Impedance spectroscopy workstation for electrical characterization of ionically conducting glasses

by

Hitendra K. Patel

A Thesis Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Department: Materials Science and Engineering
Major: Ceramic Engineering

Iowa State University
Ames, Iowa
1989

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

Impedance spectroscopy (IS) is a powerful technique for characterizing many electrical properties of materials and plays an important role in science and engineering. It involves a relatively simple electrical measurement whose results may often be correlated with mass transport, dielectric properties, defects, microstructure and compositional influences on the conductance of solids.

Historically, IS developed from Cole-Cole plots [1, 2, 3, 4], i.e., graphs of ϵ' and ϵ'' for dielectric systems in the complex plane, which were first plotted as early as 1941 by Cole and Cole [5]. Impedance spectroscopy today, however, uses some or all of the four related functions: impedance (Z), admittance (Y), modulus (M) and the dielectric permittivity (ϵ). Bound or mobile charges in ionic, mixed electronic-ionic, semiconductor and insulator materials have been investigated using IS [1, 2, 4, 6]. IS has been especially powerful in determining the conductance and dielectric properties of glasses [4, 6, 7, 8].

Isothermal electrical measurements made at different temperatures have been used successfully to determine thermally activated ionic conductivities. Activation energies for dc conduction due to ionic motion in glass have been determined for many years by this technique [6, 7, 8, 9, 10].

While the theory of IS is well established, its growth has been slowed by the lack of convenient, fast and automated instrumentation. A single instrument capable of frequency and temperature scanning, data collection and data storage is not commercially available. Factors which contribute to this are that the sample holder (cell) is user, frequency and material specific; many samples are atmospheric sensitive and the need for strict environment control is an added concern; and finally, analysis software for easy conversion of raw data into meaningful results is often limited in capability.

Commercial instruments are, however, available which can automatically measure the impedance as a function of frequency over wide ranges, and these are easily interfaced to personal computers (PCs). Temperature controllers with high stability and computer interfaceability are also available commercially. The furnace for the temperature controller may be easily built or bought. Sample holders, however, have to be designed by the user to meet specific requirements such as size, ease of handling, temperature, frequency range and environment.

The goal of this project is to set up an automated impedance spectroscopy workstation. It must be accurate, precise, fast, convenient to use, and be able to generate reproducible results. Once experimental conditions are established by the operator, the workstation must be able to automatically control the entire experiment. Analyses of data using both numerical and graphical methods should be available to enhance data interpretation.

CHAPTER 2. BACKGROUND

The purpose of this chapter is to give a background for the application of impedance spectroscopy as a tool for materials characterization. A discussion of ionically conducting glass is included because the final application of this technique will be the electrical characterization of fast ion conducting glasses.

Impedance Spectroscopy

The IS section is broken into three parts. Impedance theory, definitions and derivations of impedance-related functions are presented first. This is followed by a description of the basic IS experiment. Finally, an elementary analysis of impedance spectra is given to show the types of information that can be obtained from the spectra.

Impedance theory and impedance related functions

The concept of electrical resistance was introduced by G. Ohm (1787- 1854) who observed a constant of proportionality between the applied dc voltage and measured current, and this relationship is known today as Ohm's Law in [11]. In 1883, O. Heaviside in [12] was the first to develop the concept of electrical impedance, and this was further mathematically developed into vector and complex notation by A. E.

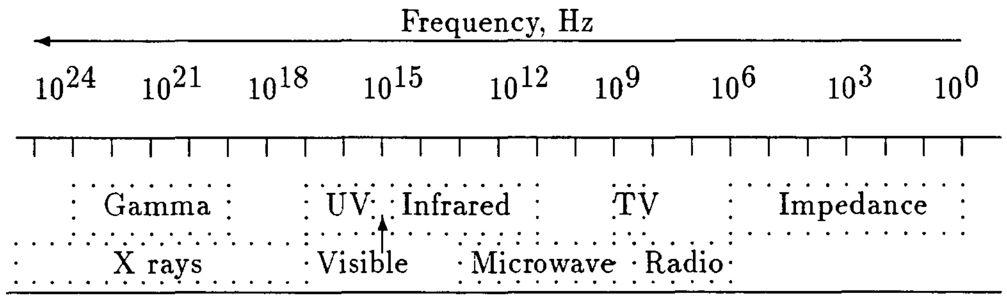


Figure 2.1: The electromagnetic spectrum

Kennelly and C. P. Steinmetz in [12]. Electrical impedance is a fundamental concept in electrical engineering and materials scientists have extended it and the branch of electrical measurements into IS.

The impedance of a material is defined as the ratio of the voltage to the current and is in general a complex quantity. Impedance, as used in the name impedance spectroscopy describes, both the technique of measurement and the frequency or energy range of the technique. The frequency spectrum for IS typically extends from approximately 1 Hz to 1 MHz and this is shown in Figure 2.1.

The input signal The input stimulus is a monochromatic voltage given by $v(t) = V_m sin(\omega t + \theta)$ and has an amplitude V_m, an angular frequency ω and phase shift θ [13]. When this voltage is applied to a material, a sinusoidal current with the same frequency but a different amplitude and phase shift is observed (see Figure 2.2). These sinusoidal signals are transformed to phasors (see next part) and the ratio of the voltage phasor and the current phasor is defined as the impedance.

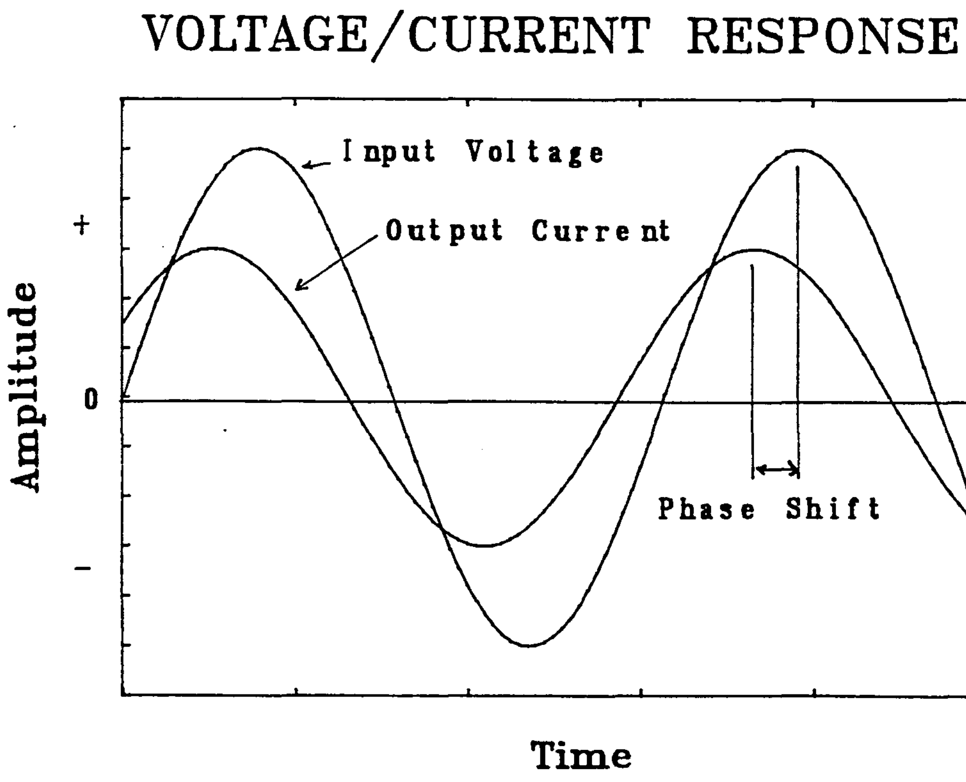


Figure 2.2: Typical voltages and current responses of material

Phasor A phasor is a complex number that carries the amplitude and phase angle information of a sinusoidal function. Euler's identity relates the complex exponential function to the sinusoidal function and is given below [14]:

$$e^{\pm j\theta} = cos\theta \pm jsin\theta \tag{2.1}$$

where $j \equiv \sqrt{-1}$. This complex exponential may be represented by one of the three equivalent notations for phasors:

polar form $\quad V^{*} = \mid V \mid \angle\theta$

rectangular form $\quad V^* = \mid V \mid cos\theta \pm j \mid V \mid sin\theta$

exponential form $\quad V^* = \mid V \mid e^{\pm j\theta}$.

All phasors will be represented with the symbol " * " and are not to be confused with the complex conjugate. The amplitude of a sinusoidal signal is defined as the magnitude and will appear between two parallel bars. In rectangular (cartesian) coordinates, a phasor Z^* may be represented by

$$Z^* = Z' + jZ''. \tag{2.2}$$

This is graphically shown in Figure 2.3 where both the rectangular and polar form are represented. Complex computations with phasors are greatly simplified by adding

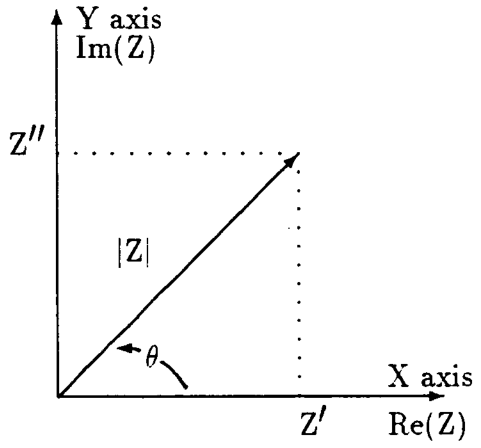


Figure 2.3: Impedance plotted as a phasor

and subtracting in the cartesian form and multiplying and dividing in the polar form. The process for addition in cartesian form is to simply sum the real parts to get the real component and to sum the imaginary parts to yield the imaginary component.

Multiplying two phasors in the polar form simply requires multiplying the magnitudes of the two components to get the resultant magnitude and summing the phases to get the resultant phase. Division requires dividing the magnitudes and subtracting the phases. The reader is referred to other references for a complete description [13, 14].

