appendix was utilized for the evaluation of the relative weights for these three sensors in this section and the many subsequent sections.

The following considerations of linearity or nonlinearity of the seven sensors were taken in assessing relative weights of this sub-criterion and in ranking the sensors accordingly:

Thermocouple: fair linearity, to facilitate estimating the relative weights in the 7X7 matrix, the thermocouple was relatively ranked among the seven sensors in rank (2).

Thermister: nonlinear, ranked among the seven sensors (3).

RTD: the most linear, or the best, ranked among the seven sensors (1).

Bimetallic: nonlinear due to nonlinearity of characteristic equation, ranked (6).

Thermometer: nonlinear due to nonlinearity of coefficient of linear expansion of both liquid and the glass, but ranked (5) since coefficient of linear expansion of glass is smaller than that for any metal.

Pyrometer: nonlinear due to nonlinearity of governing equations of radiation phenomena (Stefan-Boltzmann law of total power of radiant flux), ranked (4).

LCD: nearly linear since it is mainly a silicon based semiconductor sensor, ranked nearly (2) like the thermocouple, but thermocouple is two folds better.

It is an advantage for any temperature measurement system to have a linear response of the sensor, since little or no electronic circuits are needed to correct for nonlinearity in addition the sensor is more electronically compatible and easily connected to transmitters and signal conditioning circuits and the total system error becomes smaller. Based on the previous considerations, the following matrix represents the relative weights of the seven sensors against Temperature Curve sub-criterion as it appears in the software:

$$A_{temp\ curve} = [a_{ij}] = \begin{pmatrix} 1.0 & 3.0 & 0.3333 & 5.0 & 5.0 & 4.0 & 2.0 \\ 0.3333 & 1.0 & 0.1667 & 2.0 & 2.0 & 1.0 & 0.3333 \\ 3.0 & 6.0 & 1.0 & 6.0 & 6.0 & 6.0 & 4.0 \\ 0.2 & 0.5 & 0.1667 & 1.0 & 1.0 & 0.3333 & 0.25 \\ 0.2 & 0.5 & 0.1667 & 1.0 & 1.0 & 0.3333 & 0.25 \\ 0.25 & 1.0 & 0.1667 & 3.0 & 3.0 & 1.0 & 0.5 \\ 0.5 & 3.0 & 0.25 & 4.0 & 4.0 & 2.0 & 1.0 \end{pmatrix}$$

An example of the relative weight estimation in the above matrix is the relative weight of the thermocouple with respect to the thermister, entry a_{12} in the above matrix, this weight is set 3, because thermocouple is weakly more preferred than thermister with respect to Temperature Curve sub-criterion, while that of an RTD with respect to the bimetallic is 6 because RTD relative weight lies mid way between being strongly more important and very strongly more important than bimetallic with respect to the Temperature Curve sub-criteria. The same analogy applies to the rest of the entries in the matrix.

The software results in terms of the three sensors: the thermocouple, the thermister, and the RTD weights against the Temperature Curve sub-criterion in the case study are shown in Table 4-3.

Table 4-3: Weights of the three alternatives against the Temperature Curve sub-criterion.

Temperature Curve matrix:				
1	3	0.3333		
0.3333	1	0.1667		
3	6	1		
		Thermocouple	Thermister	RTD
Alternatives Weight Vector =		0.25099	0.09602	0.65299
Consistency Index = 0.00918				
Consistency Ratio = 0.01583				

Sensitivity Sub-criterion: values of the incremental ratio of the sensor's output to the input temperature (the sensitivity) for the seven sensors [7] are as follows:

Thermocouple: low sensitivity on the order of 20-80 μV/°C, to facilitate estimating the relative weights in the 7X7 matrix, the thermocouple was relatively ranked among the sensors in rank (4).

Thermister: very high sensitivity, negative temperature coefficient (NTC) thermisters can have sensitivities on the order of 4.0 % Ω/Ω/◦C, the most sensitive of all sensors, ranked among the seven sensors (1).

RTD: medium sensitivity on the order of 0.39 % Ω/Ω/◦C, (~4milli /◦C) ranked among the seven sensors (2).

Bimetallic: low sensitivity on the order of 20 ppm (mm/mm/◦C), ranked (5).

Thermometer: low sensitivity on the order of 8.5 ppm (mm/mm/◦C), ranked (7).

Pyrometer: low sensitivity on the order of 10 ppm◦C $^{-1}$, ranked (6).

LCD: medium sensitivity on the order of 0.19 % Ω/Ω/◦C [7], (~1.9 milli /◦C),

ranked (3).

Based on the above technical data, the judgment matrix of the relative weights for the Sensitivity sub-criterion for the seven sensors as it appears in Appendix III is:

$$A_{sensitivity} = [a_{ij}] = \begin{pmatrix} 1.0 & 0.1111 & 0.2 & 2.0 & 2.0 & 2.0 & 0.3333 \\ 9.0 & 1.0 & 4.0 & 9.0 & 9.0 & 6.0 & 4.0 \\ 5.0 & 0.25 & 1.0 & 4.0 & 5.0 & 4.0 & 2.0 \\ 0.5 & 0.1111 & 0.25 & 1.0 & 2.0 & 2.0 & 0.25 \\ 0.5 & 0.1111 & 0.2 & 0.5 & 1.0 & 1.0 & 0.25 \\ 0.5 & 0.1667 & 0.25 & 0.5 & 1.0 & 1.0 & 0.25 \\ 3.0 & 0.25 & 0.5 & 4.0 & 4.0 & 4.0 & 1.0 \end{pmatrix}$$

The relative weights of the sensors in the above matrix were based on the values of sensitivities for these sensors. For example, the thermister relative weight with respect to the thermocouple, entry a_{21} is 9 since sensitivity of the thermister is the highest among all sensors and thermocouple sensitivity is low on the order of ppm (parts per million or microns). Another example is the relative weight of the RTD with respect to the bimetallic strip thermometer, entry a_{34}, is 4 since the RTD's sensitivity is moderate on the order of 4 milli per Celsius while that of the bimetallic is very low on the order of 20 ppm. The same analogy is used for the rest of the weights (entries). The software uses these preset values to calculate weights of the seven sensors with respect to the Sensitivity sub-criterion. Results of the software in terms of sensors weights for the Sensitivity sub-criterion for the three sensors case study are shown in Table 4-4.

Table 4-4: The three alternatives case study weights for the Sensitivity sub-criterion

Sensitivity matrix:			
1	0.1111	0.2	
9	1	4	
5	0.25	1	
	Thermocouple	Thermister	RTD
Alternatives Weight Vector =	0.06225	0.70131	0.23644
Consistency Index = 0.03611			
Consistency Ratio = 0.06226			

Self Heating Sub-criterion: below are the judgments relating to the seven sensors with respect to Self Heating sub-criterion that will be used in determining the relative weights of the judgment matrix.

Thermocouple: experiences no self heating, one among the best sensors, ranked (1).

Thermister: experiences high level of self heating, the worst sensor of all, ranked (7).

RTD: experiences very low to low level of self heating, ranked (4).

Bimetallic: experiences no self heating, but can fail 'closed' (short-circuited) at end of life, ranked (2).

Thermometer: experiences no self heating, one among the best sensors, ranked (1).

Pyrometer: experiences no self heating, one among the best sensors, ranked (1).

LCD: experiences very low to low level of self heating, ranked (3).

Based on the upper judgments, the judgment matrix of the relative weights for the Self Heating sub-criterion is

$$A_{self\ heating} = [a_{ij}] = \begin{pmatrix} 1.0 & 8.0 & 3.0 & 3.0 & 1.0 & 1.0 & 2.0 \\ 0.125 & 1.0 & 0.2 & 0.25 & 0.2 & 0.1667 & 0.25 \\ 0.3333 & 5.0 & 1.0 & 0.5 & 0.5 & 0.3333 & 1.0 \\ 0.3333 & 4.0 & 2.0 & 1.0 & 1.0 & 0.5 & 1.0 \\ 1.0 & 5.0 & 2.0 & 1.0 & 1.0 & 1.0 & 2.0 \\ 1.0 & 6.0 & 3.0 & 2.0 & 1.0 & 1.0 & 1.0 \\ 0.5 & 4.0 & 1.0 & 1.0 & 0.5 & 1.0 & 1.0 \end{pmatrix}$$

The values of the relative weights in the above matrix are based on the judgments stated before. For example, since the thermocouple experiences no self heating problem at all, it is considered superior to the thermister which experiences high level of self heating, so the relative weight of the thermocouple with respect to the thermister, the a_{12} entry in the above matrix, is set mid way on Saaty's scale between very strongly more important and extremely more important and given a value of 8. Another example is the relative weight of the RTD with respect to the bimetallic, the a_{34} entry. Since the RTD experiences very low to low self heating and is ranked relative to the rest of the sensors (4) and the bimetallic ranked (2), and since self heating is not common in bimetallic, i.e. we don't, in general, talk about bimetallic self heating characteristic, then the relative importance of the bimetallic with respect to the RTD is no more than a factor of 2. The same analogy applies to the rest of the entries in the matrix. Table 4-5 shows the results of the software in terms of sensors weights for the Self Heating sub-criterion for the three sensors case study.

Table 4-5: The results for the Self Heating sub-criterion

Self-Heating Issues matrix:				
1	8	3		
0.125	1	0.2		
0.3333	5	1		
		Thermocouple	Thermsiter	RTD
Alternatives Weight Vector =		0.65715	0.06825	0.27459
Consistency Index = 0.02218				
Consistency Ratio = 0.03824				

Typical Small Size Sub-criterion: below are the judgments [7] relating to the seven sensors with respect to the Typical Small Size sub-criterion and that will be used in determining the relative weights of the judgment matrix.

Thermocouple: the smallest sensor, sizes down to 0.025 mm of the thermocouple wire diameter are present in industry. Typical size is 0.25 mm diameter, ranked relative to rest alternatives (1).

Thermister: the next smaller sensor, typical sizes for bead-type thermisters diameters range from 0.4 mm to 2.5 mm with a typical probe length 3-12.7 mm, ranked (2).

RTD: the second next smaller sensor, with an RTD diameter ranging from 1.6 mm to 6.35 mm and an RTD probe length ranging from 1.6 mm to 101.6 mm, ranked (3).

Bimetallic: has a typical strip length of 3 inches (76.2 mm), ranked (4).

Thermometer: has a typical length of 8 inches (203.2 mm), ranked (5).

Pyrometer: commercial pyrometer has a size of 54 mm X 54 mm X 147 mm, ranked (6).

LCD: commercial LCD has dimensions of 20 cm X 6.35 cm, ranked (5).

Based on the upper judgments, and based on the notion that as the size of the sensor becomes smaller it is considered better due to its faster response time and increased fitness to be installed in any place within process, then the judgment matrix of the relative weights for the Typical Small Size sub-criterion is:

$$A_{size} = [a_{ij}] = \begin{pmatrix} 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 5.0 \\ 0.5 & 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 4.0 \\ 0.3333 & 0.5 & 1.0 & 2.0 & 4.0 & 6.0 & 4.0 \\ 0.25 & 0.3333 & 0.5 & 1.0 & 2.0 & 3.0 & 2.0 \\ 0.2 & 0.25 & 0.25 & 0.5 & 1.0 & 3.0 & 1.0 \\ 0.1667 & 0.2 & 0.1667 & 0.3333 & 0.3333 & 1.0 & 0.5 \\ 0.2 & 0.25 & 0.25 & 0.5 & 1.0 & 2.0 & 1.0 \end{pmatrix}$$

The values of the relative weights in the above matrix are based on the information stated above. For example, since the thermocouple is the smallest sensor and the pyrometer is the largest the relative weight of the thermocouple with respect to the pyrometer, i.e. the a_{16} entry in the above matrix, is set mid way on Saaty's scale between strongly more important and very strongly more important and given a value of 6. Another example is the relative weight of the pyrometer with respect to the thermometer, the a_{65} entry. Since the pyrometer is a 3-D device and the thermometer is only approximately one dimension (length), then the relative weight is set to 0.3333 indicating that pyrometer is weakly less important than the liquid-in-glass thermometer. The same analogy applies to the rest of the entries in the matrix. Table 4-6 shows the results of the software in terms of sensor weights for the Typical Small Size sub-criterion for the three sensors case study.

Table 4-6: The software results for the Typical Small Size sub-criterion

Typical Small Size matrix:			
1	2	3	
0.5	1	2	
0.3333	0.5	1	
	Thermocouple	Thermister	RTD
Alternatives Weight Vector =	0.53896	0.29725	0.16377
Consistency Index = 0.00458			
Consistency Ratio = 0.00791			

Typical Fast Thermal Time Constant Sub-criterion: below are values of the average response time for the seven sensors. These values will be used in determining the relative weights of the judgment matrix.

Thermocouple: 0.01 seconds (see Appendix IV), among the fastest sensors, ranked relative to rest alternatives (1).

Thermister: typical response time 2 seconds [7], ranked (2).

RTD: 5 seconds [7] , ranked (4).

Bimetallic: 20 seconds [24], the slowest of all sensors, ranked (6).

Thermometer: 10 seconds [25], ranked (5).

Pyrometer: 0.01 seconds [26], among the fastest sensors, ranked (1).

LCD: 3 seconds [27], ranked (3).

Based on the upper judgments, and based on the notion that it is an advantage for the sensor in a process to have smaller response time then the judgment matrix of the relative weights for the Fast Thermal Time Constant sub-criterion is:

$$A_{fast\ th\ time\ cons} = [a_{ij}] = \begin{pmatrix} 1.0 & 3.0 & 4.0 & 6.0 & 5.0 & 1.0 & 3.0 \\ 0.3333 & 1.0 & 2.0 & 4.0 & 3.0 & 0.3333 & 1.0 \\ 0.25 & 0.5 & 1.0 & 2.0 & 2.0 & 0.25 & 1.0 \\ 0.1667 & 0.25 & 0.5 & 1.0 & 0.5 & 0.1667 & 0.3333 \\ 0.2 & 0.3333 & 0.5 & 2.0 & 1.0 & 0.2 & 0.3333 \\ 1.0 & 3.0 & 4.0 & 6.0 & 5.0 & 1.0 & 3.0 \\ 0.3333 & 1.0 & 1.0 & 3.0 & 3.0 & 0.3333 & 1.0 \end{pmatrix}$$

The values of the relative weights in the above matrix were based on the response time values for the seven sensors stated above. For example, since the thermocouple and the pyrometer are the fastest sensors and the bimetallic strip is the slowest, then entries a_{14} and a_{64} are set midway on Saaty's scale between strongly more important and very strongly more important and given a value of 6. Another example is the relative weight of

the LCD with respect to the pyrometer. The response time of an LCD is 3 seconds and that of pyrometer is 0.01 seconds, then the pyrometer is relatively weakly more important than the LCD and the entry a_{67} is given a value of 3. The same analogy applies to the rest of the entries in the matrix. Table 4-7 shows the results of the software for the Typical Fast Thermal Time Constant sub-criterion for the three sensors case study.

Table 4-7: The software results for the Typical Fast Thermal Time Constant sub-criterion.

Typical Fast Thermal Time Constant matrix:			
1	3	4	
0.3333	1	2	
0.25	0.5	1	
	Thermocouple	Thermister	RTD
Alternatives Weight Vector =	0.62323	0.23948	0.13728
Consistency Index = 0.00915			
Consistency Ratio = 0.01578			

Long Term Stability and Accuracy Sub-criterion: below are descriptions of the long term stability and accuracy behavior for the seven sensors.

Thermocouple: thermocouple long term stability and accuracy is okay (see Appendix IV), it experiences drift and needs calibration. Its resolution is within ± 1.0 °C. It is ranked relative to rest alternatives (6).

Thermister: good stability and accuracy, its resolution ranges from ± 0.1 °C to ± 0.001 °C, ranked (2).

RTD: RTD is the most stable and accurate of all sensors, its resolution can reach ± 0.00001 °C, ranked (1).

Bimetallic: bimetallic thermometer experiences drift, its resolution ranges from ± 1.0 °C to ± 2.0 °C, ranked (7).

Thermometer: thermometer exhibits fair stability, accuracy of ± 0.1 °C, ranked (4).

Pyrometer: pyrometer has good resolution ranging from ± 0.1 °C to ± 1.0 °C, ranked (5).

LCD: LCD has good stability, resolution of 0.001-0.1 °C, ranked (3).

Based on the upper judgments, and based on the notion that long term stability and accuracy is an advantage for a sensor then the judgment matrix of the relative weights for the Long Term Stability and Accuracy sub-criterion is:

$$A_{long\ term\ stability} = [a_{ij}] = \begin{pmatrix} 1.0 & 0.25 & 0.1667 & 2.0 & 0.3333 & 0.5 & 0.25 \\ 4.0 & 1.0 & 0.3333 & 4.0 & 3.0 & 3.0 & 2.0 \\ 6.0 & 3.0 & 1.0 & 8.0 & 4.0 & 5.0 & 3.0 \\ 0.5 & 0.25 & 0.125 & 1.0 & 0.3333 & 0.3333 & 0.25 \\ 3.0 & 0.3333 & 0.25 & 3.0 & 1.0 & 2.0 & 0.5 \\ 2.0 & 0.3333 & 0.2 & 3.0 & 0.5 & 1.0 & 0.3333 \\ 4.0 & 0.5 & 0.3333 & 4.0 & 2.0 & 3.0 & 1.0 \end{pmatrix}$$

The values of the relative weights in the above matrix were based on the descriptions for the seven sensors stated above. For example, since the RTD exhibits the best stability and accuracy and the thermister exhibits good stability and accuracy, a relative weight of 0.3333 is given to the entry a_{23} in the above matrix. The same analogy applies to the rest of the entries in the matrix. Table 4-8 shows the results of the software for the Long Term Stability and Accuracy sub-criterion for the three sensors case study.

Table 4-8: The software results for the Long Term Stability and Accuracy sub-criterion

Long Term Stability and Accuracy matrix:					
1	0.25	0.1667			
4	1	0.3333			
6	3	1			
			Thermocouple	Thermister	RTD
Alternatives Weight Vector =			0.08695	0.27371	0.63933
Consistency Index = 0.02704					
Consistency Ratio = 0.04663					

Corrosion Resistance Sub-criterion: below are descriptions of the corrosion characteristics for the seven sensors.

Thermocouple: low corrosion resistance [7, 8]. It is ranked relative to rest alternatives (4).

Thermister: good corrosion resistance, but corrodes in acidic, alkali media, ranked [23] (2).

RTD: RTD has good corrosion resistance due to its highly inert platinum wire [7, 8], ranked (1).

Bimetallic: experiences normal corrosion of component metals, ranked (3).

Thermometer: has good corrosion resistance due to its inert glass capillary tube, ranked (1).

Pyrometer: pyrometer has no direct contact with the medium, so it is not exposed to corrosion due to medium, ranked (1).

LCD: LCD has good corrosion resistance, relatively ranked [7] (2).

Based on the upper judgments, and based on the notion that corrosion resistance is an advantage for a sensor then the judgment matrix of the relative weights for the Corrosion Resistance sub-criterion is:

$$\boldsymbol{A}_{corrosion\ resis.} = [\boldsymbol{a}_{ij}] = \begin{pmatrix} 1.0 & 0.25 & 0.1667 & 0.5 & 0.1667 & 0.1667 & 0.25 \\ 4.0 & 1.0 & 0.3333 & 2.0 & 0.25 & 0.25 & 1.0 \\ 6.0 & 3.0 & 1.0 & 4.0 & 1.0 & 1.0 & 4.0 \\ 2.0 & 0.5 & 0.25 & 1.0 & 0.25 & 0.25 & 0.5 \\ 6.0 & 4.0 & 1.0 & 4.0 & 1.0 & 1.0 & 3.0 \\ 6.0 & 4.0 & 1.0 & 4.0 & 1.0 & 1.0 & 3.0 \\ 4.0 & 1.0 & 0.25 & 2.0 & 0.3333 & 0.3333 & 1.0 \end{pmatrix}$$

The values of the relative weights in the above matrix were based on the descriptions for the seven sensors stated above. For example, since the thermocouple exhibits the worst corrosion resistance and the thermometer exhibits excellent corrosion resistance, the entry a_{15} in the above matrix is given a value of 0.1667. The same analogy applies to the rest of the entries in the matrix. Table 4-9 shows the results of the software for the Corrosion Resistance sub-criterion for the three sensors case study.

Table 4-9: The software results for the Corrosion Resistance sub-criterion

Corrosion Resistance matrix:				
1	0.25	0.1667		
4	1	0.3333		
6	3	1		
		Thermocouple	Thermister	RTD
Alternatives Weight Vector =		0.08695	0.27371	0.63933
Consistency Index = 0.02704				
Consistency Ratio = 0.04663				

Cost Sub-criterion: below is cost information for the seven sensors. Putting into mind that cost minimization is a pursuit of any temperature measurement system then the relative rank of the seven sensors would appear as indicated beside each.

Thermocouple: low –medium cost. Typical cost values depending on the thermocouple type range from $5 for many thermocouples to $150 for some products of the high temperature measuring K-type thermocouple. So it is one of the least expensive choices. It is ranked relative to rest alternatives (1).

Thermister: low –medium cost. Typical cost values depend on the thermister features and range from $0.6 to $28, relatively close cost to thermocouple's, ranked (1).

RTD: high cost. High-accuracy wire-wound RTDs can be as costly as $5000, $700 is not uncommon. See Table 2.2, ranked the worst alternative relative to the Cost sub-criterion (3).

Bimetallic: medium cost, cost ranges from $10 to $53, ranked (2).

Thermometer: medium cost, typical cost values range from $15 to $67, ranked (2).

Pyrometer: very expensive, cost values for pyrometers can range from $200 up to more than $3000, ranked (3).

LCD: medium cost, cost ranges from $10 to $50, relatively ranked (2).

Based on the upper judgments, the following is the relative weights judgment matrix for the Cost sub-criterion.

$$\boldsymbol{A}_{cost} = [\boldsymbol{a}_{ij}] = \begin{pmatrix} 1.0 & 1.0 & 6.0 & 3.0 & 3.0 & 6.0 & 3.0 \\ 1.0 & 1.0 & 6.0 & 3.0 & 3.0 & 6.0 & 3.0 \\ 0.1667 & 0.1667 & 1.0 & 0.25 & 0.25 & 1.0 & 0.25 \\ 0.3333 & 0.3333 & 4.0 & 1.0 & 1.0 & 4.0 & 1.0 \\ 0.3333 & 0.3333 & 4.0 & 1.0 & 1.0 & 4.0 & 1.0 \\ 0.1667 & 0.1667 & 1.0 & 0.25 & 0.25 & 1.0 & 0.25 \\ 0.3333 & 0.3333 & 4.0 & 1.0 & 1.0 & 4.0 & 1.0 \end{pmatrix}$$

Evaluating the relative weights in the above matrix is based on comparing the relative ranks given to the seven sensors against the cost sub-criterion. For example, since the thermocouple and the thermister on average exhibit the least cost sensor choices, and come rank 1 while the industrial thermometer exhibits a medium cost alternative with a rank 2, then both the thermocouple and the thermsiter are equally weakly more important than the thermometer and entries $\boldsymbol{a_{15}}$ and $\boldsymbol{a_{25}}$ are assigned value 3.0. On the other hand, since the thermocouple is the most preferred alternative with respect to the Cost sub-criterion with rank 1 and because the RTD and the pyrometer are the most expensive alternatives among all alternatives having a rank 3 then the RTD and the pyrometer score badly relative to the thermocouple. You can consider them as being mid way between strongly less important and very strongly less important than the thermocouple and thus entries $\boldsymbol{a_{31}}$ and $\boldsymbol{a_{61}}$ are

given value 0.1667. Table 4-10 shows the results of the software for the Cost sub-criterion for the three sensors case study.

Table 4-10 The software results for the Cost sub-criterion

Cost matrix:				
1	1	6		
1	1	6		
0.1667	0.1667	1		
		Thermocouple	Thermister	RTD
Alternatives Weight Vector =		0.46153	0.46153	0.07693
Consistency Index = 0				
Consistency Ratio = 0				

4.3.5.2 Sub-criteria Weights Interpretation

The discussion in this section and the next section relates to the Automotives application. Criteria and sub-criteria weights for the other two applications: the Chemical Process and the HVAC applications will be interpreted and explained in section 4.3.5.4.

Static Sub-criteria Relative Weights: the 11 sub-criteria comprising the Static criterion are relatively pair-wise compared using Saaty scale. Recall from section 2.2.1 the important requirements for the selection of temperature sensors for a certain application. These requirements are mentioned here again to remind of the relative importance for each:

1- Temperature range
2- Accuracy
3- Response time
4- Sensitivity
5- Corrosion conditions and resistance
6- Breaking down due to wear and tear
7- Interchangeability
8- Variations in temperature – temperature shock

9- Pressure conditions

The scheme for prioritizing the 23 sub-criteria-although not seen here- was first laid on comparing the 23 sub-criteria as a whole, and as if they all belong to the same parent criterion, which is in this case, the overall goal; the selection of the best sensor. In doing so, the 23 sub-criteria are passed on and assimilated to one of the above requirements. The requirements are followed here from top to bottom in a descending order of preference or priority for evaluating the relative preferences of the 23 sub-criteria, i.e. starting from temperature range requirement with the first priority, then passing on to accuracy requirement with the second priority and so forth.

To elaborate, every single sub-criterion of the 23 sub-criteria is passed on the requirements and assimilated to one of them. Throughout the work for this section, if multiple sub-criteria of the 23 sub-criteria are related in different degrees to the same requirement of the above, only the one with the closest relation to the requirement is passed on at one time and the next with the second degree relationship is passed on after the rest of sub-criteria that assimilate the next requirements are passed on. For example, the Long Term Stability and Accuracy sub-criterion, the NIST Standards sub-criterion, the Point or Area Measurement sub-criterion, the Self-Heating Issues sub-criterion , the Temperature Curve sub-criterion, the Extension Wires sub-criterion, and the Temperature Measurement sub-criterion all are related and assimilated to the accuracy requirement in different degrees of relationship. However only one sub-criterion of the seven afore mentioned sub-criteria with the closest similitude (degree of relationship) to the accuracy requirement, which is the Long Term Stability and Accuracy sub-criterion in this case, is passed on at a time for purposes of prioritization. The other six are periodically passed on one by one after the next requirements are assimilated with the rest of the sub-criteria, i.e. after response time, sensitivity,… etc. requirements have been assimilated with the remaining sub-criteria.

Based on this scheme the relative preference of the aforementioned seven sub-criteria from the most important to the least important is: the Long Term Stability sub-criterion, the NIST Standards sub-criterion, the Point or Area Measurement sub-criterion, the Self-Heating Issues sub-criterion, the Temperature Curve sub-criterion, the Extension Wires sub-criterion, and the Temperature Measurement sub-criterion, respectively. The outcome of this prioritization scheme is 23 sub-criteria relatively ranked from (1) to (23). Table 4-11 depicts the ranks for the 23 sub-criteria under this prioritization scheme.

Table 4-11: The ranks for the 23 Sub-criteria with respect to the overall goal.

Criteria	Sub-Criteria	Rank
Static Criteria	Maximum Operating Temperature	1
	Minimum Operating Temperature	1
	Temperature Curve	15
	Sensitivity	6
	Self-Heating Issues	13
	Long Term Stability and Accuracy	2
	Typical Temperature Coefficient	16
	Extension Wires	17
	Long Wire runs from Sensor	21
	Measurement Parameter	22
	Temperature Measurement	20
Dynamic Characteristics	Stimulation Electronics required	5
	Existence of Maximum Sensitivity Region	18
	Typical Fast Thermal Time Constant	3
Environmental Parameters	Typical Small Size	12
	Noise Immunity	14
	Fragility-Durability Characteristics	8
	High Thermal Gradient Environment	9
	Corrosion Resistance	7
Others	Point or Area Measurement	11
	Manufacturing Variances	19
	NIST Standards exist	10
	Cost	4

The next step is to categorize these 23 sub-criteria into their 4 parent criteria, namely, into: Static criteria, Dynamic criteria, Environmental criteria, and Others criteria categories. The prioritization scheme will be utilized here to help prioritize different sub-criteria inside the same parent criterion (see Table 4.11). Putting in mind that operating temperature range

comprises both minimum and maximum operating temperatures then the relative preferences (ranks) that apply to the 11 sub-criteria comprising the Static criterion alongside their interpretation become as follows:

Maximum Operating Temperature: is assimilated to temperature range requirement, and since the temperature range requirement has requirement priority (1) to be satisfied, then the Maximum Operating Temperature sub-criterion also has priority (1) among other sub-criteria inside the Static criterion. So it is ranked (1).

Minimum Operating Temperature: is assimilated to temperature range requirement, it has the same priority as the Maximum Operating Temperature sub-criterion and ranked (1).

Temperature Curve: is realized in sensors and assimilated to accuracy requirement. A linear response of a sensor provides for accurate sensor measurements while non-linear responses add to sensor inaccuracy and error. Since it is prioritized in the second pass on requirements while the more relevant sub-criterion to the accuracy requirement, which is the Long Term Stability and Accuracy sub-criterion is prioritized in the first pass, so it is ranked less important than the Long Term Stability and Accuracy sub-criterion in the Static category and has a new rank (5).

Sensitivity: is assimilated to sensitivity requirement and ranked relatively inside Static criterion (3).

Self-Heating Issues: assimilated to accuracy, ranked inside Static criterion (4).

Long Term Stability and Accuracy: assimilated to accuracy, ranked (2).

Typical Temperature Coefficient: assimilated to sensitivity, ranked (6).

Extension Wires: assimilated to accuracy, ranked (7).

Long Wire Runs from Sensor: assimilated to accuracy, ranked (9).

Measurement Parameter: assimilated to accuracy, ranked (10).

Temperature Measurement: assimilated to accuracy, ranked (8).

Having set the relative preferences for the 11 Static sub-criteria, the following judgment matrix is established based on Saaty scale of relative importance:

$$A_{sub\text{-}static} = [a_{ij}] = \begin{pmatrix} 1.0 & 1.0 & 5.0 & 4.0 & 4.0 & 2.0 & 5.0 & 6.0 & 7.0 & 8.0 & 6.0 \\ 1.0 & 1.0 & 5.0 & 4.0 & 4.0 & 2.0 & 5.0 & 6.0 & 7.0 & 8.0 & 6.0 \\ 0.2 & 0.2 & 1.0 & 0.3333 & 0.3333 & 0.25 & 1.0 & 2.0 & 4.0 & 5.0 & 3.0 \\ 0.25 & 0.25 & 3.0 & 1.0 & 2.0 & 0.5 & 3.0 & 3.0 & 5.0 & 6.0 & 4.0 \\ 0.25 & 0.25 & 3.0 & 0.5 & 1.0 & 0.3333 & 3.0 & 5.0 & 6.0 & 8.0 & 4.0 \\ 0.5 & 0.5 & 4.0 & 2.0 & 3.0 & 1.0 & 4.0 & 5.0 & 6.0 & 8.0 & 5.0 \\ 0.2 & 0.2 & 1.0 & 0.3333 & 0.3333 & 0.25 & 1.0 & 1.0 & 4.0 & 6.0 & 3.0 \\ 0.1667 & 0.1667 & 0.5 & 0.3333 & 0.2 & 0.2 & 1.0 & 1.0 & 3.0 & 4.0 & 1.0 \\ 0.1429 & 0.1429 & 0.25 & 0.2 & 0.1667 & 0.1667 & 0.25 & 0.3333 & 1.0 & 2.0 & 0.3333 \\ 0.125 & 0.125 & 0.2 & 0.1667 & 0.125 & 0.125 & 0.1667 & 0.25 & 0.5 & 1.0 & 0.25 \\ 0.1667 & 0.1667 & 0.3333 & 0.25 & 0.25 & 0.2 & 0.3333 & 1.0 & 3.0 & 4.0 & 1.0 \end{pmatrix}$$

The interpretation of entries values for the above matrix is an easy task after the 11 sub-criteria have been properly prioritized. For example, since Maximum and Minimum Operating Temperature sub-criteria rank 1 while Temperature Curve sub-criterion ranks 5, then both Maximum and Minimum Operating Temperature sub-criteria are strongly more important than Temperature Curve sub-criterion and their relative weights according to Saaty's scale is given a value 5, so both entries a_{13} and a_{23} have a value 5. The self heating is an important characteristic that plays decisive role in sensor's accuracy, so the Self heating Issues sub-criterion ranks 4 among the 11 sub-criteria. The Typical Temperature Coefficient, on the other hand, is less important for a sensor- yet retains good importance- than Self Heating Issue sub-criterion, so it ranks 6, if these considerations are taken into mind, then the Self heating Issues sub-criterion can be considered weakly more important than the Typical Temperature Coefficient sub-criterion with three folds and thus entry a_{57} has a value 3. The worst scoring sub-criterion among all 11 sub-criteria is the Measurement Parameter sub-criterion (see Appendix IV for meaning of measurement

parameter), the reason behind is that whether you are measuring resistance or voltage by your sensor, it makes a slight difference, both measurement parameters will eventually give you a representation of the temperature measured. This gives rise to the relatively low rank given to the Measurement Parameter sub-criteria which is (10). However, Measurement Parameter sub-criterion still has a slight effect on the final goal in the sense that because temperature measurement through voltage difference created via thermoelectric effect in thermocouple, for example, is an inherent phenomenon that happens whenever a thermo element conductor is exposed to a temperature difference. It does not need an external power source to drive the sensor nor does it need software to convert the thermo voltage to temperature. The thermo voltage has a direct relation to the temperature difference, needs no signal conditioning, and can be directly looked up from standard tables. Resistance temperature measurement, on the other hand, relies on passing direct current through the sensor, the thermister, for example, then measuring the electric resistance of the sensor then relating this resistance to the temperature being measured. It normally needs computer software to convert these resistance values into a value of the temperature according to the characteristic Temperature-Resistance equation for the sensor. All these stages have their own contribution to the total error in the sensor's temperature reading, so thermo voltage parameter is generally better, however, this need not prevent the use of passive devices- those that use direct current to drive them- and thus the overall relative importance of Measurement Parameter sub-criterion is the lowest. Because the Minimum Operating Temperature sub-criterion is the most important sub-criterion among the 11 sub-criteria that a certain alternative sensor must fulfill with rank (1) and because the Measurement Parameter is the least important (10), then the Minimum Operating Temperature sub-criterion is midway between very strongly more important

and extremely more important than the Measurement Parameter sub-criterion and thus entry $a_{10\,2}$ is a fraction 0.125.

The same analogy can be applied to the rest of entries. Having been set into the software, this 11X 11 matrix is then used to calculate the weights of the individual sub-criteria that comprise the Static criterion. Table 4.12 shows the sub-criteria weights calculated by the software for the Static criterion.

Table 4-12: Static Sub-criteria Weights

Sub-Criteria Static matrix:											List of Sub-criteria
1.0	1.0	5.0	4.0	4.0	2.0	5.0	6.0	7.0	8.0	6.0	(1) Maximum Operating Temperature
1.0	1.0	5.0	4.0	4.0	2.0	5.0	6.0	7.0	8.0	6.0	(2) Minimum Operating Temperature
0.2	0.2	1.0	0.3333	0.3333	0.25	1.0	2.0	4.0	5.0	3.0	(3) Temperature Curve
0.25	0.25	3.0	1.0	2.0	0.5	3.0	3.0	5.0	6.0	4.0	(4) Sensitivity
0.25	0.25	3.0	0.5	1.0	0.3333	3.0	5.0	6.0	8.0	4.0	(5) Self Heating
0.5	0.5	4.0	2.0	3.0	1.0	4.0	5.0	6.0	8.0	5.0	(6) Stability and Accuracy
0.2	0.2	1.0	0.3333	0.3333	0.25	1.0	1.0	4.0	6.0	3.0	(7) Temperature Coefficient
0.1667	0.1667	0.5	0.3333	0.2	0.2	1.0	1.0	3.0	4.0	1.0	(8) Extension Wires
0.1429	0.1429	0.25	0.2	0.1667	0.1667	0.25	0.3333	1.0	2.0	0.3333	(9) Long Wire Runs
0.125	0.125	0.2	0.1667	0.125	0.125	0.1667	0.25	0.5	1.0	0.25	(10) Measurement Parameter
0.1667	0.1667	0.3333	0.25	0.25	0.2	0.3333	1.0	3.0	4.0	1.0	(11) Temperature Measurement

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Static sub-criteria Weight Vector =	0.22120	0.22120	0.05379	0.09837	0.09777	0.15040	0.05234	0.03704
(9)	(10)	(11)						
0.01983	0.01452	0.03355						

Consistency Index = 0.08281

Consistency Ratio = 0.05209

Dynamic Sub-criteria Relative Weights: in this section, the three Dynamic sub-criteria are also ranked after being assimilated to the requirements mentioned in the previous section. These sub-criteria and their relative ranks are:

Stimulation Electronics Required: this sub-criterion refers to the extent a nominated alternative sensor is in need to be driven by an external source of electrical power, namely,

direct current, or it is self-driven and the consequent need for the set of stimulation electronics and/ or the need for signal conditioning circuits, interface circuits, analogue to digital converter circuits, lead wires circuits, or signal amplification circuits. It is ranked relative to the other two sub-criteria with respect to the Dynamic criterion (2).

Existence of Maximum Sensitivity Region: for certain sensors, the thermister for example, there is a region in its resistance-temperature characteristic equation in which the sensitivity-the slope of the curve- is abnormally maximum. This maximum sensitivity region is considered a disadvantage in the sensor's characteristics because the sensor's behavior is extremely non-linear and unpredictable [7] which adds up sharply to inaccurate readings and difficulty of characterizing the resistance-temperature curve for that sensor. It is ranked (3).

Typical Fast Thermal Time Constant: this sub-criterion refers to how fast the sensor behaves in responding to a step change in the measured variable, the temperature in this case. In any sensing system, fast time response is regarded as an advantage for the sensor. This sub-criterion is assimilated to response time requirement and ranked inside Dynamic criterion (1).

Based on the relative ranks for the Dynamic sub-criteria, the following judgment matrix is established:

$$\boldsymbol{A}_{\text{Sub dyn.}} = [a_{ij}] = \begin{pmatrix} 1.0 & 2.0 & 0.1667 \\ 0.5 & 1.0 & 0.1667 \\ 6.0 & 6.0 & 1.0 \end{pmatrix}$$

Because the most important sub-criterion among the three Dynamic sub-criteria is the Typical Fast Thermal Time Constant and the least important sub-criterion is the Existence of Maximum Sensitivity Region sub-criterion, then it is fair to consider the Typical Fast Thermal Time Constant sub-criterion mid way between strongly and very strongly more important than the Existence of Maximum Sensitivity Region sub-criterion and thus entry

a_{16} is given a value 0.1667. After the above Dynamic sub-criteria judgment matrix has been introduced into the software, the software can then calculate the weights for individual Dynamic sub-criteria. Table 4-13 shows these weights.

Table 4-13 Dynamic sub-criteria weights.

Sub-Criteria Dynamic matrix:					
1	2	0.1667			
0.5	1	0.1667			
6	6	1			
			Stimulation Electronics	Maximum Sensitivity	Time Constant
Dynamic sub-criteria Weight Vector =			0.16019	0.10093	0.73888
Consistency Index = 0.02722					
Consistency Ratio = 0.04694					

Environmental Sub-criteria Relative Weights: the Environmental sub-criteria alongside their relative ranks are:

Typical Small Size: it is an advantage for a sensor to be small-sized for three reasons:

1- It becomes more similar to a point-measurement sensor. Point measurement sensors measure temperature more accurately than area measurement sensors because of their fastness in reaching thermal equilibrium with the sensed medium. Small-sized sensors reach thermal equilibrium faster because of their small thermal mass. Area measurement sensors, on the other hand, are slower in response and a temperature gradient arises through the different parts (points) of the sensor, hence you can see industrial products of area measurement sensors having temperature averaging capabilities. In general, area measurement sensors add up to the total sensor error because of the temperature gradient.

2- The small-sized sensor is faster in response.

3- The small-sized sensor fits better -in terms of size-in places of closed compartments and or vessels, and can be more customized in varying industrial environments.

The above considerations suggest the relatively high importance of the small size sub-criterion, but since corrosion resistance concerns us more-it appears frankly in the requirements list and small size doesn't- so it is comfortable to rank it among the five sub-criteria that comprise the Dynamic criterion (3).

Noise Immunity: noise immunity is an advantage for a sensor. A sensor that is prone to electrical and electromagnetic noise from neighboring electrical or electronic devices, such as motors for example, is considered a bad alternative. Noise immunity at the final end adds up to the total sensor's accuracy. It is ranked relative to the rest four sub-criteria (4).

Fragility-Durability Characteristics: it is an important sub-criterion that relates to the sensor's reliability. It is ranked (2).

High Thermal Gradient Environment: it is important for a sensor to withstand and cope with high thermal gradients that may be encountered in harsh environments-like chemical processes for example. Sensors that don't withstand temperature and pressure gradients often experience thermal cracks and the final damage of the sensor giving rise to maintenance and replacement costs. It is given rank (5).

Corrosion Resistance: it is an exceptionally important sub-criterion among other Environmental sub-criteria since it is frankly mentioned in the requirements list. It is given rank (1).

The Environmental sub-criteria judgment matrix is:

$$\boldsymbol{A}_{\text{Sub Env.}} = [\boldsymbol{a}_{ij}] = \begin{pmatrix} 1.0 & 3.0 & 0.3333 & 4.0 & 0.25 \\ 0.3333 & 1.0 & 0.25 & 3.0 & 0.2 \\ 3.0 & 4.0 & 1.0 & 5.0 & 0.5 \\ 0.25 & 0.3333 & 0.2 & 1.0 & 0.1667 \\ 4.0 & 5.0 & 2.0 & 6.0 & 1.0 \end{pmatrix}$$

Fragility-Durability Characteristics sub-criterion ranks (2) while Typical Small Size sub-criterion ranks (3), so it is safe to consider the Fragility-Durability Characteristics sub-criterion weakly more important than the Typical Small Size sub-criterion and thus entry a_{31} has a value 3. But since the Corrosion Resistance sub-criterion ranks (1) while the Fragility-Durability Characteristics sub-criterion and the High Thermal Gradient Environment sub-criterion rank (2) and (5) respectively then entries a_{53} and a_{54} can have in the above matrix values 2 and 6 respectively. After the Environmental sub-criteria judgment matrix has been introduced into the software, the software calculates the weights for the individual Environmental sub-criteria. Table 4-14 shows these weights.

Table 4-14: Environmental sub-criteria weights.