Impedance The impedance has been defined as the ratio of the voltage phasor to the current phasor at a particular frequency. It is represented by the letter Z^* and has the units of ohms (Ω). Impedance can be written as:

$$Z^*(\omega) = Z'(\omega) - jZ''(\omega) \tag{2.3}$$

where the real part of the impedance Z' is the resistive component R and the imaginary part Z'' is the reactive component X. The real part is a measure of the energy dissipated in a material and the reactive part is the energy stored by the material during the application of the sinusoidal voltage. The reactive component has a negative sign because IS measurements usually involve capacitive and rarely inductive elements. To analyze the impedance, it is customary to plot the impedance in the Z' versus $-Z''$ plane.

Admittance The admittance Y^* is the ratio of the current phasor to the voltage phasor and is, therefore, the reciprocal of the impedance. It is measured in siemens and is commonly expressed as mhos ($\mho$). The real component of admittance is called the conductance (G) and the imaginary component is called the susceptance (B). The reciprocal of the impedance is a complex operation and is given by:

$$Y^* \equiv \frac{1}{Z^*} = \frac{Z' + jZ''}{\mid Z \mid^2} \equiv G + jB \tag{2.4}$$

Dielectric permittivity The dielectric permittivity ϵ^* describes the material's ability to store electrical energy. The dielectric permittivity is given by

$$\epsilon^* = \epsilon_O k^* = \epsilon_O(k' - jk'') \tag{2.5}$$

where ϵ_O is the dielectric permittivity of free space and has units of F/m and k^* is the relative dielectric permittivity and is a ratio of the dielectric permittivity of the material to the dielectric permittivity of vacuum. k' is called the relative dielectric constant and determines the polarizability of a material. k'' is defined as the relative loss factor and describes the ohmic losses in a material. A material is said to be a good conductor if $k'' \gg k'$ and a good insulator if $k'' \ll k'$. The ratio of k'' to k' is called the loss tangent or dissipation factor, tan δ. The inverse of the dissipation factor is called the quality factor Q and is a frequently used design parameter during the manufacture of electronic devices and circuits. Good insulators usually have dissipation factors less than 0.001 and relative dielectric constants less than 30. Materials used in capacitors are called dielectrics and have a k' greater than 30 and a dissipation factor less then 0.001. Appendix B shows that the relative dielectric permittivity and the impedance are related by the equation shown below.

$$k^* = -j\frac{k_O}{\epsilon_O \omega Z^*} , \tag{2.6}$$

where k_O is the cell constant of the material and is the ratio of the distance d between the parallel plates of the capacitor to the surface area A perpendicular to the electric field.

Electrical modulus The electrical modulus is the inverse of the dielectric permittivity ϵ^*, has units of m/F and is given by

$$M^* = \frac{1}{\epsilon^*} = j\frac{\epsilon_o \omega Z^*}{k_o}. \tag{2.7}$$

This form of data presentation for conducting materials was introduced by Macedo et al. [3] and popularized by Hodge et al. [2] for the reason that it emphasizes bulk property at the expense of interfacial polarization.

Conductivity Conductivity has units of ℧/cm and is a measure of the current through a material. It is derived from the admittance by accounting for the effect of the cell constant k_o:

$$\sigma^* = k_o A^* = \frac{k_o}{Z^*}. \tag{2.8}$$

Resistivity The resistivity ρ^* is the inverse of the conductivity and has units Ωcm. It can be calculated from the impedance and the cell constant as shown:

$$\rho^* = \frac{1}{\sigma^*} = \frac{Z^*}{k_o} = \frac{R}{k_o} + j\frac{X}{k_o} \tag{2.9}$$

The resistivity listed in most handbooks of physical properties is the zero frequency limit of the real part of the resistivity. The imaginary part of the resistivity for nonmagnetic materials is a measure of the material's capacitive properties.

As shown above, all of the related functions can be derived from the impedance. Interpretation of a particular property for a particular type of material, however, may be facilitated by choosing one of the related impedance functions.

Impedance spectroscopy measurement

The general approach in IS is to apply a known voltage/or current to a material and measure the resulting current or voltage. Three different types of electrical stimuli are used in IS and these are summarized here [12].

In transient measurements, a fixed voltage of magnitude V_O is applied to a material and the resulting current is recorded as a function of time i(t) (see Figure 2.4). The ratio $v_O/i(t)$ is then Fourier transformed from the time domain to the frequency domain to get the frequency dependent impedance. The disadvantages of this process include (1) the need to Fourier analyze the results, (2) the data acquisition rate has to be greater than the inverse of the relaxation time for the various processes within the material, (3) the frequency spectrum is not directly controlled and (4) electrode to sample contact problems may occur due to blocking electrodes.

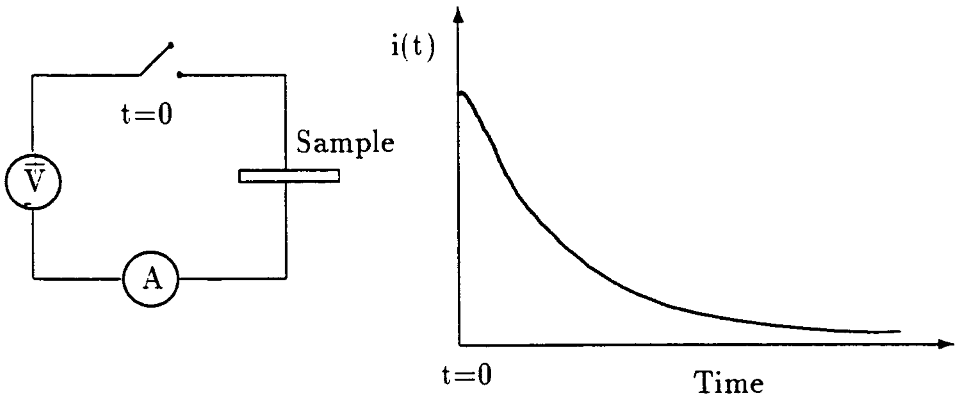


Figure 2.4: Transient measurement

Another technique in IS is to take the ratio of an applied random (white) noise voltage signal v(t) to the resulting current. The ratio is then Fourier transformed to the frequency domain to yield the impedance. The Fourier transformation of random signals is cumbersome and computationally difficult. Though this is a relatively straight forward experiment, it is hampered also by the problems mentioned above for the transient measurements.

The approach used by most impedance spectroscopists today is to measure the impedance directly in the frequency domain. A constant amplitude sinusoidal voltage is applied at one frequency and the phase shift and the amplitude of the current is measured at that frequency. The ratio of the voltage phasor to the current phasor is the frequency dependent impedance. Today, impedance analyzers using microprocessors provide a comprehensive range of impedance and frequency response measuring capabilities. Instruments are commercially available with frequency ranges from 10 μHz to 40 MHz with a resolution up to fifty thousand measurements over the range. Computer automation of these instruments is made possible through parallel and serial communication ports. In general, the advantages of these instruments are their wide impedance range, their choice of many different measurement modes, their ease of use and their wide frequency range.

The characterization procedure of a material using IS is to fit the experimental data to an equivalent circuit using ideal lumped circuit elements. The choice of a circuit, however, is not unique, because an infinite number of different circuits all having the same impedance response can be used to fit the experimental data. A plausible physical model from theory or practice is therefore used to determined the best equivalent circuit for the material. The experimental data are curve fitted to the circuit, and the values for circuit elements are calculated using either linear regression, complex linear or nonlinear least-squares fitting. The frequency response of the ideal circuit is calculated and compared to the experimental data. The magnitudes of the various circuit elements are adjusted until either a good match to the experimental data is found or the circuit is abandoned for another in which a better fit can be achieved. One major disadvantage of this technique is that the processes contributing to the impedance within a material are distributed over space and their microscopic properties may be also distributed. The sum of each process therefore may not be described by a lump circuit element. In such cases, the experimental data may be approximated to an equivalent circuit where a resistor implies a conductive path for a species and a capacitor implies polarization of some species. A system is characterized once an equivalent circuit is found which is able to predict the impedance of a material, especially under different physical conditions.

Analysis of Impedance Spectra

According to dielectric theory, IS measures two fundamental electrical characteristics of a material - the relative dielectric constant and the real conductance. The relative dielectric constant measures the polarization or capacitive properties and the conductance measures the resistive properties of a material. To illustrate this, a treatment of an ideal lossy dielectric material exhibiting a single relaxation phenomenon is presented first. This is then followed by the treatment of a real lossy dielectric, the example used being that of a fast ion conducting (FIC) glass.

Ideal lossy dielectric

In a lossy material, both conduction and polarization of species occur, and these cause ohmic loss and energy storage, respectively. A material having these properties can be represented by a parallel resistor-capacitor (RC) circuit where the ohmic losses are due to a single component resistor and the polarization is due to a capacitor. The impedance of a such a circuit is given by

$$Z^* = \frac{R}{1+(\omega RC)^2} - j\frac{\omega R^2 C}{1+(\omega RC)^2} \tag{2.10}$$

The relaxation time τ for this circuit is the product of the resistance and the capacitance, $\tau = RC$; the relaxation frequency is the inverse of the relaxation time. At frequencies less than the relaxation frequency, the impedance of the capacitor ($1/\omega$C) is large, therefore the current passes mainly through the resistor. At frequencies greater than the relaxation frequency, the impedance of the capacitor becomes small and the current passes mainly through the capacitor. The frequency at which the impedance of the capacitor and the resistor are equal is just the relaxation frequency. The complete relaxation phenomena will only be observed in an experiment if the frequency range of the experiment spans frequencies both lower and higher than the relaxation frequency.