Sub-Criteria Environmental matrix:

1	3	0.3333	4	0.25
0.3333	1	0.25	3	0.2
3	4	1	5	0.5
0.25	0.3333	0.2	1	0.1667
4	5	2	6	1

	Small Size	Noise Immunity	Fragility-Durability	Thermal Gradient
Environmental sub-criteria Weight Vector =	0.15165	0.08646	0.28264	0.04767
	Corrosion Resistance			
	0.43158			

Consistency Index = 0.06347

Consistency Ratio = 0.05667

Others Sub-criteria Relative Weights: the Others sub-criteria alongside their relative ranks are:

Point or Area Measurement: it is an advantage for a sensor to be similar to a point-measurement sensor. Nonetheless, area measurement does not prevent use of area measuring sensors for temperature measurement. In fact area measuring sensors are widely industrially employed. This sub-criterion is ranked (3).

Manufacturing Variances: it is an advantage for a sensor to be homogenous and invariant from batch to batch during manufacturing. It is ranked (4).

NIST Standards Exist: NIST is an American body that is concerned about different standards for materials and manufacturing technologies, the abbreviation NIST stands for National Institute of Standards and Technology. It is considered an advantage for a sensor to be compliant with NIST standards, and this is what is sometimes referred to in reference books as interchangeability. Compliance with NIST standards (interchangeability) leads to a final better accuracy of the sensor. NIST standards sub-criterion is ranked (2).

Cost: in many low -accuracy demanding sensor selection situations, the cost sub-criterion has the preference to all other sensor's sub-criteria. The same applies if a group of sensor systems with large numbers of sensors are needed for a certain application, as is the case in the HVAC application. In this respect, it is an advantage for a sensor to have low price. However, cost should not be the final criterion that overbalances sensor's choice. In fact, for the long-run operability and reliability, other sub-criteria should not be overlooked or sacrificed in favor of cost. Cost is given rank (1). The Others sub-criteria judgment matrix is:

$$\boldsymbol{A}_{\text{Sub Others.}} = [\boldsymbol{a}_{ij}] = \begin{pmatrix} 1.0 & 3.0 & 0.5 & 0.25 \\ 0.3333 & 1.0 & 0.3333 & 0.2 \\ 2.0 & 3.0 & 1.0 & 0.3333 \\ 4.0 & 5.0 & 3.0 & 1.0 \end{pmatrix}$$

Because the NIST standards sub-criterion ranks (2) while the Manufacturing Variances sub-criterion ranks (4) then it is safe to consider NIST standards sub-criterion weakly more important than the Manufacturing Variances sub-criterion, and thus entry $\boldsymbol{a}_{32}$ can be given a value 3. After The Others sub-criteria judgment matrix has been introduced into

the software, the software calculates the weights of the Others sub-criteria. Table 4.15 shows these weights.

Table 4-15 Others sub-criteria weights.

Sub-Criteria Others matrix:

1	3	0.5	0.25
0.3333	1	0.3333	0.2
2	3	1	0.3333
4	5	3	1

	Point Area Measurement	Manufacturing Variances	NIST Standards	Cost
Others sub-criteria Weight Vector =	0.15750	0.07747	0.22913	0.53589

Consistency Index = 0.03752

Consistency Ratio = 0.04169

4.3.5.3 Criteria Weights Interpretation

In the prioritization scheme followed in this thesis, the 23 sub-criteria are prioritized as a whole against the overall goal, then they are sorted out into their parent criteria and a new prioritization scheme was followed inside each criterion yet making use of and depending on the previous 23 sub-criteria prioritization scheme. Now, for estimating the relative importance of one criterion relative to another with respect to the final goal, a third scheme was adopted. This scheme simply comprises assigning relative scores (weights) to each sub-criterion of the 23 sub-criteria after they have been ranked from (1) to (23) in the first stage. These relative scores are then aggregated (simply summed up) for sub-criteria that belong to the same parent criterion in order to obtain a total score for that criterion. By doing this, a score for each criterion is obtained. The next step is to relate these scores to each other-by simple division- of the four criteria aggregated score in order to obtain the relative weights of the criteria which will be entered in the criteria judgment matrix. This work was done separately, and the criteria judgment matrix for the Automotives application is obtained as follows:

$$A_{\text{criteria}} = [a_{ij}] = \begin{pmatrix} 1.0 & 4.0 & 3.0 & 4.0 \\ 0.25 & 1.0 & 0.5 & 1.0 \\ 0.3333 & 2.0 & 1.0 & 2.0 \\ 0.25 & 1.0 & 0.5 & 1.0 \end{pmatrix}$$

Looking at the above matrix, it is clear that the Static criterion is the most important criterion among the four criteria, it has a relative weight with respect to the Dynamic, Environmental, Others criteria of 4, 3, 4 (entries a_{12}, a_{13}, a_{14} in the matrix) respectively. This can easily be figured out if we recall that the Static criterion contains 11 sub-criteria among which lie the most important and the second most important of all 23 sub-criteria, the Maximum , Minimum Operating Temperature and the Long Term Stability and Accuracy sub-criteria. The second important criterion is evident to be the Environmental, having a weight of 2 relative to both Dynamic and Others criteria (entries a_{32} and a_{34}). The Dynamic and Others criteria are equally important criteria, this can be deduced from their relative weights with respect to each other (value 1 for entries a_{24} and a_{42}). Having been introduced into the software, the judgment matrix can then be used by the software to calculate the weights for the criteria with respect to the final goal. Table 4-16 shows these weights.

Table 4-16: Criteria weights for the Automotives application.

Criteria matrix:

1	4	3	4
0.25	1	0.5	1
0.3333	2	1	2
0.25	1	0.5	1

	Static	Dynamic	Environmental	Others
Criteria Weight Vector =	0.53637	0.12159	0.22045	0.12159

Consistency Index = 0.00687

Consistency Ratio = 0.00763

Having introduced all the matrices representing relative weights of the alternatives with respect to each sub-criterion, the relative weights of the sub-criteria with respect to the criteria, and the relative weights of the criteria with respect to the goal for a certain application in the software then the results of the software for any application case study the user applies to the software in the form of alternatives scores can easily be obtained by simply pressing the Select button on the second tab. Table 4-17 shows the results for the three sensors: the thermocouple, the thermister, and the RTD case study for the automotive catalytic converter application.

Table 4-17 Scores for the thermocouple, the thermister, and the RTD for the Automotives application.

Sensor	Score	Rank
Thermocouple	0.37849	1
Thermister	0.27560	3
RTD	0.34589	2

4.3.5.4 Variations in Components Weights for the Three Applications

The interpretation of the components weights in the previous three sections applies only to the Automotives application, which is set as the default application in the software as the user opens it. New sets of weights are introduced into the software that take care of the special requirements for the other two applications: the Chemical Process and the HVAC (Heating, Ventilating and Air Conditioning) applications.

The Chemical Process Application

The following are considerations pertaining to the Chemical Process application that have to be taken into account when varying the component weights of the Chemical

Process application:

1- Chemical processes are harsh environments in which the proposed sensor faces high rates of corrosion, and in many cases high temperatures and pressures and high

temperature gradients. The proposed sensor is then to be exceptionally corrosion with high thermal gradient resistance.

2- Normally, the rate of temperature change for the chemical medium in which the proposed sensor is to be put is high (< 1.0 ^{0}C/ min), and so the sensor should have fast response time to cope with fast varying medium temperatures.

3- Chemical Process applications usually need accurate temperature measurement and control because of the nature of the chemical reaction. Catastrophic loss of material, energy, equipment and /or human lives can be possible if temperature measurement and control were not maintained within narrow ranges of accuracy and precision.

4- Sensors used in chemical processes are normally enclosed into closed compartments and/or closed vessels or reactors, distillation columns, mixers, evaporators, heat exchangers…etc. hence the need for a small-sized sensor that fits into these enclosures.

Based on these requirements for the Chemical Process application, the following scheme is proposed to take care of these requirements and to create new weights for the Chemical Process application:

1- To account for the corrosion resistance and other environment-related requirements, the weight of the Environmental criterion entered in Table 4.16 for the Automotive application is increased by a percentage of 60 %. This means that its relative weights in the criteria judgment matrix for the Chemical Process application with respect to the other three criteria will increase.

2- To account for the fast response time requirement of the sensor and other dynamic-behavior related issues, the weight of the Dynamic criterion is increased by a percentage of 35 %. This means that its relative weights in the criteria judgment matrix with respect to the Static and Others criteria will increase.

Of course these increases to the Environmental and Dynamic criteria will be charged to the other two remaining criteria, namely, the Static and Others criteria in the same proportion that the Static criterion is more important than the Others criterion, i.e. these increases will be charged in a proportion of 4:1.

3- The Long Term Stability and Accuracy sub-criterion relative weights inside the 11X 11 Static sub-criteria judgment matrix will be increased by a factor of 1 relative importance on Saaty's scale relative to the rest of the eleven sub-criteria while the Static criterion weight with respect to the goal will remain unchanged.

4- The Typical Fast Thermal Time Constant sub-criterion relative weights inside the 3X3 Dynamic sub-criteria judgment matrix will be increased by a factor of 1 relative importance on Saaty's scale relative to the rest of the three sub-criteria.

5- The Typical small size, the Corrosion Resistance, and the High Thermal Gradient Environment sub-criteria relative weights in the 5X5 Environmental Parameters sub-criteria judgment matrix will be increased by a factor of 1 relative importance on Saaty's scale relative to the rest two sub-criteria but will not be increased one against each other.

Based on these amendments, the new Criteria judgment matrix for the Chemical Process application is as follows:

$$\boldsymbol{A}_{\text{criteria}} = [\boldsymbol{a}_{ij}] = \begin{pmatrix} 1.0 & 2.0 & 1.0 & 4.0 \\ 0.5 & 1.0 & 0.5 & 2.0 \\ 1.0 & 2.0 & 1.0 & 4.0 \\ 0.25 & 0.5 & 0.25 & 1.0 \end{pmatrix}$$

Note the essential change to the relative weights of the various components of the criteria matrix before the amendments for the Automotives application and after the amendments for the Chemical Process application. For example, the relative weight of the Static criterion with respect to the Environmental criterion, entry $\boldsymbol{a}_{13}$, was 3.0 before the

amendments indicating weak importance of the Static criterion relative to the Environmental criterion in the Automotives application. This weight drastically changed to 1.0 after the amendments for the Chemical Process application indicating equal importance of the two criteria with respect to the goal which makes sense in an application where corrosion and other detrimental effects are to be minimized. The relative weight of the Dynamic criterion with respect to the Static criterion, entry a_{21}, was 0.25 before the amendments indicating that Dynamic criterion is mid way between being weakly and strongly less important than the Static criterion in the Automotives application. This weight also changed remarkably to a value of 0.5 after the amendments indicating that the Dynamic criterion became mid way between being equally and weakly less important than the Static criterion in the Chemical Process application which makes sense for an application in which fast response is needed. Having introduced the above matrix into the software for the Chemical Process application, the new weights of the four criteria will be calculated. Table 4.18 shows the software criteria new weights values for the Chemical Process application versus their old values in the Automotives application for purposes of comparison in addition to % increase or decrease in each criterion value before and after the amendments.

Table 4-18: Criteria weights for the Chemical Process application.

Criteria Matrix (Chemical Process):

1	2	1	4
0.5	1	0.5	2
1	2	1	4
0.25	0.5	0.25	1

	Static	Dynamic	Environmental	Others
Automotives Criteria Weight Vector =	0.53637	0.12159	0.22045	0.12159
Chemical Process Criteria Weight Vector =	0.36363	0.18181	0.36363	0.09090
% increase or decrease =	- 32	+ 49	+ 65	- 25

Consistency Index (Chemical Process) = 0

Consistency Ratio (Chemical Process) = 0

The rest of the relative weights amendments other than amendments for the criteria matrix are introduced into the software. Table 4-19 shows the new weights for the Long Term Stability and Accuracy, Typical Fast Thermal Time Constant, Corrosion Resistance, Typical Small Size, and High Thermal Gradient Environment sub-criteria in the Chemical Process application versus their corresponding old values in the Automotives application in addition to % increase in their values.

Table 4-19: New sub-criteria weights for the Chemical Process application versus old values for the Automotives application and % increase in weights.

Sub-Criterion	Old value	New value	% increase
Long Term Stability and Accuracy	0.15040	0.19869	32
Typical Fast Thermal Time Constant	0.73887	0.76708	4
Corrosion Resistance	0.43157	0.47902	5
Typical Small Size	0.15165	0.17142	13
High Thermal Gradient Environment	0.04767	0.05228	10

Figure 4.6 shows a 3D-column chart depicting values for these sub-criteria in both Automotives and Chemical Process applications.

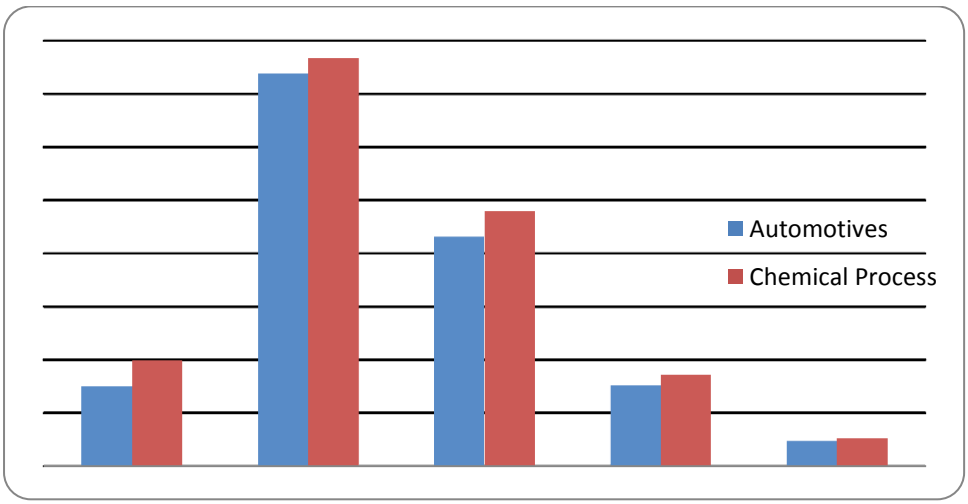

Figure 4.6: Values of sub-criteria in both Automotives and Chemical Process applications.

Having introduced all these amendments into the software project in the Chemical Process file, the results for the best alternative sensor among a set of nominated alternative sensors can be obtained. Table 4-20 shows the new scores for the three alternative sensors: the thermocouple, the thermister, and the RTD case study in the Chemical Process application. It also shows the old scores for the same sensors in the Automotives application for purposes of comparison.

Table 4-20: Scores for the three sensor case study in the Chemical Process application

Sensor	Automotives	Chemical	Rank (Chemical)
Thermocouple	0.37849	0.38179	1
Thermister	0.27560	0.26806	3
RTD	0.34589	0.35013	2

The HVAC Application

The following are the basic HVAC applications sensor selection considerations (see Appendix V [28]):

1- High accuracy because of the need to control the consumption of energy during heating and cooling within narrow tolerances for cost consumption purposes. Low accuracy sensors or sensor systems can be responsible for a large amount of energy loss and thus lager sums of wasted money.

2- Reliability and quality of the proposed sensor or system of sensors.

3- Initial cost, maintenance and replacement costs. The cost issue is very important in talking about the HVAC application because a group of sensors in large numbers (automated system of sensors) are normally needed.

4- Lead wire characteristics are also enhanced in the HVAC applications because of the electromagnetic noise peculiar to HVAC applications.

5- Consideration must be given to moisture, vibration, temperature extremes, condensation, vandalism, and other aggressive environments but to a less degree than in the case of Chemical Process application.

Based on these requirements, the following amendments are introduced into the components weights:

1- Since a high level of accuracy is needed in the HVAC application and since Static sub-criteria like: Temperature Curve, Self Heating, Long Term Stability and Accuracy, Extension Wires, Long Wire Runs from Sensor and Temperature Measurement all add to sensors accuracy then the scheme is to increase the Static criterion weight in the Automotives application by 20 % and to increase the relative weights of these six sub-criteria inside the 11X11 static sub-criteria matrix by a factor of 1 relative importance on Saaty's scale.

2- The relative weights of the Fragility and Durability, Noise Immunity, and Corrosion resistance sub-criteria will be increased inside the Environmental 5X5 sub-criteria judgment matrix by a factor of 2 relative importance on Saaty's scale to account for requirements of reliability, electromagnetic interference, and aggressive environments, respectively, but without increasing the overall Environmental criterion weight.

3- The Others criterion weight will be increased by a percentage of 40 % over its value in the Automotives application. In addition to this, the cost sub-criterion relative weights inside the 4X4 Others sub-criteria matrix with respect to the remaining three sub-criteria will be increased by a factor of 2 relative importance on Saaty's scale.

The criteria judgment matrix after these amendments for the HVAC application is:

$$\boldsymbol{A}_{\text{criteria}} = [\boldsymbol{a}_{ij}] = \begin{pmatrix} 1.0 & 9.0 & 6.0 & 4.0 \\ 0.1111 & 1.0 & 0.5 & 0.3333 \\ 0.1667 & 2.0 & 1.0 & 0.5 \\ 0.25 & 3.0 & 2.0 & 1.0 \end{pmatrix}$$

Table 4-21 shows criteria weights as they appear in the software after the amendments have been introduced for the HVAC application; it also shows their respective values in the Automotives application for purposes of comparison in addition to % increase or decrease in these sub-criteria values.

Table 4-21: Criteria weights for the HVAC application.

Criteria Matrix (HVAC):

1	9	6	4
0.1111	1	0.5	0.3333
0.1667	2	1	0.5
0.25	3	2	1

	Static	Dynamic	Environmental	Others
Automotives Criteria Weight Vector =	0.53637	0.12159	0.22045	0.12159
HVAC Criteria Weight Vector =	0.64295	0.06228	0.10835	0.18639
% increase or decrease =	+ 20	- 49	- 51	+ 53

Consistency Index = 0.00692

Consistency Ratio = 0.00769

The rest of the relative weights amendments other than amendments for the criteria matrix are introduced into the software. Table 4-22 shows the new weights for the Temperature Curve, the Self Heating Issues, the Long Term Stability and Accuracy, the Extension Wires, the Long Wire Runs from Sensor, the Temperature Measurement, the Fragility-Durability Characteristics, the Noise Immunity, the Corrosion Resistance, and the Cost sub-criteria new weights in the HVAC application versus their corresponding old values in the Automotives application in addition to percentage increase in these sub-criteria values.

Table 4-22: New sub-criteria weights for the HVAC application versus old values for the Automotives application.

Sub-Criterion	Old value	New value	% Increase
Temperature Curve	0.05379	0.06348	18
Self-Heating Issues	0.09777	0.11344	16
Long Term Stability and Accuracy	0.15040	0.18940	26
Extension Wires	0.03704	0.04448	20
Long Wire Runs from Sensor	0.01983	0.02295	16
Temperature Measurement	0.03355	0.03764	12
Fragility-Durability Characteristics	0.28264	0.30877	9
Noise Immunity	0.08646	0.10695	24
Corrosion Resistance	0.43158	0.45559	6
Cost	0.53589	0.63425	18

Figure 4.7 shows a 3D-column chart depicting the values of these sub-criteria for the HVAC and Automotives applications.

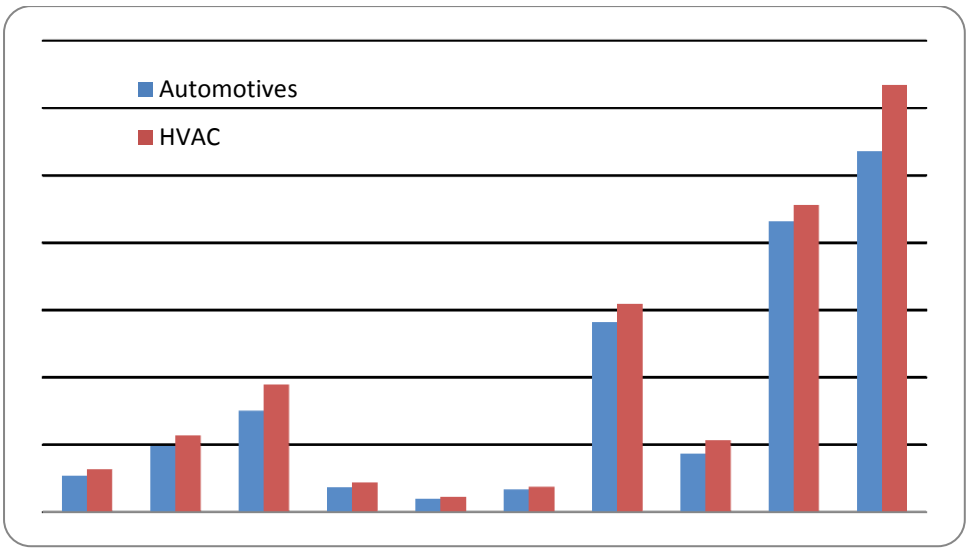

Figure 4.7: Values of sub-criteria in both Automotives and HVAC applications.

Having introduced all these amendments into the software project in the HVAC file, the results for the best alternative sensor among a set of nominated alternative sensors can be obtained. Table 4-23 shows the new scores for the three alternative sensors: the

thermocouple, the thermister, and the RTD case study in the HVAC application. It also shows the old scores for the same sensors in the Automotives application for purposes of comparison.

Table 4-23: Scores for the three sensor case study in the HVAC application.

Sensor	Automotives	HVAC	Rank (HVAC)
Thermocouple	0.37849	0.35968	1
Thermister	0.27560	0.28670	3
RTD	0.34589	0.35362	2

Figure 4.8 shows a 3D-column chart depicting values of the Static, Dynamic, Enironmental, and Others criteria weights in the Automtives, Chemical Process, and HVAC applications for purposes of comparison.

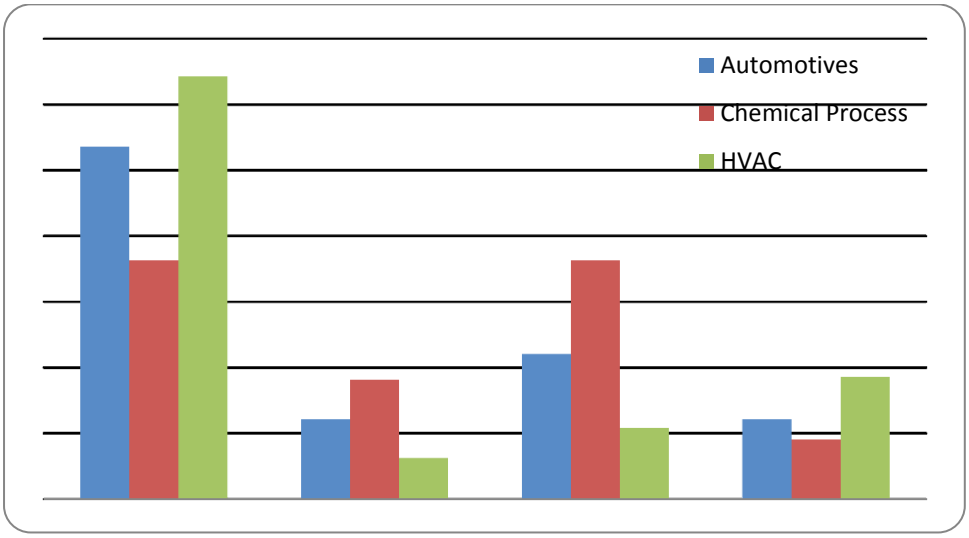

Figure 4.8: Values of criteria weights in the Automotives, the Chemical Process and the HVAC applications.

Figure 4.9 shows a 3D-column chart depicting final scores of the three sensors: the thermocouple, the thermister, and the RTD case study in the Automtives, Chemical Process, and HVAC applications.

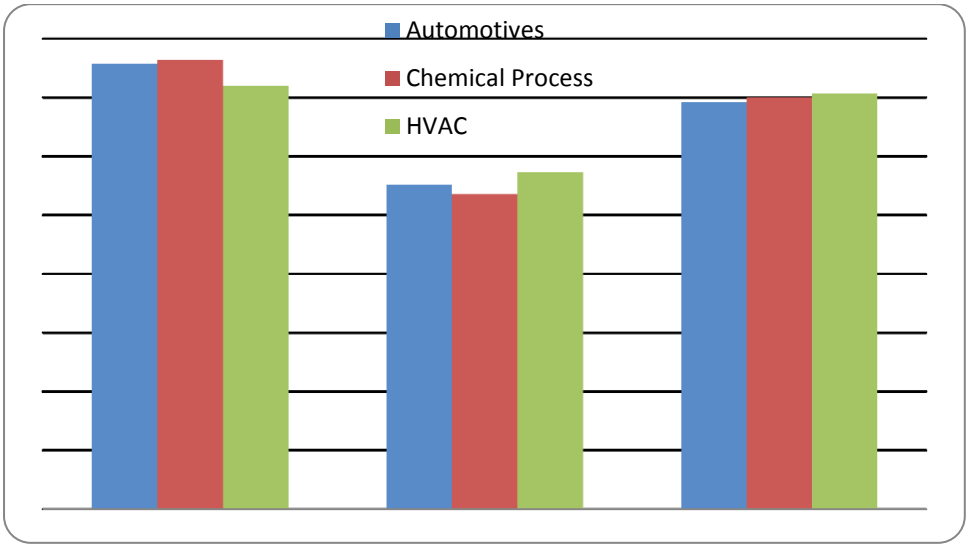

Figure 4.9: Values of sensors' final scores in the Automotives, the Chemical Process and the HVAC applications.

4.3.6 Components Weights Calculation

In this step, relative comparison weights of each checked alternative sensor against other checked sensors are retrieved by the system from the input values, and weights of all components in the hierarchal structure are consequently determined and calculated by the software. Most of these sub-criteria and therefore parent criteria in addition to the alternatives and the goal are separate and distinct entities and are therefore considered independent components such that AHP method can be used. Interdependencies are minimum between most criteria and can be assumed independent with minimum effects on the final judgment.

After the system retrieves the weights of the alternatives in the lower level, it aggregates them to obtain weights of upper immediate parent components in the immediate upper levels. Specifically speaking, this step consists of the following three sub-steps:

1- Starting from the twenty-three 7X7 matrices of relative weights of the alternatives, the software calculates the score of each alternative against each sub-criterion as was interpreted in the previous section.
2- Using the 11X11, 3X3, 5X5, 4X4 matrices of relative weights of the Static, Dynamic, Environmental, and Others sub-criteria against their respective parent criteria, and using the 4X4 matrix of relative weights of the four criteria against the final goal, the software calculates the scores of the different components in the hierarchy in the sub-criteria and criteria level.
3- Finally, the software, lumps scores of alternatives against sub-criteria from the first step and scores of different components from the second step and integrates them all to obtain the final contribution (score) of each alternative sensor against the goal. This score is the final outcome of the AHP method and is the measure of preference of the alternatives towards our final goal such that the alternative sensor with the largest score is considered the best (most preferred) and the one with the smallest score the worst (the least preferred) and values in between are arranged in preference according to descending order of score value.

4.3.7 Performing the Consistency Test

The software then computes the consistency index and the consistency ratio based on equations (3.8), (3.9), and (3.10).

4.3.8 Displaying the Final Results

After the user does the necessary selections of the intended application, the restrictions and the sensors and presses the Select button, the software then displays the final results on the software console. These results include:

1- The final scores of the selected sensors against the goal, these scores are shown vertically from top to bottom in the same order the selected sensors appear in on the second tab.
2- The list of all matrices representing the relative weights of the alternative sensors with respect to sub-criteria, the relative weights of the sub-criteria with respect to criteria, and the relative weights of criteria with respect to the goal
3- Values of consistency index and consistency ratio for the whole set of matrices.

The next chapter deals with applying the proposed software to the case study of choosing the best alternative sensor from among the three sensors: the thermocouple, the thermister, and the RTD in the automotive catalytic converter application.

After being introduced to the software, the user may want to look at the base code the program was built-in. This code is shown for the three applications in Appendix VI.

Chapter Five Case Study

5.1 Case Study Description

The case study which will be applied here to the software is the selection between three alternative sensors: the thermocouple, the thermister, and the RTD in the Automotives catalytic converter application.

5.2 Automotive Catalytic Converter Description

A catalytic converter is a device which chemically converts harmful exhaust gases, produced by the internal combustion engine as by-products of the fuel combustion process, into harmless carbon dioxide, water vapor, and nitrogen gas. Essentially, the catalytic converter is used to complete the oxidation process for hydrocarbons and carbon monoxide, in addition to reducing oxides of nitrogen (NOx) back to simple nitrogen and carbon dioxide.

The converter is constructed such that the converter shell contains a substrate material. There are two types of converter substrates: Pelletized, which consists of many small-sized ceramic pellets and Monolithic, which is a ceramic "honeycomb" material. The surface of the substrate material is coated with a thin film of precious metals (rhodium, platinum / palladium, and cerium) which acts as a chemical catalyst. Its function is to assist in the chemical reactions that are required to lower the emission levels to be within acceptable environmental regulations. As engine exhaust gases flow through the converter, they contact the coated surface which initiates the catalytic process. As exhaust and catalyst temperatures rise, the following reactions occur:

- Oxides of nitrogen (NOx) are reduced into simple nitrogen (N2) and carbon dioxide (CO2).
- Hydrocarbons (HC) and carbon monoxide (CO) are oxidized to produce water and carbon dioxide.

Figure 5.1 depicts a commercial converter and the various parts and chemical reactions that occur within the converter.

Catalyst operating efficiency is greatly affected by two factors; operating temperature and feed gas composition. The catalyst begins to operate at around 288 ^{0}C; however, efficient purification does not take place until the catalyst reaches at least 400 ^{0}C. Also, the converter feed gases (exhaust gases coming out of engine) must alternate rapidly between high CO content, to reduce NOx emissions, and high O2 content, to oxidize HC and CO emissions. To ensure that the catalytic converter has the feed gas composition it needs, a closed loop control system is designed to rapidly alternate the air/fuel ratio slightly rich (air-to-fuel mass ratio lower than 14.7: 1), then slightly lean (air-to-fuel mass ratio higher than 14.7: 1) of stoichiometry. By doing this, the carbon monoxide and oxygen content of the exhaust gas also alternates with the air/fuel ratio. Temperature sensors are used in the catalytic converter to measure the temperature of the inlet and outlet gas for two purposes: to indicate the maximum temperature the converter can tolerate before the substrate material melts, and for loop control purposes of the air/fuel ratio.

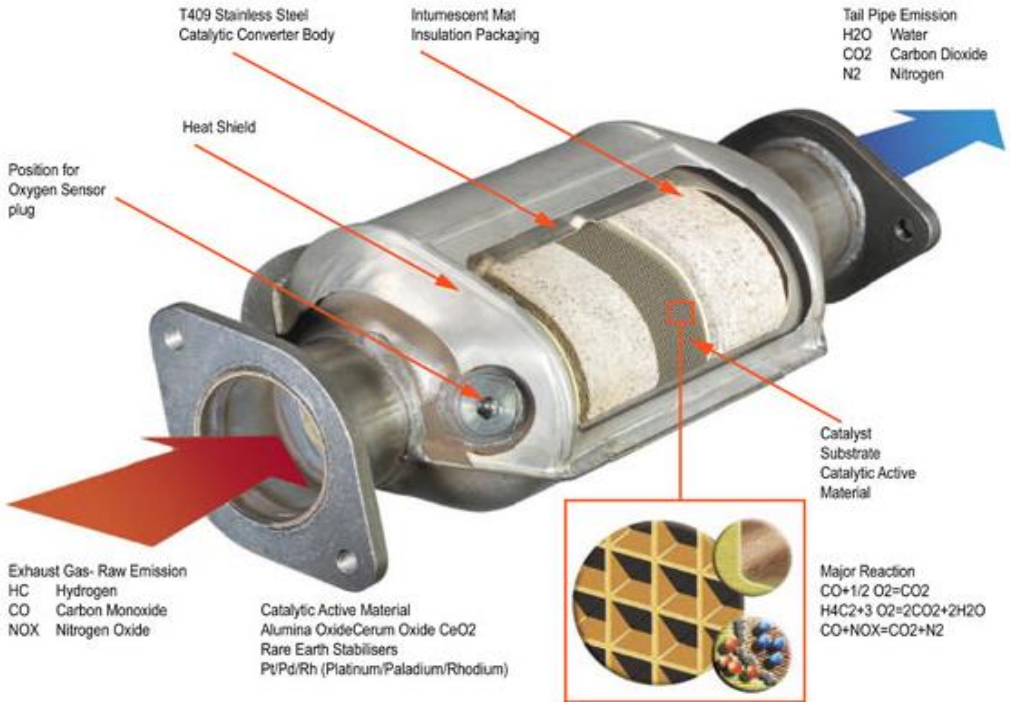

Figure 5.1: A commercial catalytic converter with the parts and chemical reactions that occur within the converter.

5.3 Application Operating Conditions

The Automotive catalytic converter application operates in the temperature range 500-750 ^{0}C (773-1023 K). The maximum temperature could reach in cases of malfunctioning and melting of the substrate material up to 870 ^{0}C (1143 K). The resolution of industrial sensors employed practically for the application is 1 % of the temperature range, i.e. (5-7.5) ^{0}C. The response time is 5-10 seconds. Normally employed sensors for the automotive application in industry are wide variety with wide range of customized features including: the thermocouple, the thermister, the RTD, the infrared pyrometer, the thermocouple pyrometer, the LCD pyrometer, the infrared laser sighting pyrometer …etc.

5.4 Applying the Software to the Case Study

After the application operating conditions have been determined, the user can now enter them into the software and get the result of the best sensor. More specifically, on the first tab the user chooses the application "Automotives", he or she chooses the nearest temperature range in the software to the application temperature range. The nearest temperature range on the first tab is 700-1150 K. The user then chooses the software nearest resolution to the application resolution, this is 1.0 ^{0}C as per provided by the software. Next, he or she specifies the response time, in this case it is chosen 5 seconds. Figure 5.2 depicts these choices. After the user completes his choices on the first tab he moves to the second tab where he checks in the intended sensors: the thermocouple, the thermister, and the RTD. Figure 5.3 shows these checked sensors. The user then them presses the Select button.

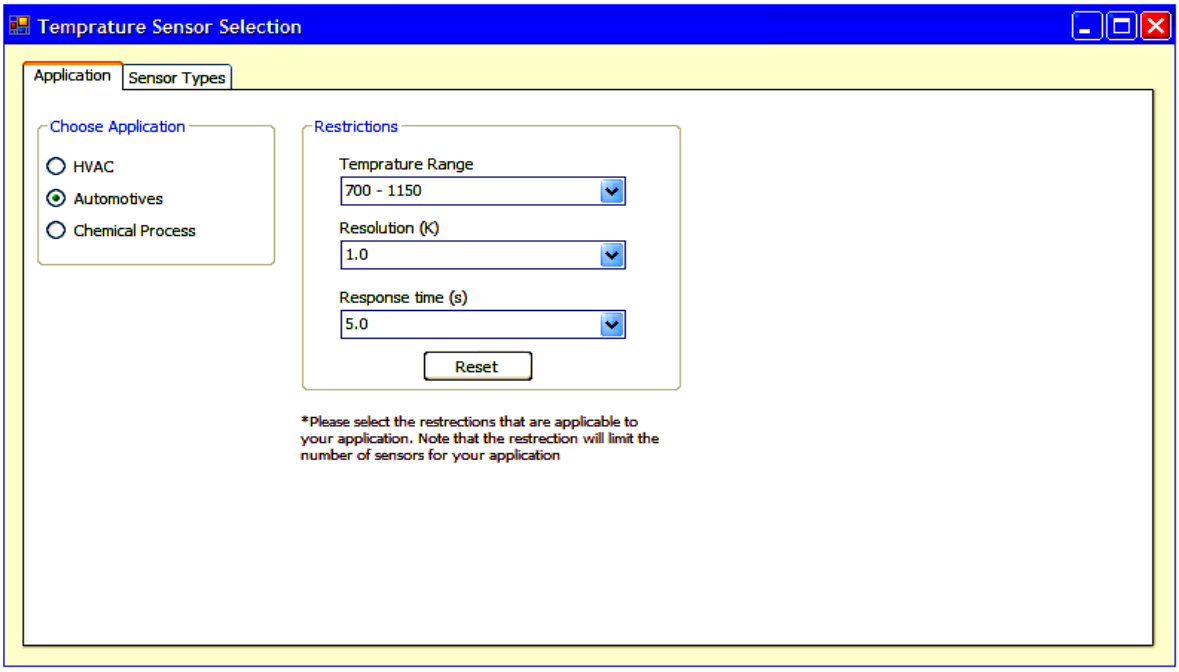

Figure 5.2: Choices on the first tab for the case study.

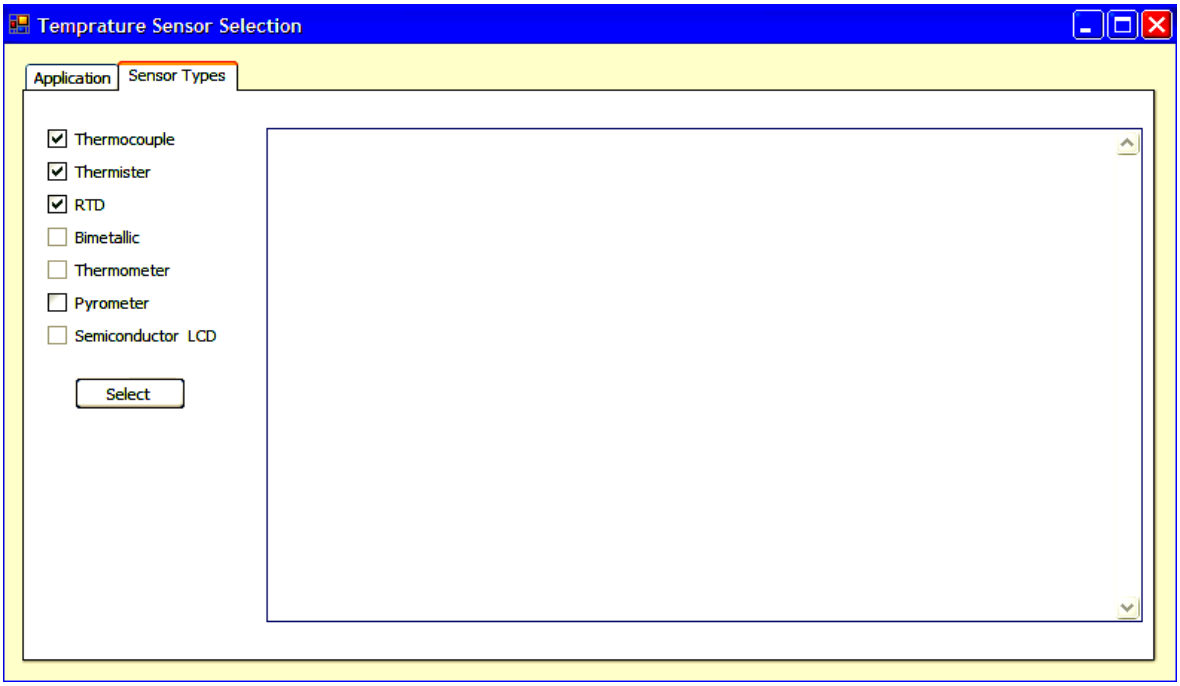

Figure 5.3: Checked in sensors on the second tab.

5.5 Software Results

Appendix VII shows the complete list of the software results for the three sensors: the thermocouple, the thermister, and the RTD automotive catalytic converter case study.

Table 5-1summarizes the three alternatives weights (scores) with respect to the 23 sub-criteria, the 4 criteria weights with respect to the goal, the synthesis weight (value) of the 23 sub-criteria towards the final goal, and the score of each alternative against each criterion.

Table 5-1: Weights of alternatives, sub-criteria, criteria and synthesis values for sub-criteria and the alternatives.

Criteria	Weights of Criteria	Sub-criteria	Weights of sub-criteria	Synthesis value	Thermocouple	Thermister	RTD
C1	0.53637	CS1	0.22119	0.11863	0.42858	0.14283	0.42858
		CS2	0.22119	0.11863	0.5	0.25	0.25
		CS3	0.05379	0.02885	0.25099	0.09602	0.65299
		CS4	0.09836	0.05275	0.06225	0.70131	0.23644
		CS5	0.09777	0.05244	0.65715	0.06825	0.27460
		CS6	0.15040	0.08067	0.086955	0.27371	0.63933
		CS7	0.05233	0.02806	0.09602	0.65299	0.25099
		CS8	0.03038	0.01629	0.07693	0.46154	0.46154
		CS9	0.01983	0.01063	0.19999	0.60000	0.19999
		CS10	0.01452	0.00778	0.62322	0.13729	0.23948
		CS11	0.03355	0.01799	0.09642	0.28422	0.61936
	Score of each alternative against first criterion				0.17481	0.15043	0.20743
C2	0.12159	CS12	0.16019	0.01947	0.62322	0.13728	0.23948
		CS13	0.10093	0.01227	0.46153	0.07693	0.46153
		CS14	0.73887	0.08983	0.62322	0.23948	0.13728
	Score of each alternative against second criterion				0.07378	0.02513	0.02268
C3	0.22045	CS15	0.15164	0.03342	0.53896	0.29726	0.16378
		CS16	0.08645	0.01905	0.09339	0.68529	0.22132
		CS17	0.28264	0.06230	0.65299	0.09602	0.25099
		CS18	0.04767	0.01050	0.68064	0.20141	0.11794
		CS19	0.43157	0.09513	0.08696	0.27371	0.63933
	Score of each alternative against third criterion				0.07557	0.05767	0.08720
C4	0.12159	CS20	0.15750	0.01915	0.53896	0.29726	0.16378
		CS21	0.07747	0.00941	0.09602	0.25099	0.65299
		CS22	0.22913	0.02786	0.44444	0.11111	0.44444
		CS23	0.53589	0.06519	0.46153	0.46153	0.07693
	Score of each alternative against second criterion				0.05369	0.04123	0.02667

Table 5-2 lists values of consistency index (CI) and consistency ratio (CR) for the matrices of the different components in the hierarchal structure.

Table 5-2: Criteria and sub-criteria factors used as basis for comparison between alternative sensors.

Criteria	Sub-Criteria	CI	CR
Static Criterion	Maximum Operating Temperature	0	0
CI = 0.08281	Minimum Operating Temperature	0	0
CR= 0.05208	Temperature Curve	0.00918	0.01583
	Sensitivity	0.03622	0.06225
	Self-Heating Issues	0.02218	0.03824
	Long Term Stability and Accuracy	0.02705	0.04663
	Typical Temperature Coefficient	0.00918	0.01583
	Extension Wires	0	0
	Long Wire runs from Sensor	0	0
	Measurement Parameter	0.00915	0.01578
	Temperature Measurement	0.04333	0.07471
Dynamic Characteristics	Stimulation Electronics required	0.00915	0.01578
CI = 0.02722	Existence of Maximum Sensitivity Region	0	0
CR = 0.04694	Typical Fast Thermal Time Constant	0.00915	0.01578
Environmental Parameters	Typical Small Size	0.00459	0.00791
CI = 0.06346	Noise Immunity	0.02710	0.04672
CR = 0.05666	Fragility-Durability Characteristics	0.00918	0.01583
	High Thermal Gradient Environment	0.01235	0.02129
	Corrosion Resistance	0.02705	0.04663
Others	Point or Area Measurement	0.00459	0.00791
CI = 0.03752	Manufacturing Variances	0.00918	0.01583
CR = 0.04169	Standards exist	0	0
	Cost	0	0
The four-criteria matrix		CI = 0.00687	CR = 0.00763

Table 5-3 shows the final scores for the three temperature sensors, the one with the largest score is the best, the thermocouple, with a score of 0.37849 and rank 1, the second ranked sensor is the RTD with a score of 0.34589, and the least preferred sensor is the thermister with a score of 0.27560. Note that the scores are arranged from top to bottom in the same order the checked alternative sensors appear in.

Table 5-3: The software final results: the three sensors scores.

Sensor	Score	Rank
Thermocouple	0.37849	1
Thermister	0.27560	3
RTD	0.34589	2

5.6 Sensitivity Analysis

This section tackles the Sensitivity Analysis applied to the case study using the software. Sensitivity Analysis for any system of input and output dependent variables refers to intended variations or perturbations in the input variables of the system for the purpose of monitoring changes in the output dependent variables. In any system, Sensitivity Analysis gives deeper understanding of the relationships that govern the system and allows for developing and optimizing the system and avoiding critical conditions which make the system unpredictable. In this thesis four variations were made and the results studied: variations in the alternative relative weights with respect to the other alternatives in the 23 matrices, variations in the relative weight of the criteria and sub-criteria, variation in the application, and variations in the number of alternatives that fit a certain application. All these variations will be applied using the software, and to simplify the situation they will be applied based on the case study described in the previous sections.

5.6.1 Case 1: Alternative Weights Variation

According to Table 5.3, the thermocouple alternative is the best alternative, having a score of 0.37849 while the second preferred sensor is the RTD having a score of 0.34589 and the worst choice is the thermister with a score of 0.27560. In this section the relative weight of the RTD will be increased by 1 relative weight unit on Saaty's scale. This means adding 1 to each entry in all the 23 matrices where the RTD appears, i.e. the addition would occur to the third row of each of the 23 alternative matrices. Then the new scores of the alternatives are monitored and discussed. This operation has been performed in the software, and the new scores of the alternative sensors were as in Table 5-4.

Table 5-4: Case 1 Sensitivity Analysis Results.

Sensor	Old Score	New score	New Rank
Thermocouple	0.37849	0.35457	2
Thermister	0.27560	0.24957	3
RTD	0.34589	0.39585	1

Figure 5.4 shows a 3D-column chart manifesting the old and new scores of the sensors.

Figure 5.4: Case 1 Sensitivity Analysis results.

5.6.2 Case 2: Sub-criterion Relative Weights Variation

In this case of Sensitivity Analysis the variation will be made to the Long Term Stability and Accuracy sub-criterion inside the Static criterion and the scores monitored. The relative weights of this sub-criterion among the 11 Static sub-criteria will be increased by a factor of 1 on Saaty's scale while the Static criterion overall score would remain unchanged to ensure that the change in the results is due to this sub-criterion effect and not from others. The procedure is merely to increase the whole values of the sixth row of the 11X11 Static sub-criteria matrix by one and the corresponding necessary changes in the reciprocals. This was done in the software, although not shown here, and the new scores of the three alternatives were as in Table 5-5.

Table 5-5: Case 2 Sensitivity Analysis Results.

Sensor	Old Score	New score	New Rank
Thermocouple	0.37849	0.37016	1
Thermister	0.27560	0.27616	3
RTD	0.34589	0.35368	2

Figure 5.5 shows the 3D-column chart depicting these results.

Figure 5.5: Case 2 Sensitivity Analysis results.

5.6.3 Case 3: Dynamic Criterion Relative Weights Variation

In this case of Sensitivity Analysis the relative weight of the Dynamic criterion is increased by a factor of 1 relative importance on Saaty's scale while the remaining criteria weights were kept unchanged. The results for this case were as in Table 5-6.

Table 5-6: Case 3 Sensitivity Analysis Results.

Sensor	Old Score	New score	New Rank
Thermocouple	0.37849	0.39531	1
Thermister	0.27560	0.27022	3
RTD	0.34589	0.33446	2

Figure 5.6 shows the new scores.

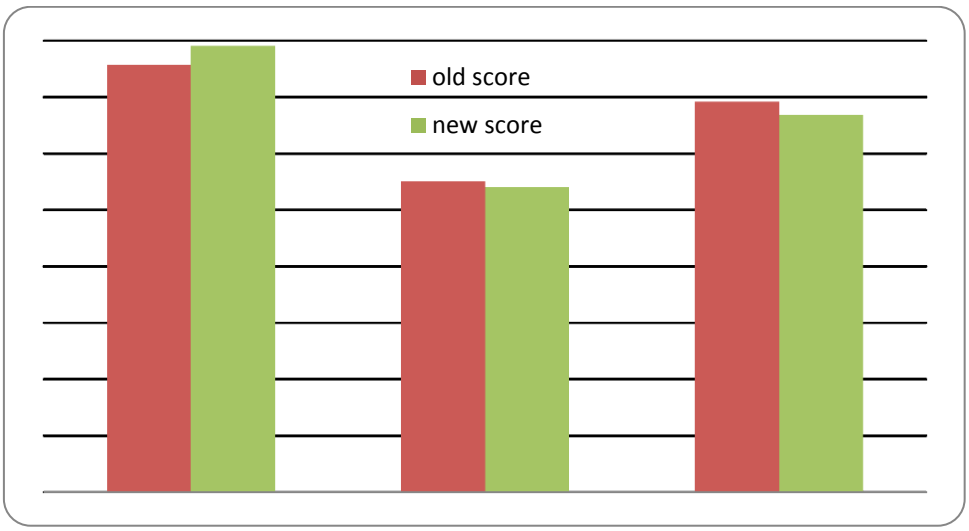

Figure 5.6: Case 3 Sensitivity Analysis results.

5.6.4 Case 4: Changing the Application

The three sensors: the thermocouple, the thermister, and the RTD case study is applied to the three different applications: Automotives, Chemical Process, and the HVAC applications, and the variations in the alternatives scores monitored. Table 5-7 shows the score of the three sensors against each application.

Table 5-7: Scores of the three sensors in the three applications.

Sensor	Automotives	Chemical Process	HVAC
Thermocouple	0.37849	0.38179	0.35968
Thermister	0.27560	026806	0.28670
RTD	0.34589	0.35013	0.35362

5.6.5 Case 5: Increasing Number of Sensors

In this case, the results are monitored upon introducing a new viable alternative sensor. In other words, scores for the three sensors case study are compared to those obtained when the pyrometer for example, is introduced among the alternative sensors and results

discussed. The scores for the four sensors: the thermocouple, the thermister, the RTD, and the pyrometer case are as in Table 5-8.

Table 5-8: Case 5 Sensitivity Analysis Results.

Sensor	Old Score	New score	% decrease (score)	New Rank
Thermocouple	0.37849	0.26910	29	1
Thermister	0.27560	0.20988	24	4
RTD	0.34589	0.26403	24	2
Pyrometer	-	0.25697	-	3

Figure 5.7 shows the results for this case.

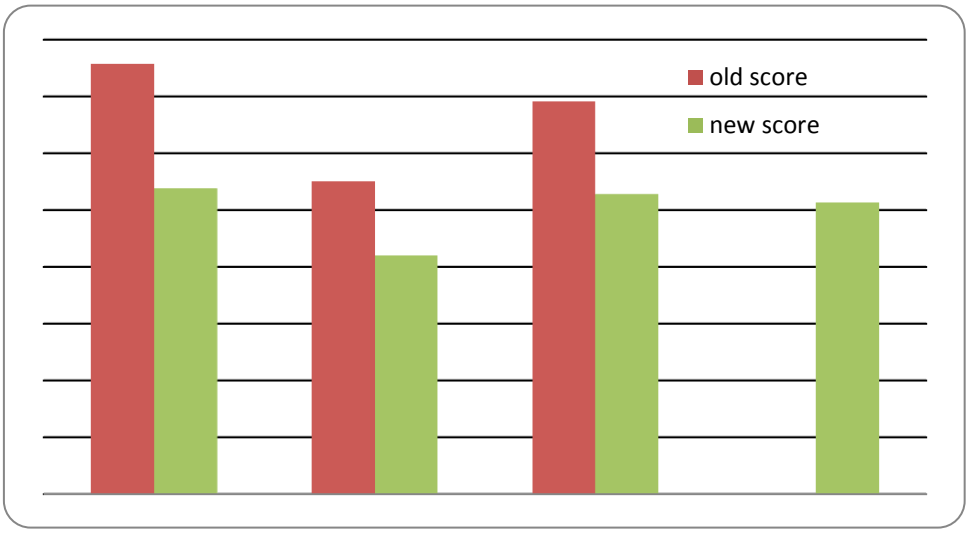

Figure 5.7: Case 5 Sensitivity Analysis results.

The next Chapter deals with the discussion part of the results presented in chapter 4, results presented in the case study, and results presented in the Sensitivity Analysis section.

Chapter Six Discussion of Results

6.1 Chapter Four Discussion

6.1.1 Alternatives Weights Discussion

It can be seen from Table 4.2 that the best scoring sensors against the Maximum Operating Temperature sub-criterion in the automotive catalytic converter application are the thermocouple and the RTD while the worst scoring is the thermister. This is because the first two sensors have the closest maximum operating temperature to the catalytic converter operating temperature while the thermister has the farthest operating temperature from that of the catalytic converter.

It can be seen from Table 4.3 that the best scoring sensor against the Temperature Curve sub-criterion in the automotive catalytic converter application is the RTD with a weight of 0.65299 while the worst scoring is the thermister with a weight of 0.09602, and that the thermocouple comes in between with a weight of 0.25099. This is because the RTD has the most linear Temperature-resistance relationship while the thermister has the most non-linear relationship and the thermocouple has good linearity relationship. The value of the consistency ratio is 0.01583 lying within acceptable limits indicating coherence and consistency in decision maker's judgments of the alternatives relative weights.

It can be seen from Table 4.4 that the best scoring sensor against the Sensitivity sub-criterion in the automotive catalytic converter application is the thermister with a weight of 0.70131 while the worst scoring is the thermocouple with a weight of 0.06225, and that the RTD comes in between with a weight of 0.23644. This is easily understood if we see sensitivity values for the three sensors and remember that the most sensitive of all three

sensors is the thermister and the least of the three is the thermocouple, and with RTD sensitivity value in between.

It can be seen from Table 4.5 that the best scoring sensor against the Self Heating sub-criterion in the automotive catalytic converter application is the thermocouple with a weight of 0.65715 while the worst scoring is the thermister with a weight of 0.06825, and that the RTD comes in between with a weight of 0.27459. This makes sense because the thermocouple experiences the least amount of self-heating while the thermister experiences much self-heating. The RTD, on the other hand, experiences moderate levels of self heating issues. The consistency ratio for the Self Heating sub-criterion matrix is 0.03824 which falls within acceptable limits and indicates consistent decision maker judgments.

It can be seen from Table 4.6 that the best scoring sensor against the Small Size sub-criterion in the automotive catalytic converter application is the thermocouple with a weight of 0.53896 while the worst scoring is the RTD with a weight of 0.16377, and that the thermister comes in between with a weight of 0.29725. This is understandable because the thermocouple is the smallest-sized sensor while the RTD is largest. The thermister's size, on the other hand, lies in between.

It can be seen from Table 4.7 that the best scoring sensor against the Time Constant sub-criterion in the automotive catalytic converter application is the thermocouple with a weight of 0.62323 while the worst scoring is the RTD with a weight of 0.13729, and that the Thermister comes in between with a weight of 0.23948. This makes sense because the thermocouple is the fastest sensor among all three sensors while the RTD is the slowest sensor. The thermister, on the other hand, has moderate value of response time.

It can be seen from Table 4.8 that the best scoring sensor against the Long Term Stability and Accuracy sub-criterion in the automotive catalytic converter application is the RTD with a weight of 0.63933 while the worst scoring is the thermocouple with a weight of

0.08695, and that the thermister comes in between with a weight of 0.27371. This can be figured out since the thermocouple is the least accurate of the three sensors while the RTD is the most accurate. The thermister, on the other hand, retains moderate levels of accuracy.

The results presented in Table 4.9 in terms of sensors weights are the same value as those presented in Table 4.8 and suggest the preference of the RTD amongst the three sensors with respect to corrosion resistance capability.

The results presented in Table 4.10 concerning the sensors Cost sub-criterion are rational in the sense that both the thermocouple and the thermister are relatively low cost alternatives and thus their weights are 0.46153 while the RTD is a very expensive alternative owing to a weight of only 0.07693.

6.1.2 Sub-criteria Weights Discussion

According to Table 4.12, the top most five important sub-criteria that make up the Static criterion in a descending order of importance, except for the first two, are: the Maximum Operating Temperature, the Minimum Operating Temperature, the Long Term Stability and Accuracy, the Sensitivity, and the Self-Heating Issues sub-criteria having weights of: 0.22120, 0.22120, 0.15040, 0.09837, and 0.09777 respectively with a total sum importance for the five sub-criteria with respect to the whole Static criterion of 0.78894. In fact, these are the basic important sub-criteria that make up Static behavior a sensor. As is explicit in the table, the consistency ratio is 0.05209 which is within acceptable limits and which indicates coherent decision maker's judgments on the Static sub-criteria relative weights.

According to Table 4.13, the most important sub-criterion that almost determines the sensor's dynamic behavior and accounts for 74 % of the total Dynamic criterion weight is the Typical Fast Thermal Time Constant having a weight of 0.73888 with respect to the Dynamic criterion.

According to Table 4.14, the top most two important sub-criteria that make up the Environmental criterion are: the Corrosion Resistance and the Fragility-Durability Characteristics sub-criteria having weights of: 0.43158 and 0.28264 respectively. These two sub-criteria comprise together about 71 % of the total Environmental criterion weight. In fact, these are the basic important sub-criteria that stand for sensor's resistance to environment. As can be noticed in the table, the consistency ratio is 0.05667 which is within acceptable limits indicating consistent decision maker's judgments on the Environmental sub-criteria relative weights.

According to Table 4.15, the top most two important sub-criteria that make up the Others criterion are: the Cost and the Existence of NIST Standards sub-criteria having weights of: 0.53589 and 0.22913 respectively. These two sub-criteria comprise together about 77 % of the total Others criterion weight. As can be seen in the table, the consistency ratio is 0.04169 which is within acceptable limits indicating consistent decision maker's judgments on the Others sub-criteria relative weights.

6.1.3 Criteria Weights Discussion

As Table 4.16 shows, the most important criterion in the selection of any temperature sensor in the Automotives application is the Static criterion with an overall score towards the goal of 0.53637. Static criterion pertains to those static qualities that are inherent in the sensor architecture and that relate to the basic technical characteristics that make a sensor. In the light of this, the score makes sense. On the other hand, the score of the Environmental criterion is 0.22045, suggesting a second-importance place of the criterion after the Static criterion. This also makes sense and matches well with view of experts in the field of sensors who state that the choice of any temperature sensor is dictated by the technical qualities that the sensor has to meet on the first scale, and on the environmental considerations, or alternatively, the medium characteristics that the sensor will be placed in

on the second scale. The relative weight (importance) of the Static criterion with respect to the Environmental criterion can be obtained from the criteria judgment matrix or from simply dividing the two scores. This is a factor of almost 2.43 which can be considered fair value, it is not too high, ignoring the importance of the Environmental criterion nor is it too small ignoring the more important Static criterion. Finally, the Dynamic and Others criteria came last important informing that response time and other dynamic response behavior-related characteristics are just third place in determining best temperature sensor with a weight of almost 0.12159 for each against the Static criterion.

6.1.4 Alternatives Final Scores Discussion

The thermocouple alternative is the best choice (rank 1) for the automotive catalytic converter application in the three sensors case study with an overall score of 0.37849 as Table 4.17 suggests. The second preferred alternative according to the same table is the RTD with an overall score of 0.34589 while the thermister comes last preference with an overall score of 0.27560. These results can be matched generally with views of experts in the field who state that almost the best sensor alternative for any application is just the thermocouple. Thermocouple is the simplest to install, the least expensive, the smallest size, the most durable and reliable, the fastest, the least interface electronic circuits-demanding senor of all or even it does not electronic devices at all. It retains reasonable accuracy and even good in many low accuracy-demanding applications, as is the case in the automotive catalytic converter, in addition to it experiences no self heating. It is a point measurement sensor with well-established traceable NIST standards. All in all, it is the best. The second best choice, the RTD, retains many of the good qualities that the thermocouple has, but it suffers from serious drawbacks such as: fragility, high cost, relatively slow response time, very low to low self heating issues, large size, and because it is an area measurement sensor it suffers from effects of high thermal environment

temperature gradients. Needless to say, that the thermister comes last because of the many drawbacks it shares with the RTD besides the high level of self heating issues it experiences, its non-standardized technical data owing to a larger amount of uncertainty in its measurements, and the manufacturing variances that accompany their use.

It should, however, be stated here that the thermocouple and the RTD final scores are close to each other (0.37849 is close to 0.34589) which poses a challenge in discriminating between the relative preferences of the thermocouple to the RTD and a challenge to the extent to which the thermocouple remains preferable to the RTD, i.e. if, for certain temperature measurement application, input values to the software in terms of components weights were revised then to what extent the thermocouple remains first preference and the RTD the second. This challenge can be resolved by means of Sensitivity analysis which reveals our system robustness and solidity, this work was done in the sensitivity section in the previous chapter. In general, other decision making problems that employ AHP and that contain alternatives scores well far apart from each other are more explicit and obvious in denoting the preference of the alternatives. For example, the preference of the first alternative in a certain decision problem having three alternative scores: 0.50, 0.30, and 0.20 is clearer and more obvious having a value 0.5 well far apart from the second alternative score 0.3. Large amount of input variations will need to be passed before the preference order between say the first and the second alternatives changes.

6.1.5 Chemical Process Weights Discussion

According to Table 4.18, the weights of the criteria components for the Chemical Process application are: 0.36363, 0.18181, 0.36363, and 0.09090 for the Static, Dynamic, Environmental, and Others criteria, respectively. We can glimpse the exceptional importance of the Environmental and Dynamic criteria in the Chemical Process application by noticing the drastic increase in their weights relative to their weights in the Automotives

application. The weight of the Environmental criterion has increased from 0.22045 to 0.36363 with a percentage increase of 65 % while the Dynamic criterion weight has increased from 0.12159 to 0.18181 with a percentage increase of 49 %. Also notice the dramatic increase in their relative weights with respect to the other two remaining criteria. For example, in the Automotives application, the relative weight of the Environmental criterion with respect to the Static criterion is 0.412 (the reciprocal of 2.43) indicating that the Static criterion is more important than the Environmental criterion. Now, the relative weight has changed to 1.0 indicating that the Environmental criterion became equal importance in the Chemical Process application with the Static criterion. Moreover, the relative weight of the Static criterion with respect to the Dynamic criterion was 4.41 (0.53637/0.12159) in the Automotives application. This relative weight has now changed to 2.0 in the Chemical Process application.

Under these new weights for the Chemical Process application, the three sensors: the thermocouple, the thermister, and the RTD would score differently against the overall goal. The new scores for the three sensors case study presented in Table 4.20 reveal an increase in the final thermocouple and RTD score and a decrease in the final score of the thermsiter. This is because the thermocouple fits slightly better with respect to the other two sensors in terms of response time, small size, and high thermal gradient environment resistance. However, since these characteristics are relatively minor in determining sensor's overall performance, i.e. the rest of the 23 sub-criteria are far more important than them then the increase in the thermocouple final score came small (the score changed only from 0.37849 to 0.38179). The increase in the RTD final score came also because of increased suitability of the RTD in meeting the special requirements pertaining to the Chemical Process application. This increased suitability is mainly attributed to sub-criteria like: RTD's excellent stability and accuracy and its corrosion resistance. The thermister

experienced a decrease in its final score because of its many drawbacks that force it to retreat against these new requirements of the Chemical Process application. To nominate some: thermister's relatively slow response time, medium stability and accuracy, decreased thermal gradient resistance, and non-existence of traceable standards.

6.1.6 HVAC Weights Discussion

According to Table 4.21, the weights of the criteria components for the Chemical Process application are: 0.64295, 0.06228, 0.10835, and 0.18639 for the Static, Dynamic, Environmental, and Others criteria, respectively. We can notice the increase in the importance (weights) of the Static and the Others criteria in the HVAC application relative to their corresponding weights in the Automotives application. For example, the Static criterion weight has increased from 0.53637 to 0.64295 with an increase of about 20 %, and the increase in the Others (mainly cost) criterion is 53 %. This increase came at the expense of the Environmental criterion which has decreased from 0.22045 to 0.10835 with a percentage decrease of 51 % and at the expense of the Dynamic criterion weight has decreased from 0.12159 to 0.06228 with a percentage decrease of 49 %. The Static criterion remained the most important but farther more important than the rest criteria-except for the Others criteria. For example, it became more important than the Dynamic criterion with relative importance (weight) 0.64295/0.06228 which is a value around 10.3, a drastic change from its corresponding value in the Automotives 4.41 (0.53637/0.12159). It also became more important than the Environmental criterion by a factor of around 6.0. However, its importance against the Others criterion has decreased from 4.41 (0.53637/0.12159) in the Automotives application to 3.45 (0.64295/0.18639) in the Chemical Process application.

The new scores for the three sensors case study in the HVAC application presented in Table 4.23 reveal an increase in the final Thermister and RTD score and a decrease in the

final score of the thermocouple. This is because the thermocouple scores badly on issues concerning Stability and Accuracy besides its bad corrosion resistant behavior. The RTD score has increased, on the other hand, because of its excellent stability and accuracy characteristics besides its good corrosion resistance performance. The thermister score has also increased because of its relatively low cost, excellent noise immunity and good corrosion resistance characteristics. Although the overall thermocouple score has decreased, it remained the most preferred alternative with a final score 0.35968, however, the RTD final score became very close to that of the thermocouple with a value 0.35362.

6.2 Sensitivity Analysis Discussion

6.2.1 Case 1 Discussion

It can be clearly seen from Table 5.4 that increasing the relative weights of the RTD alternative in the 23 sub-criteria matrices by a factor of 1 relative importance on Saaty's scale resulted in dominance of the RTD alternative over the thermocouple alternative, i.e. the thermocouple was the most preferred sensor choice before the increase while the RTD became the most preferred after the increase was employed to the system. This reveals and confirms the challenging decision situation when the scores of alternatives obtained by AHP fall close to each other and slightly apart, in which case the decision maker cannot decide sharply of the preference of one alternative to the other, rather, the close-scored alternatives are almost the same preference.

6.2.2 Case 2 Discussion

It can clearly be seen from Table 5.5 that although increasing the relative weights of the Long Term Stability and Accuracy sub-criterion by a factor of 1 on Saaty's scale has decreased the final score of the thermocouple alternative and has increased the final score

of the RTD alternative it did not change the preferences (ranks) of the three alternatives and that the thermocouple remained the most preferred (rank 1).

6.2.3 Case 3 Discussion

It can be clearly seen from Table 5.6 that increasing the Dynamic criterion relative weight by a factor of 1 relative importance on Saaty's scale has increased the thermocouple final score and decreased the thermister and the RTD final scores, this is because the thermocouple scores the best on the response time sub-criterion. This change also made the preference of the thermocouple to the RTD more distinct and sharp. Now the thermocouple final score increased from 0.37849 to 0.39531 and the RTD score decreased from 0.34589 to 0.33446. The distance between the two alternatives before the change was 0.04403 (0.37849-0.33446) has enlarged to 0.06085 (0.39531-0.33446) indicating sharper decision of the thermocouple preference to the RTD.

6.2.4 Case 4 Discussion

Results of Table 5.7 confirm the view of experts that not only does an alternative temperature sensor selection depend on its inherent characteristics but also it depends on the specific application and the peculiar environment (medium) the sensor is to be put in. The table also evidently reveals the increased suitability of the RTD and the decreased suitability of the thermocouple to the HVAC application temperature sensing. This is due to the fact that the RTD is the best choice with regard to stability and accuracy characteristics, while many factors gather to worsen the thermocouple choice in this regard. The final score of the RTD in the HVAC application is 0.35362 became very close and strong rival a value to the value of the final score of the thermocouple 0.35968, suggesting both the thermocouple and the RTD are almost the most preferred sensors in the HVAC application.

6.2.5 Case 5 Discussion

As table 5.8 shows the introduction of a new candidate sensor can totally change the scene. When the pyrometer is introduced into the set of alternative sensors available for the selection process, it came third place preferred with a strong score 0.25697 (this value is comparable to those of the thermocouple's and the RTD's, 0.26910 and 0.26403 respectively), and the thermister choice retreated to a fourth place preference. All the sensors' scores: the hermocouple's, the thermister's, and the RTD's have decreased, but the decrease experienced by the thermocouple was the largest, about 29 %, this indicates that the introduction of the pyrometer was at the expense of the thermocouple to a larger degree than it was to the thermister and the RTD (decrease in their final score both was only 24 %).

6.3 Conclusions

This study presents one new addition to the multitude of the Analytic Hierarchy Process (AHP) applications and fields of use. The advantage of AHP method implementation in selecting the optimum temperature sensor in a certain application is that the multi-criteria decision making process is based on objective break down of the whole decision problem into a hierarchy of multiple layers (levels) that can be further broken down into low-leveled sub-layers each of which is being well defined and given an objective weight that can be integrated through the whole hierarchy to obtain an objective evaluation of the alternative candidate sensors under study rather than the decision problem is based upon one level of assessment and is subject to subjective evaluation of the selection by decision makers and expertise in the field. This study highlighted the evaluative criteria and sub-criteria that relate to the selection of temperature sensors. Those criteria with high weights through the hierarchy can be regarded as being the most important and critical in

evaluation of best candidate temperature sensors and can be lumped together in a bundle and may be used as first assessment or screening stage for the selection process in other situations. One more advantage of AHP method in the selection of temperature sensors is that it has the capability to handle qualitative (verbal) as well as quantitative judgments of the alternatives and reflect these judgments into measurable quantitative final scores when ranking the alternatives. The outcome of the study in terms of alternatives final scores not only gives a rank to the candidate alternative sensors, but also gives a quantitative measure of the degree of dominance of one alternative over the others. This dominance or preference, of say the best alternative sensor, the thermocouple in the case study presented in this thesis, and inferiority of the least preferred alternative sensor, the thermister in this case, was further tested by means of sensitivity analysis to investigate to what degree the best alternative sensor remains dominant and the inferior sensor remains inferior. Inputs to the sensitivity analysis problem were variations in criteria and sub-criteria weights and variations in the expert's evaluation of the relative weights for one alternative sensor against the 23 sub-criteria in addition to variations in the application the nominated sensors are to be used in and variations in the sensors final scores due to the introduction of a new candidate sensor. The results showed the robustness of the proposed work and software to the variations carried out in all cases Sensitivity Analysis except for the case of revising the expert's evaluation of the relative weights of one alternative sensor with regard to other alternative sensors against the 23 sub-criteria. This challenge can be circumvented if we notice the closeness of the most preferred sensor, the thermocouple, final score to that of the second preferred alternative sensor, the RTD's. This closeness in final scores reveals the unique challenge that is inherent in the decision problem itself, the best sensor, and not the proposed method nor the proposed computer program. The selection of the best temperature sensor decision problem is confusing and problematic in

itself owing to the very contradicting features that are present in the different alternatives. You may find the best durability, response time, small size, and cost in the thermocouple alternative but you will be, on the other hand, frowned when you confront the several defects of the same alternative, the thermocouple, exemplified in relatively low accuracy, low corrosion resistance, extension wires problems, …etc. The same applies to the RTD, you will find some merits you are looking for in the RTD but always you will find multiple drawbacks that worsen the RTD's alternative. The merits of the two sensors, the thermocouple and the RTD seem to balance each other and the same applies to their disadvantages. Under these circumstances, it becomes evident why scores of these two sensors are close to each other and why the sensors decision making problem becomes challenging. Anyway, in industrial environment, one should better treat the selection cases one by one paying much attention to the application and environment under concern and the specific technical characteristics of the alternative sensors that may be widely customized and extremely variant and to match these specific characteristics with the 23 sub-criteria matrices introduced in this thesis and to revise the entries of these 23 matrices for better matching to the real industrial-field selection case, or other new criteria that can be added to the assessment process and have significant contribution, especially if area of application differs, or old sensors that can be eliminated in favor to new generations of sensors. New versions of fabricated sensors in industry in each of the sensors categories that have superior features can also be compared. These new sensors with new features may affect the degree of dominance of the alternative sensors when pair-wise compared.

Future Work

The study opens the door wide to apply AHP method in selecting other types of sensors in many other areas, these devices may include: chemical composition sensors, meteorological air pollution sensors, blood pressure and blood chemistry measurement sensors, and many other applications and fields of study. In this sense, future work may include AHP method implementation in one of these fields.

References

1. Nasser Y. El-Awar, David J. Geer, Theodore J. Krellner, Peter J. Straub, Keystone Thermometrics (September 14, 1999), AUTOMOTIVE TEMPERATURE SENSING. [online] [accessed May 2010]. Available from URL: http://www.thermometrics.com/assets/images/autoapp.PDF..
2. David M. Scott. Industrial Process Sensors. Dupont Company, Experimental Station, Wilmington, Delware, U.S.A. CRC Press, Taylor & Francis Group, LLC, 2008.
3. http://www.ametekcalibration.com/UK/Products/Temperature_calibrators.aspx

   Temperature Calibrators-Product Selection Guide

   P-CP-2003-US.pdf
4. J. W. Dally, W. F. Riley, and K. G. McConnell. Instrumentation for Engineering Measurements. John Wiley & Sons, Inc; 1984.
5. W. Gopel, J. Hesse, J.N.Zemel. Sensors A Comprehensive Survey, volume 1, Fundamentals and General Aspects. VCH Publishers Inc., New York; 1989.
6. D. M. Considine, G. D. Considine. Process Instruments and Controls Handbook, 3rd Ed. McGraw-Hill Book Company, Inc.; 1985.
7. R. J. Stephenson, A. M. Moulin, M. E. Welland, J. Burns, M. Sapoff, R. P. Reed, R. Frank, J. Fraden, J. V. Nicholas, F. Pavese, J. Stasiek, T. Madaj, J. Mikielewics, and B. Culshaw, 1999. Measurement, Instrumentation, and Sensors Handbook CRCnetbase, CRC Press LLC.
8. C. Hagart-Alexander, 1985. Jones' Instrument Technology Volume 2, Measurement of Temperature and Chemical Composition, 4th ed., Butterworth & Co. (Publishers) Ltd.

9. Paul Goodwin and George Wright. Decision Analysis for Management Judgment. John Wiley & Sons Ltd; 1998.
10. Chang, C. W., Wu, C. R., Lin, C. T., & Chen, H. C. (2007). An Application of AHP and sensitivity analysis for selecting the best slicing machine. Computers and Industrial Engineering; 52: 296-307.
11. Winston, W. L., 1994. Operations Research. Applications and Algorithms, 3rd ed., Duxbury Press. An Imprint of Wadsworth Publishing Company. Belmont, California.
12. Dyer, J. S. (1990) Remarks on the Analytic Hierarchy Process, Management Science, 36, 249-258.
13. Vaida, O. S., Kumar, S. (2006). Analytic hierarchy process: An overview of applications. European Journal of Operational Research; 169: 1-29.
14. Yurdakul, M. (2004). AHP as a strategic decision-making tool to justify machine tool selection. Journal of Materials Processing Technology, 146 (3) 365-376.
15. Hsu, P. F., Wu, C. R., & Li, Y. T. (2008). Selection of infectious medical waste disposal firms by using the analytic hierarchy process and sensitivity analysis. Waste Management; 28: 1386-1394.
16. Chang, C. W., Wu, C. R., Lin, C. T., & Chen, H. C. (2007). An Application of AHP and sensitivity analysis for selecting the best slicing machine. Computers and Industrial Engineering; 52: 296-307.
17. Okada, H., Styles, S.W., & Grismer, M.E. (2008). Application of the Analytic Hierarchy Process to irrigationproject improvement. Part II. How professionals evaluate an irrigation project for its improvement. Agricultural Water Management; 95: 205-210.

18. Vavra, I., Bydzovsky, J., Falckbart, K., Tejada, J., Kopera, L., Kovakova, E., Temst, K., & Bruynseraede, Y. (2001). Fe/Cr sensor for the milliKelvin temperature range. Sensors and Actuators; 91: 177-179.

19. Hoa, C. H., Cha, Y. H. C., Prakasha, S., Potwina, G., Doerr, H. J., Deshpandey, C. V., Bunshah, R. F., & Zellerb, M. (1995). Electrical resistance drift of molybdenum silicide yhin film temperature sensors. Thin Solid Films; 260: 232-238.

20. Bianchi, R. A., Santos, F. V. D., Karam, J. M., Courtois, B., Pressecq, F., & Sifflet, S. (1998). CMOS-compatible smart temperature sensors. Microelectronics Journal; 29: 627-636.

21. Han, Y., & Kim, S. J. (2008). Diode temperature sensor array for measuring micro-scale surface temperatures with high resolution. Sensors and Actuators; 14: 52-58.Altshuller GS. Creativity as an Exact Science. New York: Gordon and Breach 1984; 228.

22. Wyoming Department of Environmental Quality, (4/29/05), EXAMPLE COMPLIANCE ASSURANCE MONITORING PLAN: CATALYTIC CONVERTER FOR CONTROL OF NOx AND CO. [online] [accessed August 2010]. Available from URL: http://deq.state.wy.us/aqd/downloads/OperPermits/CatConEx5n.pdf.

23. Watlow Electric Manufacturing Company, Single Iteration Division, 909 Horan Dr. Fenton, MO 63026 (2010). How To Choose The Right Temperature Sensor, White Paper WP05-001. [online] [accessed January 2010]. Available from URL: http://www.singleiteration.com/library/document.cfm?id=51

24. Made-in-China.com China manufacturer directory China products China suppliers China trade China company (2010). [online] [accessed August 2010]. Available from URL: www.made-in-china.com/.../China-Bimetal-Thermometers-WSS-.html

25. Volker Thomsen, Spectro Analytical Instruments, 160 Authority Dr., Fitchburg, MA 01420(1998). Response Time of a Thermometer. THE PHYSICS TEACHER, Volume 36 (December), 540-541. http://www.df.uba.ar/users/sgil/physics_paper_doc/papers_phys/termo/cooling2.pdf (accessed August 2010)
26. WINTRONICS,INC. Wintronics Calbration Services & Infrared Thermometer (2005) . [online] [accessed August 2010]. Available from URL: www.wintron.com/infrared/irproducts.htm
27. Made-in-China.com China manufacturer directory China products China suppliers China trade China company (2010). [online] [accessed August 2010]. Available from URL: http://www.made-in-china.com/showroom/corshare/product-detailYbjnWswVAukQ/China-Mems-Semiconductor-Alcohol-Sensor-SAT178B-.html
28. CURTIS J. KLAASSEN, PE, Iowa Energy Center (2001). Selecting and Specifying Building Automation System Sensors Considerations for upgrading sensor performance. HPAC Engineering, (July), 64-66. http://www.df.uba.ar/users/sgil/physics_paper_doc/papers_phys/termo/cooling2.pdf (accessed August 2010)

APPENDICES

Appendix I: Thermocouples to British Standards

Type	Conductors (positive conductor first)	BS 1041, Part 4: 1966 Tolerance on temperature	Output for indicated temperature cold junction at 0°C	Service temperature. max intermittent service in bracket
B	Platinum: 30% Rhodium	0 to 1100°C± 3°C	1.241 mV at 500°C	0 to 1500°C (1700°C)
	Platinum: 6% Rhodium	1100 to 1550°C±4°C		Better life expectancy at high temperature than types R&S
K	Nickel: Chromium/Constantan (Chromel/Constantan) (Chromel/Advance)	0 to 400°C±3°C	6.317 mV at 100°C	-200 to 850°C (1100°C) resistant to oxidizing atmospheres
J	Iron/Constantan	0 to 300°C±3°C 300 to 850°C±1%	5.268 mV at 100°C	-200 to 850°C (1100°C) low cost, suitable for general use
K	Nickel: Chromium/ Nickel: Aluminium (Chromel/Alumel)	0 to 400°C±3°C 400 to 1100°C±0.75%	4.095 mV at 100°C	-200 to 1100°C (1300°C) good general purpose, best in oxidizing atmosphere
R	Platinum: 13%Rhodium/Platinum	0 to 1100°C±1°C 1100 to 1400°C±2°C 1400°C±3°C	4.471 mV at 500°C	0 to 1500°C(1700°C) high temperature corrosion resistant
S	Platinum: 10%Rhodium/Platinum	0 to 1100°C±1°C 1100 to 1400°C±2°C 1400°C±3°C	4.471 mV at 500°C	0 to 1500°C(1700°C) high temperature corrosion resistant
T	Copper/Constantan; (Copper/Advance)	0 to 100°C±1°C 100 to 400°C±1%	4.277 mV at 100°C	-250 to 400°C(500°C) high resistance to corrosion by water
	Rhodium: Iridium/Rhodium	composition and accuracy to be agreed with manufacturer	6.4 mV at 1200°C	0 to 2000°C(2100°C)
	Tungsten: Rhenium 5% Tungsten: Rhenium 26%	accuracy to be agreed with manufacturer	8.890 mV at 500°C	0 to 2300°C(2600°C)
	Tungsten/Molybdenum	composition and accuracy to be agreed with manufacturer	-	1250 to 2600°C

Appendix II: AMETEK CALIBRATION INSTRUMENT

Industrial Temperature Measurement Pages 1, 7-12, 18-23

(Omitted)

Appendix III: List of Matrices in the software for the seven sensors the Three Applications: the Automotives, the Chemical Process, and the HVAC

1- Automotives Application

Maximum Opearting Temperature Matrix:

{1.0,3.0,1.0,9.0,4.0,6.0,4.0},
{0.3333,1.0,0.3333,6.0,2.0,5.0,2.0},
{1.0,3.0,1.0,9.0,4.0,6.0,4.0},
{0.1111,0.1667,0.1111,1.0,0.1667,0.3333,0.1667},
{0.25,0.5,0.25,6.0,1.0,4.0,1.0},
{0.1667,0.2,0.1667,3.0,0.25,1.0,0.25},
{0.25,0.5,0.25,6.0,1.0,4.0,1.0}

Minimum Operating Tempearture Matrix:

{1.0,2.0,2.0,0.5,0.5,0.125,0.3333},
{0.5,1.0,1.0,0.3333,0.3333,0.125,0.25},
{0.5,1.0,1.0,0.3333,0.3333,0.125,0.25},
{2.0,3.0,3.0,1.0,1.0,0.25,0.5},
{2.0,3.0,3.0,1.0,1.0,0.25,0.5},
{8.0,8.0,8.0,4.0,4.0,1.0,4.0},
{3.0,4.0,4.0,2.0,2.0,0.25,1.0}

Temperature Curve Matrix:

{1.0,3.0,0.3333,5.0,5.0,4.0,2.0},
{0.3333,1.0,0.1667,2.0,2.0,1.0,0.3333},
{3.0,6.0,1.0,6.0,6.0,6.0,4.0},
{0.2,0.5,0.1667,1.0,1.0,0.3333,0.25},
{0.2,0.5,0.1667,1.0,1.0,0.3333,0.25},
{0.25,1.0,0.1667,3.0,3.0,1.0,0.5},
{0.5,3.0,0.25,4.0,4.0,2.0,1.0}

Sensitivity Matrix:

{1.0,0.1111,0.2,2.0,2.0,2.0,0.3333},
{9.0,1.0,4.0,9.0,9.0,6.0,4.0},
{5.0,0.25,1.0,4.0,5.0,4.0,2.0},
{0.5,0.1111,0.25,1.0,2.0,2.0,0.25},
{0.5,.1111,0.2,0.5,1.0,1.0,0.25},
{0.5,0.1667,0.25,0.5,1.0,1.0,0.25},
{3.0,0.25,0.5,4.0,4.0,4.0,1.0}

Self Heating Issues Matrix:

{1.0,8.0,3.0,3.0,1.0,1.0,2.0},
{0.125,1.0,0.2,0.25,0.2,0.1667,0.25},
{0.3333,5.0,1.0,0.5,0.5,0.3333,1.0},
{0.3333,4.0,2.0,1.0,1.0,0.5,1.0},

{1.0,5.0,2.0,1.0,1.0,1.0,2.0},
{1.0,6.0,3.0,2.0,1.0,1.0,1.0},
{0.5,4.0,1.0,1.0,0.5,1.0,1.0}

Long Term Stability and Accuracy Matrix:

{1.0,0.25,0.1667,2.0,0.3333,0.5,0.25},
{4.0,1.0,0.3333,4.0,3.0,3.0,2.0},
{6.0,3.0,1.0,8.0,4.0,5.0,3.0},
{0.5,0.25,0.125,1.0,0.3333,0.3333,0.25},
{3.0,0.3333,0.25,3.0,1.0,2.0,0.5},

{2.0,0.3333,0.2,3.0,0.5,1.0,0.3333},
{4.0,0.5,0.3333,4.0,2.0,3.0,1.0}

Typical Temperature Coefficient Matrix:

{1.0,0.1667,0.3333,4.0,4.0,4.0,0.5},
{6.0,1.0,3.0,6.0,6.0,6.0,6.0},
{3.0,0.3333,1.0,5.0,6.0,5.0,2.0},
{0.25,0.1667,0.2,1.0,1.0,1.0,0.2},
{0.25,0.1667,0.1667,1.0,1.0,1.0,0.2},
{0.25,0.1667,0.2,1.0,1.0,1.0,0.3333},
{2.0,0.1667,0.5,5.0,5.0,3.0,1.0}

Extension Wires Matrix:

{1.0,0.1667,0.16667,0.125,0.125,0.125,0.125},
{6.0,1.0,1.0,0.25,0.25,0.25,0.25},
{6.0,1.0,1.0,0.25,0.25,0.25,0.25},
{8.0,4.0,4.0,1.0,1.0,1.0,1.0}
{8.0,4.0,4.0,1.0,1.0,1.0,1.0},
{8.0,4.0,4.0,1.0,1.0,1.0,1.0},
{8.0,4.0,4.0,1.0,1.0,1.0,1.0}

LongWireMatrix:

{1.0,0.3333,1.0,0.1667,0.1667,0.1667,0.1667},
{3.0,1.0,3.0,0.5,0.5,0.5,0.5},
{1.0,0.3333,1.0,0.1667,0.1667,0.1667,0.1667},
{6.0,2.0,6.0,1.0,1.0,1.0,1.0},
{6.0,2.0,6.0,1.0,1.0,1.0,1.0},
{6.0,2.0,6.0,1.0,1.0,1.0,1.0},
{6.0,2.0,6.0,1.0,1.0,1.0,1.0}