The complex plane plot of impedance for a typical RC circuit is given in Figure 2.5 which appears as a semicircle with diameter R and center located at (R/2,0) in the x-y plane. The frequency is a parameter in this plot and increases in the direction of the arrow, from the right side of the plot to the left. The magnitude of the resistance of the resistor may be determined by the intersection of the impedance arc at the low frequency (right) side of the semicircle with the x axis. As frequency increases,

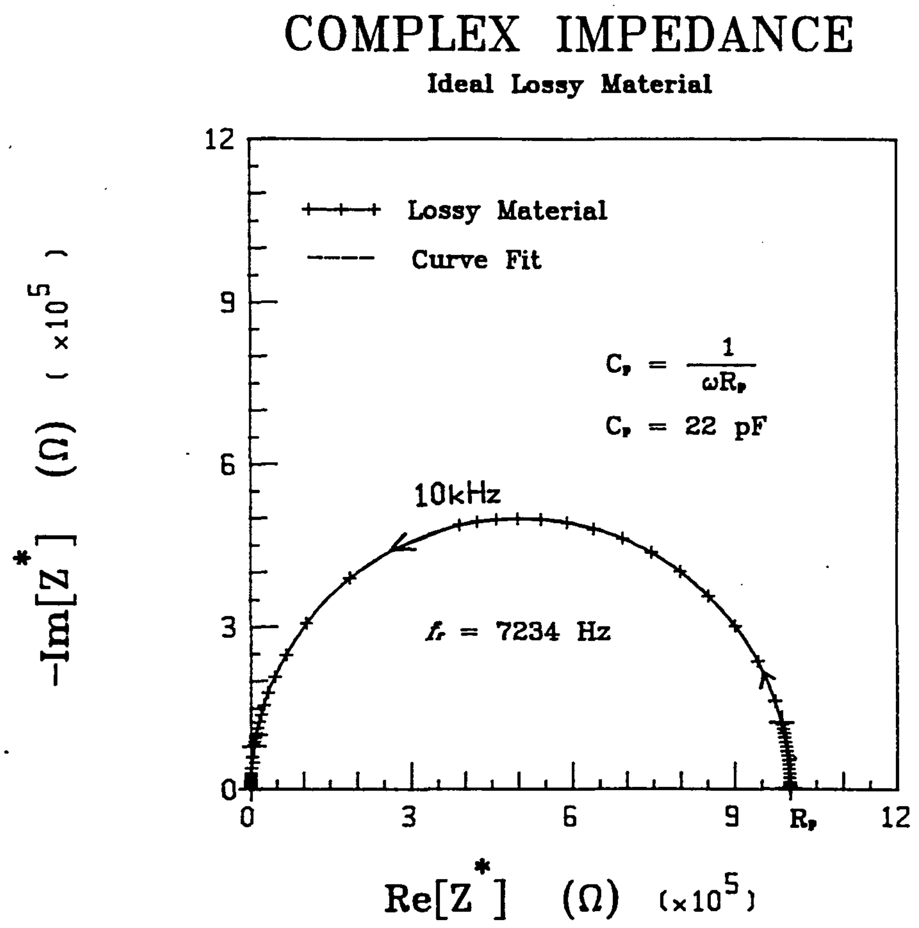


Figure 2.5: Complex impedance plane plot of lossy material

the magnitude of the imaginary impedance goes through a maximum value at the relaxation frequency. Once the relaxation frequency and the resistance are known, the capacitance can be determined using the relaxation frequency equation

$$C = \frac{1}{\omega R}. \tag{2.11}$$

At high frequencies, both the real and imaginary part of the impedance go to zero. For ideal lossy materials, the resistance and capacitance are easily determined from the impedance plot. The value for the conductivity and relative dielectric constant can be calculated, if the cell constant k_O is known, from Equation 2.8 and Equation 2.6, respectively.

Real lossy dielectric

Solid electrolytes fall into the class of lossy dielectrics. The relative dielectric constant for most electrolytes lies between 5 and 30 [15]. The resistive contribution arises from the thermally activated jumping of ions over potential energy barriers in the solid electrolyte. The first approximation for an equivalent circuit for a solid electrolyte is, therefore, a parallel RC circuit.

A complex impedance plot for a typical glassy solid electrolyte, silver phosphate ($AgPO_3$), where silver is the mobile ion, is presented to characterize the behavior of lossy dielectric materials (see Figure 2.6). There are two typical features which are observed in impedance plane plots for glassy solid electrolytes. (1) The center of the circle is usually below the x axis (arcs of this type are called depressed semicircles and arise because the conductivity relaxation occurs from mobile ions which are distributed throughout the material in unequal chemical sites); and (2) because of the presence of other relaxations such as grain boundaries, contact polarization and surface adlayers, additional or even overlapping arcs may occur in the complex plane.

The center of the semicircle for the $AgPO_3$ glass is, as is expected below the x axis. A least-square fit routine for the equation of a circle is used to locate the center and calculate the resistance of the glass. The capacitance is dispersive due to relaxation of each mobile ion at a different frequency. The solid line semicircle in

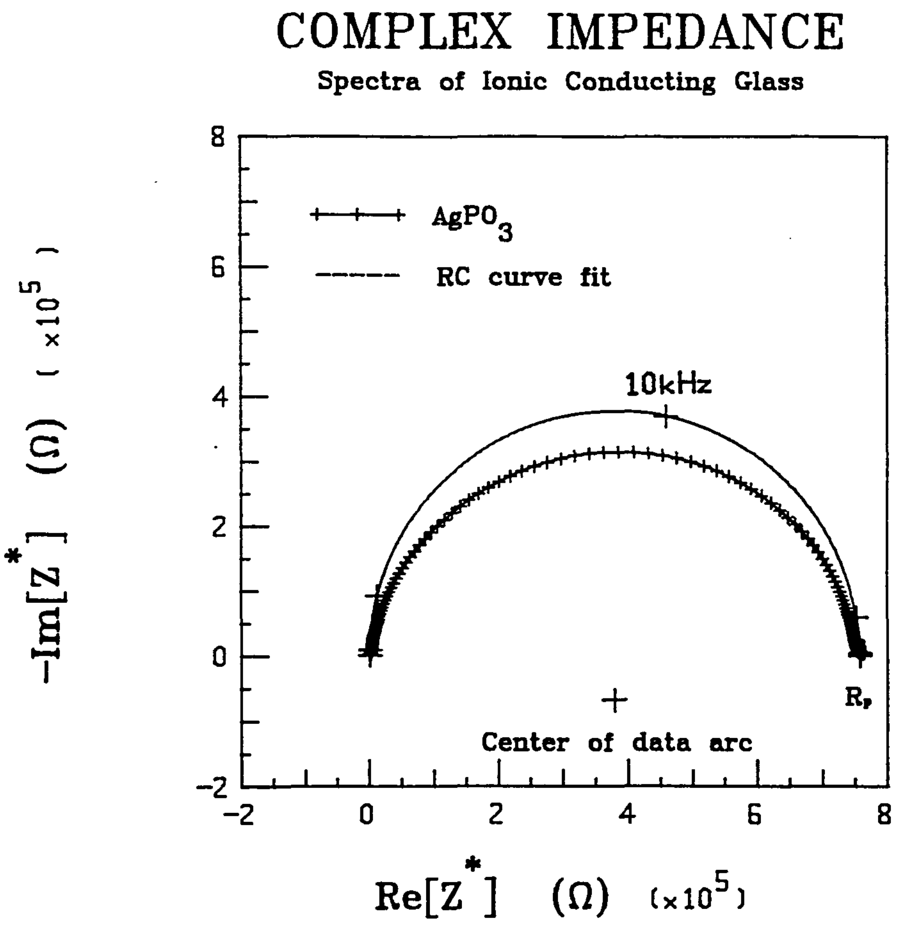


Figure 2.6: Impedance spectra of a ionic conducting glass

Figure 2.6 is the curve fit of the data if the capacitance is determined using the parallel RC model - the difference between the ideal behavior and the glass is dramatic.

The relative dielectric constant is a function of frequency, temperature and polarizability of the glass (see Appendix B). At high temperature and low frequencies, where the mobile ions in the glass are capable of contributing to the polarizability of the glass, the relative dielectric constant reaches quite large values. As the temperature is lowered, the thermally activated conduction of the mobile ions drops off dramatically, their contribution to the polarization decreases, and the relative dielectric constant decreases. At highest frequency and lowest temperature, the mobile ion contribution to the relative dielectric constant has been effectively frozen out, and all that remains is the contribution from the instantaneous atomic polarization of ions in the glass. This is the relative dielectric constant usually reported for glasses. Plots of the relative dielectric constant at different temperatures therefore should all level off to the same value at high frequencies (see Figure 2.7).

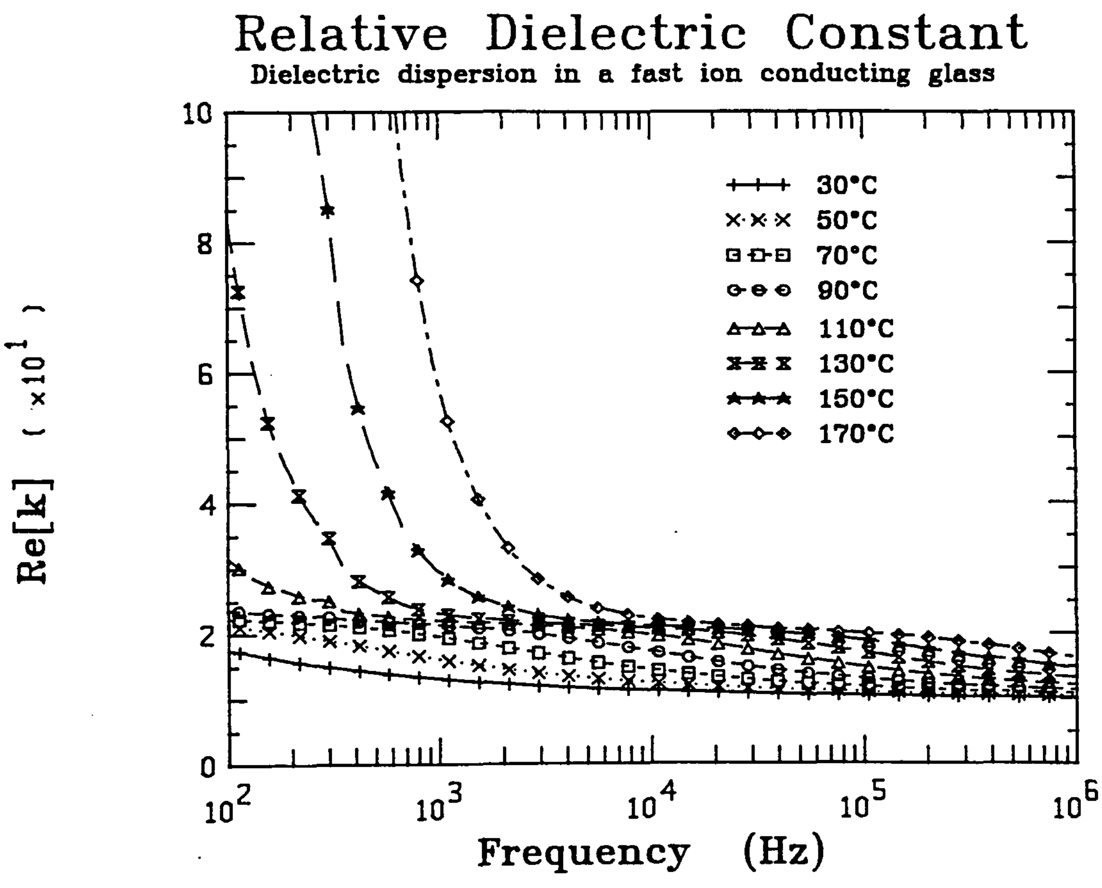


Figure 2.7: Dispersive behavior of the relative dielectric constant

The activation energy for dc conductivity for glass is determined by plotting the dc conductivity measured at different temperatures versus the inverse of the temperature. The data are displayed in this manner because the conductivity has an Arrhenius dependence on temperature