Measurement Parameter Matrix:

{1.0,4.0,3.0,5.0,5.0,6.0,1.0},
{0.25,1.0,0.5,4.0,4.0,5.0,0.3333},
{0.3333,2.0,1.0,3.0,3.0,4.0,0.3333},
{0.2,0.25,0.3333,1.0,1.0,3.0,0.2},
{0.2,0.25,0.3333,1.0,1.0,3.0,0.2},
{0.1667,0.2,0.25,0.3333,0.3333,1.0,0.1667},
{1.0,3.0,3.0,5.0,5.0,6.0,1.0}

Temperature Measurement Matrix:

{1.0,0.25,0.2,0.3333,0.3333,0.5,0.1667},
{4.0,1.0,0.3333,1.0,3.0,3.0,0.1667},
{5.0,3.0,1.0,4.0,5.0,5.0,0.5},

{3.0,1.0,0.25,1.0,2.0,2.0,0.25},
{3.0,0.3333,0.2,0.5,1.0,1.0,0.1667},
{2.0,0.3333,0.2,0.5,1.0,1.0,0.2},
{6.0,6.0,2.0,4.0,6.0,5.0,1.0}

Stimulation Electronics Matrix:

{1.0,4.0,3.0,1.0,1.0,2.0,6.0},
{0.25,1.0,0.5,0.2,0.2,0.25,3.0},
{0.3333,2.0,1.0,0.25,0.25,0.5,3.0},
{1.0,5.0,4.0,1.0,1.0,2.0,6.0},
{1.0,5.0,4.0,1.0,1.0,3.0,6.0},
{0.5,4.0,2.0,0.5,0.3333,1.0,4.0},
{0.1667,0.3333,0.3333,0.1667,0.1667,0.25,1.0}

Existence of Maximum Sensitivity Region Matrix:

{1.0,6.0,1.0,4.0,1.0,2.0,1.0},
{0.1667,1.0,0.1667,0.25,0.1667,0.2,0.125},
{1.0,6.0,1.0,4.0,1.0,2.0,1.0},
{0.25,4.0,0.25,1.0,0.25,0.3333,0.1667},
{1.0,6.0,1.0,4.0,1.0,3.0,1.0},
{0.5,5.0,0.5,3.0,0.3333,1.0,0.3333},
{1.0,8.0,1.0,6.0,1.0,3.0,1.0}

Typical Fast Thermal Time Constant Matrix:

{1.0,3.0,4.0,6.0,5.0,1.0,3.0},
{0.3333,1.0,2.0,4.0,3.0,0.3333,1.0},
{0.25,0.5,1.0,2.0,2.0,0.25,1.0},
{0.1667,0.25,0.5,1.0,0.5,0.1667,0.3333},
{0.2,0.3333,0.5,2.0,1.0,0.2,0.3333},
{1.0,3.0,4.0,6.0,5.0,1.0,3.0},
{0.3333,1.0,1.0,3.0,3.0,0.3333,1.0}

Typiacl Small Size Matrix:

{1.0,2.0,3.0,4.0,5.0,6.0,5.0},
{0.5,1.0,2.0,3.0,4.0,5.0,4.0},
{0.3333,0.5,1.0,2.0,4.0,6.0,4.0},
{0.25,0.3333,0.5,1.0,2.0,3.0,2.0},
{0.2,0.25,0.25,0.5,1.0,3.0,1.0},
{0.1667,0.2,0.1667,0.3333,0.3333,1.0,0.5},
{0.2,0.25,0.25,0.5,1.0,2.0,1.0}

Noise Immunity Matrix:

{1.0,0.1667,0.3333,0.25,0.25,0.5,0.25},
{6.0,1.0,4.0,3.0,3.0,5.0,3.0},
{3.0,0.25,1.0,0.5,0.5,2.0,0.5},
{4.0,0.3333,2.0,1.0,1.0,3.0,1.0},
{4.0,0.3333,2.0,1.0,1.0,4.0,1.0},
{2.0,0.2,0.5,0.3333,0.25,1.0,0.25},
{4.0,0.3333,2.0,1.0,1.0,4.0,1.0}

Fragility-Durability Characteristics Matrix:

{1.0,6.0,3.0,6.0,8.0,3.0,4.0},
{0.1667,1.0,0.3333,2.0,3.0,0.3333,0.5},

{0.3333,3.0,1.0,3.0,4.0,2.0,3.0},
{0.1667,0.5,0.3333,1.0,3.0,0.25,0.3333},
{0.125,0.3333,0.25,0.3333,1.0,0.25,0.3333},
{0.3333,3.0,0.5,4.0,4.0,1.0,3.0},
{0.25,2.0,0.3333,3.0,3.0,0.3333,1.0}

High Thermal Gradient Environment Matrix:

{1.0,4.0,5.0,7.0,8.0,4.0,6.0},
{0.25,1.0,2.0,5.0,6.0,2.0,4.0},
{0.2,0.5,1.0,3.0,3.0,0.5,2.0},
{0.1429,0.2,0.3333,1.0,1.0,0.25,0.5},
{0.125,0.1667,0.3333,1.0,1.0,0.2,0.3333},
{0.25,0.5,2.0,4.0,5.0,1.0,4.0},
{0.1667,0.25,0.5,2.0,3.0,0.25,1.0}

Corrosion Resistance Matrix:

{1.0,0.25,0.1667,0.5,0.1667,0.1667,0.25},
{4.0,1.0,0.3333,2.0,0.25,0.25,1.0},
{6.0,3.0,1.0,4.0,1.0,1.0,4.0},
{2.0,0.5,0.25,1.0,0.25,0.25,0.5},
{6.0,4.0,1.0,4.0,1.0,1.0,3.0},
{6.0,4.0,1.0,4.0,1.0,1.0,3.0},
{4.0,1.0,0.25,2.0,0.3333,0.3333,1.0}

Point or Area Measurement Matrix:

{1.0,2.0,3.0,4.0,6.0,6.0,4.0},
{0.5,1.0,2.0,3.0,4.0,4.0,3.0},
{0.3333,0.5,1.0,2.0,3.0,4.0,2.0},
{0.25,0.3333,0.5,1.0,2.0,2.0,0.5},
{0.1667,0.25,0.3333,0.5,1.0,1.0,0.5},
{0.1667,0.25,0.25,0.5,1.0,1.0,2.0},
{0.25,0.5,0.3333,2.0,2.0,0.5,1.0}

Manufacturing Variances Matrix:

{1.0, 0.3333, 0.1667,0.25,0.5,0.2,0.25},
{3.0, 1.0, 0.3333,0.5,2.0,0.25,0.5},
{6.0, 3.0, 1.0,4.0,3.0,6.0,3.0},
{4.0,2.0,0.25,1.0,4.0,0.3333,0.5},
{2.0,0.5,0.3333,0.25,1.0,0.2,0.25},
{5.0,4.0,0.1667,3.0,5.0,1.0,2.0},
{4.0,2.0,0.3333,2.0,4.0,0.5,1.0}

NIST Standards Matrix:

{1.0 , 4.0, 1.0,1.0,1.0,4.0,5.0},
{0.25, 1.0, 0.25,0.25,0.25,1.0,2.0},
{1.0, 4.0, 1.0,1.0,1.0,4.0,5.0},
{1.0,4.0,1.0,1.0,1.0,4.0,5.0},
{1.0,4.0,1.0,1.0,1.0,4.0,5.0},
{0.25,1.0,0.25,0.25,0.25,1.0,2.0},
{0.2,0.5,0.2,0.2,0.2,0.5,1.0}

Cost Matrix:

{1.0,1.0,6.0,3.0,3.0,6.0,3.0},

{1.0,1.0,6.0,3.0,3.0,6.0,3.0},
{0.1667,0.1667,1.0,0.25,0.25,1.0,0.25},
{0.3333,0.3333,4.0,1.0,1.0,4.0,1.0},
{0.3333,0.3333,4.0,1.0,1.0,4.0,1.0},
{0.1667,0.1667,1.0,0.25,0.25,1.0,0.25},
{0.3333,0.3333,4.0,1.0,1.0,4.0,1.0}

Criteria Matrix:

{1.0,4.0,3.0,4.0},
{0.25,1.0,0.5,1.0},
{0.3333,2.0,1.0,2.0},
{0.25,1.0,0.5,1.0}

Sub-criteria Static Matrix:

{1.0,1.0,5.0,4.0,4.0,2.0,5.0,6.0,7.0,8.0,6.0},
{1.0,1.0,5.0,4.0,4.0,2.0,5.0,6.0,7.0,8.0,6.0},
{0.2,0.2,1.0,0.3333,0.3333,0.25,1.0,2.0,4.0,5.0,3.0},
{0.25,0.25,3.0,1.0,2.0,0.5,3.0,3.0,5.0,6.0,4.0},
{0.25,0.25,3.0,0.5,1.0,0.3333,3.0,5.0,6.0,8.0,4.0},
{0.5,0.5,4.0,2.0,3.0,1.0,4.0,5.0,6.0,8.0,5.0},
{0.2,0.2,1.0,0.3333,0.3333,0.25,1.0,1.0,4.0,6.0,3.0},
{0.1667,0.1667,0.5,0.3333,0.2,0.2,1.0,1.0,3.0,4.0,1.0},
{0.1429,0.1429,0.25,0.2,0.1667,0.1667,0.25,0.3333,1.0,2.0,0.3333},
{0.125,0.125,0.2,0.1667,0.125,0.125,0.1667,0.25,0.5,1.0,0.25},
{0.1667,0.1667,0.3333,0.25,0.25,0.2,0.3333,1.0,3.0,4.0,1.0}

Sub-criteria Dynamic Matrix:

{1.0,2.0,0.1667},
{0.5,1.0,0.1667},
{6.0,6.0,1.0}

Sub-criteria Environmental Matrix:

{1.0,3.0,0.3333,4.0,0.25},
{0.3333,1.0,0.25,3.0,0.2},
{3.0,4.0,1.0,5.0,0.5},
{0.25,0.3333,0.2,1.0,0.1667},
{4.0,5.0,2.0,6.0,1.0}

Others Matrix:

{1.0,3.0,0.5,0.25},
{0.3333,1.0,0.3333,0.2},
{2.0,3.0,1.0,0.3333},
{4.0,5.0,3.0,1.0}

2- Chemical Process Application

Maximum Opearting Temperature Matrix:

{1.0,3.0,1.0,9.0,4.0,6.0,4.0},
{0.3333,1.0,0.3333,6.0,2.0,5.0,2.0},
{1.0,3.0,1.0,9.0,4.0,6.0,4.0},
{0.1111,0.1667,0.1111,1.0,0.1667,0.3333,0.1667},
{0.25,0.5,0.25,6.0,1.0,4.0,1.0},
{0.1667,0.2,0.1667,3.0,0.25,1.0,0.25},
{0.25,0.5,0.25,6.0,1.0,4.0,1.0}

Minimum Operating Tempearture Matrix:

{1.0,2.0,2.0,0.5,0.5,0.125,0.3333},
{0.5,1.0,1.0,0.3333,0.3333,0.125,0.25},
{0.5,1.0,1.0,0.3333,0.3333,0.125,0.25},
{2.0,3.0,3.0,1.0,1.0,0.25,0.5},
{2.0,3.0,3.0,1.0,1.0,0.25,0.5},
{8.0,8.0,8.0,4.0,4.0,1.0,4.0},
{3.0,4.0,4.0,2.0,2.0,0.25,1.0}

Temperature Curve Matrix:

{1.0,3.0,0.3333,5.0,5.0,4.0,2.0},
{0.3333,1.0,0.1667,2.0,2.0,1.0,0.3333},
{3.0,6.0,1.0,6.0,6.0,6.0,4.0},
{0.2,0.5,0.1667,1.0,1.0,0.3333,0.25},
{0.2,0.5,0.1667,1.0,1.0,0.3333,0.25},
{0.25,1.0,0.1667,3.0,3.0,1.0,0.5},
{0.5,3.0,0.25,4.0,4.0,2.0,1.0}

Sensitivity Matrix:

{1.0,0.1111,0.2,2.0,2.0,2.0,0.3333},
{9.0,1.0,4.0,9.0,9.0,6.0,4.0},
{5.0,0.25,1.0,4.0,5.0,4.0,2.0},
{0.5,0.1111,0.25,1.0,2.0,2.0,0.25},
{0.5,.1111,0.2,0.5,1.0,1.0,0.25},
{0.5,0.1667,0.25,0.5,1.0,1.0,0.25},
{3.0,0.25,0.5,4.0,4.0,4.0,1.0}

Self Heating Issues Matrix:

{1.0,8.0,3.0,3.0,1.0,1.0,2.0},
{0.125,1.0,0.2,0.25,0.2,0.1667,0.25},
{0.3333,5.0,1.0,0.5,0.5,0.3333,1.0},
{0.3333,4.0,2.0,1.0,1.0,0.5,1.0},
{1.0,5.0,2.0,1.0,1.0,1.0,2.0},
{1.0,6.0,3.0,2.0,1.0,1.0,1.0},
{0.5,4.0,1.0,1.0,0.5,1.0,1.0}

Long Term Stability and Accuracy Matrix:

{1.0,0.25,0.1667,2.0,0.3333,0.5,0.25},
{4.0,1.0,0.3333,4.0,3.0,3.0,2.0},
{6.0,3.0,1.0,8.0,4.0,5.0,3.0},
{0.5,0.25,0.125,1.0,0.3333,0.3333,0.25},
{3.0,0.3333,0.25,3.0,1.0,2.0,0.5},

{2.0,0.3333,0.2,3.0,0.5,1.0,0.3333},
{4.0,0.5,0.3333,4.0,2.0,3.0,1.0}

Typical Temperature Coefficient Matrix:

{1.0,0.1667,0.3333,4.0,4.0,4.0,0.5},
{6.0,1.0,3.0,6.0,6.0,6.0,6.0},
{3.0,0.3333,1.0,5.0,6.0,5.0,2.0},
{0.25,0.1667,0.2,1.0,1.0,1.0,0.2},
{0.25,0.1667,0.1667,1.0,1.0,1.0,0.2},
{0.25,0.1667,0.2,1.0,1.0,1.0,0.3333},
{2.0,0.1667,0.5,5.0,5.0,3.0,1.0}

Extension Wires Matrix:

{1.0,0.1667,0.16667,0.125,0.125,0.125,0.125},
{6.0,1.0,1.0,0.25,0.25,0.25,0.25},
{6.0,1.0,1.0,0.25,0.25,0.25,0.25},
{8.0,4.0,4.0,1.0,1.0,1.0,1.0},
{8.0,4.0,4.0,1.0,1.0,1.0,1.0},
{8.0,4.0,4.0,1.0,1.0,1.0,1.0},
{8.0,4.0,4.0,1.0,1.0,1.0,1.0}

LongWireMatrix:

{1.0,0.3333,1.0,0.1667,0.1667,0.1667,0.1667},
{3.0,1.0,3.0,0.5,0.5,0.5,0.5},
{1.0,0.3333,1.0,0.1667,0.1667,0.1667,0.1667},
{6.0,2.0,6.0,1.0,1.0,1.0,1.0},
{6.0,2.0,6.0,1.0,1.0,1.0,1.0},
{6.0,2.0,6.0,1.0,1.0,1.0,1.0},
{6.0,2.0,6.0,1.0,1.0,1.0,1.0}

Measurement Parameter Matrix:

{1.0,4.0,3.0,5.0,5.0,6.0,1.0},
{0.25,1.0,0.5,4.0,4.0,5.0,0.3333},
{0.3333,2.0,1.0,3.0,3.0,4.0,0.3333},
{0.2,0.25,0.3333,1.0,1.0,3.0,0.2},
{0.2,0.25,0.3333,1.0,1.0,3.0,0.2},
{0.1667,0.2,0.25,0.3333,0.3333,1.0,0.1667},
{1.0,3.0,3.0,5.0,5.0,6.0,1.0}

Temperature Measurement Matrix:

{1.0,0.25,0.2,0.3333,0.3333,0.5,0.1667},
{4.0,1.0,0.3333,1.0,3.0,3.0,0.1667},
{5.0,3.0,1.0,4.0,5.0,5.0,0.5},
{3.0,1.0,0.25,1.0,2.0,2.0,0.25},
{3.0,0.3333,0.2,0.5,1.0,1.0,0.1667},
{2.0,0.3333,0.2,0.5,1.0,1.0,0.2},
{6.0,6.0,2.0,4.0,6.0,5.0,1.0}

Stimulation Electronics Matrix:

{1.0,4.0,3.0,1.0,1.0,2.0,6.0},
{0.25,1.0,0.5,0.2,0.2,0.25,3.0},
{0.3333,2.0,1.0,0.25,0.25,0.5,3.0},
{1.0,5.0,4.0,1.0,1.0,2.0,6.0},
{1.0,5.0,4.0,1.0,1.0,3.0,6.0},

{0.5,4.0,2.0,0.5,0.3333,1.0,4.0},
{0.1667,0.3333,0.3333,0.1667,0.1667,0.25,1.0}

Existence of Maximum Sensitivity Region Matrix:

{1.0,6.0,1.0,4.0,1.0,2.0,1.0},
{0.1667,1.0,0.1667,0.25,0.1667,0.2,0.125},
{1.0,6.0,1.0,4.0,1.0,2.0,1.0},
{0.25,4.0,0.25,1.0,0.25,0.3333,0.1667},
{1.0,6.0,1.0,4.0,1.0,3.0,1.0},
{0.5,5.0,0.5,3.0,0.3333,1.0,0.3333},
{1.0,8.0,1.0,6.0,1.0,3.0,1.0}

Typical Fast Thermal Time Constant Matrix:

{1.0,3.0,4.0,6.0,5.0,1.0,3.0},
{0.3333,1.0,2.0,4.0,3.0,0.3333,1.0},
{0.25,0.5,1.0,2.0,2.0,0.25,1.0},
{0.1667,0.25,0.5,1.0,0.5,0.1667,0.3333},
{0.2,0.3333,0.5,2.0,1.0,0.2,0.3333},
{1.0,3.0,4.0,6.0,5.0,1.0,3.0},
{0.3333,1.0,1.0,3.0,3.0,0.3333,1.0}

Typiacl Small Size Matrix:

{1.0,2.0,3.0,4.0,5.0,6.0,5.0},
{0.5,1.0,2.0,3.0,4.0,5.0,4.0},
{0.3333,0.5,1.0,2.0,4.0,6.0,4.0},
{0.25,0.3333,0.5,1.0,2.0,3.0,2.0},
{0.2,0.25,0.25,0.5,1.0,3.0,1.0},
{0.1667,0.2,0.1667,0.3333,0.3333,1.0,0.5},
{0.2,0.25,0.25,0.5,1.0,2.0,1.0}

Noise Immunity Matrix:

{1.0,0.1667,0.3333,0.25,0.25,0.5,0.25},
{6.0,1.0,4.0,3.0,3.0,5.0,3.0},
{3.0,0.25,1.0,0.5,0.5,2.0,0.5},
{4.0,0.3333,2.0,1.0,1.0,3.0,1.0},
{4.0,0.3333,2.0,1.0,1.0,4.0,1.0},
{2.0,0.2,0.5,0.3333,0.25,1.0,0.25},
{4.0,0.3333,2.0,1.0,1.0,4.0,1.0}

Fragility-Durability Characteristics Matrix:

{1.0,6.0,3.0,6.0,8.0,3.0,4.0},
{0.1667,1.0,0.3333,2.0,3.0,0.3333,0.5},
{0.3333,3.0,1.0,3.0,4.0,2.0,3.0},
{0.1667,0.5,0.3333,1.0,3.0,0.25,0.3333},
{0.125,0.3333,0.25,0.3333,1.0,0.25,0.3333},
{0.3333,3.0,0.5,4.0,4.0,1.0,3.0},
{0.25,2.0,0.3333,3.0,3.0,0.3333,1.0},

High Thermal Gradient Environment Matrix:

{1.0,4.0,5.0,7.0,8.0,4.0,6.0},
{0.25,1.0,2.0,5.0,6.0,2.0,4.0},
{0.2,0.5,1.0,3.0,3.0,0.5,2.0},
{0.1429,0.2,0.3333,1.0,1.0,0.25,0.5},
{0.125,0.1667,0.3333,1.0,1.0,0.2,0.3333},

{0.25,0.5,2.0,4.0,5.0,1.0,4.0},
{0.1667,0.25,0.5,2.0,3.0,0.25,1.0}

Corrosion Resistance Matrix:

{1.0,0.25,0.1667,0.5,0.1667,0.1667,0.25},
{4.0,1.0,0.3333,2.0,0.25,0.25,1.0},
{6.0,3.0,1.0,4.0,1.0,1.0,4.0},
{2.0,0.5,0.25,1.0,0.25,0.25,0.5},
{6.0,4.0,1.0,4.0,1.0,1.0,3.0},
{6.0,4.0,1.0,4.0,1.0,1.0,3.0},
{4.0,1.0,0.25,2.0,0.3333,0.3333,1.0}

Point or Area Measurement Matrix:

{1.0,2.0,3.0,4.0,6.0,6.0,4.0},
{0.5,1.0,2.0,3.0,4.0,4.0,3.0},
{0.3333,0.5,1.0,2.0,3.0,4.0,2.0},
{0.25,0.3333,0.5,1.0,2.0,2.0,0.5},
{0.1667,0.25,0.3333,0.5,1.0,1.0,0.5},
{0.1667,0.25,0.25,0.5,1.0,1.0,2.0},
{0.25,0.5,0.3333,2.0,2.0,0.5,1.0}

Manufacturing Variances Matrix:

{1.0, 0.3333, 0.1667,0.25,0.5,0.2,0.25},
{3.0, 1.0, 0.3333,0.5,2.0,0.25,0.5},
{6.0, 3.0, 1.0,4.0,3.0,6.0,3.0},
{4.0,2.0,0.25,1.0,4.0,0.3333,0.5},
{2.0,0.5,0.3333,0.25,1.0,0.2,0.25},
{5.0,4.0,0.1667,3.0,5.0,1.0,2.0},
{4.0,2.0,0.3333,2.0,4.0,0.5,1.0}

NIST Standards Matrix:

{1.0 , 4.0, 1.0,1.0,1.0,4.0,5.0},
{0.25, 1.0, 0.25,0.25,0.25,1.0,2.0},
{1.0, 4.0, 1.0,1.0,1.0,4.0,5.0},
{1.0,4.0,1.0,1.0,1.0,4.0,5.0},
{1.0,4.0,1.0,1.0,1.0,4.0,5.0},
{0.25,1.0,0.25,0.25,0.25,1.0,2.0},
{0.2,0.5,0.2,0.2,0.2,0.5,1.0}

Cost Matrix:

{1.0,1.0,6.0,3.0,3.0,6.0,3.0},
{1.0,1.0,6.0,3.0,3.0,6.0,3.0},
{0.1667,0.1667,1.0,0.25,0.25,1.0,0.25},
{0.3333,0.3333,4.0,1.0,1.0,4.0,1.0},
{0.3333,0.3333,4.0,1.0,1.0,4.0,1.0},
{0.1667,0.1667,1.0,0.25,0.25,1.0,0.25},
{0.3333,0.3333,4.0,1.0,1.0,4.0,1.0}

Criteria Matrix:

{1.0,2.0,1.0,4.0},
{0.5,1.0,0.5,2.0},
{1.0,2.0,1.0,4.0},
{0.25,0.5,0.25,1.0}

Sub-criteria Static Matrix:

{1.0,1.0,5.0,4.0,4.0,1.0,5.0,6.0,7.0,8.0,6.0},
{1.0,1.0,5.0,4.0,4.0,1.0,5.0,6.0,7.0,8.0,6.0},
{0.2,0.2,1.0,0.3333,0.3333,0.2,1.0,2.0,4.0,5.0,3.0},
{0.25,0.25,3.0,1.0,2.0,0.3333,3.0,3.0,5.0,6.0,4.0},
{0.25,0.25,3.0,0.5,1.0,0.25,3.0,5.0,6.0,8.0,4.0},
{1.0,1.0,5.0,3.0,4.0,1.0,5.0,6.0,7.0,9.0,6.0},
{0.2,0.2,1.0,0.3333,0.3333,0.2,1.0,1.0,4.0,6.0,3.0},
{0.1667,0.1667,0.5,0.3333,0.2,0.1667,1.0,1.0,3.0,4.0,1.0},
{0.1429,0.1429,0.25,0.2,0.1667,0.1429,0.25,0.3333,1.0,2.0,0.3333},
{0.125,0.125,0.2,0.1667,0.125,0.1111,0.1667,0.25,0.5,1.0,0.25},
{0.1667,0.1667,0.3333,0.25,0.25,0.1667,0.3333,1.0,3.0,4.0,1.0}

Sub-criteria Dynamic Matrix:

{1.0,2.0,0.1429},
{0.5,1.0,0.1429},
{7.0,7.0,1.0}

Sub-criteria Environmental Matrix:

{1.0,4.0,0.5,4.0,0.25},
{0.25,1.0,0.25,2.0,0.1667},
{2.0,4.0,1.0,4.0,0.3333},
{0.25,0.5,0.25,1.0,0.1667},
{4.0,6.0,3.0,6.0,1.0}

Others Matrix:

{1.0,3.0,0.5,0.25},
{0.3333,1.0,0.3333,0.2},
{2.0,3.0,1.0,0.3333},
{4.0,5.0,3.0,1.0}

3- HVAC Application

Maximum Opearting Temperature Matrix:

{1.0,3.0,1.0,9.0,4.0,6.0,4.0},
{0.3333,1.0,0.3333,6.0,2.0,5.0,2.0},
{1.0,3.0,1.0,9.0,4.0,6.0,4.0},
{0.1111,0.1667,0.1111,1.0,0.1667,0.3333,0.1667},
{0.25,0.5,0.25,6.0,1.0,4.0,1.0},
{0.1667,0.2,0.1667,3.0,0.25,1.0,0.25},
{0.25,0.5,0.25,6.0,1.0,4.0,1.0}

Minimum Operating Tempearture Matrix:

{1.0,2.0,2.0,0.5,0.5,0.125,0.3333},
{0.5,1.0,1.0,0.3333,0.3333,0.125,0.25},
{0.5,1.0,1.0,0.3333,0.3333,0.125,0.25},
{2.0,3.0,3.0,1.0,1.0,0.25,0.5},
{2.0,3.0,3.0,1.0,1.0,0.25,0.5},
{8.0,8.0,8.0,4.0,4.0,1.0,4.0},
{3.0,4.0,4.0,2.0,2.0,0.25,1.0}

Temperature Curve Matrix:

{1.0,3.0,0.3333,5.0,5.0,4.0,2.0},
{0.3333,1.0,0.1667,2.0,2.0,1.0,0.3333},
{3.0,6.0,1.0,6.0,6.0,6.0,4.0},
{0.2,0.5,0.1667,1.0,1.0,0.3333,0.25},
{0.2,0.5,0.1667,1.0,1.0,0.3333,0.25},
{0.25,1.0,0.1667,3.0,3.0,1.0,0.5},
{0.5,3.0,0.25,4.0,4.0,2.0,1.0}

Sensitivity Matrix:

{1.0,0.1111,0.2,2.0,2.0,2.0,0.3333},
{9.0,1.0,4.0,9.0,9.0,6.0,4.0},
{5.0,0.25,1.0,4.0,5.0,4.0,2.0},
{0.5,0.1111,0.25,1.0,2.0,2.0,0.25},
{0.5,.1111,0.2,0.5,1.0,1.0,0.25},
{0.5,0.1667,0.25,0.5,1.0,1.0,0.25},
{3.0,0.25,0.5,4.0,4.0,4.0,1.0}

Self Heating Issues Matrix:

{1.0,8.0,3.0,3.0,1.0,1.0,2.0},
{0.125,1.0,0.2,0.25,0.2,0.1667,0.25},
{0.3333,5.0,1.0,0.5,0.5,0.3333,1.0},
{0.3333,4.0,2.0,1.0,1.0,0.5,1.0},
{1.0,5.0,2.0,1.0,1.0,1.0,2.0},
{1.0,6.0,3.0,2.0,1.0,1.0,1.0},
{0.5,4.0,1.0,1.0,0.5,1.0,1.0}

Long Term Stability and Accuracy Matrix:

{1.0,0.25,0.1667,2.0,0.3333,0.5,0.25},
{4.0,1.0,0.3333,4.0,3.0,3.0,2.0},
{6.0,3.0,1.0,8.0,4.0,5.0,3.0},
{0.5,0.25,0.125,1.0,0.3333,0.3333,0.25},
{3.0,0.3333,0.25,3.0,1.0,2.0,0.5},

{2.0,0.3333,0.2,3.0,0.5,1.0,0.3333},
{4.0,0.5,0.3333,4.0,2.0,3.0,1.0}

Typical Temperature Coefficient Matrix:

{1.0,0.1667,0.3333,4.0,4.0,4.0,0.5},
{6.0,1.0,3.0,6.0,6.0,6.0,6.0},
{3.0,0.3333,1.0,5.0,6.0,5.0,2.0},
{0.25,0.1667,0.2,1.0,1.0,1.0,0.2},
{0.25,0.1667,0.1667,1.0,1.0,1.0,0.2},
{0.25,0.1667,0.2,1.0,1.0,1.0,0.3333},
{2.0,0.1667,0.5,5.0,5.0,3.0,1.0}

Extension Wires Matrix:

{1.0,0.1667,0.16667,0.125,0.125,0.125,0.125},
{6.0,1.0,1.0,0.25,0.25,0.25,0.25},
{6.0,1.0,1.0,0.25,0.25,0.25,0.25},
{8.0,4.0,4.0,1.0,1.0,1.0,1.0},
{8.0,4.0,4.0,1.0,1.0,1.0,1.0},
{8.0,4.0,4.0,1.0,1.0,1.0,1.0},
{8.0,4.0,4.0,1.0,1.0,1.0,1.0}

LongWireMatrix:

{1.0,0.3333,1.0,0.1667,0.1667,0.1667,0.1667},
{3.0,1.0,3.0,0.5,0.5,0.5,0.5},
{1.0,0.3333,1.0,0.1667,0.1667,0.1667,0.1667},
{6.0,2.0,6.0,1.0,1.0,1.0,1.0},
{6.0,2.0,6.0,1.0,1.0,1.0,1.0},
{6.0,2.0,6.0,1.0,1.0,1.0,1.0},
{6.0,2.0,6.0,1.0,1.0,1.0,1.0}

Measurement Parameter Matrix:

{1.0,4.0,3.0,5.0,5.0,6.0,1.0},
{0.25,1.0,0.5,4.0,4.0,5.0,0.3333},
{0.3333,2.0,1.0,3.0,3.0,4.0,0.3333},
{0.2,0.25,0.3333,1.0,1.0,3.0,0.2},
{0.2,0.25,0.3333,1.0,1.0,3.0,0.2},
{0.1667,0.2,0.25,0.3333,0.3333,1.0,0.1667},
{1.0,3.0,3.0,5.0,5.0,6.0,1.0}

Temperature Measurement Matrix:

{1.0,0.25,0.2,0.3333,0.3333,0.5,0.1667},
{4.0,1.0,0.3333,1.0,3.0,3.0,0.1667},
{5.0,3.0,1.0,4.0,5.0,5.0,0.5},
{3.0,1.0,0.25,1.0,2.0,2.0,0.25},
{3.0,0.3333,0.2,0.5,1.0,1.0,0.1667},
{2.0,0.3333,0.2,0.5,1.0,1.0,0.2},
{6.0,6.0,2.0,4.0,6.0,5.0,1.0}

Stimulation Electronics Matrix:

{1.0,4.0,3.0,1.0,1.0,2.0,6.0},
{0.25,1.0,0.5,0.2,0.2,0.25,3.0},
{0.3333,2.0,1.0,0.25,0.25,0.5,3.0},
{1.0,5.0,4.0,1.0,1.0,2.0,6.0},
{1.0,5.0,4.0,1.0,1.0,3.0,6.0},

{0.5,4.0,2.0,0.5,0.3333,1.0,4.0},
{0.1667,0.3333,0.3333,0.1667,0.1667,0.25,1.0}

Existence of Maximum Sensitivity Region Matrix:

{1.0,6.0,1.0,4.0,1.0,2.0,1.0},
{0.1667,1.0,0.1667,0.25,0.1667,0.2,0.125},
{1.0,6.0,1.0,4.0,1.0,2.0,1.0},
{0.25,4.0,0.25,1.0,0.25,0.3333,0.1667},
{1.0,6.0,1.0,4.0,1.0,3.0,1.0},
{0.5,5.0,0.5,3.0,0.3333,1.0,0.3333},
{1.0,8.0,1.0,6.0,1.0,3.0,1.0}

Typical Fast Thermal Time Constant Matrix:

{1.0,3.0,4.0,6.0,5.0,1.0,3.0},
{0.3333,1.0,2.0,4.0,3.0,0.3333,1.0},
{0.25,0.5,1.0,2.0,2.0,0.25,1.0},
{0.1667,0.25,0.5,1.0,0.5,0.1667,0.3333},
{0.2,0.3333,0.5,2.0,1.0,0.2,0.3333},
{1.0,3.0,4.0,6.0,5.0,1.0,3.0},
{0.3333,1.0,1.0,3.0,3.0,0.3333,1.0}

Typiacl Small Size Matrix:

{1.0,2.0,3.0,4.0,5.0,6.0,5.0},
{0.5,1.0,2.0,3.0,4.0,5.0,4.0},
{0.3333,0.5,1.0,2.0,4.0,6.0,4.0},
{0.25,0.3333,0.5,1.0,2.0,3.0,2.0},
{0.2,0.25,0.25,0.5,1.0,3.0,1.0},
{0.1667,0.2,0.1667,0.3333,0.3333,1.0,0.5},
{0.2,0.25,0.25,0.5,1.0,2.0,1.0}

Noise Immunity Matrix:

{1.0,0.1667,0.3333,0.25,0.25,0.5,0.25},
{6.0,1.0,4.0,3.0,3.0,5.0,3.0},
{3.0,0.25,1.0,0.5,0.5,2.0,0.5},
{4.0,0.3333,2.0,1.0,1.0,3.0,1.0},
{4.0,0.3333,2.0,1.0,1.0,4.0,1.0},
{2.0,0.2,0.5,0.3333,0.25,1.0,0.25},
{4.0,0.3333,2.0,1.0,1.0,4.0,1.0}

Fragility-Durability Characteristics Matrix:

{1.0,6.0,3.0,6.0,8.0,3.0,4.0},
{0.1667,1.0,0.3333,2.0,3.0,0.3333,0.5},
{0.3333,3.0,1.0,3.0,4.0,2.0,3.0},
{0.1667,0.5,0.3333,1.0,3.0,0.25,0.3333},
{0.125,0.3333,0.25,0.3333,1.0,0.25,0.3333},
{0.3333,3.0,0.5,4.0,4.0,1.0,3.0},
{0.25,2.0,0.3333,3.0,3.0,0.3333,1.0},

High Thermal Gradient Environment Matrix:

{1.0,4.0,5.0,7.0,8.0,4.0,6.0},
{0.25,1.0,2.0,5.0,6.0,2.0,4.0},
{0.2,0.5,1.0,3.0,3.0,0.5,2.0},
{0.1429,0.2,0.3333,1.0,1.0,0.25,0.5},
{0.125,0.1667,0.3333,1.0,1.0,0.2,0.3333},

{0.25,0.5,2.0,4.0,5.0,1.0,4.0},
{0.1667,0.25,0.5,2.0,3.0,0.25,1.0}

Corrosion Resistance Matrix:

{1.0,0.25,0.1667,0.5,0.1667,0.1667,0.25},
{4.0,1.0,0.3333,2.0,0.25,0.25,1.0},
{6.0,3.0,1.0,4.0,1.0,1.0,4.0},
{2.0,0.5,0.25,1.0,0.25,0.25,0.5},
{6.0,4.0,1.0,4.0,1.0,1.0,3.0},
{6.0,4.0,1.0,4.0,1.0,1.0,3.0},
{4.0,1.0,0.25,2.0,0.3333,0.3333,1.0}

Point or Area Measurement Matrix:

{1.0,2.0,3.0,4.0,6.0,6.0,4.0},
{0.5,1.0,2.0,3.0,4.0,4.0,3.0},
{0.3333,0.5,1.0,2.0,3.0,4.0,2.0},
{0.25,0.3333,0.5,1.0,2.0,2.0,0.5},
{0.1667,0.25,0.3333,0.5,1.0,1.0,0.5},
{0.1667,0.25,0.25,0.5,1.0,1.0,2.0},
{0.25,0.5,0.3333,2.0,2.0,0.5,1.0}

Manufacturing Variances Matrix:

{1.0, 0.3333, 0.1667,0.25,0.5,0.2,0.25},
{3.0, 1.0, 0.3333,0.5,2.0,0.25,0.5},
{6.0, 3.0, 1.0,4.0,3.0,6.0,3.0},
{4.0,2.0,0.25,1.0,4.0,0.3333,0.5},
{2.0,0.5,0.3333,0.25,1.0,0.2,0.25},
{5.0,4.0,0.1667,3.0,5.0,1.0,2.0},
{4.0,2.0,0.3333,2.0,4.0,0.5,1.0}

NIST Standards Matrix:

{1.0 , 4.0, 1.0,1.0,1.0,4.0,5.0},
{0.25, 1.0, 0.25,0.25,0.25,1.0,2.0},
{1.0, 4.0, 1.0,1.0,1.0,4.0,5.0},
{1.0,4.0,1.0,1.0,1.0,4.0,5.0},
{1.0,4.0,1.0,1.0,1.0,4.0,5.0},
{0.25,1.0,0.25,0.25,0.25,1.0,2.0},
{0.2,0.5,0.2,0.2,0.2,0.5,1.0}

Cost Matrix:

{1.0,1.0,6.0,3.0,3.0,6.0,3.0},
{1.0,1.0,6.0,3.0,3.0,6.0,3.0},
{0.1667,0.1667,1.0,0.25,0.25,1.0,0.25},
{0.3333,0.3333,4.0,1.0,1.0,4.0,1.0},
{0.3333,0.3333,4.0,1.0,1.0,4.0,1.0},
{0.1667,0.1667,1.0,0.25,0.25,1.0,0.25},
{0.3333,0.3333,4.0,1.0,1.0,4.0,1.0}

Criteria Matrix:

{1.0,9.0,6.0,4.0},
{0.1111,1.0,0.5,0.3333},
{0.1667,2.0,1.0,0.5},
{0.25,3.0,2.0,1.0}

Sub-criteria Static Matrix:

{1.0,1.0,4.0,4.0,3.0,1.0,5.0,5.0,6.0,8.0,5.0},
{1.0,1.0,4.0,4.0,3.0,1.0,5.0,5.0,6.0,8.0,5.0},
{0.25,0.25,1.0,0.5,0.3333,0.25,2.0,2.0,4.0,6.0,3.0},
{0.25,0.25,2.0,1.0,1.0,0.3333,3.0,2.0,4.0,6.0,3.0},
{0.3333,0.3333,3.0,1.0,1.0,0.3333,4.0,5.0,6.0,9.0,4.0},
{1.0,1.0,4.0,3.0,3.0,1.0,5.0,5.0,6.0,9.0,5.0},
{0.2,0.2,0.5,0.3333,0.25,0.2,1.0,0.5,3.0,6.0,3.0},
{0.2,0.2,0.5,0.5,0.2,0.2,2.0,1.0,3.0,5.0,1.0},
{0.1667,0.1667,0.25,0.25,0.1667,0.1667,0.3333,0.3333,1.0,3.0,0.3333},
{0.125,0.125,0.1667,0.1667,0.1111,0.1111,0.1667,0.2,0.3333,1.0,0.2},
{0.2,0.2,0.3333,0.3333,0.25,0.2,0.3333,1.0,3.0,5.0,1.0}

Sub-criteria Dynamic Matrix:

{1.0,2.0,0.1667},
{0.5,1.0,0.1667},
{6.0,6.0,1.0}

Sub-criteria Environmental Matrix:

{1.0,1.0,0.2,4.0,0.1667},
{1.0,1.0,0.25,5.0,0.2},
{5.0,4.0,1.0,7.0,0.5},
{0.25,0.2,0.1429,1.0,0.125},
{6.0,5.0,2.0,8.0,1.0}

Others Matrix:

{1.0,3.0,0.5,0.1667},
{0.3333,1.0,0.3333,0.1429},
{2.0,3.0,1.0,0.2},
{6.0,7.0,5.0,1.0}

Appendix IV [c.f. 6, 23]: Comparison between the Thermocouple, the Thermister, and the RTD against the 23 Sub-criteria

Sub-criterion	Thermocouple	Thermistor	RTD
Maximum Operating Temperature (°C)	2300	1000	850
Minimum Operating Temperature (°C)	-200	-100	-200
Temperature Curve	Fair linearity*	Poor linearity*	Good Linearity*
Point or Area measurement	Point	Area	Area
Sensitivity	Low*	Very high*	Medium*
Measurement Parameter	Voltage	Resistance	Resistance
Temperature Measurement	differential	Absolute	Absolute
Stimulation Electronics required	None	Yes	Yes
Self-Heating Issues	No self heating*	High*	Very low to low*
Existence of Maximum Sensitivity Region	No	Yes	No
Standards Exist	Yes	No	Yes
Manufacturing Variances	Inhomogeniety	Batch-to-batch	Lowest
Typical Small Size	0.01″ Small to large*	0.1′ Small to medium*	0.1′ Large*
Typical Fast Thermal Time Constant	0.01 sec	0.1 sec	0.1 sec
Long Term Stability and Accuracy	OK Poor to fair*	Good poor*	Best good*
Noise immunity	OK if shielded	best	Good
Fragility/Durability	Best	OK	Good

High thermal gradient environment	Best	OK	OK
Typical Temperature Coefficient	+ 0.4 % °C $^{-1}$	-3 to -5 % °C $^{-1}$	+ 0.4 to + 0.5% °C $^{-1}$
Corrosion resistance	Low	Good	Good
Extension Wires	Same alloy	Any kind	Any kind
Long Wire runs from Sensor	No	OK	No
Cost	Low-medium	Low-medium	High

* These comparisons were taken from reference [6].

Appendix V [28]: Selecting and Specifying Building Automation System Sensors Considerations for Upgrading Sensor Performance (Omitted)

Appendix VI: Programming Code for the Software

1. AHP.cs File

```
using System;
using System.Collections.Generic;
using System.Text;

namespace AhpCaseStudy1GUI
{
   class AHP
   {
      public static int APPLICATION_HVAC = 0;
      public static int APPLICATION_AUTOMOTIVES = 1;
      public static int APPLICATION_CHEMICAL_REACTIONS = 2;

      public static int SENSOR_THERMOCOUPLE = 0;
      public static int SENSOR_THERMISTER = 1;
      public static int SENSOR_RTD = 2;
      public static int SENSOR_BIMETALLIC = 3;
      public static int SENSOR_THERMOMETER = 4;
      public static int SENSOR_PYROMETER = 5;
      public static int SENSOR_LCD_DISPLAY = 6;

      public static int TEMP_RANGE_20_200 = 0;
      public static int TEMP_RANGE_200_700 = 1;
      public static int TEMP_RANGE_700_950 = 2;
      public static int TEMP_RANGE_950_1150 = 3;
      public static int TEMP_RANGE_1150_1300 = 4;
      public static int TEMP_RANGE_Minus_1300_2700 = 5;
      public static int TEMP_RANGE_Minus_2700_3300 = 6;
      public static int TEMP_RANGE_Minus_200_950 = 7;
      public static int TEMP_RANGE_Minus_700_1150 = 8;


      public static int ACURACY_1 = 0;
      public static int ACURACY_POINT_1 = 1;
      public static int ACURACY_POINT_O_1 = 2;
      public static int ACURACY_POINT_O_O_1 = 3;
      public static int ACURACY_POINT_O_O_O_O_1 = 4;

      public static int RESPONSE_TIME_POINT_O_1 = 0;
      public static int RESPONSE_TIME_POINT_2 = 1;
      public static int RESPONSE_TIME_POINT_3 = 2;
      public static int RESPONSE_TIME_1 = 3;
      public static int RESPONSE_TIME_10 = 4;
      public static int RESPONSE_TIME_20 = 5;

   }
}
```

2. AHPSubCriteria.cs File

```
using System;
using System.Collections.Generic;
using System.Text;

namespace AhpCaseStudy1GUI
{

   class AHPSubCriteria
   {

      public double[,] MaximumTempMatrix;
      public double[,] MinimumTempMatrix;
      public double[,] TempCurveMtrix;
      public double[,] MaxSensitivityMatrix;
      public double[,] SelfHeatingMatrix;
      public double[,] LongTermStabilityMatrix;
      public double[,] TypTempCoeffMatrix;
      public double[,] ExtWiresMatrix;
      public double[,] LongWireMatrix;
      public double[,] MeasureParaMatrix;
      public double[,] TempMeasureMatrix;
      public double[,] StimulationElecMatrix;
      public double[,] TypOutputLevelMatrix;
      public double[,] TypFastThertimeConsMatrix;
      public double[,] TypSmallSizMatrix;
      public double[,] NoiseImmunityMatrix;
      public double[,] FraDurMatrix;
      public double[,] HiThGrEnMatrix;
      public double[,] CorrResMatrix;
      public double[,] PointAreaMeasMatrix;
      public double[,] ManuVarMatrix;
      public double[,] NistStanMatrix;
      public double[,] CostMatrix;

      public void CreateSubMatrices(int size)
      {
         MaximumTempMatrix = new double[size, size];
         MinimumTempMatrix = new double[size, size];
         TempCurveMtrix = new double[size, size];
         MaxSensitivityMatrix = new double[size, size];
         SelfHeatingMatrix = new double[size, size];
         LongTermStabilityMatrix = new double[size, size];
         TypTempCoeffMatrix = new double[size, size];
         ExtWiresMatrix = new double[size, size];
         LongWireMatrix = new double[size, size];
         MeasureParaMatrix = new double[size, size];
         TempMeasureMatrix = new double[size, size];
         StimulationElecMatrix = new double[size, size];
         TypOutputLevelMatrix = new double[size, size];
         TypFastThertimeConsMatrix = new double[size, size];
         TypSmallSizMatrix = new double[size, size];
         NoiseImmunityMatrix = new double[size, size];
         FraDurMatrix = new double[size, size];
         HiThGrEnMatrix = new double[size, size];
         CorrResMatrix = new double[size, size];
         PointAreaMeasMatrix = new double[size, size];
         ManuVarMatrix = new double[size, size];
         NistStanMatrix = new double[size, size];
         CostMatrix = new double[size, size];
      }

      public void FillMatrices(int[] IndeciesArray)
      {
         CreateSubMatrices(IndeciesArray.Length);
      }
   }
}
```

3. Automotives.cs File

```
using System;
using System.Collections.Generic;
using System.Text;

namespace AhpCaseStudy1GUI
{
   class Automotives
   {
      public System.Windows.Forms.TextBox textBoxResults;

      void Print(double[,] mat, int nDimensionSize, string strTitle)
      {
         if (strTitle == "")
            strTitle = "Matrix";

         //System.Console.WriteLine(strTitle);
         textBoxResults.AppendText("\n");
         textBoxResults.AppendText(strTitle);
         textBoxResults.AppendText("\n");
         textBoxResults.AppendText("\n");
         for (int i = 0; i < nDimensionSize; i++)
         {
            for (int j = 0; j < nDimensionSize; j++)
            {
               //System.Console.Write("{0}\t\t", mat[i, j]);
               textBoxResults.AppendText("     "+mat[i, j]);
               //textBoxResults.Update();
            }
            //System.Console.Write("\n");
            textBoxResults.AppendText("\n");
         }

         //System.Console.Write("\n\n");
         textBoxResults.AppendText("\n\n");
      }

      void Print(double[] vector, int nDimensionSize)
      {
         //System.Console.WriteLine("Relative Weight Vector = ");
         textBoxResults.AppendText("Relative Weight Vector = ");
         for (int i = 0; i < nDimensionSize; i++)
         {
            //System.Console.Write("{0}\t\t", vector[i]);
            textBoxResults.AppendText("     " + vector[i]);
         }
         //System.Console.Write("\n");
         textBoxResults.AppendText("\n");
      }

      void Scale(double[] vector, double scaleFactor)
      {
         for (int i = 0; i < vector.Length; i++)
         {
            vector[i] = vector[i] * scaleFactor;//vector[i] *= scaleFactor;
         }
         Print(vector, vector.Length);
      }

      double GetSummationOfVectorElements(double[] vector)
      {
         double Summation = 0.0;
         for (int i = 0; i < vector.Length; i++)
         {
            Summation = Summation + vector[i];
         }
         return Summation;
      }

      double[] CalculateWeightsForEachCriterion(double[,] expertAssesmentMatrix, int nDimensionSize, string strTitle)
      {
         //check for vald input values
         double[] weightVector = new double[nDimensionSize];
```

```
        if (expertAssesmentMatrix == null)
            return weightVector;

        Print(expertAssesmentMatrix, nDimensionSize, strTitle);

        //Calculate the weight factor for each colmun
        //The result is a vetor
        double[] wieghtFactor = new double[nDimensionSize];
        for (int j = 0; j < nDimensionSize; j++)
        {
            double result = 0.0;
            for (int k = 0; k < nDimensionSize; k++)
            {
                result += expertAssesmentMatrix[k, j]; //result = result + mat1[k, j];
            }
            wieghtFactor[j] = result;
        }

        double[,] mat1ImmediatResult = new double[nDimensionSize, nDimensionSize];
        for (int j = 0; j < nDimensionSize; j++)
        {
            for (int k = 0; k < nDimensionSize; k++)
            {
                expertAssesmentMatrix[k, j] = expertAssesmentMatrix[k, j] / wieghtFactor[j];
            }
        }

        //Calculate the weight factor for each colmun
        //The result is a vetor
        for (int j = 0; j < nDimensionSize; j++)
        {
            double result = 0.0;
            for (int k = 0; k < nDimensionSize; k++)
            {
                result += expertAssesmentMatrix[j, k]; //result = result + mat1[k, j];
            }
            weightVector[j] = result / nDimensionSize;
        }

        Print(weightVector, nDimensionSize);

//System.Console.WriteLine("_____
_____\n");


        return weightVector;
    }

    void FillSubMatrix(double[,] SourceMatrix, double[,] destinationSubMatrix, int[] IndeciesArray)
    {
        if (SourceMatrix == null || destinationSubMatrix == null || IndeciesArray == null)
            return;

        for (int i = 0; i < IndeciesArray.Length; i++)
        {
            for (int j = 0; j < IndeciesArray.Length; j++)
            {
                //destinationSubMatrix[IndeciesArray[i], IndeciesArray[j]] = SourceMatrix[IndeciesArray[i], IndeciesArray[j]];
                destinationSubMatrix[i, j] = SourceMatrix[IndeciesArray[i], IndeciesArray[j]];
            }
        }
    }

    double ComputeConsistencyIndex(double[,] Matrix, double[] vector, int nDimensionSize)
    {
        //check for vald input values
        double[] transposeVector = new double[nDimensionSize];
        double[] randomIndex = { 1.0, 0.5, 0.58, 0.9, 1.12, 1.24, 1.32 };

        if (Matrix == null)
            return 0.0;

        for (int j = 0; j < nDimensionSize; j++)
        {
            double result = 0.0;
            for (int k = 0; k < nDimensionSize; k++)
```

```
            {
                result += (Matrix[j, k] * vector[k]);
            }
            transposeVector[j] = result;
        }
        double division = 0.0;
        for (int k = 0; k < nDimensionSize; k++)
        {
            division += (transposeVector[k] / vector[k]);
            //transposeVector[k] = transposeVector[k]/vector[k];

        }
        division = division / nDimensionSize;

        double consistencyIndex = (division - nDimensionSize) / (nDimensionSize - 1);

        if (consistencyIndex < 0.0 && consistencyIndex > -0.0005)
            consistencyIndex = 0.0;

        if (consistencyIndex < 0.0005)
            consistencyIndex = 0.0;

        textBoxResults.AppendText("Consistency Index = ");
        textBoxResults.AppendText("" + consistencyIndex + "\n");

        textBoxResults.AppendText("\nConsistency Ratio = ");
        if( nDimensionSize > randomIndex.Length )
            textBoxResults.AppendText("" + (consistencyIndex/1.59));
        else
            textBoxResults.AppendText("" + (consistencyIndex/randomIndex[nDimensionSize-1]));
        textBoxResults.AppendText("\n");
        textBoxResults.AppendText("_____\n");
        return consistencyIndex;
    }

    public double[] SelectBestSensor(int[] IndeciesArray)
    {
        double[] SensorRanks = new double[IndeciesArray.Length];
        AHPSubCriteria ReturnSubMatrices = new AHPSubCriteria();
        ReturnSubMatrices.CreateSubMatrices(IndeciesArray.Length);
        AHPSubCriteria ReturnSubMatricesNotNormalized = new AHPSubCriteria();
        ReturnSubMatricesNotNormalized.CreateSubMatrices(IndeciesArray.Length);

        double[,] MaximumTempMatrix = new double[,] {
                {1.0,3.0,1.0,9.0,4.0,6.0,4.0},
                {0.3333,1.0,0.3333,6.0,2.0,5.0,2.0},
                {1.0,3.0,1.0,9.0,4.0,6.0,4.0},
                {0.1111,0.1667,0.1111,1.0,0.1667,0.3333,0.1667},
                {0.25,0.5,0.25,6.0,1.0,4.0,1.0},
                {0.1667,0.2,0.1667,3.0,0.25,1.0,0.25},
                {0.25,0.5,0.25,6.0,1.0,4.0,1.0}
        };
        FillSubMatrix(MaximumTempMatrix, ReturnSubMatrices.MaximumTempMatrix, IndeciesArray);
        double[] MaximumTempMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.MaximumTempMatrix,
IndeciesArray.Length, "Maximum Operating Temprature: ");

        FillSubMatrix(MaximumTempMatrix, ReturnSubMatricesNotNormalized.MaximumTempMatrix, IndeciesArray);
        double ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.MaximumTempMatrix,
MaximumTempMatrixResult, IndeciesArray.Length);
        ////////////////////////////////////////////////////////////////////////////////////////////////
        double[,] MinimumTempMatrix = new double[,] {
                {1.0,2.0,2.0,0.5,0.5,0.125,0.3333},
                {0.5,1.0,1.0,0.3333,0.3333,0.125,0.25},
                {0.5,1.0,1.0,0.3333,0.3333,0.125,0.25},
                {2.0,3.0,3.0,1.0,1.0,0.25,0.5},
                {2.0,3.0,3.0,1.0,1.0,0.25,0.5},
                {8.0,8.0,8.0,4.0,4.0,1.0,4.0},
                {3.0,4.0,4.0,2.0,2.0,0.25,1.0}
        };
        FillSubMatrix(MinimumTempMatrix, ReturnSubMatrices.MinimumTempMatrix, IndeciesArray);
        double[] MinimumTempMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.MinimumTempMatrix,
IndeciesArray.Length, "Minimum Operating Temprature: ");

        FillSubMatrix(MinimumTempMatrix, ReturnSubMatricesNotNormalized.MinimumTempMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.MinimumTempMatrix,
MinimumTempMatrixResult, IndeciesArray.Length);
```

```
        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] TempCurveMtrix = new double[,] {
            {1.0,3.0,0.3333,5.0,5.0,4.0,2.0},
            {0.3333,1.0,0.1667,2.0,2.0,1.0,0.3333},
            {3.0,6.0,1.0,6.0,6.0,6.0,4.0},
            {0.2,0.5,0.1667,1.0,1.0,0.3333,0.25},
            {0.2,0.5,0.1667,1.0,1.0,0.3333,0.25},
            {0.25,1.0,0.1667,3.0,3.0,1.0,0.5},
            {0.5,3.0,0.25,4.0,4.0,2.0,1.0}
        };
        FillSubMatrix(TempCurveMtrix, ReturnSubMatrices.TempCurveMtrix, IndeciesArray);
        double[] TempCurveMtrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.TempCurveMtrix,
IndeciesArray.Length, "Temperature Curve:");

        FillSubMatrix(TempCurveMtrix, ReturnSubMatricesNotNormalized.TempCurveMtrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.TempCurveMtrix, TempCurveMtrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] MaxSensitivityMatrix = new double[,] {
            {1.0,0.1111,0.2,2.0,2.0,2.0,0.3333},
            {9.0,1.0,4.0,9.0,9.0,6.0,4.0},
            {5.0,0.25,1.0,4.0,5.0,4.0,2.0},
            {0.5,0.1111,0.25,1.0,2.0,2.0,0.25},
            {0.5,.1111,0.2,0.5,1.0,1.0,0.25},
            {0.5,0.1667,0.25,0.5,1.0,1.0,0.25},
            {3.0,0.25,0.5,4.0,4.0,4.0,1.0}
        };
        FillSubMatrix(MaxSensitivityMatrix, ReturnSubMatrices.MaxSensitivityMatrix, IndeciesArray);
        double[] MaxSensitivityMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.MaxSensitivityMatrix,
IndeciesArray.Length, "Maximum Sensitivity Region:");

        FillSubMatrix(MaxSensitivityMatrix, ReturnSubMatricesNotNormalized.MaxSensitivityMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.MaxSensitivityMatrix,
MaxSensitivityMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] SelfHeatingMatrix = new double[,] {
            {1.0,8.0,3.0,3.0,1.0,1.0,2.0},
            {0.125,1.0,0.2,0.25,0.2,0.1667,0.25},
            {0.3333,5.0,1.0,0.5,0.5,0.3333,1.0},
            {0.3333,4.0,2.0,1.0,1.0,0.5,1.0},
            {1.0,5.0,2.0,1.0,1.0,1.0,2.0},
            {1.0,6.0,3.0,2.0,1.0,1.0,1.0},
            {0.5,4.0,1.0,1.0,0.5,1.0,1.0}
        };
        FillSubMatrix(SelfHeatingMatrix, ReturnSubMatrices.SelfHeatingMatrix, IndeciesArray);
        double[] SelfHeatingMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.SelfHeatingMatrix,
IndeciesArray.Length, "Self-Heating Issues:");

        FillSubMatrix(SelfHeatingMatrix, ReturnSubMatricesNotNormalized.SelfHeatingMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.SelfHeatingMatrix,
SelfHeatingMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] LongTermStabilityMatrix = new double[,] {
            {1.0,0.25,0.1667,2.0,0.3333,0.5,0.25},
            {4.0,1.0,0.3333,4.0,3.0,3.0,2.0},
            {6.0,3.0,1.0,8.0,4.0,5.0,3.0},
            {0.5,0.25,0.125,1.0,0.3333,0.3333,0.25},
            {3.0,0.3333,0.25,3.0,1.0,2.0,0.5},
            {2.0,0.3333,0.2,3.0,0.5,1.0,0.3333},
            {4.0,0.5,0.3333,4.0,2.0,3.0,1.0}
        };
        FillSubMatrix(LongTermStabilityMatrix, ReturnSubMatrices.LongTermStabilityMatrix, IndeciesArray);
        double[] LongTermStabilityMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.LongTermStabilityMatrix,
IndeciesArray.Length, "Long Term Stability and Accuracy:");

        FillSubMatrix(LongTermStabilityMatrix, ReturnSubMatricesNotNormalized.LongTermStabilityMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.LongTermStabilityMatrix,
LongTermStabilityMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] TypTempCoeffMatrix = new double[,] {
            {1.0,0.1667,0.3333,4.0,4.0,4.0,0.5},
```

```
            {6.0,1.0,3.0,6.0,6.0,6.0,6.0},
            {3.0,0.3333,1.0,5.0,6.0,5.0,2.0},
            {0.25,0.1667,0.2,1.0,1.0,1.0,0.2},
            {0.25,0.1667,0.1667,1.0,1.0,1.0,0.2},
            {0.25,0.1667,0.2,1.0,1.0,1.0,0.3333},
            {2.0,0.1667,0.5,5.0,5.0,3.0,1.0}
        };
        FillSubMatrix(TypTempCoeffMatrix, ReturnSubMatrices.TypTempCoeffMatrix, IndeciesArray);
        double[] TypTempCoeffMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.TypTempCoeffMatrix,
IndeciesArray.Length, "Typical Temperature Coefficient:");

        FillSubMatrix(TypTempCoeffMatrix, ReturnSubMatricesNotNormalized.TypTempCoeffMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.TypTempCoeffMatrix,
TypTempCoeffMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////////
        double[,] ExtWiresMatrix = new double[,] {
            {1.0,0.1667,0.16667,0.125,0.125,0.125,0.125},
            {6.0,1.0,1.0,0.25,0.25,0.25,0.25},
            {6.0,1.0,1.0,0.25,0.25,0.25,0.25},
            {8.0,4.0,4.0,1.0,1.0,1.0,1.0},
            {8.0,4.0,4.0,1.0,1.0,1.0,1.0},
            {8.0,4.0,4.0,1.0,1.0,1.0,1.0},
            {8.0,4.0,4.0,1.0,1.0,1.0,1.0}
        };
        FillSubMatrix(ExtWiresMatrix, ReturnSubMatrices.ExtWiresMatrix, IndeciesArray);
        double[] ExtWiresMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.ExtWiresMatrix,
IndeciesArray.Length, "Extension Wires:");

        FillSubMatrix(ExtWiresMatrix, ReturnSubMatricesNotNormalized.ExtWiresMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.ExtWiresMatrix, ExtWiresMatrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////////
        double[,] LongWireMatrix = new double[,] {
            {1.0,0.3333,1.0,0.1667,0.1667,0.1667,0.1667},
            {3.0,1.0,3.0,0.5,0.5,0.5,0.5},
            {1.0,0.3333,1.0,0.1667,0.1667,0.1667,0.1667},
            {6.0,2.0,6.0,1.0,1.0,1.0,1.0},
            {6.0,2.0,6.0,1.0,1.0,1.0,1.0},
            {6.0,2.0,6.0,1.0,1.0,1.0,1.0},
            {6.0,2.0,6.0,1.0,1.0,1.0,1.0}
        };
        FillSubMatrix(LongWireMatrix, ReturnSubMatrices.LongWireMatrix, IndeciesArray);
        double[] LongWireMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.LongWireMatrix,
IndeciesArray.Length, "Long Wire Runs From Sensor:");

        FillSubMatrix(LongWireMatrix, ReturnSubMatricesNotNormalized.LongWireMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.LongWireMatrix, LongWireMatrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////////
        double[,] MeasureParaMatrix = new double[,] {
            {1.0,4.0,3.0,5.0,5.0,6.0,1.0},
            {0.25,1.0,0.5,4.0,4.0,5.0,0.3333},
            {0.3333,2.0,1.0,3.0,3.0,4.0,0.3333},
            {0.2,0.25,0.3333,1.0,1.0,3.0,0.2},
            {0.2,0.25,0.3333,1.0,1.0,3.0,0.2},
            {0.1667,0.2,0.25,0.3333,0.3333,1.0,0.1667},
            {1.0,3.0,3.0,5.0,5.0,6.0,1.0}
        };
        FillSubMatrix(MeasureParaMatrix, ReturnSubMatrices.MeasureParaMatrix, IndeciesArray);
        double[] MeasureParaMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.MeasureParaMatrix,
IndeciesArray.Length, "Measurement Parameter:");

        FillSubMatrix(MeasureParaMatrix, ReturnSubMatricesNotNormalized.MeasureParaMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.MeasureParaMatrix,
MeasureParaMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////////
        double[,] TempMeasureMatrix = new double[,] {
            {1.0,0.25,0.2,0.3333,0.3333,0.5,0.1667},
            {4.0,1.0,0.3333,1.0,3.0,3.0,0.1667},
            {5.0,3.0,1.0,4.0,5.0,5.0,0.5},
            {3.0,1.0,0.25,1.0,2.0,2.0,0.25},
            {3.0,0.3333,0.2,0.5,1.0,1.0,0.1667},
```

```
            {2.0,0.3333,0.2,0.5,1.0,1.0,0.2},
            {6.0,6.0,2.0,4.0,6.0,5.0,1.0}
        };
        FillSubMatrix(TempMeasureMatrix, ReturnSubMatrices.TempMeasureMatrix, IndeciesArray);
        double[] TempMeasureMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.TempMeasureMatrix,
IndeciesArray.Length, "Temperature Measurement:");

        FillSubMatrix(TempMeasureMatrix, ReturnSubMatricesNotNormalized.TempMeasureMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.TempMeasureMatrix,
TempMeasureMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] StimulationElecMatrix = new double[,] {
            {1.0,4.0,3.0,1.0,1.0,2.0,6.0},
            {0.25,1.0,0.5,0.2,0.2,0.25,3.0},
            {0.3333,2.0,1.0,0.25,0.25,0.5,3.0},
            {1.0,5.0,4.0,1.0,1.0,2.0,6.0},
            {1.0,5.0,4.0,1.0,1.0,3.0,6.0},
            {0.5,4.0,2.0,0.5,0.3333,1.0,4.0},
            {0.1667,0.3333,0.3333,0.1667,0.1667,0.25,1.0}
        };
        FillSubMatrix(StimulationElecMatrix, ReturnSubMatrices.StimulationElecMatrix, IndeciesArray);
        double[] StimulationElecMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.StimulationElecMatrix,
IndeciesArray.Length, "Stimulation Electronics Required:");

        FillSubMatrix(StimulationElecMatrix, ReturnSubMatricesNotNormalized.StimulationElecMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.StimulationElecMatrix,
StimulationElecMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] TypOutputLevelMatrix = new double[,] {
            {1.0,6.0,1.0,4.0,1.0,2.0,1.0},
            {0.1667,1.0,0.1667,0.25,0.1667,0.2,0.125},
            {1.0,6.0,1.0,4.0,1.0,2.0,1.0},
            {0.25,4.0,0.25,1.0,0.25,0.3333,0.1667},
            {1.0,6.0,1.0,4.0,1.0,3.0,1.0},
            {0.5,5.0,0.5,3.0,0.3333,1.0,0.3333},
            {1.0,8.0,1.0,6.0,1.0,3.0,1.0}
        };
        FillSubMatrix(TypOutputLevelMatrix, ReturnSubMatrices.TypOutputLevelMatrix, IndeciesArray);
        double[] TypOutputLevelMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.TypOutputLevelMatrix,
IndeciesArray.Length, "Existence of Maximum Sensitivity Region :");

        FillSubMatrix(TypOutputLevelMatrix, ReturnSubMatricesNotNormalized.TypOutputLevelMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.TypOutputLevelMatrix,
TypOutputLevelMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] TypFastThertimeConsMatrix = new double[,] {
            {1.0,3.0,4.0,6.0,5.0,1.0,3.0},
            {0.3333,1.0,2.0,4.0,3.0,0.3333,1.0},
            {0.25,0.5,1.0,2.0,2.0,0.25,1.0},
            {0.1667,0.25,0.5,1.0,0.5,0.1667,0.3333},
            {0.2,0.3333,0.5,2.0,1.0,0.2,0.3333},
            {1.0,3.0,4.0,6.0,5.0,1.0,3.0},
            {0.3333,1.0,1.0,3.0,3.0,0.3333,1.0}
        };
        FillSubMatrix(TypFastThertimeConsMatrix, ReturnSubMatrices.TypFastThertimeConsMatrix, IndeciesArray);
        double[] TypFastThertimeConsMatrixResult =
CalculateWeightsForEachCriterion(ReturnSubMatrices.TypFastThertimeConsMatrix, IndeciesArray.Length, "Typical Fast Thermal
Time Constant:");

        FillSubMatrix(TypFastThertimeConsMatrix, ReturnSubMatricesNotNormalized.TypFastThertimeConsMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.TypFastThertimeConsMatrix,
TypFastThertimeConsMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] TypSmallSizMatrix = new double[,] {
            {1.0,2.0,3.0,4.0,5.0,6.0,5.0},
            {0.5,1.0,2.0,3.0,4.0,5.0,4.0},
            {0.3333,0.5,1.0,2.0,4.0,6.0,4.0},
            {0.25,0.3333,0.5,1.0,2.0,3.0,2.0},
            {0.2,0.25,0.25,0.5,1.0,3.0,1.0},
            {0.1667,0.2,0.1667,0.3333,0.3333,1.0,0.5},
            {0.2,0.25,0.25,0.5,1.0,2.0,1.0}
        };
```

```
        FillSubMatrix(TypSmallSizMatrix, ReturnSubMatrices.TypSmallSizMatrix, IndeciesArray);
        double[] TypSmallSizMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.TypSmallSizMatrix,
IndeciesArray.Length, "Typical Small Size:");

        FillSubMatrix(TypSmallSizMatrix, ReturnSubMatricesNotNormalized.TypSmallSizMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.TypSmallSizMatrix,
TypSmallSizMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] NoiseImmunityMatrix = new double[,] {
            {1.0,0.1667,0.3333,0.25,0.25,0.5,0.25},
            {6.0,1.0,4.0,3.0,3.0,5.0,3.0},
            {3.0,0.25,1.0,0.5,0.5,2.0,0.5},
            {4.0,0.3333,2.0,1.0,1.0,3.0,1.0},
            {4.0,0.3333,2.0,1.0,1.0,4.0,1.0},
            {2.0,0.2,0.5,0.3333,0.25,1.0,0.25},
            {4.0,0.3333,2.0,1.0,1.0,4.0,1.0}
        };
        FillSubMatrix(NoiseImmunityMatrix, ReturnSubMatrices.NoiseImmunityMatrix, IndeciesArray);
        double[] NoiseImmunityMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.NoiseImmunityMatrix,
IndeciesArray.Length, "Noise Immunity:");

        FillSubMatrix(NoiseImmunityMatrix, ReturnSubMatricesNotNormalized.NoiseImmunityMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.NoiseImmunityMatrix,
NoiseImmunityMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] FraDurMatrix = new double[,] {
            {1.0,6.0,3.0,6.0,8.0,3.0,4.0},
            {0.1667,1.0,0.3333,2.0,3.0,0.3333,0.5},
            {0.3333,3.0,1.0,3.0,4.0,2.0,3.0},
            {0.1667,0.5,0.3333,1.0,3.0,0.25,0.3333},
            {0.125,0.3333,0.25,0.3333,1.0,0.25,0.3333},
            {0.3333,3.0,0.5,4.0,4.0,1.0,3.0},
            {0.25,2.0,0.3333,3.0,3.0,0.3333,1.0}

        };
        FillSubMatrix(FraDurMatrix, ReturnSubMatrices.FraDurMatrix, IndeciesArray);
        double[] FraDurMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.FraDurMatrix, IndeciesArray.Length,
"Fragility-Durability:");

        FillSubMatrix(FraDurMatrix, ReturnSubMatricesNotNormalized.FraDurMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.FraDurMatrix, FraDurMatrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] HiThGrEnMatrix = new double[,] {
            {1.0,4.0,5.0,7.0,8.0,4.0,6.0},
            {0.25,1.0,2.0,5.0,6.0,2.0,4.0},
            {0.2,0.5,1.0,3.0,3.0,0.5,2.0},
            {0.1429,0.2,0.3333,1.0,1.0,0.25,0.5},
            {0.125,0.1667,0.3333,1.0,1.0,0.2,0.3333},
            {0.25,0.5,2.0,4.0,5.0,1.0,4.0},
            {0.1667,0.25,0.5,2.0,3.0,0.25,1.0}
        };
        FillSubMatrix(HiThGrEnMatrix, ReturnSubMatrices.HiThGrEnMatrix, IndeciesArray);
        double[] HiThGrEnMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.HiThGrEnMatrix,
IndeciesArray.Length, "High Thermal Gradient Environment:");

        FillSubMatrix(HiThGrEnMatrix, ReturnSubMatricesNotNormalized.HiThGrEnMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.HiThGrEnMatrix, HiThGrEnMatrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] CorrResMatrix = new double[,] {
            {1.0,0.25,0.1667,0.5,0.1667,0.1667,0.25},
            {4.0,1.0,0.3333,2.0,0.25,0.25,1.0},
            {6.0,3.0,1.0,4.0,1.0,1.0,4.0},
            {2.0,0.5,0.25,1.0,0.25,0.25,0.5},
            {6.0,4.0,1.0,4.0,1.0,1.0,3.0},
            {6.0,4.0,1.0,4.0,1.0,1.0,3.0},
            {4.0,1.0,0.25,2.0,0.3333,0.3333,1.0}
        };
        FillSubMatrix(CorrResMatrix, ReturnSubMatrices.CorrResMatrix, IndeciesArray);
        double[] CorrResMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.CorrResMatrix, IndeciesArray.Length,
"Corrosion Resistance:");
```

```
        FillSubMatrix(CorrResMatrix, ReturnSubMatricesNotNormalized.CorrResMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.CorrResMatrix, CorrResMatrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////////
        double[,] PointAreaMeasMatrix = new double[,] {
            {1.0,2.0,3.0,4.0,6.0,6.0,4.0},
            {0.5,1.0,2.0,3.0,4.0,4.0,3.0},
            {0.3333,0.5,1.0,2.0,3.0,4.0,2.0},
            {0.25,0.3333,0.5,1.0,2.0,2.0,0.5},
            {0.1667,0.25,0.3333,0.5,1.0,1.0,0.5},
            {0.1667,0.25,0.25,0.5,1.0,1.0,2.0},
            {0.25,0.5,0.3333,2.0,2.0,0.5,1.0}
        };
        FillSubMatrix(PointAreaMeasMatrix, ReturnSubMatrices.PointAreaMeasMatrix, IndeciesArray);
        double[] PointAreaMeasMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.PointAreaMeasMatrix,
IndeciesArray.Length, "Point or Area Measurement:");

        FillSubMatrix(PointAreaMeasMatrix, ReturnSubMatricesNotNormalized.PointAreaMeasMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.PointAreaMeasMatrix,
PointAreaMeasMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////////
        double[,] ManuVarMatrix = new double[,] {
            {1.0, 0.3333, 0.1667,0.25,0.5,0.2,0.25},
            {3.0, 1.0, 0.3333,0.5,2.0,0.25,0.5},
            {6.0, 3.0, 1.0,4.0,3.0,6.0,3.0},
            {4.0,2.0,0.25,1.0,4.0,0.3333,0.5},
            {2.0,0.5,0.3333,0.25,1.0,0.2,0.25},
            {5.0,4.0,0.1667,3.0,5.0,1.0,2.0},
            {4.0,2.0,0.3333,2.0,4.0,0.5,1.0}
        };
        FillSubMatrix(ManuVarMatrix, ReturnSubMatrices.ManuVarMatrix, IndeciesArray);
        double[] ManuVarMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.ManuVarMatrix,
IndeciesArray.Length, "Manufacturing Variances:");

        FillSubMatrix(ManuVarMatrix, ReturnSubMatricesNotNormalized.ManuVarMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.ManuVarMatrix, ManuVarMatrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////////
        double[,] NistStanMatrix = new double[,] {
            {1.0   , 4.0, 1.0,1.0,1.0,4.0,5.0},
            {0.25, 1.0, 0.25,0.25,0.25,1.0,2.0},
            {1.0, 4.0, 1.0,1.0,1.0,4.0,5.0},
            {1.0,4.0,1.0,1.0,1.0,4.0,5.0},
            {1.0,4.0,1.0,1.0,1.0,4.0,5.0},
            {0.25,1.0,0.25,0.25,0.25,1.0,2.0},
            {0.2,0.5,0.2,0.2,0.2,0.5,1.0}
        };
        FillSubMatrix(NistStanMatrix, ReturnSubMatrices.NistStanMatrix, IndeciesArray);
        double[] NistStanMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.NistStanMatrix, IndeciesArray.Length,
"Standards Exist:");

        FillSubMatrix(NistStanMatrix, ReturnSubMatricesNotNormalized.NistStanMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.NistStanMatrix, NistStanMatrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////////
        double[,] CostMatrix = new double[,] {
            {1.0,1.0,6.0,3.0,3.0,6.0,3.0},
            {1.0,1.0,6.0,3.0,3.0,6.0,3.0},
            {0.1667,0.1667,1.0,0.25,0.25,1.0,0.25},
            {0.3333,0.3333,4.0,1.0,1.0,4.0,1.0},
            {0.3333,0.3333,4.0,1.0,1.0,4.0,1.0},
            {0.1667,0.1667,1.0,0.25,0.25,1.0,0.25},
            {0.3333,0.3333,4.0,1.0,1.0,4.0,1.0}
        };
        FillSubMatrix(CostMatrix, ReturnSubMatrices.CostMatrix, IndeciesArray);
        double[] CostMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.CostMatrix, IndeciesArray.Length,
"Cost:");

        FillSubMatrix(CostMatrix, ReturnSubMatricesNotNormalized.CostMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.CostMatrix, CostMatrixResult,
IndeciesArray.Length);
```

```
        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] weightsOfCriteria = new double[,] {
            {1.0,4.0,3.0,4.0},
            {0.25,1.0,0.5,1.0},
            {0.3333,2.0,1.0,2.0},
            {0.25,1.0,0.5,1.0}
        };

        double[] weightsOfCriteriaResults = CalculateWeightsForEachCriterion(weightsOfCriteria, 4, "Weights of Criteria:");

        double[,] weightsOfCriteria2 = new double[,] {
            {1.0,4.0,3.0,4.0},
            {0.25,1.0,0.5,1.0},
            {0.3333,2.0,1.0,2.0},
            {0.25,1.0,0.5,1.0}
        };
        ConsistencyIndex = ComputeConsistencyIndex(weightsOfCriteria2, weightsOfCriteriaResults, 4);
        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] Weightssubcriteriastatic = new double[,] {
            {1.0,1.0,5.0,4.0,4.0,2.0,5.0,6.0,7.0,8.0,6.0},
            {1.0,1.0,5.0,4.0,4.0,2.0,5.0,6.0,7.0,8.0,6.0},
            {0.2,0.2,1.0,0.3333,0.3333,0.25,1.0,2.0,4.0,5.0,3.0},
            {0.25,0.25,3.0,1.0,2.0,0.5,3.0,3.0,5.0,6.0,4.0},
            {0.25,0.25,3.0,0.5,1.0,0.3333,3.0,5.0,6.0,8.0,4.0},
            {0.5,0.5,4.0,2.0,3.0,1.0,4.0,5.0,6.0,8.0,5.0},
            {0.2,0.2,1.0,0.3333,0.3333,0.25,1.0,1.0,4.0,6.0,3.0},
            {0.1667,0.1667,0.5,0.3333,0.2,0.2,1.0,1.0,3.0,4.0,1.0},
            {0.1429,0.1429,0.25,0.2,0.1667,0.1667,0.25,0.3333,1.0,2.0,0.3333},
            {0.125,0.125,0.2,0.1667,0.125,0.125,0.1667,0.25,0.5,1.0,0.25},
            {0.1667,0.1667,0.3333,0.25,0.25,0.2,0.3333,1.0,3.0,4.0,1.0}
        };
        double[] weightsSubCriteriaStaticResult = CalculateWeightsForEachCriterion(Weightssubcriteriastatic, 11, "Weights of Sub-
Criteria Static:");

        double[,] Weightssubcriteriastatic2 = new double[,] {
            {1.0,1.0,5.0,4.0,4.0,2.0,5.0,6.0,7.0,8.0,6.0},
            {1.0,1.0,5.0,4.0,4.0,2.0,5.0,6.0,7.0,8.0,6.0},
            {0.2,0.2,1.0,0.3333,0.3333,0.25,1.0,2.0,4.0,5.0,3.0},
            {0.25,0.25,3.0,1.0,2.0,0.5,3.0,3.0,5.0,6.0,4.0},
            {0.25,0.25,3.0,0.5,1.0,0.3333,3.0,5.0,6.0,8.0,4.0},
            {0.5,0.5,4.0,2.0,3.0,1.0,4.0,5.0,6.0,8.0,5.0},
            {0.2,0.2,1.0,0.3333,0.3333,0.25,1.0,1.0,4.0,6.0,3.0},
            {0.1667,0.1667,0.5,0.3333,0.2,0.2,1.0,1.0,3.0,4.0,1.0},
            {0.1429,0.1429,0.25,0.2,0.1667,0.1667,0.25,0.3333,1.0,2.0,0.3333},
            {0.125,0.125,0.2,0.1667,0.125,0.125,0.1667,0.25,0.5,1.0,0.25},
            {0.1667,0.1667,0.3333,0.25,0.25,0.2,0.3333,1.0,3.0,4.0,1.0}
        };
        ConsistencyIndex = ComputeConsistencyIndex(Weightssubcriteriastatic2, weightsSubCriteriaStaticResult, 11);
        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] weightssubCriteriaDynamic = new double[,] {
            {1.0,2.0,0.1667},
            {0.5,1.0,0.1667},
            {6.0,6.0,1.0}
        };
        double[] weightsSubCriteriaDynamicResult = CalculateWeightsForEachCriterion(weightssubCriteriaDynamic, 3, "Weights of
Sub-Criteria Dynamic:");

        double[,] weightssubCriteriaDynamic2 = new double[,] {
            {1.0,2.0,0.1667},
            {0.5,1.0,0.1667},
            {6.0,6.0,1.0}
        };
        ConsistencyIndex = ComputeConsistencyIndex(weightssubCriteriaDynamic2, weightsSubCriteriaDynamicResult, 3);
        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] weightssubCriteriaEnv = new double[,] {
            {1.0,3.0,0.3333,4.0,0.25},
            {0.3333,1.0,0.25,3.0,0.2},
            {3.0,4.0,1.0,5.0,0.5},
            {0.25,0.3333,0.2,1.0,0.1667},
            {4.0,5.0,2.0,6.0,1.0}
        };
        double[] weightsSubCriteriaEnvironmentalResult = CalculateWeightsForEachCriterion(weightssubCriteriaEnv, 5, "Weights of
Sub-Criteria Environmental:");

        double[,] weightssubCriteriaEnv2 = new double[,] {
```

```
            {1.0,3.0,0.3333,4.0,0.25},
            {0.3333,1.0,0.25,3.0,0.2},
            {3.0,4.0,1.0,5.0,0.5},
            {0.25,0.3333,0.2,1.0,0.1667},
            {4.0,5.0,2.0,6.0,1.0}
        };
        ConsistencyIndex = ComputeConsistencyIndex(weightssubCriteriaEnv2, weightsSubCriteriaEnvironmentalResult, 5);
        ////////////////////////////////////////////////////////////////////////////////////////////
        double[,] Others = new double[,] {
            {1.0,3.0,0.5,0.25},
            {0.3333,1.0,0.3333,0.2},
            {2.0,3.0,1.0,0.3333},
            {4.0,5.0,3.0,1.0}
        };
        double[] weightsSubCriteriaOthersResult = CalculateWeightsForEachCriterion(Others, 4, "Weights of Sub-Criteria Others:");

        double[,] Others2 = new double[,] {
            {1.0,3.0,0.5,0.25},
            {0.3333,1.0,0.3333,0.2},
            {2.0,3.0,1.0,0.3333},
            {4.0,5.0,3.0,1.0}
        };
        ConsistencyIndex = ComputeConsistencyIndex(Others2, weightsSubCriteriaOthersResult, 4);
        ////////////////////////////////////////////////////////////////////////////////////////////////////
        //Calculate sub criteria aggregate weights with respect to final goal
        Scale(weightsSubCriteriaStaticResult, weightsOfCriteriaResults[0]);
        Scale(weightsSubCriteriaDynamicResult, weightsOfCriteriaResults[1]);
        Scale(weightsSubCriteriaEnvironmentalResult, weightsOfCriteriaResults[2]);
        Scale(weightsSubCriteriaOthersResult, weightsOfCriteriaResults[3]);

        ////////////////////////////////////////////////////////////////////////////////////////////////////
        //Calculate sub criteria aggregate weights for ThermoCouple (alternative 1)
        double ThermoCoupleFinalScore = 0.0;
        double ThermisterFinalScore = 0.0;
        double RTDFinalScore = 0.0;
        double BimetallicFinalScore = 0.0;
        double ThermometerFinalScore = 0.0;
        double PyrometerFinalScore = 0.0;
        double LCDDidplayFinalScore = 0.0;

        for (int k = 0; k < IndeciesArray.Length; k++)
        {
           if (IndeciesArray[k] == AHP.SENSOR_THERMOCOUPLE)
           {
              double[] ThermoCoupleAggregateSubCriteriaStatic = new double[11];
              double[] ThermoCoupleAggregateSubCriteriaDynamic = new double[3];
              double[] ThermoCoupleAggregateSubCriteriaEnvironmental = new double[5];
              double[] ThermoCoupleAggregateSubCriteriaOthers = new double[4];

              ThermoCoupleAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
              ThermoCoupleAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
              ThermoCoupleAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
              ThermoCoupleAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
              ThermoCoupleAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
              ThermoCoupleAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
              ThermoCoupleAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
              ThermoCoupleAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
              ThermoCoupleAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
              ThermoCoupleAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
              ThermoCoupleAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

              ThermoCoupleFinalScore = ThermoCoupleFinalScore +
GetSummationOfVectorElements(ThermoCoupleAggregateSubCriteriaStatic);

              ThermoCoupleAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] *
weightsSubCriteriaDynamicResult[0];
              ThermoCoupleAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] *
weightsSubCriteriaDynamicResult[1];
              ThermoCoupleAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] *
weightsSubCriteriaDynamicResult[2];

              ThermoCoupleFinalScore = ThermoCoupleFinalScore +
GetSummationOfVectorElements(ThermoCoupleAggregateSubCriteriaDynamic);

              ThermoCoupleAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[0];
```

```
            ThermoCoupleAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
            ThermoCoupleAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[2];
            ThermoCoupleAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[3];
            ThermoCoupleAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[4];

            ThermoCoupleFinalScore = ThermoCoupleFinalScore +
GetSummationOfVectorElements(ThermoCoupleAggregateSubCriteriaEnvironmental);

            ThermoCoupleAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
            ThermoCoupleAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
            ThermoCoupleAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
            ThermoCoupleAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

            ThermoCoupleFinalScore = ThermoCoupleFinalScore +
GetSummationOfVectorElements(ThermoCoupleAggregateSubCriteriaOthers);
            SensorRanks[k] = ThermoCoupleFinalScore;

            System.Console.Write("\n\n");
            System.Console.Write("Thermo Couple Final Score = {0}", ThermoCoupleFinalScore);
        }
        ////////////////////////////////////////////////////////////////////////////////////////////////////////////
        //Calculate sub criteria aggregate weights for Thermister (alternative 2)
        else if (IndeciesArray[k] == AHP.SENSOR_THERMISTER)
        {
            double[] ThermisterAggregateSubCriteriaStatic = new double[11];
            double[] ThermisterAggregateSubCriteriaDynamic = new double[3];
            double[] ThermisterAggregateSubCriteriaEnvironmental = new double[5];
            double[] ThermisterAggregateSubCriteriaOthers = new double[4];

            ThermisterAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
            ThermisterAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
            ThermisterAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
            ThermisterAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
            ThermisterAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
            ThermisterAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
            ThermisterAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
            ThermisterAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
            ThermisterAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
            ThermisterAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
            ThermisterAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

            ThermisterFinalScore = ThermisterFinalScore + GetSummationOfVectorElements(ThermisterAggregateSubCriteriaStatic);

            ThermisterAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] * weightsSubCriteriaDynamicResult[0];
            ThermisterAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] * weightsSubCriteriaDynamicResult[1];
            ThermisterAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] *
weightsSubCriteriaDynamicResult[2];

            ThermisterFinalScore = ThermisterFinalScore +
GetSummationOfVectorElements(ThermisterAggregateSubCriteriaDynamic);

            ThermisterAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[0];
            ThermisterAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
            ThermisterAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[2];
            ThermisterAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[3];
            ThermisterAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[4];

            ThermisterFinalScore = ThermisterFinalScore +
GetSummationOfVectorElements(ThermisterAggregateSubCriteriaEnvironmental);

            ThermisterAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
            ThermisterAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
            ThermisterAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
            ThermisterAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

            ThermisterFinalScore = ThermisterFinalScore + GetSummationOfVectorElements(ThermisterAggregateSubCriteriaOthers);

            System.Console.Write("\n\n");
            System.Console.Write("Thermister Final Score = {0}", ThermisterFinalScore);
```

```
                SensorRanks[k] = ThermisterFinalScore;
            }
            ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
            //Calculate sub criteria aggregate weights for RTD (alternative 3)
            else if (IndeciesArray[k] == AHP.SENSOR_RTD)
            {
                double[] RTDAggregateSubCriteriaStatic = new double[11];
                double[] RTDAggregateSubCriteriaDynamic = new double[3];
                double[] RTDAggregateSubCriteriaEnvironmental = new double[5];
                double[] RTDAggregateSubCriteriaOthers = new double[4];

                RTDAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
                RTDAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
                RTDAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
                RTDAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
                RTDAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
                RTDAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
                RTDAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
                RTDAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
                RTDAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
                RTDAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
                RTDAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

                RTDFinalScore = RTDFinalScore + GetSummationOfVectorElements(RTDAggregateSubCriteriaStatic);

                RTDAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] * weightsSubCriteriaDynamicResult[0];
                RTDAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] * weightsSubCriteriaDynamicResult[1];
                RTDAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] * weightsSubCriteriaDynamicResult[2];

                RTDFinalScore = RTDFinalScore + GetSummationOfVectorElements(RTDAggregateSubCriteriaDynamic);

                RTDAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[0];
                RTDAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
                RTDAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[2];
                RTDAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[3];
                RTDAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[4];

                RTDFinalScore = RTDFinalScore + GetSummationOfVectorElements(RTDAggregateSubCriteriaEnvironmental);

                RTDAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
                RTDAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
                RTDAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
                RTDAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

                RTDFinalScore = RTDFinalScore + GetSummationOfVectorElements(RTDAggregateSubCriteriaOthers);
                SensorRanks[k] = RTDFinalScore;
            }
            else if (IndeciesArray[k] == AHP.SENSOR_BIMETALLIC)
            {
                double[] BimetallicAggregateSubCriteriaStatic = new double[11];
                double[] BimetallicAggregateSubCriteriaDynamic = new double[3];
                double[] BimetallicAggregateSubCriteriaEnvironmental = new double[5];
                double[] BimetallicAggregateSubCriteriaOthers = new double[4];

                BimetallicAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
                BimetallicAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
                BimetallicAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
                BimetallicAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
                BimetallicAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
                BimetallicAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
                BimetallicAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
                BimetallicAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
                BimetallicAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
                BimetallicAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
                BimetallicAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

                BimetallicFinalScore = BimetallicFinalScore + GetSummationOfVectorElements(BimetallicAggregateSubCriteriaStatic);

                BimetallicAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] * weightsSubCriteriaDynamicResult[0];
                BimetallicAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] * weightsSubCriteriaDynamicResult[1];
                BimetallicAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] *
weightsSubCriteriaDynamicResult[2];
```

```
                BimetallicFinalScore = BimetallicFinalScore +
GetSummationOfVectorElements(BimetallicAggregateSubCriteriaDynamic);

                BimetallicAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[0];
                BimetallicAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
                BimetallicAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[2];
                BimetallicAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[3];
                BimetallicAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[4];

                BimetallicFinalScore = BimetallicFinalScore +
GetSummationOfVectorElements(BimetallicAggregateSubCriteriaEnvironmental);

                BimetallicAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
                BimetallicAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
                BimetallicAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
                BimetallicAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

                BimetallicFinalScore = BimetallicFinalScore + GetSummationOfVectorElements(BimetallicAggregateSubCriteriaOthers);
                SensorRanks[k] = BimetallicFinalScore;
            }
            else if (IndeciesArray[k] == AHP.SENSOR_THERMOMETER)
            {
                double[] ThermometerAggregateSubCriteriaStatic = new double[11];
                double[] ThermometerAggregateSubCriteriaDynamic = new double[3];
                double[] ThermometerAggregateSubCriteriaEnvironmental = new double[5];
                double[] ThermometerAggregateSubCriteriaOthers = new double[4];

                ThermometerAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
                ThermometerAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
                ThermometerAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
                ThermometerAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
                ThermometerAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
                ThermometerAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
                ThermometerAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
                ThermometerAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
                ThermometerAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
                ThermometerAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
                ThermometerAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

                ThermometerFinalScore = ThermometerFinalScore +
GetSummationOfVectorElements(ThermometerAggregateSubCriteriaStatic);

                ThermometerAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] * weightsSubCriteriaDynamicResult[0];
                ThermometerAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] * weightsSubCriteriaDynamicResult[1];
                ThermometerAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] *
weightsSubCriteriaDynamicResult[2];

                ThermometerFinalScore = ThermometerFinalScore +
GetSummationOfVectorElements(ThermometerAggregateSubCriteriaDynamic);

                ThermometerAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[0];
                ThermometerAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
                ThermometerAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[2];
                ThermometerAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[3];
                ThermometerAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[4];

                ThermometerFinalScore = ThermometerFinalScore +
GetSummationOfVectorElements(ThermometerAggregateSubCriteriaEnvironmental);

                ThermometerAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
                ThermometerAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
                ThermometerAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
                ThermometerAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

                ThermometerFinalScore = ThermometerFinalScore +
GetSummationOfVectorElements(ThermometerAggregateSubCriteriaOthers);
                SensorRanks[k] = ThermometerFinalScore;
            }
```

```
            else if (IndeciesArray[k] == AHP.SENSOR_PYROMETER)
            {
                double[] PyrometerAggregateSubCriteriaStatic = new double[11];
                double[] PyrometerAggregateSubCriteriaDynamic = new double[3];
                double[] PyrometerAggregateSubCriteriaEnvironmental = new double[5];
                double[] PyrometerAggregateSubCriteriaOthers = new double[4];

                PyrometerAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
                PyrometerAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
                PyrometerAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
                PyrometerAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
                PyrometerAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
                PyrometerAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
                PyrometerAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
                PyrometerAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
                PyrometerAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
                PyrometerAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
                PyrometerAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

                PyrometerFinalScore = PyrometerFinalScore + GetSummationOfVectorElements(PyrometerAggregateSubCriteriaStatic);

                PyrometerAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] * weightsSubCriteriaDynamicResult[0];
                PyrometerAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] * weightsSubCriteriaDynamicResult[1];
                PyrometerAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] *
weightsSubCriteriaDynamicResult[2];

                PyrometerFinalScore = PyrometerFinalScore +
GetSummationOfVectorElements(PyrometerAggregateSubCriteriaDynamic);

                PyrometerAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[0];
                PyrometerAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
                PyrometerAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[2];
                PyrometerAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[3];
                PyrometerAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[4];

                PyrometerFinalScore = PyrometerFinalScore +
GetSummationOfVectorElements(PyrometerAggregateSubCriteriaEnvironmental);

                PyrometerAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
                PyrometerAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
                PyrometerAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
                PyrometerAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

                PyrometerFinalScore = PyrometerFinalScore + GetSummationOfVectorElements(PyrometerAggregateSubCriteriaOthers);
                SensorRanks[k] = PyrometerFinalScore;
            }
            else if (IndeciesArray[k] == AHP.SENSOR_LCD_DISPLAY)
            {
                double[] LCDDidplayAggregateSubCriteriaStatic = new double[11];
                double[] LCDDidplayAggregateSubCriteriaDynamic = new double[3];
                double[] LCDDidplayAggregateSubCriteriaEnvironmental = new double[5];
                double[] LCDDidplayAggregateSubCriteriaOthers = new double[4];

                LCDDidplayAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
                LCDDidplayAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
                LCDDidplayAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
                LCDDidplayAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
                LCDDidplayAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
                LCDDidplayAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
                LCDDidplayAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
                LCDDidplayAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
                LCDDidplayAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
                LCDDidplayAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
                LCDDidplayAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

                LCDDidplayFinalScore = LCDDidplayFinalScore +
GetSummationOfVectorElements(LCDDidplayAggregateSubCriteriaStatic);

                LCDDidplayAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] * weightsSubCriteriaDynamicResult[0];
                LCDDidplayAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] * weightsSubCriteriaDynamicResult[1];
                LCDDidplayAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] *
weightsSubCriteriaDynamicResult[2];
```

```
                LCDDidplayFinalScore = LCDDidplayFinalScore +
GetSummationOfVectorElements(LCDDidplayAggregateSubCriteriaDynamic);

                LCDDidplayAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[0];
                LCDDidplayAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
                LCDDidplayAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[2];
                LCDDidplayAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[3];
                LCDDidplayAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[4];

                LCDDidplayFinalScore = LCDDidplayFinalScore +
GetSummationOfVectorElements(LCDDidplayAggregateSubCriteriaEnvironmental);

                LCDDidplayAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
                LCDDidplayAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
                LCDDidplayAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
                LCDDidplayAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

                LCDDidplayFinalScore = LCDDidplayFinalScore +
GetSummationOfVectorElements(LCDDidplayAggregateSubCriteriaOthers);
                SensorRanks[k] = LCDDidplayFinalScore;
            }
        }

        textBoxResults.AppendText("\n\n");
        textBoxResults.AppendText("\nSensor Ranks: \n");
        for (int j = 0; j < SensorRanks.Length; j++)
        {
            textBoxResults.AppendText("" + SensorRanks[j] + "\n");
        }

        return SensorRanks;
    }
  }
}
```