$$\sigma = \sigma_o e^{\frac{-E_a}{kT}} \tag{2.12}$$

where σ_o is the limiting conductivity, E_a is the activation energy, k is the Boltzman constant and T is the temperature. An Arrhenius plot for the $AgPO_3$ glass is shown in Figure 2.8, where the activation energy has been determined from the slope of the least-squares fit line and the limiting conductivity from the y intercept of the line.

The above discussion for characterizing the electrical properties of glass using IS describes the major capabilities of the technique, but it is, however, far from complete. For a more complete discussion of different techniques for analyzing impedance spectra data, the reader is referred to the book by MacDonald [12]. Much mathematical modeling of dispersive behavior using models such as Debye and Williams-Watts functions and Dyre's uniform distribution law have been studied [4, 12, 16, 17].

The purpose of this chapter was to provide the reader with a background to understand and appreciate impedance spectroscopy measurements and to show the motivation for the construction of an IS workstation.

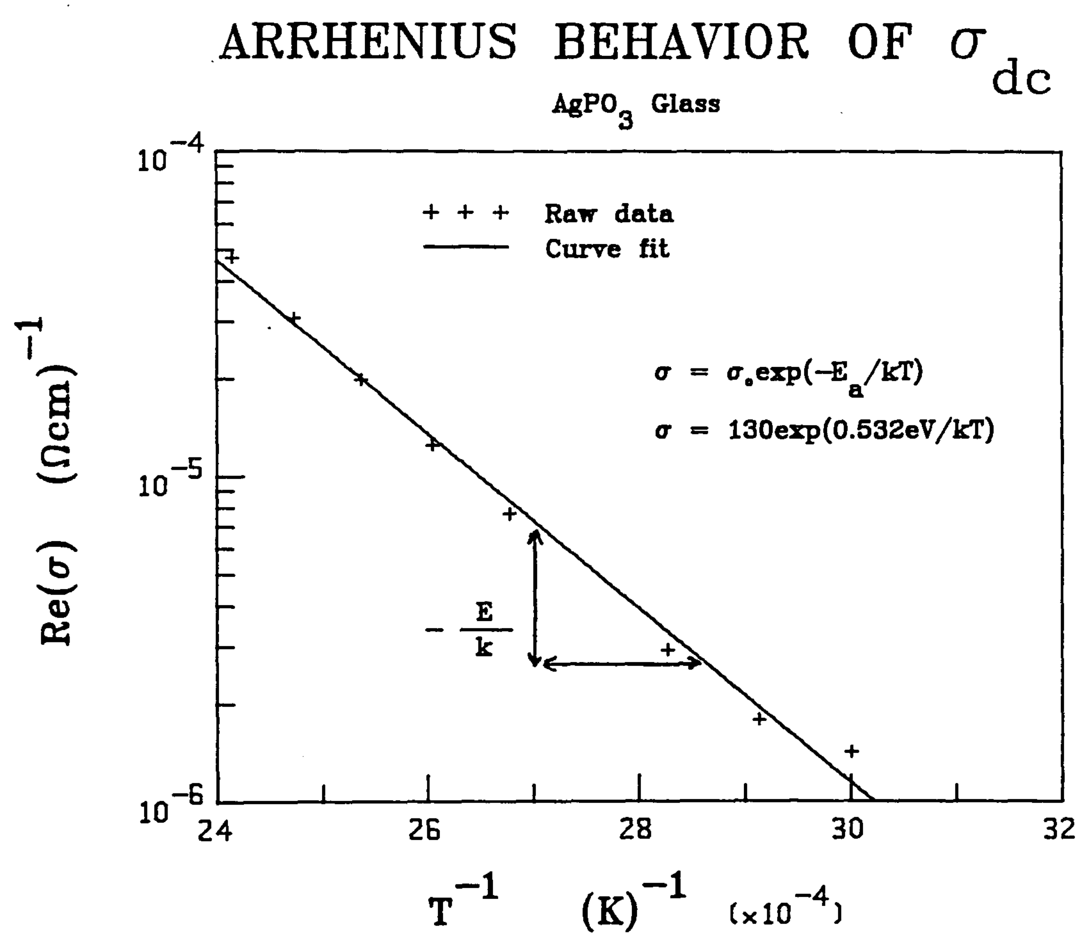


Figure 2.8: Thermally activated conductivity exhibiting Arrhenius behavior

CHAPTER 3. IMPEDANCE WORKSTATION

Automated multifrequency instrumentation for impedance spectroscopy was reported as early as 1955 by Weingarten [18]. Since then, systems based on frequency or network analyzers for impedance measurements have been reported by Morse in 1974 [19], and later by Klein and Ploof [20], Engstrom and Wang [21], Staudt and Schon [22], and Boukamp [23]. Much of the work reported there emphasized new instrumentation and techniques in measuring impedance. Today this problem has been largely overcome with the availability of conventional wide frequency and impedance range impedance analyzers from companies such as DuPont, Hewlett-Packard, Schlumberger Technologies and Tektronix. Microcomputers are now much faster, more versatile and have higher data storage capabilities compared to those used in the past. Interfacing between instruments has been simplified through standardized serial and/or parallel communication ports. High level computer languages and commercially available software are also now available that have greatly simplified the software programming of the computer interfaced instrumentation. With these advantages in mind, it is anticipated that the workstation described in this thesis will have substantially more power and capability than impedance spectroscopy workstations reported in the past and presently available.

In this chapter, an impedance spectroscopy workstation for studying fast ion conducting glasses is described. The workstation is designed and built to achieve the following performance criteria: (1) it must have a frequency range of at least 1 Hz to 1 MHz, (2) a temperature range of at least 100 K to 600 K, (3) the sample under study must be protected from mechanical abuse and environmental contamination and reaction during measurement, (4) the software must be able to operate the complete experiment unattended once the experimental conditions are set, (5) after measurements are made, the data must be stored in a systematic manner for later identification and retrieval, (6) the software must be user-friendly during experiment setup and data analysis, (7) the data analysis software must provide high level plotting and analysis functions and (8) data output after analysis to a data file, plotter or printer must be available.

Automation of Instrumentation

The automation of the workstation can be divided into the assembling and programming of the equipment. The assembling includes the selection of instrumentation, the design of the conductivity cell and the furnace, and the calibration of temperature controller and thermocouples. The programming encompasses the development of a user friendly interface for configuring the instruments, setting up the experimental conditions, controlling the entire IS experiment, and storing the impedance data.

Instrumentation hardware

The design for the impedance workstation is based on the automated impedance measuring system reported by S. Martin [6]. The designs of other reported impedance workstations do vary [19, 20, 21, 22, 23], but, the general concept of the setup is similar. A block diagram of the impedance measuring facility is shown in Figure 3.1. To meet the objectives of both ease of programming and flexibility in the system, the general purpose interface bus (GPIB) which conforms to the IEEE 488-1978 standard is used to link the instruments. Each component in the block diagram was commercially bought except for the conductivity cell and the furnace. A general description of each instrument and their capabilities is given next. A complete explanation of the design, materials and implementation of the conductivity cell and the furnace is emphasized.

Impedance analyzer The quality of the impedance analyzer determines the quality of the measured data in an IS experiment. The Solartron 1260 Impedance Gain-Phase Analyzer made by Schlumberger Technologies was selected after a very thorough evaluation of other commercial products. The Hewlett-Packard 4192A was the most easy to use but was not selected due to the wider frequency capability 10 μHz to 32 MHz compared to 100 Hz to 15 MHz and measurement resolution of fifty thousand points of the Solartron. The HP4192A had a similar impedance range (10 mΩ to 100 MΩ) and was much faster and just as accurate and precise during measurements as the Solartron 1260. The wide frequency range of the Solartron 1260 compared to the HP4194A is dramatically shown in Figure 3.2 where the real and