4. ChemicalProcess.cs File

```
using System;
using System.Collections.Generic;
using System.Text;

namespace AhpCaseStudy1GUI
{
   class ChemicalReactions
   {
      public System.Windows.Forms.TextBox textBoxResults;

      void Print(double[,] mat, int nDimensionSize, string strTitle)
      {
         if (strTitle == "")
            strTitle = "Matrix";

         //System.Console.WriteLine(strTitle);
         textBoxResults.AppendText("\n");
         textBoxResults.AppendText(strTitle);
         textBoxResults.AppendText("\n");
         textBoxResults.AppendText("\n");
         for (int i = 0; i < nDimensionSize; i++)
         {
            for (int j = 0; j < nDimensionSize; j++)
            {
               //System.Console.Write("{0}\t\t", mat[i, j]);
               textBoxResults.AppendText("     " + mat[i, j]);
               //textBoxResults.Update();
            }
            //System.Console.Write("\n");
            textBoxResults.AppendText("\n");
         }

         //System.Console.Write("\n\n");
         textBoxResults.AppendText("\n\n");
      }

      void Print(double[] vector, int nDimensionSize)
      {
         //System.Console.WriteLine("Relative Weight Vector = ");
         textBoxResults.AppendText("Relative Weight Vector = ");
         for (int i = 0; i < nDimensionSize; i++)
         {
            //System.Console.Write("{0}\t\t", vector[i]);
            textBoxResults.AppendText("     " + vector[i]);
         }
         //System.Console.Write("\n");
         textBoxResults.AppendText("\n");
      }

      void Scale(double[] vector, double scaleFactor)
      {
         for (int i = 0; i < vector.Length; i++)
         {
            vector[i] = vector[i] * scaleFactor;//vector[i] *= scaleFactor;
         }
         Print(vector, vector.Length);
      }

      double GetSummationOfVectorElements(double[] vector)
      {
         double Summation = 0.0;
         for (int i = 0; i < vector.Length; i++)
         {
            Summation = Summation + vector[i];
         }
         return Summation;
      }

      double[] CalculateWeightsForEachCriterion(double[,] expertAssesmentMatrix, int nDimensionSize, string strTitle)
      {
         //check for vald input values
         double[] weightVector = new double[nDimensionSize];
```

```
        if (expertAssesmentMatrix == null)
            return weightVector;

        Print(expertAssesmentMatrix, nDimensionSize, strTitle);

        //Calculate the weight factor for each colmun
        //The result is a vetor
        double[] wieghtFactor = new double[nDimensionSize];
        for (int j = 0; j < nDimensionSize; j++)
        {
            double result = 0.0;
            for (int k = 0; k < nDimensionSize; k++)
            {
                result += expertAssesmentMatrix[k, j]; //result = result + mat1[k, j];
            }
            wieghtFactor[j] = result;
        }

        double[,] mat1ImmediatResult = new double[nDimensionSize, nDimensionSize];
        for (int j = 0; j < nDimensionSize; j++)
        {
            for (int k = 0; k < nDimensionSize; k++)
            {
                expertAssesmentMatrix[k, j] = expertAssesmentMatrix[k, j] / wieghtFactor[j];
            }
        }

        //Calculate the weight factor for each colmun
        //The result is a vetor
        for (int j = 0; j < nDimensionSize; j++)
        {
            double result = 0.0;
            for (int k = 0; k < nDimensionSize; k++)
            {
                result += expertAssesmentMatrix[j, k]; //result = result + mat1[k, j];
            }
            weightVector[j] = result / nDimensionSize;
        }

        Print(weightVector, nDimensionSize);

//System.Console.WriteLine("_____
_____\n");


        return weightVector;
    }

    void FillSubMatrix(double[,] SourceMatrix, double[,] destinationSubMatrix, int[] IndeciesArray)
    {
        if (SourceMatrix == null || destinationSubMatrix == null || IndeciesArray == null)
            return;

        for (int i = 0; i < IndeciesArray.Length; i++)
        {
            for (int j = 0; j < IndeciesArray.Length; j++)
            {
                //destinationSubMatrix[IndeciesArray[i], IndeciesArray[j]] = SourceMatrix[IndeciesArray[i], IndeciesArray[j]];
                destinationSubMatrix[i, j] = SourceMatrix[IndeciesArray[i], IndeciesArray[j]];
            }
        }
    }

    double ComputeConsistencyIndex(double[,] Matrix, double[] vector, int nDimensionSize)
    {
        //check for vald input values
        double[] transposeVector = new double[nDimensionSize];
        double[] randomIndex = { 1.0, 0.5, 0.58, 0.9, 1.12, 1.24, 1.32 };

        if (Matrix == null)
            return 0.0;

        for (int j = 0; j < nDimensionSize; j++)
        {
            double result = 0.0;
            for (int k = 0; k < nDimensionSize; k++)
```

```
            {
                result += (Matrix[j, k] * vector[k]);
            }
            transposeVector[j] = result;
        }
        double division = 0.0;
        for (int k = 0; k < nDimensionSize; k++)
        {
            division += (transposeVector[k] / vector[k]);
            //transposeVector[k] = transposeVector[k]/vector[k];

        }
        division = division / nDimensionSize;

        double consistencyIndex = (division - nDimensionSize) / (nDimensionSize - 1);

        if (consistencyIndex < 0.0 && consistencyIndex > -0.0005)
            consistencyIndex = 0.0;

        if (consistencyIndex < 0.0005)
            consistencyIndex = 0.0;

        textBoxResults.AppendText("Consistency Index = ");
        textBoxResults.AppendText("" + consistencyIndex + "\n");

        textBoxResults.AppendText("\nConsistency Ratio = ");
        if (nDimensionSize > randomIndex.Length)
            textBoxResults.AppendText("" + (consistencyIndex / 1.59));
        else
            textBoxResults.AppendText("" + (consistencyIndex / randomIndex[nDimensionSize - 1]));
        textBoxResults.AppendText("\n");
        textBoxResults.AppendText("_____\n");
        return consistencyIndex;
    }

    public double[] SelectBestSensor(int[] IndeciesArray)
    {
        double[] SensorRanks = new double[IndeciesArray.Length];
        AHPSubCriteria ReturnSubMatrices = new AHPSubCriteria();
        ReturnSubMatrices.CreateSubMatrices(IndeciesArray.Length);
        AHPSubCriteria ReturnSubMatricesNotNormalized = new AHPSubCriteria();
        ReturnSubMatricesNotNormalized.CreateSubMatrices(IndeciesArray.Length);

        double[,] MaximumTempMatrix = new double[,] {
                {1.0,3.0,1.0,9.0,4.0,6.0,4.0},
                {0.3333,1.0,0.3333,6.0,2.0,5.0,2.0},
                {1.0,3.0,1.0,9.0,4.0,6.0,4.0},
                {0.1111,0.1667,0.1111,1.0,0.1667,0.3333,0.1667},
                {0.25,0.5,0.25,6.0,1.0,4.0,1.0},
                {0.1667,0.2,0.1667,3.0,0.25,1.0,0.25},
                {0.25,0.5,0.25,6.0,1.0,4.0,1.0}
        };
        FillSubMatrix(MaximumTempMatrix, ReturnSubMatrices.MaximumTempMatrix, IndeciesArray);
        double[] MaximumTempMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.MaximumTempMatrix,
IndeciesArray.Length, "Maximum Operating Temprature: ");

        FillSubMatrix(MaximumTempMatrix, ReturnSubMatricesNotNormalized.MaximumTempMatrix, IndeciesArray);
        double ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.MaximumTempMatrix,
MaximumTempMatrixResult, IndeciesArray.Length);
        ////////////////////////////////////////////////////////////////////////////////////////////////
        double[,] MinimumTempMatrix = new double[,] {
                {1.0,2.0,2.0,0.5,0.5,0.125,0.3333},
                {0.5,1.0,1.0,0.3333,0.3333,0.125,0.25},
                {0.5,1.0,1.0,0.3333,0.3333,0.125,0.25},
                {2.0,3.0,3.0,1.0,1.0,0.25,0.5},
                {2.0,3.0,3.0,1.0,1.0,0.25,0.5},
                {8.0,8.0,8.0,4.0,4.0,1.0,4.0},
                {3.0,4.0,4.0,2.0,2.0,0.25,1.0}
        };
        FillSubMatrix(MinimumTempMatrix, ReturnSubMatrices.MinimumTempMatrix, IndeciesArray);
        double[] MinimumTempMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.MinimumTempMatrix,
IndeciesArray.Length, "Minimum Operating Temprature: ");

        FillSubMatrix(MinimumTempMatrix, ReturnSubMatricesNotNormalized.MinimumTempMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.MinimumTempMatrix,
MinimumTempMatrixResult, IndeciesArray.Length);
```

```
        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] TempCurveMtrix = new double[,] {
            {1.0,3.0,0.3333,5.0,5.0,4.0,2.0},
            {0.3333,1.0,0.1667,2.0,2.0,1.0,0.3333},
            {3.0,6.0,1.0,6.0,6.0,6.0,4.0},
            {0.2,0.5,0.1667,1.0,1.0,0.3333,0.25},
            {0.2,0.5,0.1667,1.0,1.0,0.3333,0.25},
            {0.25,1.0,0.1667,3.0,3.0,1.0,0.5},
            {0.5,3.0,0.25,4.0,4.0,2.0,1.0}
        };
        FillSubMatrix(TempCurveMtrix, ReturnSubMatrices.TempCurveMtrix, IndeciesArray);
        double[] TempCurveMtrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.TempCurveMtrix,
IndeciesArray.Length, "Temperature Curve:");

        FillSubMatrix(TempCurveMtrix, ReturnSubMatricesNotNormalized.TempCurveMtrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.TempCurveMtrix, TempCurveMtrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] MaxSensitivityMatrix = new double[,] {
            {1.0,0.1111,0.2,2.0,2.0,2.0,0.3333},
            {9.0,1.0,4.0,9.0,9.0,6.0,4.0},
            {5.0,0.25,1.0,4.0,5.0,4.0,2.0},
            {0.5,0.1111,0.25,1.0,2.0,2.0,0.25},
            {0.5,.1111,0.2,0.5,1.0,1.0,0.25},
            {0.5,0.1667,0.25,0.5,1.0,1.0,0.25},
            {3.0,0.25,0.5,4.0,4.0,4.0,1.0}
        };
        FillSubMatrix(MaxSensitivityMatrix, ReturnSubMatrices.MaxSensitivityMatrix, IndeciesArray);
        double[] MaxSensitivityMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.MaxSensitivityMatrix,
IndeciesArray.Length, "Maximum Sensitivity Region:");

        FillSubMatrix(MaxSensitivityMatrix, ReturnSubMatricesNotNormalized.MaxSensitivityMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.MaxSensitivityMatrix,
MaxSensitivityMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] SelfHeatingMatrix = new double[,] {
            {1.0,8.0,3.0,1.0,1.0,2.0},
            {0.125,1.0,0.2,0.25,0.2,0.1667,0.25},
            {0.3333,5.0,1.0,0.5,0.5,0.3333,1.0},
            {0.3333,4.0,2.0,1.0,1.0,0.5,1.0},
            {1.0,5.0,2.0,1.0,1.0,1.0,2.0},
            {1.0,6.0,3.0,2.0,1.0,1.0,1.0},
            {0.5,4.0,1.0,1.0,0.5,1.0,1.0}
        };
        FillSubMatrix(SelfHeatingMatrix, ReturnSubMatrices.SelfHeatingMatrix, IndeciesArray);
        double[] SelfHeatingMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.SelfHeatingMatrix,
IndeciesArray.Length, "Self-Heating Issues:");

        FillSubMatrix(SelfHeatingMatrix, ReturnSubMatricesNotNormalized.SelfHeatingMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.SelfHeatingMatrix,
SelfHeatingMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] LongTermStabilityMatrix = new double[,] {
            {1.0,0.25,0.1667,2.0,0.3333,0.5,0.25},
            {4.0,1.0,0.3333,4.0,3.0,3.0,2.0},
            {6.0,3.0,1.0,8.0,4.0,5.0,3.0},
            {0.5,0.25,0.125,1.0,0.3333,0.3333,0.25},
            {3.0,0.3333,0.25,3.0,1.0,2.0,0.5},
            {2.0,0.3333,0.2,3.0,0.5,1.0,0.3333},
            {4.0,0.5,0.3333,4.0,2.0,3.0,1.0}
        };
        FillSubMatrix(LongTermStabilityMatrix, ReturnSubMatrices.LongTermStabilityMatrix, IndeciesArray);
        double[] LongTermStabilityMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.LongTermStabilityMatrix,
IndeciesArray.Length, "Long Term Stability and Accuracy:");

        FillSubMatrix(LongTermStabilityMatrix, ReturnSubMatricesNotNormalized.LongTermStabilityMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.LongTermStabilityMatrix,
LongTermStabilityMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] TypTempCoeffMatrix = new double[,] {
            {1.0,0.1667,0.3333,4.0,4.0,4.0,0.5},
```

```
                {6.0,1.0,3.0,6.0,6.0,6.0,6.0},
                {3.0,0.3333,1.0,5.0,6.0,5.0,2.0},
                {0.25,0.1667,0.2,1.0,1.0,1.0,0.2},
                {0.25,0.1667,0.1667,1.0,1.0,1.0,0.2},
                {0.25,0.1667,0.2,1.0,1.0,1.0,0.3333},
                {2.0,0.1667,0.5,5.0,5.0,3.0,1.0}
            };
            FillSubMatrix(TypTempCoeffMatrix, ReturnSubMatrices.TypTempCoeffMatrix, IndeciesArray);
            double[] TypTempCoeffMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.TypTempCoeffMatrix,
IndeciesArray.Length, "Typical Temperature Coefficient:");

            FillSubMatrix(TypTempCoeffMatrix, ReturnSubMatricesNotNormalized.TypTempCoeffMatrix, IndeciesArray);
            ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.TypTempCoeffMatrix,
TypTempCoeffMatrixResult, IndeciesArray.Length);

            ////////////////////////////////////////////////////////////////////////////////////////////
            double[,] ExtWiresMatrix = new double[,] {
                {1.0,0.1667,0.16667,0.125,0.125,0.125,0.125},
                {6.0,1.0,1.0,0.25,0.25,0.25,0.25},
                {6.0,1.0,1.0,0.25,0.25,0.25,0.25},
                {8.0,4.0,4.0,1.0,1.0,1.0,1.0},
                {8.0,4.0,4.0,1.0,1.0,1.0,1.0},
                {8.0,4.0,4.0,1.0,1.0,1.0,1.0},
                {8.0,4.0,4.0,1.0,1.0,1.0,1.0}
            };
            FillSubMatrix(ExtWiresMatrix, ReturnSubMatrices.ExtWiresMatrix, IndeciesArray);
            double[] ExtWiresMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.ExtWiresMatrix,
IndeciesArray.Length, "Extension Wires:");

            FillSubMatrix(ExtWiresMatrix, ReturnSubMatricesNotNormalized.ExtWiresMatrix, IndeciesArray);
            ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.ExtWiresMatrix, ExtWiresMatrixResult,
IndeciesArray.Length);

            ////////////////////////////////////////////////////////////////////////////////////////////
            double[,] LongWireMatrix = new double[,] {
                {1.0,0.3333,1.0,0.1667,0.1667,0.1667,0.1667},
                {3.0,1.0,3.0,0.5,0.5,0.5,0.5},
                {1.0,0.3333,1.0,0.1667,0.1667,0.1667,0.1667},
                {6.0,2.0,6.0,1.0,1.0,1.0,1.0},
                {6.0,2.0,6.0,1.0,1.0,1.0,1.0},
                {6.0,2.0,6.0,1.0,1.0,1.0,1.0},
                {6.0,2.0,6.0,1.0,1.0,1.0,1.0}
            };
            FillSubMatrix(LongWireMatrix, ReturnSubMatrices.LongWireMatrix, IndeciesArray);
            double[] LongWireMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.LongWireMatrix,
IndeciesArray.Length, "Long Wire Runs From Sensor:");

            FillSubMatrix(LongWireMatrix, ReturnSubMatricesNotNormalized.LongWireMatrix, IndeciesArray);
            ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.LongWireMatrix, LongWireMatrixResult,
IndeciesArray.Length);

            ////////////////////////////////////////////////////////////////////////////////////////////
            double[,] MeasureParaMatrix = new double[,] {
                {1.0,4.0,3.0,5.0,5.0,6.0,1.0},
                {0.25,1.0,0.5,4.0,4.0,5.0,0.3333},
                {0.3333,2.0,1.0,3.0,3.0,4.0,0.3333},
                {0.2,0.25,0.3333,1.0,1.0,3.0,0.2},
                {0.2,0.25,0.3333,1.0,1.0,3.0,0.2},
                {0.1667,0.2,0.25,0.3333,0.3333,1.0,0.1667},
                {1.0,3.0,3.0,5.0,5.0,6.0,1.0}
            };
            FillSubMatrix(MeasureParaMatrix, ReturnSubMatrices.MeasureParaMatrix, IndeciesArray);
            double[] MeasureParaMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.MeasureParaMatrix,
IndeciesArray.Length, "Measurement Parameter:");

            FillSubMatrix(MeasureParaMatrix, ReturnSubMatricesNotNormalized.MeasureParaMatrix, IndeciesArray);
            ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.MeasureParaMatrix,
MeasureParaMatrixResult, IndeciesArray.Length);

            ////////////////////////////////////////////////////////////////////////////////////////////
            double[,] TempMeasureMatrix = new double[,] {
                {1.0,0.25,0.2,0.3333,0.3333,0.5,0.1667},
                {4.0,1.0,0.3333,1.0,3.0,3.0,0.1667},
                {5.0,3.0,1.0,4.0,5.0,5.0,0.5},
                {3.0,1.0,0.25,1.0,2.0,2.0,0.25},
                {3.0,0.3333,0.2,0.5,1.0,1.0,0.1667},
```

```
            {2.0,0.3333,0.2,0.5,1.0,1.0,0.2},
            {6.0,6.0,2.0,4.0,6.0,5.0,1.0}
        };
        FillSubMatrix(TempMeasureMatrix, ReturnSubMatrices.TempMeasureMatrix, IndeciesArray);
        double[] TempMeasureMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.TempMeasureMatrix,
IndeciesArray.Length, "Temperature Measurement:");

        FillSubMatrix(TempMeasureMatrix, ReturnSubMatricesNotNormalized.TempMeasureMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.TempMeasureMatrix,
TempMeasureMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] StimulationElecMatrix = new double[,] {
            {1.0,4.0,3.0,1.0,1.0,2.0,6.0},
            {0.25,1.0,0.5,0.2,0.2,0.25,3.0},
            {0.3333,2.0,1.0,0.25,0.25,0.5,3.0},
            {1.0,5.0,4.0,1.0,1.0,2.0,6.0},
            {1.0,5.0,4.0,1.0,1.0,3.0,6.0},
            {0.5,4.0,2.0,0.5,0.3333,1.0,4.0},
            {0.1667,0.3333,0.3333,0.1667,0.1667,0.25,1.0}
        };
        FillSubMatrix(StimulationElecMatrix, ReturnSubMatrices.StimulationElecMatrix, IndeciesArray);
        double[] StimulationElecMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.StimulationElecMatrix,
IndeciesArray.Length, "Stimulation Electronics Required:");

        FillSubMatrix(StimulationElecMatrix, ReturnSubMatricesNotNormalized.StimulationElecMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.StimulationElecMatrix,
StimulationElecMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] TypOutputLevelMatrix = new double[,] {
            {1.0,6.0,1.0,4.0,1.0,2.0,1.0},
            {0.1667,1.0,0.1667,0.25,0.1667,0.2,0.125},
            {1.0,6.0,1.0,4.0,1.0,2.0,1.0},
            {0.25,4.0,0.25,1.0,0.25,0.3333,0.1667},
            {1.0,6.0,1.0,4.0,1.0,3.0,1.0},
            {0.5,5.0,0.5,3.0,0.3333,1.0,0.3333},
            {1.0,8.0,1.0,6.0,1.0,3.0,1.0}
        };
        FillSubMatrix(TypOutputLevelMatrix, ReturnSubMatrices.TypOutputLevelMatrix, IndeciesArray);
        double[] TypOutputLevelMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.TypOutputLevelMatrix,
IndeciesArray.Length, "Typical Output Levels Per Degree Celsius:");

        FillSubMatrix(TypOutputLevelMatrix, ReturnSubMatricesNotNormalized.TypOutputLevelMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.TypOutputLevelMatrix,
TypOutputLevelMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] TypFastThertimeConsMatrix = new double[,] {
            {1.0,3.0,4.0,6.0,5.0,1.0,3.0},
            {0.3333,1.0,2.0,4.0,3.0,0.3333,1.0},
            {0.25,0.5,1.0,2.0,2.0,0.25,1.0},
            {0.1667,0.25,0.5,1.0,0.5,0.1667,0.3333},
            {0.2,0.3333,0.5,2.0,1.0,0.2,0.3333},
            {1.0,3.0,4.0,6.0,5.0,1.0,3.0},
            {0.3333,1.0,1.0,3.0,3.0,0.3333,1.0}
        };
        FillSubMatrix(TypFastThertimeConsMatrix, ReturnSubMatrices.TypFastThertimeConsMatrix, IndeciesArray);
        double[] TypFastThertimeConsMatrixResult =
CalculateWeightsForEachCriterion(ReturnSubMatrices.TypFastThertimeConsMatrix, IndeciesArray.Length, "Typical Fast Thermal
Time Constant:");

        FillSubMatrix(TypFastThertimeConsMatrix, ReturnSubMatricesNotNormalized.TypFastThertimeConsMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.TypFastThertimeConsMatrix,
TypFastThertimeConsMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] TypSmallSizMatrix = new double[,] {
            {1.0,2.0,3.0,4.0,5.0,6.0,5.0},
            {0.5,1.0,2.0,3.0,4.0,5.0,4.0},
            {0.3333,0.5,1.0,2.0,4.0,6.0,4.0},
            {0.25,0.3333,0.5,1.0,2.0,3.0,2.0},
            {0.2,0.25,0.25,0.5,1.0,3.0,1.0},
            {0.1667,0.2,0.1667,0.3333,0.3333,1.0,0.5},
            {0.2,0.25,0.25,0.5,1.0,2.0,1.0}
        };
```

```
        FillSubMatrix(TypSmallSizMatrix, ReturnSubMatrices.TypSmallSizMatrix, IndeciesArray);
        double[] TypSmallSizMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.TypSmallSizMatrix,
IndeciesArray.Length, "Typical Small Size:");

        FillSubMatrix(TypSmallSizMatrix, ReturnSubMatricesNotNormalized.TypSmallSizMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.TypSmallSizMatrix,
TypSmallSizMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] NoiseImmunityMatrix = new double[,] {
            {1.0,0.1667,0.3333,0.25,0.25,0.5,0.25},
            {6.0,1.0,4.0,3.0,3.0,5.0,3.0},
            {3.0,0.25,1.0,0.5,0.5,2.0,0.5},
            {4.0,0.3333,2.0,1.0,1.0,3.0,1.0},
            {4.0,0.3333,2.0,1.0,1.0,4.0,1.0},
            {2.0,0.2,0.5,0.3333,0.25,1.0,0.25},
            {4.0,0.3333,2.0,1.0,1.0,4.0,1.0}
        };
        FillSubMatrix(NoiseImmunityMatrix, ReturnSubMatrices.NoiseImmunityMatrix, IndeciesArray);
        double[] NoiseImmunityMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.NoiseImmunityMatrix,
IndeciesArray.Length, "Noise Immunity:");

        FillSubMatrix(NoiseImmunityMatrix, ReturnSubMatricesNotNormalized.NoiseImmunityMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.NoiseImmunityMatrix,
NoiseImmunityMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] FraDurMatrix = new double[,] {
            {1.0,6.0,3.0,6.0,8.0,3.0,4.0},
            {0.1667,1.0,0.3333,2.0,3.0,0.3333,0.5},
            {0.3333,3.0,1.0,3.0,4.0,2.0,3.0},
            {0.1667,0.5,0.3333,1.0,3.0,0.25,0.3333},
            {0.125,0.3333,0.25,0.3333,1.0,0.25,0.3333},
            {0.3333,3.0,0.5,4.0,4.0,1.0,3.0},
            {0.25,2.0,0.3333,3.0,3.0,0.3333,1.0},

        };
        FillSubMatrix(FraDurMatrix, ReturnSubMatrices.FraDurMatrix, IndeciesArray);
        double[] FraDurMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.FraDurMatrix, IndeciesArray.Length,
"Fragility-Durability:");

        FillSubMatrix(FraDurMatrix, ReturnSubMatricesNotNormalized.FraDurMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.FraDurMatrix, FraDurMatrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] HiThGrEnMatrix = new double[,] {
            {1.0,4.0,5.0,7.0,8.0,4.0,6.0},
            {0.25,1.0,2.0,5.0,6.0,2.0,4.0},
            {0.2,0.5,1.0,3.0,3.0,0.5,2.0},
            {0.1429,0.2,0.3333,1.0,1.0,0.25,0.5},
            {0.125,0.1667,0.3333,1.0,1.0,0.2,0.3333},
            {0.25,0.5,2.0,4.0,5.0,1.0,4.0},
            {0.1667,0.25,0.5,2.0,3.0,0.25,1.0}
        };
        FillSubMatrix(HiThGrEnMatrix, ReturnSubMatrices.HiThGrEnMatrix, IndeciesArray);
        double[] HiThGrEnMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.HiThGrEnMatrix,
IndeciesArray.Length, "High Thermal Gradient Environment:");

        FillSubMatrix(HiThGrEnMatrix, ReturnSubMatricesNotNormalized.HiThGrEnMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.HiThGrEnMatrix, HiThGrEnMatrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] CorrResMatrix = new double[,] {
            {1.0,0.25,0.1667,0.5,0.1667,0.1667,0.25},
            {4.0,1.0,0.3333,2.0,0.25,0.25,1.0},
            {6.0,3.0,1.0,4.0,1.0,1.0,4.0},
            {2.0,0.5,0.25,1.0,0.25,0.25,0.5},
            {6.0,4.0,1.0,4.0,1.0,1.0,3.0},
            {6.0,4.0,1.0,4.0,1.0,1.0,3.0},
            {4.0,1.0,0.25,2.0,0.3333,0.3333,1.0}
        };
        FillSubMatrix(CorrResMatrix, ReturnSubMatrices.CorrResMatrix, IndeciesArray);
        double[] CorrResMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.CorrResMatrix, IndeciesArray.Length,
"Corrosion Resistance:");
```

```
        FillSubMatrix(CorrResMatrix, ReturnSubMatricesNotNormalized.CorrResMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.CorrResMatrix, CorrResMatrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////////
        double[,] PointAreaMeasMatrix = new double[,] {
            {1.0,2.0,3.0,4.0,6.0,6.0,4.0},
             {0.5,1.0,2.0,3.0,4.0,4.0,3.0},
             {0.3333,0.5,1.0,2.0,3.0,4.0,2.0},
             {0.25,0.3333,0.5,1.0,2.0,2.0,0.5},
             {0.1667,0.25,0.3333,0.5,1.0,1.0,0.5},
             {0.1667,0.25,0.25,0.5,1.0,1.0,2.0},
             {0.25,0.5,0.3333,2.0,2.0,0.5,1.0}
        };
        FillSubMatrix(PointAreaMeasMatrix, ReturnSubMatrices.PointAreaMeasMatrix, IndeciesArray);
        double[] PointAreaMeasMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.PointAreaMeasMatrix,
IndeciesArray.Length, "Point or Area Measurement:");

        FillSubMatrix(PointAreaMeasMatrix, ReturnSubMatricesNotNormalized.PointAreaMeasMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.PointAreaMeasMatrix,
PointAreaMeasMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////////
        double[,] ManuVarMatrix = new double[,] {
             {1.0, 0.3333, 0.1667,0.25,0.5,0.2,0.25},
             {3.0, 1.0, 0.3333,0.5,2.0,0.25,0.5},
             {6.0, 3.0, 1.0,4.0,3.0,6.0,3.0},
             {4.0,2.0,0.25,1.0,4.0,0.3333,0.5},
             {2.0,0.5,0.3333,0.25,1.0,0.2,0.25},
             {5.0,4.0,0.1667,3.0,5.0,1.0,2.0},
             {4.0,2.0,0.3333,2.0,4.0,0.5,1.0}
        };
        FillSubMatrix(ManuVarMatrix, ReturnSubMatrices.ManuVarMatrix, IndeciesArray);
        double[] ManuVarMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.ManuVarMatrix,
IndeciesArray.Length, "Manufacturing Variances:");

        FillSubMatrix(ManuVarMatrix, ReturnSubMatricesNotNormalized.ManuVarMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.ManuVarMatrix, ManuVarMatrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////////
        double[,] NistStanMatrix = new double[,] {
             {1.0   , 4.0, 1.0,1.0,1.0,4.0,5.0},
             {0.25, 1.0, 0.25,0.25,0.25,1.0,2.0},
             {1.0, 4.0, 1.0,1.0,1.0,4.0,5.0},
             {1.0,4.0,1.0,1.0,1.0,4.0,5.0},
             {1.0,4.0,1.0,1.0,1.0,4.0,5.0},
             {0.25,1.0,0.25,0.25,0.25,1.0,2.0},
             {0.2,0.5,0.2,0.2,0.2,0.5,1.0}
        };
        FillSubMatrix(NistStanMatrix, ReturnSubMatrices.NistStanMatrix, IndeciesArray);
        double[] NistStanMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.NistStanMatrix, IndeciesArray.Length,
"NIST Standards:");

        FillSubMatrix(NistStanMatrix, ReturnSubMatricesNotNormalized.NistStanMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.NistStanMatrix, NistStanMatrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////////
        double[,] CostMatrix = new double[,] {
             {1.0,1.0,6.0,3.0,3.0,6.0,3.0},
             {1.0,1.0,6.0,3.0,3.0,6.0,3.0},
             {0.1667,0.1667,1.0,0.25,0.25,1.0,0.25},
             {0.3333,0.3333,4.0,1.0,1.0,4.0,1.0},
             {0.3333,0.3333,4.0,1.0,1.0,4.0,1.0},
             {0.1667,0.1667,1.0,0.25,0.25,1.0,0.25},
             {0.3333,0.3333,4.0,1.0,1.0,4.0,1.0}
        };
        FillSubMatrix(CostMatrix, ReturnSubMatrices.CostMatrix, IndeciesArray);
        double[] CostMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.CostMatrix, IndeciesArray.Length,
"Cost:");

        FillSubMatrix(CostMatrix, ReturnSubMatricesNotNormalized.CostMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.CostMatrix, CostMatrixResult,
IndeciesArray.Length);
```

```
        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] weightsOfCriteria = new double[,] {
            {1.0,2.0,1.0,4.0},
            {0.5,1.0,0.5,2.0},
            {1.0,2.0,1.0,4.0},
            {0.25,0.5,0.25,1.0}
        };

        double[] weightsOfCriteriaResults = CalculateWeightsForEachCriterion(weightsOfCriteria, 4, "Weights of Criteria:");

        double[,] weightsOfCriteria2 = new double[,] {
            {1.0,2.0,1.0,4.0},
            {0.5,1.0,0.5,2.0},
            {1.0,2.0,1.0,4.0},
            {0.25,0.5,0.25,1.0}
        };
        ConsistencyIndex = ComputeConsistencyIndex(weightsOfCriteria2, weightsOfCriteriaResults, 4);
        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] Weightssubcriteriastatic = new double[,] {
            {1.0,1.0,5.0,4.0,4.0,1.0,5.0,6.0,7.0,8.0,6.0},
            {1.0,1.0,5.0,4.0,4.0,1.0,5.0,6.0,7.0,8.0,6.0},
            {0.2,0.2,1.0,0.3333,0.3333,0.2,1.0,2.0,4.0,5.0,3.0},
            {0.25,0.25,3.0,1.0,2.0,0.3333,3.0,3.0,5.0,6.0,4.0},
            {0.25,0.25,3.0,0.5,1.0,0.25,3.0,5.0,6.0,8.0,4.0},
            {1.0,1.0,5.0,3.0,4.0,1.0,5.0,6.0,7.0,9.0,6.0},
            {0.2,0.2,1.0,0.3333,0.3333,0.2,1.0,1.0,4.0,6.0,3.0},
            {0.1667,0.1667,0.5,0.3333,0.2,0.1667,1.0,1.0,3.0,4.0,1.0},
            {0.1429,0.1429,0.25,0.2,0.1667,0.1429,0.25,0.3333,1.0,2.0,0.3333},
            {0.125,0.125,0.2,0.1667,0.125,0.1111,0.1667,0.25,0.5,1.0,0.25},
            {0.1667,0.1667,0.3333,0.25,0.25,0.1667,0.3333,1.0,3.0,4.0,1.0}
        };
        double[] weightsSubCriteriaStaticResult = CalculateWeightsForEachCriterion(Weightssubcriteriastatic, 11, "Weights of Sub-
Criteria Static:");

        double[,] Weightssubcriteriastatic2 = new double[,] {
            {1.0,1.0,5.0,4.0,4.0,1.0,5.0,6.0,7.0,8.0,6.0},
            {1.0,1.0,5.0,4.0,4.0,1.0,5.0,6.0,7.0,8.0,6.0},
            {0.2,0.2,1.0,0.3333,0.3333,0.2,1.0,2.0,4.0,5.0,3.0},
            {0.25,0.25,3.0,1.0,2.0,0.3333,3.0,3.0,5.0,6.0,4.0},
            {0.25,0.25,3.0,0.5,1.0,0.25,3.0,5.0,6.0,8.0,4.0},
            {1.0,1.0,5.0,3.0,4.0,1.0,5.0,6.0,7.0,9.0,6.0},
            {0.2,0.2,1.0,0.3333,0.3333,0.2,1.0,1.0,4.0,6.0,3.0},
            {0.1667,0.1667,0.5,0.3333,0.2,0.1667,1.0,1.0,3.0,4.0,1.0},
            {0.1429,0.1429,0.25,0.2,0.1667,0.1429,0.25,0.3333,1.0,2.0,0.3333},
            {0.125,0.125,0.2,0.1667,0.125,0.1111,0.1667,0.25,0.5,1.0,0.25},
            {0.1667,0.1667,0.3333,0.25,0.25,0.1667,0.3333,1.0,3.0,4.0,1.0}


        };
        ConsistencyIndex = ComputeConsistencyIndex(Weightssubcriteriastatic2, weightsSubCriteriaStaticResult, 11);
        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] weightssubCriteriaDynamic = new double[,] {
            {1.0,2.0,0.1429},
            {0.5,1.0,0.1429},
            {7.0,7.0,1.0}
        };
        double[] weightsSubCriteriaDynamicResult = CalculateWeightsForEachCriterion(weightssubCriteriaDynamic, 3, "Weights of
Sub-Criteria Dynamic:");

        double[,] weightssubCriteriaDynamic2 = new double[,] {
            {1.0,2.0,0.1429},
            {0.5,1.0,0.1429},
            {7.0,7.0,1.0}
        };
        ConsistencyIndex = ComputeConsistencyIndex(weightssubCriteriaDynamic2, weightsSubCriteriaDynamicResult, 3);
        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] weightssubCriteriaEnv = new double[,] {
            {1.0,4.0,0.5,4.0,0.25},
            {0.25,1.0,0.25,2.0,0.1667},
            {2.0,4.0,1.0,4.0,0.3333},
            {0.25,0.5,0.25,1.0,0.1667},
            {4.0,6.0,3.0,6.0,1.0}
        };
        double[] weightsSubCriteriaEnvironmentalResult = CalculateWeightsForEachCriterion(weightssubCriteriaEnv, 5, "Weights of
Sub-Criteria Environmental:");
```

```
            double[,] weightssubCriteriaEnv2 = new double[,] {
                  {1.0,4.0,0.5,4.0,0.25},
                  {0.25,1.0,0.25,2.0,0.1667},
                  {2.0,4.0,1.0,4.0,0.3333},
                  {0.25,0.5,0.25,1.0,0.1667},
                  {4.0,6.0,3.0,6.0,1.0}
            };
            ConsistencyIndex = ComputeConsistencyIndex(weightssubCriteriaEnv2, weightsSubCriteriaEnvironmentalResult, 5);
            ////////////////////////////////////////////////////////////////////////////////////////
            double[,] Others = new double[,] {
                  {1.0,3.0,0.5,0.25},
                  {0.3333,1.0,0.3333,0.2},
                  {2.0,3.0,1.0,0.3333},
                  {4.0,5.0,3.0,1.0}
            };
            double[] weightsSubCriteriaOthersResult = CalculateWeightsForEachCriterion(Others, 4, "Weights of Sub-Criteria Others:");

            double[,] Others2 = new double[,] {
                 {1.0,3.0,0.5,0.25},
                  {0.3333,1.0,0.3333,0.2},
                  {2.0,3.0,1.0,0.3333},
                  {4.0,5.0,3.0,1.0}
            };
            ConsistencyIndex = ComputeConsistencyIndex(Others2, weightsSubCriteriaOthersResult, 4);
            ////////////////////////////////////////////////////////////////////////////////////////////////
            //Calculate sub criteria aggregate weights with respect to final goal
            Scale(weightsSubCriteriaStaticResult, weightsOfCriteriaResults[0]);
            Scale(weightsSubCriteriaDynamicResult, weightsOfCriteriaResults[1]);
            Scale(weightsSubCriteriaEnvironmentalResult, weightsOfCriteriaResults[2]);
            Scale(weightsSubCriteriaOthersResult, weightsOfCriteriaResults[3]);

            ////////////////////////////////////////////////////////////////////////////////////////////////
            //Calculate sub criteria aggregate weights for ThermoCouple (alternative 1)
            double ThermoCoupleFinalScore = 0.0;
            double ThermisterFinalScore = 0.0;
            double RTDFinalScore = 0.0;
            double BimetallicFinalScore = 0.0;
            double ThermometerFinalScore = 0.0;
            double PyrometerFinalScore = 0.0;
            double LCDDidplayFinalScore = 0.0;

            for (int k = 0; k < IndeciesArray.Length; k++)
            {
               if (IndeciesArray[k] == AHP.SENSOR_THERMOCOUPLE)
               {
                  double[] ThermoCoupleAggregateSubCriteriaStatic = new double[11];
                  double[] ThermoCoupleAggregateSubCriteriaDynamic = new double[3];
                  double[] ThermoCoupleAggregateSubCriteriaEnvironmental = new double[5];
                  double[] ThermoCoupleAggregateSubCriteriaOthers = new double[4];

                  ThermoCoupleAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
                  ThermoCoupleAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
                  ThermoCoupleAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
                  ThermoCoupleAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
                  ThermoCoupleAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
                  ThermoCoupleAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
                  ThermoCoupleAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
                  ThermoCoupleAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
                  ThermoCoupleAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
                  ThermoCoupleAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
                  ThermoCoupleAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

                  ThermoCoupleFinalScore = ThermoCoupleFinalScore +
GetSummationOfVectorElements(ThermoCoupleAggregateSubCriteriaStatic);

                  ThermoCoupleAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] *
weightsSubCriteriaDynamicResult[0];
                  ThermoCoupleAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] *
weightsSubCriteriaDynamicResult[1];
                  ThermoCoupleAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] *
weightsSubCriteriaDynamicResult[2];

                  ThermoCoupleFinalScore = ThermoCoupleFinalScore +
GetSummationOfVectorElements(ThermoCoupleAggregateSubCriteriaDynamic);
```

```
                ThermoCoupleAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[0];
                ThermoCoupleAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
                ThermoCoupleAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[2];
                ThermoCoupleAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[3];
                ThermoCoupleAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[4];

                ThermoCoupleFinalScore = ThermoCoupleFinalScore +
GetSummationOfVectorElements(ThermoCoupleAggregateSubCriteriaEnvironmental);

                ThermoCoupleAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
                ThermoCoupleAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
                ThermoCoupleAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
                ThermoCoupleAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

                ThermoCoupleFinalScore = ThermoCoupleFinalScore +
GetSummationOfVectorElements(ThermoCoupleAggregateSubCriteriaOthers);
                SensorRanks[k] = ThermoCoupleFinalScore;

                System.Console.Write("\n\n");
                System.Console.Write("Thermo Couple Final Score = {0}", ThermoCoupleFinalScore);
            }
            ////////////////////////////////////////////////////////////////////////////////////////////////////////////
            //Calculate sub criteria aggregate weights for Thermister (alternative 2)
            else if (IndeciesArray[k] == AHP.SENSOR_THERMISTER)
            {
                double[] ThermisterAggregateSubCriteriaStatic = new double[11];
                double[] ThermisterAggregateSubCriteriaDynamic = new double[3];
                double[] ThermisterAggregateSubCriteriaEnvironmental = new double[5];
                double[] ThermisterAggregateSubCriteriaOthers = new double[4];

                ThermisterAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
                ThermisterAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
                ThermisterAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
                ThermisterAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
                ThermisterAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
                ThermisterAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
                ThermisterAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
                ThermisterAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
                ThermisterAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
                ThermisterAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
                ThermisterAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

                ThermisterFinalScore = ThermisterFinalScore + GetSummationOfVectorElements(ThermisterAggregateSubCriteriaStatic);

                ThermisterAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] * weightsSubCriteriaDynamicResult[0];
                ThermisterAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] * weightsSubCriteriaDynamicResult[1];
                ThermisterAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] *
weightsSubCriteriaDynamicResult[2];

                ThermisterFinalScore = ThermisterFinalScore +
GetSummationOfVectorElements(ThermisterAggregateSubCriteriaDynamic);

                ThermisterAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[0];
                ThermisterAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
                ThermisterAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[2];
                ThermisterAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[3];
                ThermisterAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[4];

                ThermisterFinalScore = ThermisterFinalScore +
GetSummationOfVectorElements(ThermisterAggregateSubCriteriaEnvironmental);

                ThermisterAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
                ThermisterAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
                ThermisterAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
                ThermisterAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

                ThermisterFinalScore = ThermisterFinalScore + GetSummationOfVectorElements(ThermisterAggregateSubCriteriaOthers);
```

```
                System.Console.Write("\n\n");
                System.Console.Write("Thermister Final Score = {0}", ThermisterFinalScore);

                SensorRanks[k] = ThermisterFinalScore;
            }
            ////////////////////////////////////////////////////////////////////////////////////////////////////////
            //Calculate sub criteria aggregate weights for RTD (alternative 3)
            else if (IndeciesArray[k] == AHP.SENSOR_RTD)
            {
                double[] RTDAggregateSubCriteriaStatic = new double[11];
                double[] RTDAggregateSubCriteriaDynamic = new double[3];
                double[] RTDAggregateSubCriteriaEnvironmental = new double[5];
                double[] RTDAggregateSubCriteriaOthers = new double[4];

                RTDAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
                RTDAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
                RTDAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
                RTDAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
                RTDAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
                RTDAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
                RTDAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
                RTDAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
                RTDAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
                RTDAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
                RTDAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

                RTDFinalScore = RTDFinalScore + GetSummationOfVectorElements(RTDAggregateSubCriteriaStatic);

                RTDAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] * weightsSubCriteriaDynamicResult[0];
                RTDAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] * weightsSubCriteriaDynamicResult[1];
                RTDAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] * weightsSubCriteriaDynamicResult[2];

                RTDFinalScore = RTDFinalScore + GetSummationOfVectorElements(RTDAggregateSubCriteriaDynamic);

                RTDAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[0];
                RTDAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
                RTDAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[2];
                RTDAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[3];
                RTDAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[4];

                RTDFinalScore = RTDFinalScore + GetSummationOfVectorElements(RTDAggregateSubCriteriaEnvironmental);

                RTDAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
                RTDAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
                RTDAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
                RTDAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

                RTDFinalScore = RTDFinalScore + GetSummationOfVectorElements(RTDAggregateSubCriteriaOthers);
                SensorRanks[k] = RTDFinalScore;
            }
            else if (IndeciesArray[k] == AHP.SENSOR_BIMETALLIC)
            {
                double[] BimetallicAggregateSubCriteriaStatic = new double[11];
                double[] BimetallicAggregateSubCriteriaDynamic = new double[3];
                double[] BimetallicAggregateSubCriteriaEnvironmental = new double[5];
                double[] BimetallicAggregateSubCriteriaOthers = new double[4];

                BimetallicAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
                BimetallicAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
                BimetallicAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
                BimetallicAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
                BimetallicAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
                BimetallicAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
                BimetallicAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
                BimetallicAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
                BimetallicAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
                BimetallicAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
                BimetallicAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

                BimetallicFinalScore = BimetallicFinalScore + GetSummationOfVectorElements(BimetallicAggregateSubCriteriaStatic);

                BimetallicAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] * weightsSubCriteriaDynamicResult[0];
                BimetallicAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] * weightsSubCriteriaDynamicResult[1];
                BimetallicAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] *
weightsSubCriteriaDynamicResult[2];
```

```
            BimetallicFinalScore = BimetallicFinalScore +
GetSummationOfVectorElements(BimetallicAggregateSubCriteriaDynamic);

            BimetallicAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[0];
            BimetallicAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
            BimetallicAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[2];
            BimetallicAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[3];
            BimetallicAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[4];

            BimetallicFinalScore = BimetallicFinalScore +
GetSummationOfVectorElements(BimetallicAggregateSubCriteriaEnvironmental);

            BimetallicAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
            BimetallicAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
            BimetallicAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
            BimetallicAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

            BimetallicFinalScore = BimetallicFinalScore + GetSummationOfVectorElements(BimetallicAggregateSubCriteriaOthers);
            SensorRanks[k] = BimetallicFinalScore;
        }
        else if (IndeciesArray[k] == AHP.SENSOR_THERMOMETER)
        {
            double[] ThermometerAggregateSubCriteriaStatic = new double[11];
            double[] ThermometerAggregateSubCriteriaDynamic = new double[3];
            double[] ThermometerAggregateSubCriteriaEnvironmental = new double[5];
            double[] ThermometerAggregateSubCriteriaOthers = new double[4];

            ThermometerAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
            ThermometerAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
            ThermometerAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
            ThermometerAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
            ThermometerAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
            ThermometerAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
            ThermometerAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
            ThermometerAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
            ThermometerAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
            ThermometerAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
            ThermometerAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

            ThermometerFinalScore = ThermometerFinalScore +
GetSummationOfVectorElements(ThermometerAggregateSubCriteriaStatic);

            ThermometerAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] * weightsSubCriteriaDynamicResult[0];
            ThermometerAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] * weightsSubCriteriaDynamicResult[1];
            ThermometerAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] *
weightsSubCriteriaDynamicResult[2];

            ThermometerFinalScore = ThermometerFinalScore +
GetSummationOfVectorElements(ThermometerAggregateSubCriteriaDynamic);

            ThermometerAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[0];
            ThermometerAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
            ThermometerAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[2];
            ThermometerAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[3];
            ThermometerAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[4];

            ThermometerFinalScore = ThermometerFinalScore +
GetSummationOfVectorElements(ThermometerAggregateSubCriteriaEnvironmental);

            ThermometerAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
            ThermometerAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
            ThermometerAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
            ThermometerAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

            ThermometerFinalScore = ThermometerFinalScore +
GetSummationOfVectorElements(ThermometerAggregateSubCriteriaOthers);
            SensorRanks[k] = ThermometerFinalScore;
```

```
            }
            else if (IndeciesArray[k] == AHP.SENSOR_PYROMETER)
            {
                double[] PyrometerAggregateSubCriteriaStatic = new double[11];
                double[] PyrometerAggregateSubCriteriaDynamic = new double[3];
                double[] PyrometerAggregateSubCriteriaEnvironmental = new double[5];
                double[] PyrometerAggregateSubCriteriaOthers = new double[4];

                PyrometerAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
                PyrometerAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
                PyrometerAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
                PyrometerAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
                PyrometerAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
                PyrometerAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
                PyrometerAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
                PyrometerAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
                PyrometerAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
                PyrometerAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
                PyrometerAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

                PyrometerFinalScore = PyrometerFinalScore + GetSummationOfVectorElements(PyrometerAggregateSubCriteriaStatic);

                PyrometerAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] * weightsSubCriteriaDynamicResult[0];
                PyrometerAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] * weightsSubCriteriaDynamicResult[1];
                PyrometerAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] *
weightsSubCriteriaDynamicResult[2];

                PyrometerFinalScore = PyrometerFinalScore +
GetSummationOfVectorElements(PyrometerAggregateSubCriteriaDynamic);

                PyrometerAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[0];
                PyrometerAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
                PyrometerAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[2];
                PyrometerAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[3];
                PyrometerAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[4];

                PyrometerFinalScore = PyrometerFinalScore +
GetSummationOfVectorElements(PyrometerAggregateSubCriteriaEnvironmental);

                PyrometerAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
                PyrometerAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
                PyrometerAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
                PyrometerAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

                PyrometerFinalScore = PyrometerFinalScore + GetSummationOfVectorElements(PyrometerAggregateSubCriteriaOthers);
                SensorRanks[k] = PyrometerFinalScore;
            }
            else if (IndeciesArray[k] == AHP.SENSOR_LCD_DISPLAY)
            {
                double[] LCDDidplayAggregateSubCriteriaStatic = new double[11];
                double[] LCDDidplayAggregateSubCriteriaDynamic = new double[3];
                double[] LCDDidplayAggregateSubCriteriaEnvironmental = new double[5];
                double[] LCDDidplayAggregateSubCriteriaOthers = new double[4];

                LCDDidplayAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
                LCDDidplayAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
                LCDDidplayAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
                LCDDidplayAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
                LCDDidplayAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
                LCDDidplayAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
                LCDDidplayAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
                LCDDidplayAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
                LCDDidplayAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
                LCDDidplayAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
                LCDDidplayAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

                LCDDidplayFinalScore = LCDDidplayFinalScore +
GetSummationOfVectorElements(LCDDidplayAggregateSubCriteriaStatic);

                LCDDidplayAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] * weightsSubCriteriaDynamicResult[0];
                LCDDidplayAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] * weightsSubCriteriaDynamicResult[1];
                LCDDidplayAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] *
weightsSubCriteriaDynamicResult[2];
```

```
                LCDDidplayFinalScore = LCDDidplayFinalScore +
GetSummationOfVectorElements(LCDDidplayAggregateSubCriteriaDynamic);

                LCDDidplayAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[0];
                LCDDidplayAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
                LCDDidplayAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[2];
                LCDDidplayAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[3];
                LCDDidplayAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[4];

                LCDDidplayFinalScore = LCDDidplayFinalScore +
GetSummationOfVectorElements(LCDDidplayAggregateSubCriteriaEnvironmental);

                LCDDidplayAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
                LCDDidplayAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
                LCDDidplayAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
                LCDDidplayAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

                LCDDidplayFinalScore = LCDDidplayFinalScore +
GetSummationOfVectorElements(LCDDidplayAggregateSubCriteriaOthers);
                SensorRanks[k] = LCDDidplayFinalScore;
            }
        }

        textBoxResults.AppendText("\n\n");
        textBoxResults.AppendText("\nSensor Ranks: \n");
        for (int j = 0; j < SensorRanks.Length; j++)
        {
            textBoxResults.AppendText("" + SensorRanks[j] + "\n");
        }

        return SensorRanks;
    }
  }
}
```

5. HVAC.cs File

```
using System;
using System.Collections.Generic;
using System.Text;

namespace AhpCaseStudy1GUI
{
   class HVAC
   {
      public System.Windows.Forms.TextBox textBoxResults;

      void Print(double[,] mat, int nDimensionSize, string strTitle)
      {
         if (strTitle == "")
            strTitle = "Matrix";

         //System.Console.WriteLine(strTitle);
         textBoxResults.AppendText("\n");
         textBoxResults.AppendText(strTitle);
         textBoxResults.AppendText("\n");
         textBoxResults.AppendText("\n");
         for (int i = 0; i < nDimensionSize; i++)
         {
            for (int j = 0; j < nDimensionSize; j++)
            {
               //System.Console.Write("{0}\t\t", mat[i, j]);
               textBoxResults.AppendText("     " + mat[i, j]);
               //textBoxResults.Update();
            }
            //System.Console.Write("\n");
            textBoxResults.AppendText("\n");
         }

         //System.Console.Write("\n\n");
         textBoxResults.AppendText("\n\n");
      }

      void Print(double[] vector, int nDimensionSize)
      {
         //System.Console.WriteLine("Relative Weight Vector = ");
         textBoxResults.AppendText("Relative Weight Vector = ");
         for (int i = 0; i < nDimensionSize; i++)
         {
            //System.Console.Write("{0}\t\t", vector[i]);
            textBoxResults.AppendText("     " + vector[i]);
         }
         //System.Console.Write("\n");
         textBoxResults.AppendText("\n");
      }

      void Scale(double[] vector, double scaleFactor)
      {
         for (int i = 0; i < vector.Length; i++)
         {
            vector[i] = vector[i] * scaleFactor;//vector[i] *= scaleFactor;
         }
         Print(vector, vector.Length);
      }

      double GetSummationOfVectorElements(double[] vector)
      {
         double Summation = 0.0;
         for (int i = 0; i < vector.Length; i++)
         {
            Summation = Summation + vector[i];
         }
         return Summation;
      }

      double[] CalculateWeightsForEachCriterion(double[,] expertAssesmentMatrix, int nDimensionSize, string strTitle)
      {
         //check for vald input values
         double[] weightVector = new double[nDimensionSize];
```

```
        if (expertAssesmentMatrix == null)
            return weightVector;

        Print(expertAssesmentMatrix, nDimensionSize, strTitle);

        //Calculate the weight factor for each colmun
        //The result is a vetor
        double[] wieghtFactor = new double[nDimensionSize];
        for (int j = 0; j < nDimensionSize; j++)
        {
            double result = 0.0;
            for (int k = 0; k < nDimensionSize; k++)
            {
                result += expertAssesmentMatrix[k, j]; //result = result + mat1[k, j];
            }
            wieghtFactor[j] = result;
        }

        double[,] mat1ImmediatResult = new double[nDimensionSize, nDimensionSize];
        for (int j = 0; j < nDimensionSize; j++)
        {
            for (int k = 0; k < nDimensionSize; k++)
            {
                expertAssesmentMatrix[k, j] = expertAssesmentMatrix[k, j] / wieghtFactor[j];
            }
        }

        //Calculate the weight factor for each colmun
        //The result is a vetor
        for (int j = 0; j < nDimensionSize; j++)
        {
            double result = 0.0;
            for (int k = 0; k < nDimensionSize; k++)
            {
                result += expertAssesmentMatrix[j, k]; //result = result + mat1[k, j];
            }
            weightVector[j] = result / nDimensionSize;
        }

        Print(weightVector, nDimensionSize);

//System.Console.WriteLine("_____
_____\n");


        return weightVector;
    }

    void FillSubMatrix(double[,] SourceMatrix, double[,] destinationSubMatrix, int[] IndeciesArray)
    {
        if (SourceMatrix == null || destinationSubMatrix == null || IndeciesArray == null)
            return;

        for (int i = 0; i < IndeciesArray.Length; i++)
        {
            for (int j = 0; j < IndeciesArray.Length; j++)
            {
                //destinationSubMatrix[IndeciesArray[i], IndeciesArray[j]] = SourceMatrix[IndeciesArray[i], IndeciesArray[j]];
                destinationSubMatrix[i, j] = SourceMatrix[IndeciesArray[i], IndeciesArray[j]];
            }
        }
    }

    double ComputeConsistencyIndex(double[,] Matrix, double[] vector, int nDimensionSize)
    {
        //check for vald input values
        double[] transposeVector = new double[nDimensionSize];
        double[] randomIndex = { 1.0, 0.5, 0.58, 0.9, 1.12, 1.24, 1.32 };

        if (Matrix == null)
            return 0.0;

        for (int j = 0; j < nDimensionSize; j++)
        {
            double result = 0.0;
```

```
            for (int k = 0; k < nDimensionSize; k++)
            {
               result += (Matrix[j, k] * vector[k]);
            }
            transposeVector[j] = result;
         }
         double division = 0.0;
         for (int k = 0; k < nDimensionSize; k++)
         {
            division += (transposeVector[k] / vector[k]);
            //transposeVector[k] = transposeVector[k]/vector[k];

         }
         division = division / nDimensionSize;

         double consistencyIndex = (division - nDimensionSize) / (nDimensionSize - 1);

         if (consistencyIndex < 0.0 && consistencyIndex > -0.0005)
            consistencyIndex = 0.0;

         if (consistencyIndex < 0.0005)
            consistencyIndex = 0.0;

         textBoxResults.AppendText("Consistency Index = ");
         textBoxResults.AppendText("" + consistencyIndex + "\n");

         textBoxResults.AppendText("\nConsistency Ratio = ");
         if (nDimensionSize > randomIndex.Length)
            textBoxResults.AppendText("" + (consistencyIndex / 1.59));
         else
            textBoxResults.AppendText("" + (consistencyIndex / randomIndex[nDimensionSize - 1]));
         textBoxResults.AppendText("\n");
         textBoxResults.AppendText("_____\n");
         return consistencyIndex;
      }

      public double[] SelectBestSensor(int[] IndeciesArray)
      {
         double[] SensorRanks = new double[IndeciesArray.Length];
         AHPSubCriteria ReturnSubMatrices = new AHPSubCriteria();
         ReturnSubMatrices.CreateSubMatrices(IndeciesArray.Length);
         AHPSubCriteria ReturnSubMatricesNotNormalized = new AHPSubCriteria();
         ReturnSubMatricesNotNormalized.CreateSubMatrices(IndeciesArray.Length);

         double[,] MaximumTempMatrix = new double[,] {
               {1.0,3.0,1.0,9.0,4.0,6.0,4.0},
               {0.3333,1.0,0.3333,6.0,2.0,5.0,2.0},
               {1.0,3.0,1.0,9.0,4.0,6.0,4.0},
               {0.1111,0.1667,0.1111,1.0,0.1667,0.3333,0.1667},
               {0.25,0.5,0.25,6.0,1.0,4.0,1.0},
               {0.1667,0.2,0.1667,3.0,0.25,1.0,0.25},
               {0.25,0.5,0.25,6.0,1.0,4.0,1.0}
         };
         FillSubMatrix(MaximumTempMatrix, ReturnSubMatrices.MaximumTempMatrix, IndeciesArray);
         double[] MaximumTempMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.MaximumTempMatrix,
IndeciesArray.Length, "Maximum Operating Temprature: ");

         FillSubMatrix(MaximumTempMatrix, ReturnSubMatricesNotNormalized.MaximumTempMatrix, IndeciesArray);
         double ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.MaximumTempMatrix,
MaximumTempMatrixResult, IndeciesArray.Length);
         ////////////////////////////////////////////////////////////////////////////////////////////////
         double[,] MinimumTempMatrix = new double[,] {
               {1.0,2.0,2.0,0.5,0.5,0.125,0.3333},
               {0.5,1.0,1.0,0.3333,0.3333,0.125,0.25},
               {0.5,1.0,1.0,0.3333,0.3333,0.125,0.25},
               {2.0,3.0,3.0,1.0,1.0,0.25,0.5},
               {2.0,3.0,3.0,1.0,1.0,0.25,0.5},
               {8.0,8.0,8.0,4.0,4.0,1.0,4.0},
               {3.0,4.0,4.0,2.0,2.0,0.25,1.0}
         };
         FillSubMatrix(MinimumTempMatrix, ReturnSubMatrices.MinimumTempMatrix, IndeciesArray);
         double[] MinimumTempMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.MinimumTempMatrix,
IndeciesArray.Length, "Minimum Operating Temprature: ");

         FillSubMatrix(MinimumTempMatrix, ReturnSubMatricesNotNormalized.MinimumTempMatrix, IndeciesArray);
```

```
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.MinimumTempMatrix,
MinimumTempMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] TempCurveMtrix = new double[,] {
            {1.0,3.0,0.3333,5.0,5.0,4.0,2.0},
            {0.3333,1.0,0.1667,2.0,2.0,1.0,0.3333},
            {3.0,6.0,1.0,6.0,6.0,6.0,4.0},
            {0.2,0.5,0.1667,1.0,1.0,0.3333,0.25},
            {0.2,0.5,0.1667,1.0,1.0,0.3333,0.25},
            {0.25,1.0,0.1667,3.0,3.0,1.0,0.5},
            {0.5,3.0,0.25,4.0,4.0,2.0,1.0}
        };
        FillSubMatrix(TempCurveMtrix, ReturnSubMatrices.TempCurveMtrix, IndeciesArray);
        double[] TempCurveMtrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.TempCurveMtrix,
IndeciesArray.Length, "Temperature Curve:");

        FillSubMatrix(TempCurveMtrix, ReturnSubMatricesNotNormalized.TempCurveMtrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.TempCurveMtrix, TempCurveMtrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] MaxSensitivityMatrix = new double[,] {
            {1.0,0.1111,0.2,2.0,2.0,2.0,0.3333},
            {9.0,1.0,4.0,9.0,9.0,6.0,4.0},
            {5.0,0.25,1.0,4.0,5.0,4.0,2.0},
            {0.5,0.1111,0.25,1.0,2.0,2.0,0.25},
            {0.5,.1111,0.2,0.5,1.0,1.0,0.25},
            {0.5,0.1667,0.25,0.5,1.0,1.0,0.25},
            {3.0,0.25,0.5,4.0,4.0,4.0,1.0}
        };
        FillSubMatrix(MaxSensitivityMatrix, ReturnSubMatrices.MaxSensitivityMatrix, IndeciesArray);
        double[] MaxSensitivityMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.MaxSensitivityMatrix,
IndeciesArray.Length, "Maximum Sensitivity Region:");

        FillSubMatrix(MaxSensitivityMatrix, ReturnSubMatricesNotNormalized.MaxSensitivityMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.MaxSensitivityMatrix,
MaxSensitivityMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] SelfHeatingMatrix = new double[,] {
            {1.0,8.0,3.0,3.0,1.0,1.0,2.0},
            {0.125,1.0,0.2,0.25,0.2,0.1667,0.25},
            {0.3333,5.0,1.0,0.5,0.5,0.3333,1.0},
            {0.3333,4.0,2.0,1.0,1.0,0.5,1.0},
            {1.0,5.0,2.0,1.0,1.0,1.0,2.0},
            {1.0,6.0,3.0,2.0,1.0,1.0,1.0},
            {0.5,4.0,1.0,1.0,0.5,1.0,1.0}
        };
        FillSubMatrix(SelfHeatingMatrix, ReturnSubMatrices.SelfHeatingMatrix, IndeciesArray);
        double[] SelfHeatingMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.SelfHeatingMatrix,
IndeciesArray.Length, "Self-Heating Issues:");

        FillSubMatrix(SelfHeatingMatrix, ReturnSubMatricesNotNormalized.SelfHeatingMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.SelfHeatingMatrix,
SelfHeatingMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] LongTermStabilityMatrix = new double[,] {
            {1.0,0.25,0.1667,2.0,0.3333,0.5,0.25},
            {4.0,1.0,0.3333,4.0,3.0,3.0,2.0},
            {6.0,3.0,1.0,8.0,4.0,5.0,3.0},
            {0.5,0.25,0.125,1.0,0.3333,0.3333,0.25},
            {3.0,0.3333,0.25,3.0,1.0,2.0,0.5},
            {2.0,0.3333,0.2,3.0,0.5,1.0,0.3333},
            {4.0,0.5,0.3333,4.0,2.0,3.0,1.0}
        };
        FillSubMatrix(LongTermStabilityMatrix, ReturnSubMatrices.LongTermStabilityMatrix, IndeciesArray);
        double[] LongTermStabilityMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.LongTermStabilityMatrix,
IndeciesArray.Length, "Long Term Stability and Accuracy:");

        FillSubMatrix(LongTermStabilityMatrix, ReturnSubMatricesNotNormalized.LongTermStabilityMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.LongTermStabilityMatrix,
LongTermStabilityMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
```

```
        double[,] TypTempCoeffMatrix = new double[,] {
            {1.0,0.1667,0.3333,4.0,4.0,4.0,0.5},
            {6.0,1.0,3.0,6.0,6.0,6.0,6.0},
            {3.0,0.3333,1.0,5.0,6.0,5.0,2.0},
            {0.25,0.1667,0.2,1.0,1.0,1.0,0.2},
            {0.25,0.1667,0.1667,1.0,1.0,1.0,0.2},
            {0.25,0.1667,0.2,1.0,1.0,1.0,0.3333},
            {2.0,0.1667,0.5,5.0,5.0,3.0,1.0}
        };
        FillSubMatrix(TypTempCoeffMatrix, ReturnSubMatrices.TypTempCoeffMatrix, IndeciesArray);
        double[] TypTempCoeffMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.TypTempCoeffMatrix,
IndeciesArray.Length, "Typical Temperature Coefficient:");

        FillSubMatrix(TypTempCoeffMatrix, ReturnSubMatricesNotNormalized.TypTempCoeffMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.TypTempCoeffMatrix,
TypTempCoeffMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] ExtWiresMatrix = new double[,] {
            {1.0,0.1667,0.16667,0.125,0.125,0.125,0.125},
            {6.0,1.0,1.0,0.25,0.25,0.25,0.25},
            {6.0,1.0,1.0,0.25,0.25,0.25,0.25},
            {8.0,4.0,4.0,1.0,1.0,1.0,1.0},
            {8.0,4.0,4.0,1.0,1.0,1.0,1.0},
            {8.0,4.0,4.0,1.0,1.0,1.0,1.0},
            {8.0,4.0,4.0,1.0,1.0,1.0,1.0}
        };
        FillSubMatrix(ExtWiresMatrix, ReturnSubMatrices.ExtWiresMatrix, IndeciesArray);
        double[] ExtWiresMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.ExtWiresMatrix,
IndeciesArray.Length, "Extension Wires:");

        FillSubMatrix(ExtWiresMatrix, ReturnSubMatricesNotNormalized.ExtWiresMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.ExtWiresMatrix, ExtWiresMatrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] LongWireMatrix = new double[,] {
            {1.0,0.3333,1.0,0.1667,0.1667,0.1667,0.1667},
            {3.0,1.0,3.0,0.5,0.5,0.5,0.5},
            {1.0,0.3333,1.0,0.1667,0.1667,0.1667,0.1667},
            {6.0,2.0,6.0,1.0,1.0,1.0,1.0},
            {6.0,2.0,6.0,1.0,1.0,1.0,1.0},
            {6.0,2.0,6.0,1.0,1.0,1.0,1.0},
            {6.0,2.0,6.0,1.0,1.0,1.0,1.0}
        };
        FillSubMatrix(LongWireMatrix, ReturnSubMatrices.LongWireMatrix, IndeciesArray);
        double[] LongWireMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.LongWireMatrix,
IndeciesArray.Length, "Long Wire Runs From Sensor:");

        FillSubMatrix(LongWireMatrix, ReturnSubMatricesNotNormalized.LongWireMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.LongWireMatrix, LongWireMatrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] MeasureParaMatrix = new double[,] {
            {1.0,4.0,3.0,5.0,5.0,6.0,1.0},
            {0.25,1.0,0.5,4.0,4.0,5.0,0.3333},
            {0.3333,2.0,1.0,3.0,3.0,4.0,0.3333},
            {0.2,0.25,0.3333,1.0,1.0,3.0,0.2},
            {0.2,0.25,0.3333,1.0,1.0,3.0,0.2},
            {0.1667,0.2,0.25,0.3333,0.3333,1.0,0.1667},
            {1.0,3.0,3.0,5.0,5.0,6.0,1.0}
        };
        FillSubMatrix(MeasureParaMatrix, ReturnSubMatrices.MeasureParaMatrix, IndeciesArray);
        double[] MeasureParaMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.MeasureParaMatrix,
IndeciesArray.Length, "Measurement Parameter:");

        FillSubMatrix(MeasureParaMatrix, ReturnSubMatricesNotNormalized.MeasureParaMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.MeasureParaMatrix,
MeasureParaMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] TempMeasureMatrix = new double[,] {
            {1.0,0.25,0.2,0.3333,0.3333,0.5,0.1667},
            {4.0,1.0,0.3333,1.0,3.0,3.0,0.1667},
            {5.0,3.0,1.0,4.0,5.0,5.0,0.5},
```

```
            {3.0,1.0,0.25,1.0,2.0,2.0,0.25},
            {3.0,0.3333,0.2,0.5,1.0,1.0,0.1667},
            {2.0,0.3333,0.2,0.5,1.0,1.0,0.2},
            {6.0,6.0,2.0,4.0,6.0,5.0,1.0}
        };
        FillSubMatrix(TempMeasureMatrix, ReturnSubMatrices.TempMeasureMatrix, IndeciesArray);
        double[] TempMeasureMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.TempMeasureMatrix,
IndeciesArray.Length, "Temperature Measurement:");

        FillSubMatrix(TempMeasureMatrix, ReturnSubMatricesNotNormalized.TempMeasureMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.TempMeasureMatrix,
TempMeasureMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] StimulationElecMatrix = new double[,] {
            {1.0,4.0,3.0,1.0,1.0,2.0,6.0},
            {0.25,1.0,0.5,0.2,0.2,0.25,3.0},
            {0.3333,2.0,1.0,0.25,0.25,0.5,3.0},
            {1.0,5.0,4.0,1.0,1.0,2.0,6.0},
            {1.0,5.0,4.0,1.0,1.0,3.0,6.0},
            {0.5,4.0,2.0,0.5,0.3333,1.0,4.0},
            {0.1667,0.3333,0.3333,0.1667,0.1667,0.25,1.0}
        };
        FillSubMatrix(StimulationElecMatrix, ReturnSubMatrices.StimulationElecMatrix, IndeciesArray);
        double[] StimulationElecMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.StimulationElecMatrix,
IndeciesArray.Length, "Stimulation Electronics Required:");

        FillSubMatrix(StimulationElecMatrix, ReturnSubMatricesNotNormalized.StimulationElecMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.StimulationElecMatrix,
StimulationElecMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] TypOutputLevelMatrix = new double[,] {
            {1.0,6.0,1.0,4.0,1.0,2.0,1.0},
            {0.1667,1.0,0.1667,0.25,0.1667,0.2,0.125},
            {1.0,6.0,1.0,4.0,1.0,2.0,1.0},
            {0.25,4.0,0.25,1.0,0.25,0.3333,0.1667},
            {1.0,6.0,1.0,4.0,1.0,3.0,1.0},
            {0.5,5.0,0.5,3.0,0.3333,1.0,0.3333},
            {1.0,8.0,1.0,6.0,1.0,3.0,1.0}
        };
        FillSubMatrix(TypOutputLevelMatrix, ReturnSubMatrices.TypOutputLevelMatrix, IndeciesArray);
        double[] TypOutputLevelMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.TypOutputLevelMatrix,
IndeciesArray.Length, "Typical Output Levels Per Degree Celsius:");

        FillSubMatrix(TypOutputLevelMatrix, ReturnSubMatricesNotNormalized.TypOutputLevelMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.TypOutputLevelMatrix,
TypOutputLevelMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] TypFastThertimeConsMatrix = new double[,] {
            {1.0,3.0,4.0,6.0,5.0,1.0,3.0},
            {0.3333,1.0,2.0,4.0,3.0,0.3333,1.0},
            {0.25,0.5,1.0,2.0,2.0,0.25,1.0},
            {0.1667,0.25,0.5,1.0,0.5,0.1667,0.3333},
            {0.2,0.3333,0.5,2.0,1.0,0.2,0.3333},
            {1.0,3.0,4.0,6.0,5.0,1.0,3.0},
            {0.3333,1.0,1.0,3.0,3.0,0.3333,1.0}
        };
        FillSubMatrix(TypFastThertimeConsMatrix, ReturnSubMatrices.TypFastThertimeConsMatrix, IndeciesArray);
        double[] TypFastThertimeConsMatrixResult =
CalculateWeightsForEachCriterion(ReturnSubMatrices.TypFastThertimeConsMatrix, IndeciesArray.Length, "Typical Fast Thermal
Time Constant:");

        FillSubMatrix(TypFastThertimeConsMatrix, ReturnSubMatricesNotNormalized.TypFastThertimeConsMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.TypFastThertimeConsMatrix,
TypFastThertimeConsMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] TypSmallSizMatrix = new double[,] {
            {1.0,2.0,3.0,4.0,5.0,6.0,5.0},
            {0.5,1.0,2.0,3.0,4.0,5.0,4.0},
            {0.3333,0.5,1.0,2.0,4.0,6.0,4.0},
            {0.25,0.3333,0.5,1.0,2.0,3.0,2.0},
            {0.2,0.25,0.25,0.5,1.0,3.0,1.0},
            {0.1667,0.2,0.1667,0.3333,0.3333,1.0,0.5},
```

```
            {0.2,0.25,0.25,0.5,1.0,2.0,1.0}
        };
        FillSubMatrix(TypSmallSizMatrix, ReturnSubMatrices.TypSmallSizMatrix, IndeciesArray);
        double[] TypSmallSizMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.TypSmallSizMatrix,
IndeciesArray.Length, "Typical Small Size:");

        FillSubMatrix(TypSmallSizMatrix, ReturnSubMatricesNotNormalized.TypSmallSizMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.TypSmallSizMatrix,
TypSmallSizMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] NoiseImmunityMatrix = new double[,] {
            {1.0,0.1667,0.3333,0.25,0.25,0.5,0.25},
            {6.0,1.0,4.0,3.0,3.0,5.0,3.0},
            {3.0,0.25,1.0,0.5,0.5,2.0,0.5},
            {4.0,0.3333,2.0,1.0,1.0,3.0,1.0},
            {4.0,0.3333,2.0,1.0,1.0,4.0,1.0},
            {2.0,0.2,0.5,0.3333,0.25,1.0,0.25},
            {4.0,0.3333,2.0,1.0,1.0,4.0,1.0}
        };
        FillSubMatrix(NoiseImmunityMatrix, ReturnSubMatrices.NoiseImmunityMatrix, IndeciesArray);
        double[] NoiseImmunityMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.NoiseImmunityMatrix,
IndeciesArray.Length, "Noise Immunity:");

        FillSubMatrix(NoiseImmunityMatrix, ReturnSubMatricesNotNormalized.NoiseImmunityMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.NoiseImmunityMatrix,
NoiseImmunityMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] FraDurMatrix = new double[,] {
            {1.0,6.0,3.0,6.0,8.0,3.0,4.0},
            {0.1667,1.0,0.3333,2.0,3.0,0.3333,0.5},
            {0.3333,3.0,1.0,3.0,4.0,2.0,3.0},
            {0.1667,0.5,0.3333,1.0,3.0,0.25,0.3333},
            {0.125,0.3333,0.25,0.3333,1.0,0.25,0.3333},
            {0.3333,3.0,0.5,4.0,4.0,1.0,3.0},
            {0.25,2.0,0.3333,3.0,3.0,0.3333,1.0},

        };
        FillSubMatrix(FraDurMatrix, ReturnSubMatrices.FraDurMatrix, IndeciesArray);
        double[] FraDurMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.FraDurMatrix, IndeciesArray.Length,
"Fragility-Durability:");

        FillSubMatrix(FraDurMatrix, ReturnSubMatricesNotNormalized.FraDurMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.FraDurMatrix, FraDurMatrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] HiThGrEnMatrix = new double[,] {
            {1.0,4.0,5.0,7.0,8.0,4.0,6.0},
            {0.25,1.0,2.0,5.0,6.0,2.0,4.0},
            {0.2,0.5,1.0,3.0,3.0,0.5,2.0},
            {0.1429,0.2,0.3333,1.0,1.0,0.25,0.5},
            {0.125,0.1667,0.3333,1.0,1.0,0.2,0.3333},
            {0.25,0.5,2.0,4.0,5.0,1.0,4.0},
            {0.1667,0.25,0.5,2.0,3.0,0.25,1.0}
        };
        FillSubMatrix(HiThGrEnMatrix, ReturnSubMatrices.HiThGrEnMatrix, IndeciesArray);
        double[] HiThGrEnMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.HiThGrEnMatrix,
IndeciesArray.Length, "High Thermal Gradient Environment:");

        FillSubMatrix(HiThGrEnMatrix, ReturnSubMatricesNotNormalized.HiThGrEnMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.HiThGrEnMatrix, HiThGrEnMatrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] CorrResMatrix = new double[,] {
            {1.0,0.25,0.1667,0.5,0.1667,0.1667,0.25},
            {4.0,1.0,0.3333,2.0,0.25,0.25,1.0},
            {6.0,3.0,1.0,4.0,1.0,1.0,4.0},
            {2.0,0.5,0.25,1.0,0.25,0.25,0.5},
            {6.0,4.0,1.0,4.0,1.0,1.0,3.0},
            {6.0,4.0,1.0,4.0,1.0,1.0,3.0},
            {4.0,1.0,0.25,2.0,0.3333,0.3333,1.0}
        };
        FillSubMatrix(CorrResMatrix, ReturnSubMatrices.CorrResMatrix, IndeciesArray);
```

```
        double[] CorrResMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.CorrResMatrix, IndeciesArray.Length,
"Corrosion Resistance:");

        FillSubMatrix(CorrResMatrix, ReturnSubMatricesNotNormalized.CorrResMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.CorrResMatrix, CorrResMatrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] PointAreaMeasMatrix = new double[,] {
             {1.0,2.0,3.0,4.0,6.0,6.0,4.0},
              {0.5,1.0,2.0,3.0,4.0,4.0,3.0},
              {0.3333,0.5,1.0,2.0,3.0,4.0,2.0},
              {0.25,0.3333,0.5,1.0,2.0,2.0,0.5},
              {0.1667,0.25,0.3333,0.5,1.0,1.0,0.5},
              {0.1667,0.25,0.25,0.5,1.0,1.0,2.0},
              {0.25,0.5,0.3333,2.0,2.0,0.5,1.0}
        };
        FillSubMatrix(PointAreaMeasMatrix, ReturnSubMatrices.PointAreaMeasMatrix, IndeciesArray);
        double[] PointAreaMeasMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.PointAreaMeasMatrix,
IndeciesArray.Length, "Point or Area Measurement:");

        FillSubMatrix(PointAreaMeasMatrix, ReturnSubMatricesNotNormalized.PointAreaMeasMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.PointAreaMeasMatrix,
PointAreaMeasMatrixResult, IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] ManuVarMatrix = new double[,] {
              {1.0, 0.3333, 0.1667,0.25,0.5,0.2,0.25},
              {3.0, 1.0, 0.3333,0.5,2.0,0.25,0.5},
              {6.0, 3.0, 1.0,4.0,3.0,6.0,3.0},
              {4.0,2.0,0.25,1.0,4.0,0.3333,0.5},
              {2.0,0.5,0.3333,0.25,1.0,0.2,0.25},
              {5.0,4.0,0.1667,3.0,5.0,1.0,2.0},
              {4.0,2.0,0.3333,2.0,4.0,0.5,1.0}
        };
        FillSubMatrix(ManuVarMatrix, ReturnSubMatrices.ManuVarMatrix, IndeciesArray);
        double[] ManuVarMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.ManuVarMatrix,
IndeciesArray.Length, "Manufacturing Variances:");

        FillSubMatrix(ManuVarMatrix, ReturnSubMatricesNotNormalized.ManuVarMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.ManuVarMatrix, ManuVarMatrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] NistStanMatrix = new double[,] {
              {1.0   , 4.0, 1.0,1.0,1.0,4.0,5.0},
              {0.25, 1.0, 0.25,0.25,0.25,1.0,2.0},
              {1.0, 4.0, 1.0,1.0,1.0,4.0,5.0},
              {1.0,4.0,1.0,1.0,1.0,4.0,5.0},
              {1.0,4.0,1.0,1.0,1.0,4.0,5.0},
              {0.25,1.0,0.25,0.25,0.25,1.0,2.0},
              {0.2,0.5,0.2,0.2,0.2,0.5,1.0}
        };
        FillSubMatrix(NistStanMatrix, ReturnSubMatrices.NistStanMatrix, IndeciesArray);
        double[] NistStanMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.NistStanMatrix, IndeciesArray.Length,
"NIST Standards:");

        FillSubMatrix(NistStanMatrix, ReturnSubMatricesNotNormalized.NistStanMatrix, IndeciesArray);
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.NistStanMatrix, NistStanMatrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] CostMatrix = new double[,] {
              {1.0,1.0,6.0,3.0,3.0,6.0,3.0},
              {1.0,1.0,6.0,3.0,3.0,6.0,3.0},
              {0.1667,0.1667,1.0,0.25,0.25,1.0,0.25},
              {0.3333,0.3333,4.0,1.0,1.0,4.0,1.0},
              {0.3333,0.3333,4.0,1.0,1.0,4.0,1.0},
              {0.1667,0.1667,1.0,0.25,0.25,1.0,0.25},
              {0.3333,0.3333,4.0,1.0,1.0,4.0,1.0}
        };
        FillSubMatrix(CostMatrix, ReturnSubMatrices.CostMatrix, IndeciesArray);
        double[] CostMatrixResult = CalculateWeightsForEachCriterion(ReturnSubMatrices.CostMatrix, IndeciesArray.Length,
"Cost:");

        FillSubMatrix(CostMatrix, ReturnSubMatricesNotNormalized.CostMatrix, IndeciesArray);
```

```
        ConsistencyIndex = ComputeConsistencyIndex(ReturnSubMatricesNotNormalized.CostMatrix, CostMatrixResult,
IndeciesArray.Length);

        ////////////////////////////////////////////////////////////////////////////////////////////
        double[,] weightsOfCriteria = new double[,] {
            {1.0,9.0,6.0,4.0},
            {0.1111,1.0,0.5,0.3333},
            {0.1667,2.0,1.0,0.5},
            {0.25,3.0,2.0,1.0}
        };

        double[] weightsOfCriteriaResults = CalculateWeightsForEachCriterion(weightsOfCriteria, 4, "Weights of Criteria:");

        double[,] weightsOfCriteria2 = new double[,] {
            {1.0,9.0,6.0,4.0},
            {0.1111,1.0,0.5,0.3333},
            {0.1667,2.0,1.0,0.5},
            {0.25,3.0,2.0,1.0}
        };
        ConsistencyIndex = ComputeConsistencyIndex(weightsOfCriteria2, weightsOfCriteriaResults, 4);
        ////////////////////////////////////////////////////////////////////////////////////////////
        double[,] Weightssubcriteriastatic = new double[,] {
            {1.0,1.0,4.0,4.0,3.0,1.0,5.0,5.0,6.0,8.0,5.0},
            {1.0,1.0,4.0,4.0,3.0,1.0,5.0,5.0,6.0,8.0,5.0},
            {0.25,0.25,1.0,0.5,0.3333,0.25,2.0,2.0,4.0,6.0,3.0},
            {0.25,0.25,2.0,1.0,1.0,0.3333,3.0,2.0,4.0,6.0,3.0},
            {0.3333,0.3333,3.0,1.0,1.0,0.3333,4.0,5.0,6.0,9.0,4.0},
            {1.0,1.0,4.0,3.0,3.0,1.0,5.0,5.0,6.0,9.0,5.0},
            {0.2,0.2,0.5,0.3333,0.25,0.2,1.0,0.5,3.0,6.0,3.0},
            {0.2,0.2,0.5,0.5,0.2,0.2,2.0,1.0,3.0,5.0,1.0},
            {0.1667,0.1667,0.25,0.25,0.1667,0.1667,0.3333,0.3333,1.0,3.0,0.3333},
            {0.125,0.125,0.1667,0.1667,0.1111,0.1111,0.1667,0.2,0.3333,1.0,0.2},
            {0.2,0.2,0.3333,0.3333,0.25,0.2,0.3333,1.0,3.0,5.0,1.0}
        };
        double[] weightsSubCriteriaStaticResult = CalculateWeightsForEachCriterion(Weightssubcriteriastatic, 11, "Weights of Sub-
Criteria Static:");

        double[,] Weightssubcriteriastatic2 = new double[,] {
            {1.0,1.0,4.0,4.0,3.0,1.0,5.0,5.0,6.0,8.0,5.0},
            {1.0,1.0,4.0,4.0,3.0,1.0,5.0,5.0,6.0,8.0,5.0},
            {0.25,0.25,1.0,0.5,0.3333,0.25,2.0,2.0,4.0,6.0,3.0},
            {0.25,0.25,2.0,1.0,1.0,0.3333,3.0,2.0,4.0,6.0,3.0},
            {0.3333,0.3333,3.0,1.0,1.0,0.3333,4.0,5.0,6.0,9.0,4.0},
            {1.0,1.0,4.0,3.0,3.0,1.0,5.0,5.0,6.0,9.0,5.0},
            {0.2,0.2,0.5,0.3333,0.25,0.2,1.0,0.5,3.0,6.0,3.0},
            {0.2,0.2,0.5,0.5,0.2,0.2,2.0,1.0,3.0,5.0,1.0},
            {0.1667,0.1667,0.25,0.25,0.1667,0.1667,0.3333,0.3333,1.0,3.0,0.3333},
            {0.125,0.125,0.1667,0.1667,0.1111,0.1111,0.1667,0.2,0.3333,1.0,0.2},
            {0.2,0.2,0.3333,0.3333,0.25,0.2,0.3333,1.0,3.0,5.0,1.0}
        };
        ConsistencyIndex = ComputeConsistencyIndex(Weightssubcriteriastatic2, weightsSubCriteriaStaticResult, 11);
        ////////////////////////////////////////////////////////////////////////////////////////////
        double[,] weightssubCriteriaDynamic = new double[,] {
            {1.0,2.0,0.1667},
            {0.5,1.0,0.1667},
            {6.0,6.0,1.0}
        };
        double[] weightsSubCriteriaDynamicResult = CalculateWeightsForEachCriterion(weightssubCriteriaDynamic, 3, "Weights of
Sub-Criteria Dynamic:");

        double[,] weightssubCriteriaDynamic2 = new double[,] {
            {1.0,2.0,0.1667},
            {0.5,1.0,0.1667},
            {6.0,6.0,1.0}
        };
        ConsistencyIndex = ComputeConsistencyIndex(weightssubCriteriaDynamic2, weightsSubCriteriaDynamicResult, 3);
        ////////////////////////////////////////////////////////////////////////////////////////////
        double[,] weightssubCriteriaEnv = new double[,] {
            {1.0,1.0,0.2,4.0,0.1667},
            {1.0,1.0,0.25,5.0,0.2},
            {5.0,4.0,1.0,7.0,0.5},
            {0.25,0.2,0.1429,1.0,0.125},
            {6.0,5.0,2.0,8.0,1.0}
        };
        double[] weightsSubCriteriaEnvironmentalResult = CalculateWeightsForEachCriterion(weightssubCriteriaEnv, 5, "Weights of
Sub-Criteria Environmental:");
```

```
        double[,] weightssubCriteriaEnv2 = new double[,] {
            {1.0,1.0,0.2,4.0,0.1667},
            {1.0,1.0,0.25,5.0,0.2},
            {5.0,4.0,1.0,7.0,0.5},
            {0.25,0.2,0.1429,1.0,0.125},
            {6.0,5.0,2.0,8.0,1.0}
        };
        ConsistencyIndex = ComputeConsistencyIndex(weightssubCriteriaEnv2, weightsSubCriteriaEnvironmentalResult, 5);
        ////////////////////////////////////////////////////////////////////////////////////////
        double[,] Others = new double[,] {
            {1.0,3.0,0.5,0.1667},
            {0.3333,1.0,0.3333,0.1429},
            {2.0,3.0,1.0,0.2},
            {6.0,7.0,5.0,1.0}
        };
        double[] weightsSubCriteriaOthersResult = CalculateWeightsForEachCriterion(Others, 4, "Weights of Sub-Criteria Others:");

        double[,] Others2 = new double[,] {
             {1.0,3.0,0.5,0.1667},
            {0.3333,1.0,0.3333,0.1429},
            {2.0,3.0,1.0,0.2},
            {6.0,7.0,5.0,1.0}
        };
        ConsistencyIndex = ComputeConsistencyIndex(Others2, weightsSubCriteriaOthersResult, 4);
        ////////////////////////////////////////////////////////////////////////////////////////////////////
        //Calculate sub criteria aggregate weights with respect to final goal
        Scale(weightsSubCriteriaStaticResult, weightsOfCriteriaResults[0]);
        Scale(weightsSubCriteriaDynamicResult, weightsOfCriteriaResults[1]);
        Scale(weightsSubCriteriaEnvironmentalResult, weightsOfCriteriaResults[2]);
        Scale(weightsSubCriteriaOthersResult, weightsOfCriteriaResults[3]);

        ////////////////////////////////////////////////////////////////////////////////////////////////////
        //Calculate sub criteria aggregate weights for ThermoCouple (alternative 1)
        double ThermoCoupleFinalScore = 0.0;
        double ThermisterFinalScore = 0.0;
        double RTDFinalScore = 0.0;
        double BimetallicFinalScore = 0.0;
        double ThermometerFinalScore = 0.0;
        double PyrometerFinalScore = 0.0;
        double LCDDidplayFinalScore = 0.0;

        for (int k = 0; k < IndeciesArray.Length; k++)
        {
           if (IndeciesArray[k] == AHP.SENSOR_THERMOCOUPLE)
           {
              double[] ThermoCoupleAggregateSubCriteriaStatic = new double[11];
              double[] ThermoCoupleAggregateSubCriteriaDynamic = new double[3];
              double[] ThermoCoupleAggregateSubCriteriaEnvironmental = new double[5];
              double[] ThermoCoupleAggregateSubCriteriaOthers = new double[4];

              ThermoCoupleAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
              ThermoCoupleAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
              ThermoCoupleAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
              ThermoCoupleAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
              ThermoCoupleAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
              ThermoCoupleAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
              ThermoCoupleAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
              ThermoCoupleAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
              ThermoCoupleAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
              ThermoCoupleAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
              ThermoCoupleAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

              ThermoCoupleFinalScore = ThermoCoupleFinalScore +
GetSummationOfVectorElements(ThermoCoupleAggregateSubCriteriaStatic);

              ThermoCoupleAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] *
weightsSubCriteriaDynamicResult[0];
              ThermoCoupleAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] *
weightsSubCriteriaDynamicResult[1];
              ThermoCoupleAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] *
weightsSubCriteriaDynamicResult[2];

              ThermoCoupleFinalScore = ThermoCoupleFinalScore +
GetSummationOfVectorElements(ThermoCoupleAggregateSubCriteriaDynamic);
```

```
                ThermoCoupleAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[0];
                ThermoCoupleAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
                ThermoCoupleAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[2];
                ThermoCoupleAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[3];
                ThermoCoupleAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[4];

                ThermoCoupleFinalScore = ThermoCoupleFinalScore +
GetSummationOfVectorElements(ThermoCoupleAggregateSubCriteriaEnvironmental);

                ThermoCoupleAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
                ThermoCoupleAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
                ThermoCoupleAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
                ThermoCoupleAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

                ThermoCoupleFinalScore = ThermoCoupleFinalScore +
GetSummationOfVectorElements(ThermoCoupleAggregateSubCriteriaOthers);
                SensorRanks[k] = ThermoCoupleFinalScore;

                System.Console.Write("\n\n");
                System.Console.Write("Thermo Couple Final Score = {0}", ThermoCoupleFinalScore);
            }
            ////////////////////////////////////////////////////////////////////////////////////////////////////////////
            //Calculate sub criteria aggregate weights for Thermister (alternative 2)
            else if (IndeciesArray[k] == AHP.SENSOR_THERMISTER)
            {
                double[] ThermisterAggregateSubCriteriaStatic = new double[11];
                double[] ThermisterAggregateSubCriteriaDynamic = new double[3];
                double[] ThermisterAggregateSubCriteriaEnvironmental = new double[5];
                double[] ThermisterAggregateSubCriteriaOthers = new double[4];

                ThermisterAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
                ThermisterAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
                ThermisterAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
                ThermisterAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
                ThermisterAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
                ThermisterAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
                ThermisterAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
                ThermisterAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
                ThermisterAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
                ThermisterAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
                ThermisterAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

                ThermisterFinalScore = ThermisterFinalScore + GetSummationOfVectorElements(ThermisterAggregateSubCriteriaStatic);

                ThermisterAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] * weightsSubCriteriaDynamicResult[0];
                ThermisterAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] * weightsSubCriteriaDynamicResult[1];
                ThermisterAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] *
weightsSubCriteriaDynamicResult[2];

                ThermisterFinalScore = ThermisterFinalScore +
GetSummationOfVectorElements(ThermisterAggregateSubCriteriaDynamic);

                ThermisterAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[0];
                ThermisterAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
                ThermisterAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[2];
                ThermisterAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[3];
                ThermisterAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[4];

                ThermisterFinalScore = ThermisterFinalScore +
GetSummationOfVectorElements(ThermisterAggregateSubCriteriaEnvironmental);

                ThermisterAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
                ThermisterAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
                ThermisterAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
                ThermisterAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

                ThermisterFinalScore = ThermisterFinalScore + GetSummationOfVectorElements(ThermisterAggregateSubCriteriaOthers);
```

```
                System.Console.Write("\n\n");
                System.Console.Write("Thermister Final Score = {0}", ThermisterFinalScore);

                SensorRanks[k] = ThermisterFinalScore;
            }
            ////////////////////////////////////////////////////////////////////////////////////////////////////////
            //Calculate sub criteria aggregate weights for RTD (alternative 3)
            else if (IndeciesArray[k] == AHP.SENSOR_RTD)
            {
                double[] RTDAggregateSubCriteriaStatic = new double[11];
                double[] RTDAggregateSubCriteriaDynamic = new double[3];
                double[] RTDAggregateSubCriteriaEnvironmental = new double[5];
                double[] RTDAggregateSubCriteriaOthers = new double[4];

                RTDAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
                RTDAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
                RTDAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
                RTDAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
                RTDAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
                RTDAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
                RTDAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
                RTDAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
                RTDAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
                RTDAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
                RTDAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

                RTDFinalScore = RTDFinalScore + GetSummationOfVectorElements(RTDAggregateSubCriteriaStatic);

                RTDAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] * weightsSubCriteriaDynamicResult[0];
                RTDAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] * weightsSubCriteriaDynamicResult[1];
                RTDAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] * weightsSubCriteriaDynamicResult[2];

                RTDFinalScore = RTDFinalScore + GetSummationOfVectorElements(RTDAggregateSubCriteriaDynamic);

                RTDAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[0];
                RTDAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
                RTDAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[2];
                RTDAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[3];
                RTDAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[4];

                RTDFinalScore = RTDFinalScore + GetSummationOfVectorElements(RTDAggregateSubCriteriaEnvironmental);

                RTDAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
                RTDAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
                RTDAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
                RTDAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

                RTDFinalScore = RTDFinalScore + GetSummationOfVectorElements(RTDAggregateSubCriteriaOthers);
                SensorRanks[k] = RTDFinalScore;
            }
            else if (IndeciesArray[k] == AHP.SENSOR_BIMETALLIC)
            {
                double[] BimetallicAggregateSubCriteriaStatic = new double[11];
                double[] BimetallicAggregateSubCriteriaDynamic = new double[3];
                double[] BimetallicAggregateSubCriteriaEnvironmental = new double[5];
                double[] BimetallicAggregateSubCriteriaOthers = new double[4];

                BimetallicAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
                BimetallicAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
                BimetallicAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
                BimetallicAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
                BimetallicAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
                BimetallicAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
                BimetallicAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
                BimetallicAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
                BimetallicAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
                BimetallicAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
                BimetallicAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

                BimetallicFinalScore = BimetallicFinalScore + GetSummationOfVectorElements(BimetallicAggregateSubCriteriaStatic);

                BimetallicAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] * weightsSubCriteriaDynamicResult[0];
                BimetallicAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] * weightsSubCriteriaDynamicResult[1];
                BimetallicAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] *
weightsSubCriteriaDynamicResult[2];
```

```
            BimetallicFinalScore = BimetallicFinalScore +
GetSummationOfVectorElements(BimetallicAggregateSubCriteriaDynamic);

            BimetallicAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[0];
            BimetallicAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
            BimetallicAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[2];
            BimetallicAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[3];
            BimetallicAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[4];

            BimetallicFinalScore = BimetallicFinalScore +
GetSummationOfVectorElements(BimetallicAggregateSubCriteriaEnvironmental);

            BimetallicAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
            BimetallicAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
            BimetallicAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
            BimetallicAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

            BimetallicFinalScore = BimetallicFinalScore + GetSummationOfVectorElements(BimetallicAggregateSubCriteriaOthers);
            SensorRanks[k] = BimetallicFinalScore;
        }
        else if (IndeciesArray[k] == AHP.SENSOR_THERMOMETER)
        {
            double[] ThermometerAggregateSubCriteriaStatic = new double[11];
            double[] ThermometerAggregateSubCriteriaDynamic = new double[3];
            double[] ThermometerAggregateSubCriteriaEnvironmental = new double[5];
            double[] ThermometerAggregateSubCriteriaOthers = new double[4];

            ThermometerAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
            ThermometerAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
            ThermometerAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
            ThermometerAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
            ThermometerAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
            ThermometerAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
            ThermometerAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
            ThermometerAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
            ThermometerAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
            ThermometerAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
            ThermometerAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

            ThermometerFinalScore = ThermometerFinalScore +
GetSummationOfVectorElements(ThermometerAggregateSubCriteriaStatic);

            ThermometerAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] * weightsSubCriteriaDynamicResult[0];
            ThermometerAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] * weightsSubCriteriaDynamicResult[1];
            ThermometerAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] *
weightsSubCriteriaDynamicResult[2];

            ThermometerFinalScore = ThermometerFinalScore +
GetSummationOfVectorElements(ThermometerAggregateSubCriteriaDynamic);

            ThermometerAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[0];
            ThermometerAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
            ThermometerAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[2];
            ThermometerAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[3];
            ThermometerAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[4];

            ThermometerFinalScore = ThermometerFinalScore +
GetSummationOfVectorElements(ThermometerAggregateSubCriteriaEnvironmental);

            ThermometerAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
            ThermometerAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
            ThermometerAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
            ThermometerAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

            ThermometerFinalScore = ThermometerFinalScore +
GetSummationOfVectorElements(ThermometerAggregateSubCriteriaOthers);
            SensorRanks[k] = ThermometerFinalScore;
```

```
            }
            else if (IndeciesArray[k] == AHP.SENSOR_PYROMETER)
            {
                double[] PyrometerAggregateSubCriteriaStatic = new double[11];
                double[] PyrometerAggregateSubCriteriaDynamic = new double[3];
                double[] PyrometerAggregateSubCriteriaEnvironmental = new double[5];
                double[] PyrometerAggregateSubCriteriaOthers = new double[4];

                PyrometerAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
                PyrometerAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
                PyrometerAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
                PyrometerAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
                PyrometerAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
                PyrometerAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
                PyrometerAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
                PyrometerAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
                PyrometerAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
                PyrometerAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
                PyrometerAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

                PyrometerFinalScore = PyrometerFinalScore + GetSummationOfVectorElements(PyrometerAggregateSubCriteriaStatic);

                PyrometerAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] * weightsSubCriteriaDynamicResult[0];
                PyrometerAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] * weightsSubCriteriaDynamicResult[1];
                PyrometerAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] *
weightsSubCriteriaDynamicResult[2];

                PyrometerFinalScore = PyrometerFinalScore +
GetSummationOfVectorElements(PyrometerAggregateSubCriteriaDynamic);

                PyrometerAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[0];
                PyrometerAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
                PyrometerAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[2];
                PyrometerAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[3];
                PyrometerAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] * weightsSubCriteriaEnvironmentalResult[4];

                PyrometerFinalScore = PyrometerFinalScore +
GetSummationOfVectorElements(PyrometerAggregateSubCriteriaEnvironmental);

                PyrometerAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
                PyrometerAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
                PyrometerAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
                PyrometerAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

                PyrometerFinalScore = PyrometerFinalScore + GetSummationOfVectorElements(PyrometerAggregateSubCriteriaOthers);
                SensorRanks[k] = PyrometerFinalScore;
            }
            else if (IndeciesArray[k] == AHP.SENSOR_LCD_DISPLAY)
            {
                double[] LCDDidplayAggregateSubCriteriaStatic = new double[11];
                double[] LCDDidplayAggregateSubCriteriaDynamic = new double[3];
                double[] LCDDidplayAggregateSubCriteriaEnvironmental = new double[5];
                double[] LCDDidplayAggregateSubCriteriaOthers = new double[4];

                LCDDidplayAggregateSubCriteriaStatic[0] = MaximumTempMatrixResult[k] * weightsSubCriteriaStaticResult[0];
                LCDDidplayAggregateSubCriteriaStatic[1] = MinimumTempMatrixResult[k] * weightsSubCriteriaStaticResult[1];
                LCDDidplayAggregateSubCriteriaStatic[2] = TempCurveMtrixResult[k] * weightsSubCriteriaStaticResult[2];
                LCDDidplayAggregateSubCriteriaStatic[3] = MaxSensitivityMatrixResult[k] * weightsSubCriteriaStaticResult[3];
                LCDDidplayAggregateSubCriteriaStatic[4] = SelfHeatingMatrixResult[k] * weightsSubCriteriaStaticResult[4];
                LCDDidplayAggregateSubCriteriaStatic[5] = LongTermStabilityMatrixResult[k] * weightsSubCriteriaStaticResult[5];
                LCDDidplayAggregateSubCriteriaStatic[6] = TypTempCoeffMatrixResult[k] * weightsSubCriteriaStaticResult[6];
                LCDDidplayAggregateSubCriteriaStatic[7] = ExtWiresMatrixResult[k] * weightsSubCriteriaStaticResult[7];
                LCDDidplayAggregateSubCriteriaStatic[8] = LongWireMatrixResult[k] * weightsSubCriteriaStaticResult[8];
                LCDDidplayAggregateSubCriteriaStatic[9] = MeasureParaMatrixResult[k] * weightsSubCriteriaStaticResult[9];
                LCDDidplayAggregateSubCriteriaStatic[10] = TempMeasureMatrixResult[k] * weightsSubCriteriaStaticResult[10];

                LCDDidplayFinalScore = LCDDidplayFinalScore +
GetSummationOfVectorElements(LCDDidplayAggregateSubCriteriaStatic);

                LCDDidplayAggregateSubCriteriaDynamic[0] = StimulationElecMatrixResult[k] * weightsSubCriteriaDynamicResult[0];
                LCDDidplayAggregateSubCriteriaDynamic[1] = TypOutputLevelMatrixResult[k] * weightsSubCriteriaDynamicResult[1];
                LCDDidplayAggregateSubCriteriaDynamic[2] = TypFastThertimeConsMatrixResult[k] *
weightsSubCriteriaDynamicResult[2];
```

```
                LCDDidplayFinalScore = LCDDidplayFinalScore +
GetSummationOfVectorElements(LCDDidplayAggregateSubCriteriaDynamic);

                LCDDidplayAggregateSubCriteriaEnvironmental[0] = TypSmallSizMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[0];
                LCDDidplayAggregateSubCriteriaEnvironmental[1] = NoiseImmunityMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[1];
                LCDDidplayAggregateSubCriteriaEnvironmental[2] = FraDurMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[2];
                LCDDidplayAggregateSubCriteriaEnvironmental[3] = HiThGrEnMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[3];
                LCDDidplayAggregateSubCriteriaEnvironmental[4] = CorrResMatrixResult[k] *
weightsSubCriteriaEnvironmentalResult[4];

                LCDDidplayFinalScore = LCDDidplayFinalScore +
GetSummationOfVectorElements(LCDDidplayAggregateSubCriteriaEnvironmental);

                LCDDidplayAggregateSubCriteriaOthers[0] = PointAreaMeasMatrixResult[k] * weightsSubCriteriaOthersResult[0];
                LCDDidplayAggregateSubCriteriaOthers[1] = ManuVarMatrixResult[k] * weightsSubCriteriaOthersResult[1];
                LCDDidplayAggregateSubCriteriaOthers[2] = NistStanMatrixResult[k] * weightsSubCriteriaOthersResult[2];
                LCDDidplayAggregateSubCriteriaOthers[3] = CostMatrixResult[k] * weightsSubCriteriaOthersResult[3];

                LCDDidplayFinalScore = LCDDidplayFinalScore +
GetSummationOfVectorElements(LCDDidplayAggregateSubCriteriaOthers);
                SensorRanks[k] = LCDDidplayFinalScore;
            }
        }

        textBoxResults.AppendText("\n\n");
        textBoxResults.AppendText("\nSensor Ranks: \n");
        for (int j = 0; j < SensorRanks.Length; j++)
        {
            textBoxResults.AppendText("" + SensorRanks[j] + "\n");
        }

        return SensorRanks;
    }
  }
}
```