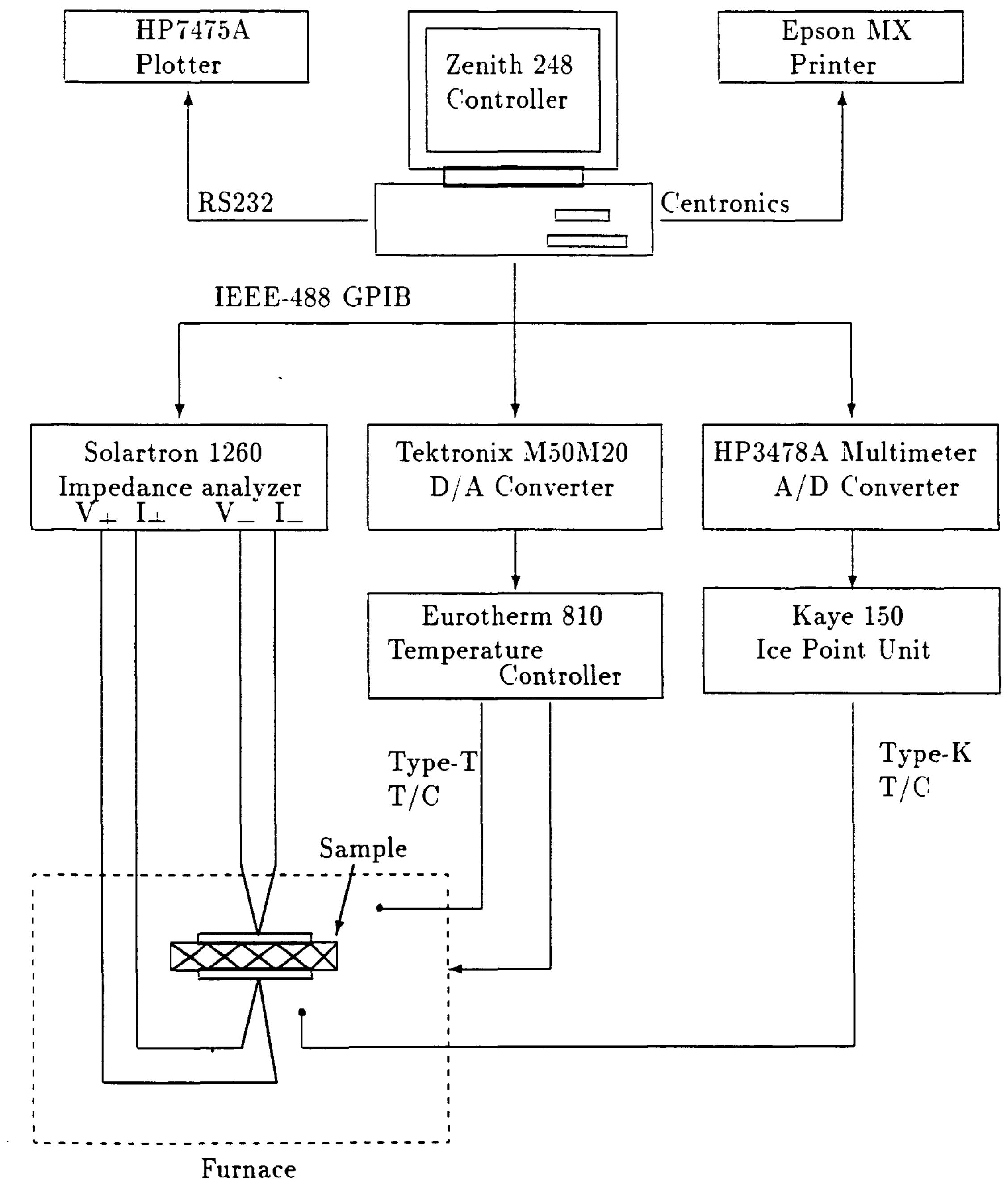
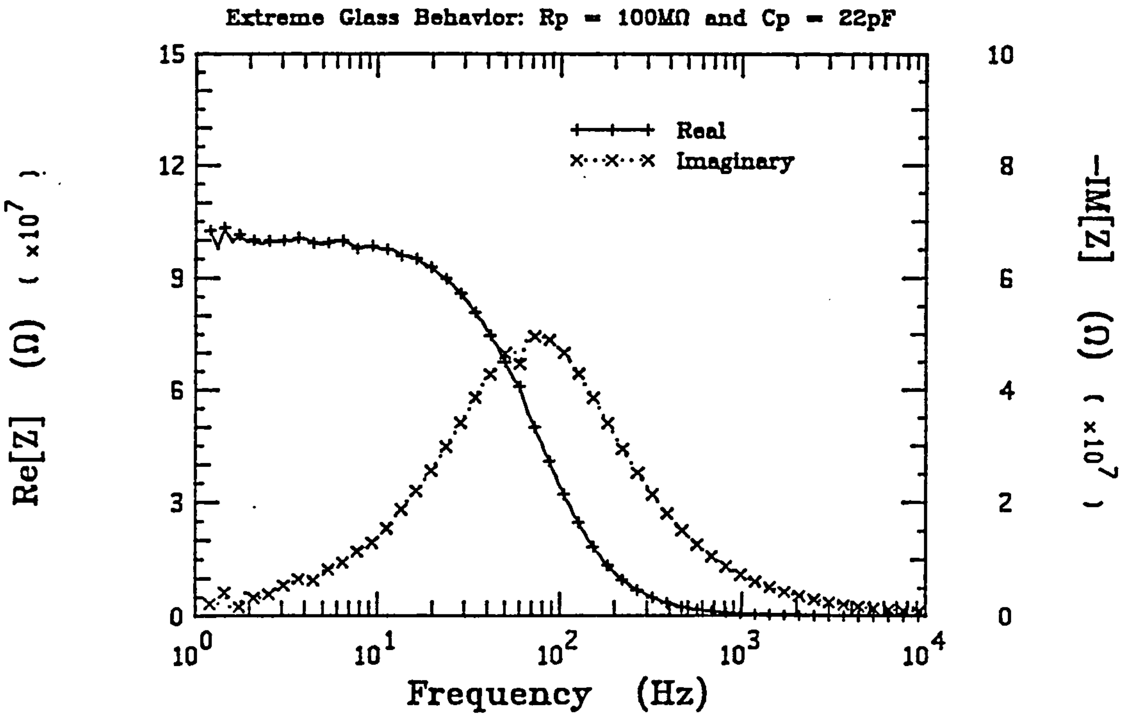


Figure 3.1: Block diagram of impedance spectroscopy workstation



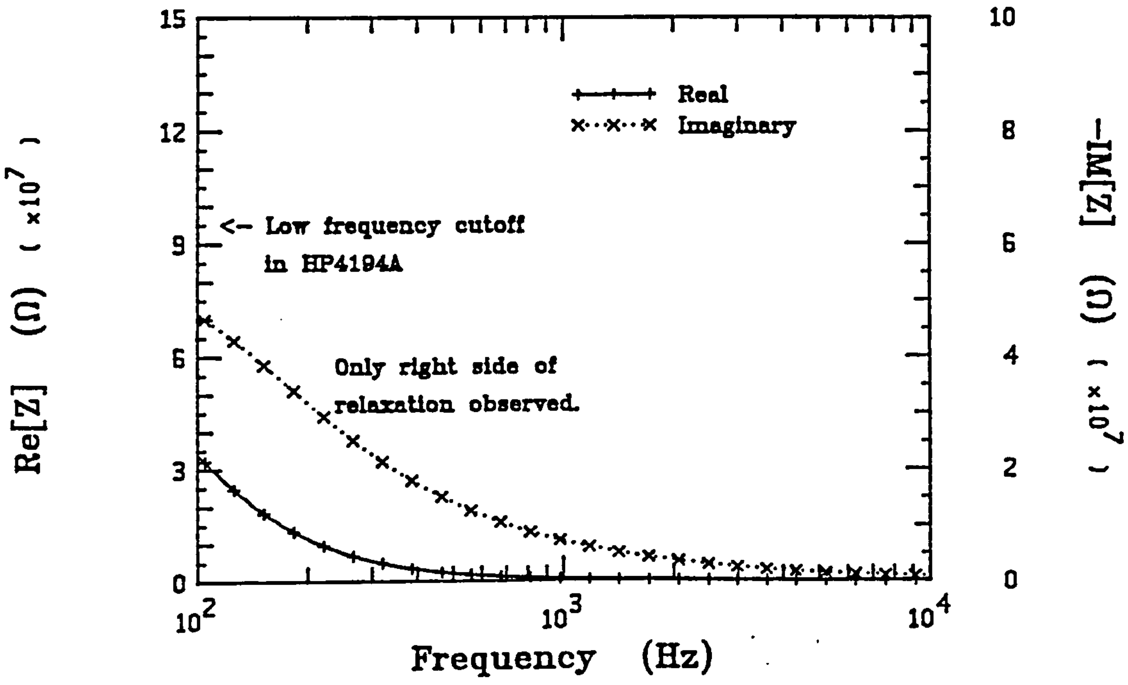


Figure 3.2: Parallel RC response of 100MΩ resistor and 22pF capacitor using (top) Solartron and (bottom) HP4194A

imaginary parts of the impedance of a parallel RC circuit with a resistance of 100 MΩ and a capacitance of 20 pF are shown. The entire relaxation spectrum of this circuit is visible using the Solartron 1260, whereas only the high frequency side of the spectrum is seen using the HP4194A. Since the above circuit elements represent the extreme of behavior that is expected for the glasses to be studied with this workstation, this test is viewed as the most critical test of an instrument's suitability for our measurements.

The Solartron is assigned to address 28 on the GPIB and is configured to both talk and listen to the computer. A complete list of commands is available to set up the Solartron and these commands can be easily sent through the GPIB from the controller. The list of commands for remote and local control, as well as a complete description of the analyzer's capabilities, can be found in the Solartron user's manual[1].

The instrument uses four terminals to measure the impedance; a schematic setup of the technique is shown in Figure 3.3, where two terminals are used to apply a voltage to the material under test and the other two terminals are used to measure the current passing through the sample. All lead connections in the four terminal configuration are made using shielded coaxial cables.

The four terminal configuration is used because the current measured through the sample is due to the voltage drop across the sample only and not the combination of the sample and the current leads as in two terminal measurements. At high frequencies, the outer shield conductor works as a return path for the measurement signal current. The same current therefore flows through both the inner conductor and the outer shield conductor, cancelling the magnetic field produced by the two op-

[1]Solartron 1260 Impedance and Gain-Phase Analyzer Operating Manual, 1987. Schlumberger Technologies, Farnborough, Hampshire, England.

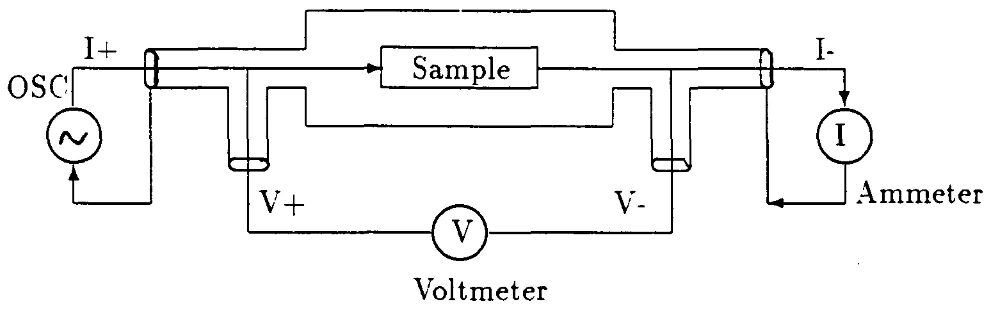


Figure 3.3: Principle of four terminal measurement

posite currents. Because minimal magnetic fields are developed, the test leads don't contribute additional measurement errors due to self or mutual inductance between the individual leads. The four terminal configuration, therefore, has a significant measuring advantage in the high frequency region by minimizing stray capacitance and residual inductance in the leads.

A limitation of this instrument is that a current must be measured for a impedance to be determined. At low frequencies, for example, the impedance of capacitors is so large that for a one volt signal applied at 100 Hz for a 2 pF capacitor, a current less than 2 nA is generated, which is below the instrument measuring limits.

The conductivity cell is connected to the impedance analyzer as shown in Figure 3.3 using shielded coaxial cables. Each cable is reserved for one of the four terminals and its length when added to the connecting wire in the cell sums to one meter. Uniformity of length will cause the same phase shift in the applied and measured signal at the point of measurement. The Solartron has a built in feature for calibrating a sample holder for line resistance and inductance and parasitic capacitance and then nulling for their effect during measurement.

Controller The controller for the workstation is an IBM compatible Zenith-248 PC AT running the Microsoft MS-DOS operating system. It has a forty megabyte hard drive, a 1.4 megabyte 3.5" floppy disk drive and a 320KByte floppy disk drive for data storage. The computer is connected to a Hewlett-Packard 7475A plotter through an asynchronous RS-232C compatible serial port and to an Epson MX printer through a parallel Centronics compatible port. GPIB capability is nonstandard to most IBM compatible PCs, and for this system is provided by a commercially available GPIB-IO card and software. The card is inserted in one of the expansion slots and is immediately ready to use with most high level computer languages. The digital to analog (D/A) and analog to digital (A/D) converters and the impedance bridge are all connected in parallel through the GPIB port to the computer.

Temperature Controller The Eurotherm 810 temperature controller is a microprocessor-based instrument with a temperature range of $\pm 250^oC$. The temperature is set through a programmable analog dc input voltage in the range of 0 to 5 volts. The relationship between the applied dc voltage V$_{applied}$ and the setpoint temperature T$_{set}$ is given by

$$T_{set} = 100V_{applied} - 250^oC. \tag{3.1}$$

Since an applied voltage is used to set the temperature controller, the following relation is given

$$V_{applied} = \frac{T_{set}}{100^oC} + 2.5Volts. \tag{3.2}$$

The temperature controller has a resolution of one degree, and to take advantage of this capability, the dc voltage source must have a resolution of 10 mV. The

Tektronix 50M20 Programmable Digital-to-Analog (D/A) Converter Card has a resolution of 5mV and an accuracy of ±10 mV and is used as the dc analog voltage source. The D/A converter and temperature controller were calibrated by comparing the applied voltage, incremented in steps of 10mV, and recording the setpoint temperature displayed on the temperature controller. The results are shown in Figure 3.4 and show that the D/A converter is indeed accurate and is in agreement with Equation 3.2.

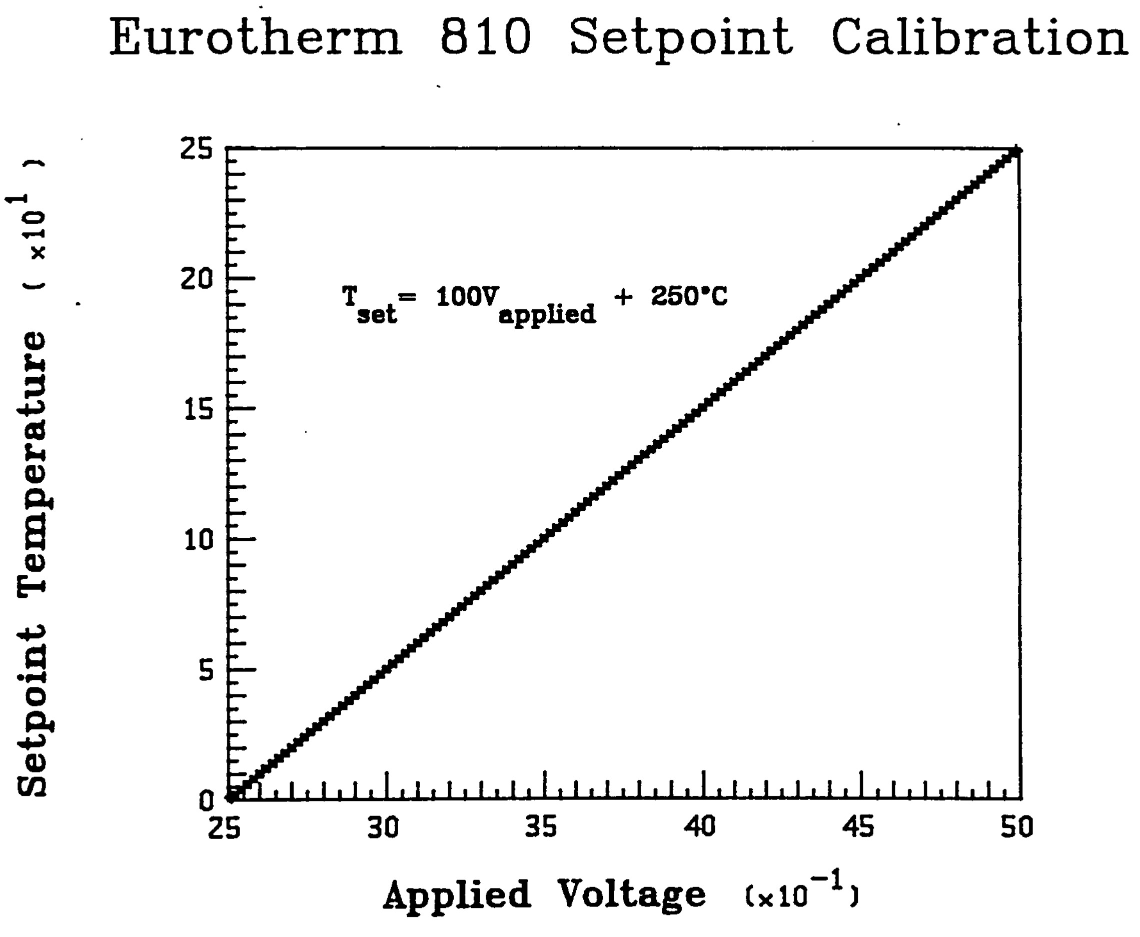


Figure 3.4: Setpoint temperature calibration of Eurotherm 810 using Tektronix 50M20 D/A converter

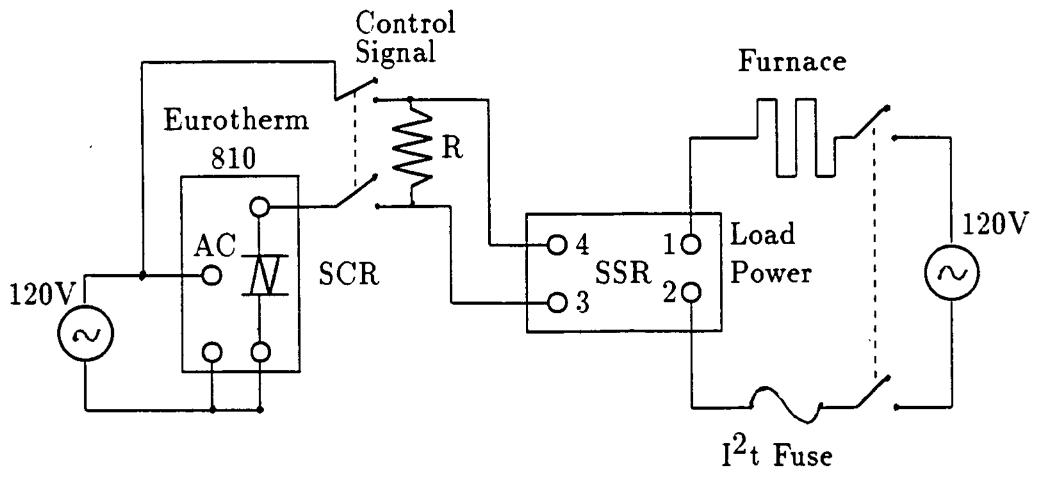


Figure 3.5: Electrical connections of temperature controller

The temperature of the furnace is measured with a Type-T thermocouple connected to the temperature controller through an internal electronic ice-point allowing the controller to calculate the temperature. The temperature is digitally displayed at all times and the setpoint can be viewed by pressing panel buttons. The calibration of the Type-T thermocouple was done at the same time as the Type-K thermocouple, and this is discussed in a separate section.

The temperature controller has an internal silicon controlled rectifier (SCR) which is used to control the on and off cycles of the furnace. An Omega solid state relay (SSR) is connected to the output of the triac to extend the temperature controller's current switching capabilities and adds a zero voltage switching and solid state reliability. The schematic of the connections of the SSR and SCR is displayed in Figure 3.5. The resistor R in Figure 3.5 serves two purposes during the on and off cycles of the SSR. During the on stage of the controller, the resistor offers a minimum load to the controllers triac to insure positive firing. During the off stage, the resistor supplies a low impedance path for the controller's leakage current. The electrical wiring of the furnace heater is given in the furnace section and is not discussed further here.