6. Program.cs File(Entry Point)

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace AhpCaseStudy1GUI
{
   static class Program
   {

      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.EnableVisualStyles();
         Application.SetCompatibleTextRenderingDefault(false);
         Application.Run(new Form1());
      }
   }
}
```

Appendix VII: Complete List of Software Results for the Three Sensors: the Thermocouple, the Thermister, and the RTD Automotive Catalytic Converter Case Study

---

Maximum Operating Temperature Matrix:
1 3 1
0.3333 1 0.3333
1 3 1

Alternatives Weight Vector = 0.42857 0.14284 0.42857

Consistency Index = 0
Consistency Ratio = 0

Minimum Operating Temperature Matrix:
1 2 2
0.5 1 1
0.5 1 1

Alternatives Weight Vector = 0.5 0.25 0.25

Consistency Index = 0
Consistency Ratio = 0

Temperature Curve Matrix:
1 3 0.3333
0.3333 1 0.1667
3 6 1

Alternatives Weight Vector = 0.25099 0.09601 0.65299

Consistency Index = 0.00918
Consistency Ratio = 0.01583

Sensitivity Matrix:
1 0.1111 0.2
9 1 4
5 0.25 1

Alternatives Weight Vector = 0.06225 0.70131 0.23643

Consistency Index = 0.03610
Consistency Ratio = 0.06225

Self-Heating Issues Matrix:
1 8 3
0.125 1 0.2
0.3333 5 1

Alternatives Weight Vector = 0.65714 0.06825 0.274591

Consistency Index = 0.02217
Consistency Ratio = 0.03823

Long Term Stability and Accuracy Matrix:
1 0.25 0.1667
4 1 0.3333
6 3 1

Alternatives Weight Vector = 0.08695 0.27371 0.63933

Consistency Index = 0.02704
Consistency Ratio = 0.04663

Typical Temperature Coefficient Matrix:

1 0.1667 0.3333
6 1 3
3 0.3333 1

Alternatives Weight Vector = 0.09601 0.65299 0.25099

Consistency Index = 0.00918
Consistency Ratio = 0.01583

Extension Wires Matrix:
1 0.1667 0.1667
6 1 1
6 1 1

Alternatives Weight Vector = 0.07692 0.46153 0.46153

Consistency Index = 0
Consistency Ratio = 0

Long Wire Runs From Sensor Matrix:
1 0.3333 1
3 1 3
1 0.3333 1

Alternatives Weight Vector = 0.19999 0.60000 0.19999

Consistency Index = 0
Consistency Ratio = 0

Measurement Parameter Matrix:
1 4 3
0.25 1 0.5
0.3333 2 1

Alternatives Weight Vector = 0.62322 0.13728 0.23948

Consistency Index = 0.00915
Consistency Ratio = 0.01578

Temperature Measurement Matrix:
1 0.25 0.2
4 1 0.3333
5 3 1

Alternatives Weight Vector = 0.09642 0.28422 0.61935

Consistency Index = 0.04333
Consistency Ratio = 0.07471

Stimulation Electronics Required Matrix:
1 4 3
0.25 1 0.5
0.3333 2 1

Alternatives Weight Vector = 0.62322 0.13728 0.23948

Consistency Index = 0.00915
Consistency Ratio = 0.01578

Existence of Maximum Sensitivity Region Matrix:
1 6 1
0.1667 1 0.1667
1 6 1

Alternatives Weight Vector = 0.46153 0.07693 0.46153

Consistency Index = 0
Consistency Ratio = 0

Typical Fast Thermal Time Constant Matrix:
1 3 4
0.3333 1 2
0.25 0.5 1
Alternatives Weight Vector = 0.62322 0.23948 0.13728

Consistency Index = 0.00915

Consistency Ratio = 0.01578

Typical Small Size Matrix:

1	2	3
0.5	1	2
0.3333	0.5	1

Alternatives Weight Vector = 0.53896 0.29725 0.16377

Consistency Index = 0.00458
Consistency Ratio = 0.00790

Noise Immunity matrix:

1	0.1667	0.33333
6	1	4
3	0.25	1

Alternatives Weight Vector = 0.09338 0.68529 0.22132

Consistency Index = 0.02710
Consistency Ratio = 0.04672

Fragility-Durability Matrix:

1	6	3
0.1667	1	0.3333
0.3333	3	1

Alternatives Weight Vector = 0.65299 0.09601 0.25099

Consistency Index = 0.00918
Consistency Ratio = 0.01583

High Thermal Gradient Environment Matrix:

1	4	5
0.25	1	2
0.2	0.5	1

Alternatives Weight Vector = 0.68064 0.20141 0.11794

Consistency Index = 0.01235
Consistency Ratio = 0.02129

Corrosion Resistance Matrix:

1	0.25	0.1667
4	1	0.3333
6	3	1

Alternatives Weight Vector = 0.08695 0.27371 0.63933

Consistency Index = 0.02704
Consistency Ratio = 0.04663

Point or Area Measurement Matrix:

1	2	3
0.5	1	2
0.3333	0.5	1

Alternatives Weight Vector = 0.53896 0.29725 0.16377

Consistency Index = 0.00458
Consistency Ratio = 0.00790

Manufacturing Variances Matrix:

1	0.3333	0.1667
3	1	0.3333
6	3	1

Alternatives Weight Vector = 0.09601 0.25099 0.65299

Consistency Index = 0.00918
Consistency Ratio = 0.01583

NIST Standards Matrix:

1 4 5
0.25 1 0.25
1 4 1

Alternatives Weight Vector = 0.44444 0.11111 0.44444

Consistency Index = 0
Consistency Ratio = 0

Cost Matrix:
1 1 6
1 1 6
0.1667 0.1667 1

Alternatives Weight Vector = 0.46153 0.46153 0.07693

Consistency Index = 0
Consistency Ratio = 0

Criteria Matrix:
1 4 3 4

0.25 1 0.5 1

0.3333 2 1 2

0.25 1 0.5 1

Criteria Weight Vector = 0.53636 0.12159 0.22045 0.12159

Consistency Index = 0.00686
Consistency Ratio = 0.00762

Sub-Criteria Static Matrix:
1.0 1.0 5.0 4.0 4.0 2.0 5.0 6.0 7.0 8.0 6.0

1.0 1.0 5.0 4.0 4.0 2.0 5.0 6.0 7.0 8.0 6.0

0.2 0.2 1.0 0.3333 0.3333 0.25 1.0 2.0 4.0 5.0 3.0

0.25 0.25 3.0 1.0 2.0 0.5 3.0 3.0 5.0 6.0 4.0

0.25 0.25 3.0 0.5 1.0 0.3333 3.0 5.0 6.0 8.0 4.0

0.5 0.5 4.0 2.0 3.0 1.0 4.0 5.0 6.0 8.0 5.0

0.2 0.2 1.0 0.3333 0.3333 0.25 1.0 1.0 4.0 6.0 3.0

0.1667 0.1667 0.5 0.3333 0.2 0.2 1.0 1.0 3.0 4.0 1.0

0.1429 0.1429 0.25 0.2 0.1667 0.1667 0.25 0.3333 1.0 2.0 0.3333

0.125 0.125 0.2 0.1667 0.125 0.125 0.1667 0.25 0.5 1.0 0.25

0.1667 0.1667 0.3333 0.25 0.25 0.2 0.3333 1.0 3.0 4.0 1.0

Sub-Criteria Static Weight Vector = 0.22118 0.22118 0.05379 0.09836 0.09777 0.15040 0.05233 0.03703 0.01983 0.01452 0.03355

Consistency Index = 0.08281
Consistency Ratio = 0.05208

Sub-Criteria Dynamic Matrix:
1 2 0.1667

0.5 1 0.1667

6 6 1

Relative Weight Vector = 0.16019 0.10093 0.73887

Consistency Index = 0.02722
Consistency Ratio = 0.04694

Sub-Criteria Environmental Matrix:

1	3	0.3333	4	0.25
0.3333	1	0.25	3	0.2
3	4	1	5	0.5
0.25	0.3333	0.2	1	0.1667
4	5	2	6	1

Relative Weight Vector = 0.15164 0.08645 0.28264 0.04767 0.43157

Consistency Index = 0.06346
Consistency Ratio = 0.05666

Sub-Criteria Others Matrix:

1	3	0.5	0.25
0.3333	1	0.3333	0.2
2	3	1	0.3333
4	5	3	1

Relative Weight Vector = 0.15750 0.07747 0.22913 0.53589

Consistency Index = 0.03752
Consistency Ratio = 0.04169

Relative Weight Vector = 0.118638 0.11863 0.02885 0.05276 0.05244 0.08066 0.02807 0.01986 0.01063 0.00778 0.01799
Relative Weight Vector = 0.01947 0.01227 0.08984
Relative Weight Vector = 0.03343 0.01905 0.06230 0.01050 0.09514
Relative Weight Vector = 0.01915 0.00942 0.02786 0.06516

Sensor Ranks
0.37849
0.27560
0.34589

Abstract in Arabic

تطبيق برمجي لا ختيار مستشعرات قياس الحرارة باستخدام طريقة التسلسل الهرمي التحليلي

اعداد: شادي محمد البعول

الملخص

الاطروحة تقدم تطبيق برمجي يعتمد على استخدام طريقة التسلسل الهرمي التحليلي (AHP) بهدف اختيار أفضل مستشعر لقياس الحرارة من بين عدة مستشعرات و لتطبيقات متعددة. تقوم منهجية الإختيار على اعطاء المستشعرات ذات الخصائص المختلفة رُتباً (Ranks) ناتجة عن تركيب الأوزان النسبية لكل مستشعر نسبة الى المستشعرات الأخرى في المستويات المختلفة للتسلسل الهرمي و بالإعتماد على معايير تقييم مستقلة. يتم حساب الوزن النسبي لكل مستشعر بالنسبة للطبقة المباشرة الأعلى في كل مستوى من مستويات التسلسل الهرمي بوساطة مقارنات تُجرى بين هذه المستشعرات مثنى مثنى (pair-wise) و التي تؤخذ من مواصفات المستشعرات المعلومة ضمن التطبيق الواحد. هذه الأوزان الثنائية مُدخلة ومتضمّنة داخل البرنامج (embedded) و يقوم البرنامج باسترجاعها بمجرد أن يحدد المستخدم للبرنامج كلاً من التطبيق الصناعي، قيود التطبيق، و المستشعرات المتوفرة. من مزايا طريقة التسلسل الهرمي التحليلي أنها طرقة تقييم لأداء البدائل مكممة وعقلانية ، هي تسمح بعملية اتخاذ قرار أسهل و أكثر تنظيماً من مجرد الأراء الشخصية الخاضعة لأراء الأفراد و المعرّضة لأحكام خاطئة. في هذه الدراسة، تتم عملية الإختيار بوساطة برنامج حاسوبي مبنيّ باستخدام لغة (C #) لتسهيل اجراء عملية الإختيار بطريقة جاهزة ومحوسبة وسهلة على المستخدم، و بالتالي هذه الدراسة تقدم المساعدة للعاملين في الصناعة و الراغبين في الإختيار بين مستشعرات حرارة متعددة.

إن البرنامج الحاسوبي المقترح متعدد الإستخدام و متنوع و قابل للتطبيق بالنسبة للعديد من حالات اختيار المستشعرات. تم تقديم مثال في هذه الأطروحة على استخدام البرنامج يتمثل بتطبيق المحول المحفّز في السيارات. يتطلب هذا التطبيق استخدام مستعرات حرارة قادرة على قياس حرارات عالية في المدى 500-750 درجة مئوية، و بدرجة حرارة قصوى قد تصل الى 870 درجة مئوية. تم الاختيار في هذا المثال بين ثلاثة أصناف من مستشعرات الحرارة: الثنائي الحراري (Thermocouples)، والثيرمستر(Thermisters)، و موازين الحرارة المتحسسة بالمقاومة (RTDs). و مع ذلك، فالبرنامج الحاسوبي قوي و قابل للتطبيق على مجوعة أوسع من الخيارات ولتطبيقات صناعية متعددة.