The temperature controller has three term control - proportional, integral and derivative (PID). The three parameters are proportional band, integral time constant and derivative time constant and can be set to give optimum control and steady state conditions in the furnace. Fine tuning of these parameters is described in the instruction manual[2] and allows as rapidly as possible the attainment of steady state conditions and optimal control at the set point.

D/A Converter The Tektronix 50M20 Digital to Analog (D/A) Converter Function Card is linked to the computer with the GPIB interface through the Tektronix MI5010 Multifunction Interface System. The address for the MI5010 is 23 and the D/A converter in placed in slot 2. Only two programming instructions are needed to operate the D/A converter - one to select the device and the other to set the voltage. The device is selected through the GPIB by first choosing address 23 and then sending the command "SEL 2" to choose the card in slot 2. The D/A converter is prompted to output a voltage with the string "VOLT #" where # means a number

[2]Eurotherm 810 Temperature Controller - Instruction Manual, 1986. Eurotherm Corporation, 11485 Sunset Hills Road, Reston, VA 22090.

between -10.24 and 10.235, the output range of the converter. The output voltage is then input to the temperature controller by the converter. Both the Tektronix units and the GPIB act as an interface for the computer to the temperature controller.

Multimeter A Hewlett-Packard 3478A multimeter is used as an analog to digital voltage converter. Temperature measurement within the conductivity cell is made with a Type-K thermocouple connected to an electronic ice point unit with the Seeback voltage generated being measured by the digital multimeter. The measured value is read by the computer through the GPIB interface at address 15 and converted to a temperature using a quartic equation (see thermocouple section).

Conductivity cell The design of a conductivity cell as mentioned above is sample and user specific. For the purpose of this project, fast ion conducting chalcogenide glasses will be studied using this workstation. Chalcogenide glasses decompose rapidly in air and as such are stable only in an inert gas environments. Loading the conductivity cell with the glass sample has to be done in a glovebox after which it is transported to the impedance workstation. The conductivity cell has to accommodate these needs, as well as allow impedance measurements to be made with accuracy and precision.

The cell is designed to be (1) hermetically sealed in order to protect the glass from decomposition, (2) small enough to pass through the glovebox exchange chamber, (3) easy to handle and manipulate during operations in the glovebox, (4) able to heat up or cool down rapidly, (5) able to withstand the temperature range of $\pm 250^{o}$C and (6) electrically noise free so as to not affect impedance measurements, especially at high frequencies, because of lead resistance, inductance, and parasitic capacitance.

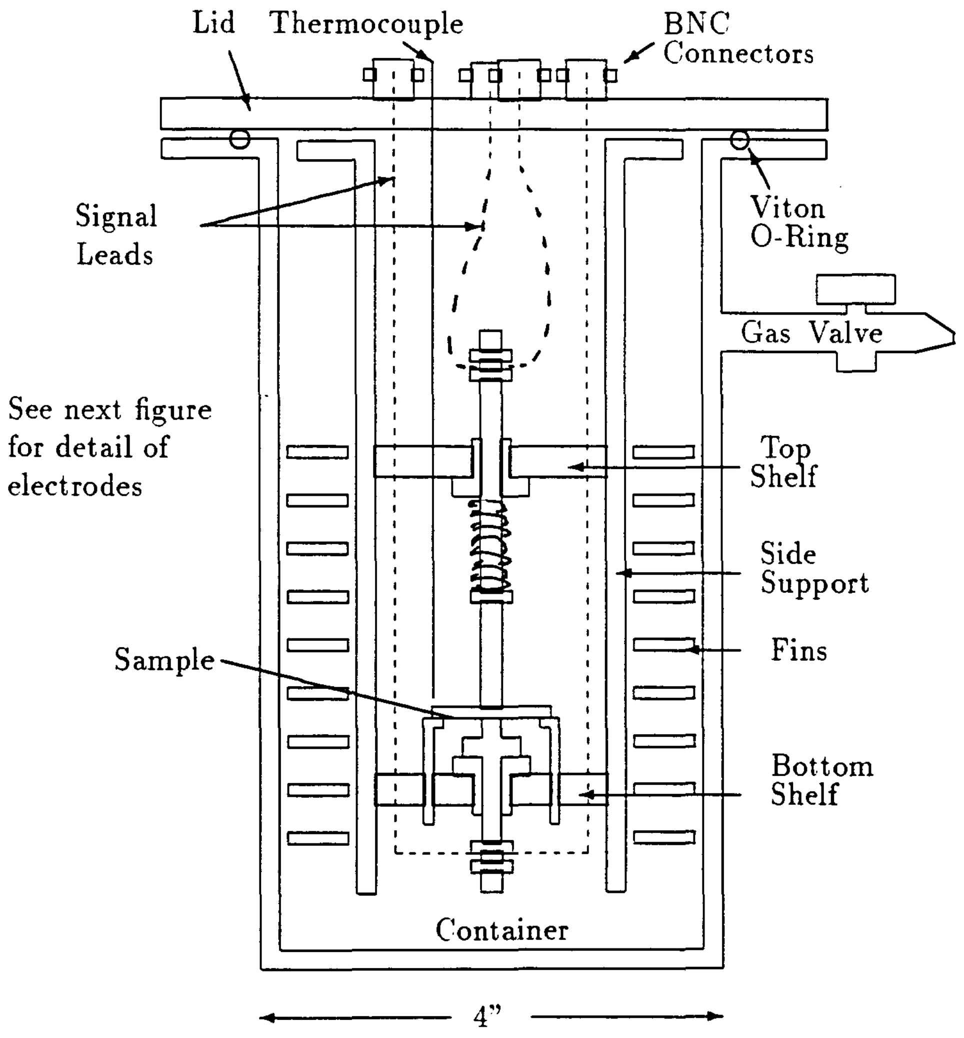


Figure 3.6: Design of conductivity cell

The discussion of the design of the cell is broken down into two parts. The physical dimensions of the cell are discussed first, followed by the electrical wiring of the cell.

The temperature range of operation of the cell is limited by the Eurotherm Temperature Controller to $\pm 250^{o}$C. The cell was, however, designed for an upper range of 500^{o}C. Brass is an appropriate material to use as it can withstand this temperature and also has a high thermal conductivity. A compact design was important, because a small cell is easy to heat and thermally control, and small lead lengths decrease measurement errors prevalent at high frequencies. The cell design was optimized to ensure maximum heating rates and yet provide thermal protection for the BNC connectors.

The support for the electrodes is made up of two brass plates (shelves) which are attached to two vertical parallel plates (side supports) and is shown in Figure 3.6. The side supports are attached with screws to the bottom side of a brass circular disk (lid), through which are made electrical BNC and thermocouple connections. This sample holder then fits into a brass cylindrical container. A Viton O-ring between the sample holder and the cylindrical container is used to form a hermetic seal. Hermetic BNC panel connectors with the O-rings replaced with Viton O-rings are attached on the top side of the lid to provide hermetic electrical connections to the inside of the cell. An Omega Type-K thermocouple with an exposed bead in an stainless steel sheath is fed through a compression fitting on the lid to provide a hermetic seal. The temperature sensor of the thermocouple is placed 2.5 mm away from the glass sample. A gas valve is attached to the upper section of the container for gas exchange. Parallel brass plates which act as fins are inserted in the lower half of the cell to increase the thermal conduction to the sample and to reduce convection currents from setting up within the cell.

Viton O-rings are used because they can withstand temperatures up to 230^{o}C.

Water cooling through copper tubing wound around the top of the lid is possible if the upper temperature limit were to increase.

After evaluating many different electrical lead designs for minimizing electrical noise while still maintaining ease of use, the design shown in Figure 3.7 was chosen. Electrical shielding from external electrical noise is provided by the brass cylindrical container of the conductivity cell. Internal interference between the measuring signals is minimized by using shielded coaxial cables.

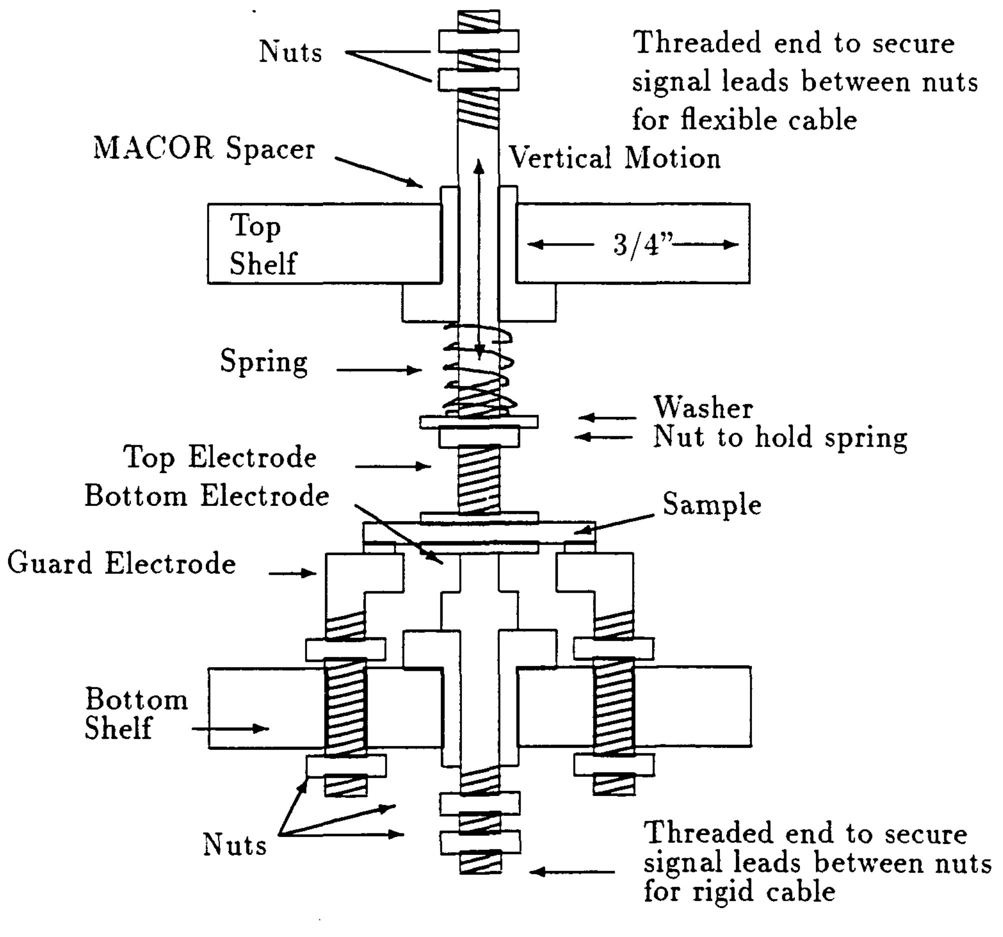


Figure 3.7: Electrode and cable connections for the conductivity cell

The electrodes in the sample holder are made out of copper with a protective gold coating. The top electrode is threaded at both ends; one end is used with a nut to hold the spring and the other end is used to secure the signal leads between two nuts. The bottom electrode is threaded only at one end for electrical connections as described above. The sample rests on the lower and immobile electrode and is held in place due to the pressure of the spring loaded top electrode. Samples as thick as half a centimeter and four centimeters in diameter may be easily loaded into the cell. These electrodes are isolated from the rest of the cell by MACOR machined spacers which have a resistivity of 10^{14} Ωcm. A guard electrode is electrically and physically connected to the lower shelf and has some mobility to move vertically and horizontally depending on the size and shape of the sample. This electrode is used to prevent conduction along the sample surface from the top electrode to the bottom electrode and vice versa; surface conduction is observed usually in high impedance materials.

The top electrode is connected to a flexible cable to allow vertical motion during sample loading. The lower electrode is immobile and is connected to a fixed cable. The advantage and reason for using a rigid cable is that it will not get kinked or bent during loading or handling. Kinks in leads can act as resistors at low frequencies and inductors at high frequencies and may contribute to measurement errors.

Commercial coaxial or shielded cables which can withstand 200^{o}C are not available and for this reason a rigid and a flexible type of high temperature ceramic insulated shielded cable was designed for the cell. The rigid shielded lead was made from platinum as the internal conductor and a silica tube as the rigid insulator. The lead is shielded using the copper wire mesh sleeve from a commercial coaxial cable. The rigid lead was shaped by heating the silica insulator to its working point. Platinum was selected over copper as the conductor for the signal carrier because of inertness. Copper, on the other hand, oxidizes easily and the conducting properties of the wire will change with time. Silica was used as the insulator because of its low dielectric constant (3.81), good electrical resistivity (4000-30000 MΩcm) and high softening point [24]. The outer conductor has the same function as before - attenuating electrical noise. Surface oxidation does not affect the outer conductor function as the skin depth for attenuating high frequency signals is orders of magnitude smaller than the thickness of the shield.

A flexible coaxial cable was designed with platinum as the central conductor, alumina beads for the flexible insulator and the same outer conductor described in the above paragraph. Alumina has a dielectric constant between 4.5 and 8.4 depending upon grade and purity and a volume resistivity greater than 10^{11} Ωcm making it a very good insulator.

The bottom electrode is connected to the low current and low voltage BNC connectors through the rigid cable while the top electrode is connected to the high current and voltage BNC connectors through the flexible cable (see Figure 3.7). In this way, the four terminals are now made into three terminals: a high (top) and low (bottom) electrode and a ground (shield) electrode. The cell connects to the outer conductors of the coaxial cable and, therefore, is also grounded.

Furnace The cell is heated in a cylindrical aluminum block using six 500 watt cartridge heaters connected in parallel and placed symmetrically around the block.

The power supply is controlled and protected through a variac with a 20A super-lag fuse using a maximum allowable voltage setting of 95VAC. The furnace design is shown in Figure 3.8. The Type-T thermocouple from the temperature controller is inserted 2.5 mm away from one of the heaters to minimize the thermal lag between the heaters and the thermocouple. Initially, the conductivity cell was water cooled and the power required to get the cell up to temperature was very high. Without water cooling, which is the case now, 800 watts is more then adequate for fast ramping to the set point.

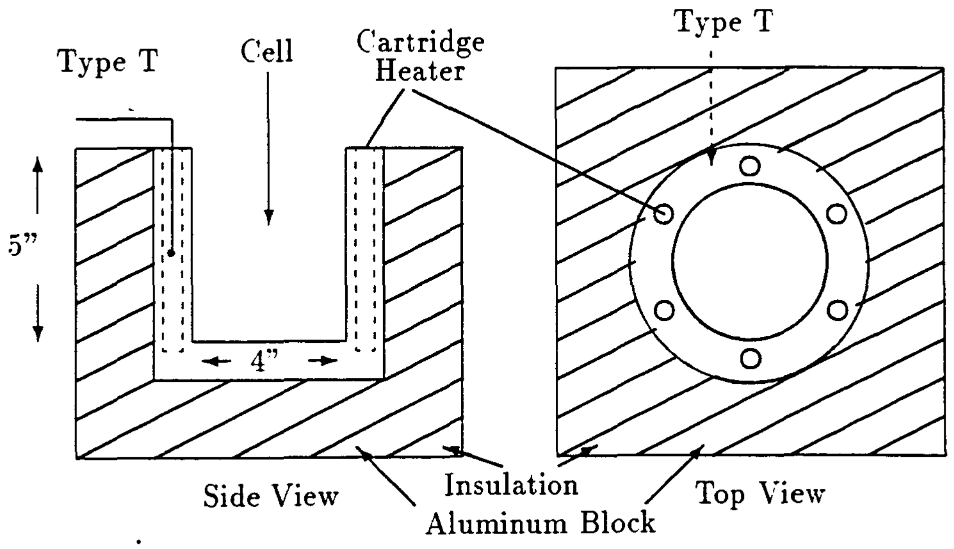


Figure 3.8: Furnace design

Sample Preparation The glass sample is a flat disc with sputtered gold circular electrodes on each side. A third electrode is sputtered on the outer edge of the glass on one side and acts as a guard electrode. A diagram of this electrode system is shown in Figure 3.9. Any surface conductivity is eliminated by providing a path to ground through the guard electrode. To calculate the cell constant for the sample, the area of the gold electrode is calculated from the diameter of the electrode and the thickness is measured using a micrometer.

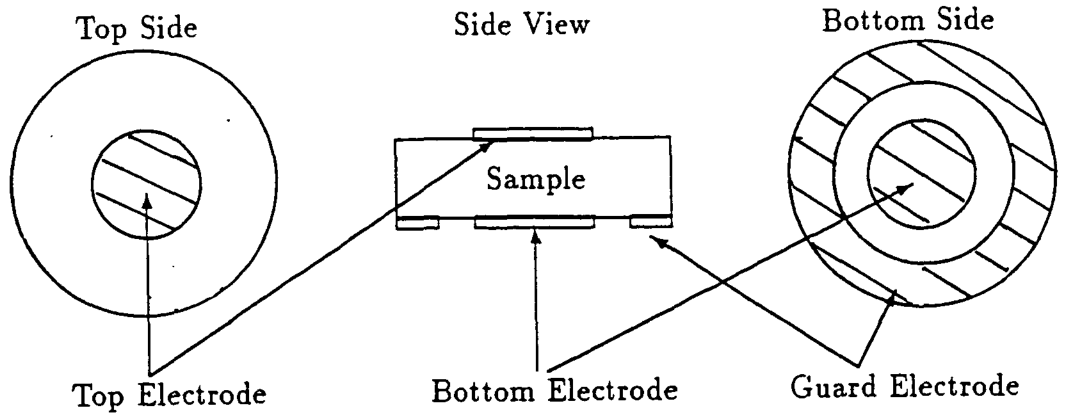


Figure 3.9: Typical electrodes used in IS experiments - parallel plate capacitor

Thermocouple calibration An Omega Type-K thermocouple is used to measure the temperature of the sample under study and an Omega Type-T thermocouple is used to measure the temperature of the furnace. Secondary calibration of the two thermocouples was done using a calibrated Type-S thermocouple from the Bureau of Standards.

Both uncalibrated thermocouples were inserted approximately one millimeter away from the Type-S thermocouple and three inches deep into a high conductivity graphite block. The block had a large enough thermal mass and high enough conductivity to assume limited thermal gradients between the thermocouples. The graphite block was inserted into the aluminum thermal block and ramped from 25-250^oC in steps of 5^oC. The Type-S and Type-K thermocouple were connected to the same ice point unit (Kaye Ice Point Reference) and then connected to a Keithley 175 and Hewlett-Packard 3478A multimeter, respectively. The voltmeter readings were input to the computer via the GPIB bus. The voltages were converted to temperature for both thermocouples using quartic equations published in the Report of Calibration for the Type-S thermocouple and in the Tektronix M41A3 instruction manual[3] for the Type-K thermocouple. The Type-T temperature was displayed on the Eurotherm 810 temperature controller and recorded manually.

The temperatures of the thermocouples were recorded when ten consecutive temperature readings spaced over five seconds differed by no more than $\pm 0.02^oC$. The results of this experiment are shown in Figure 3.10 and Figure 3.11.

Linear fits for the Type-K and Type-T results are given by

$$T_S = 0.99T_K - 0.05 \tag{3.3}$$

and

$$T_S = 0.99T_T - 0.59. \tag{3.4}$$

where the subscript represents the type of thermocouple. Both thermocouples are in close agreement with the calibrated value.

A fifth ordered polynomial with a correlation of one was curve fit to the Type-K and Type-S measurements and is given by

$$T_S = a + bT_K + cT_K^2 + dT_K^3 + eT_K^4 + fT_K^5 \tag{3.5}$$

[3]Tektronix M41A1 Low Level Amplifier M41A2-A8 Thermocouple Amplifiers, 1984. Tektronix Inc., Beaverton, Oregon.

where T_K is the measured Type-K temperature and the coefficients a, b, c, d, e and f are -1.5, 1.12, -2.419x10^{-3}, 1.665x10^{-5}, -4.644x10^{-8} and 4.346x10^{-11}, respectively. Equation 3.5 is incorporated in the programming section for temperature measurements and is used to determine the temperature of the sample.

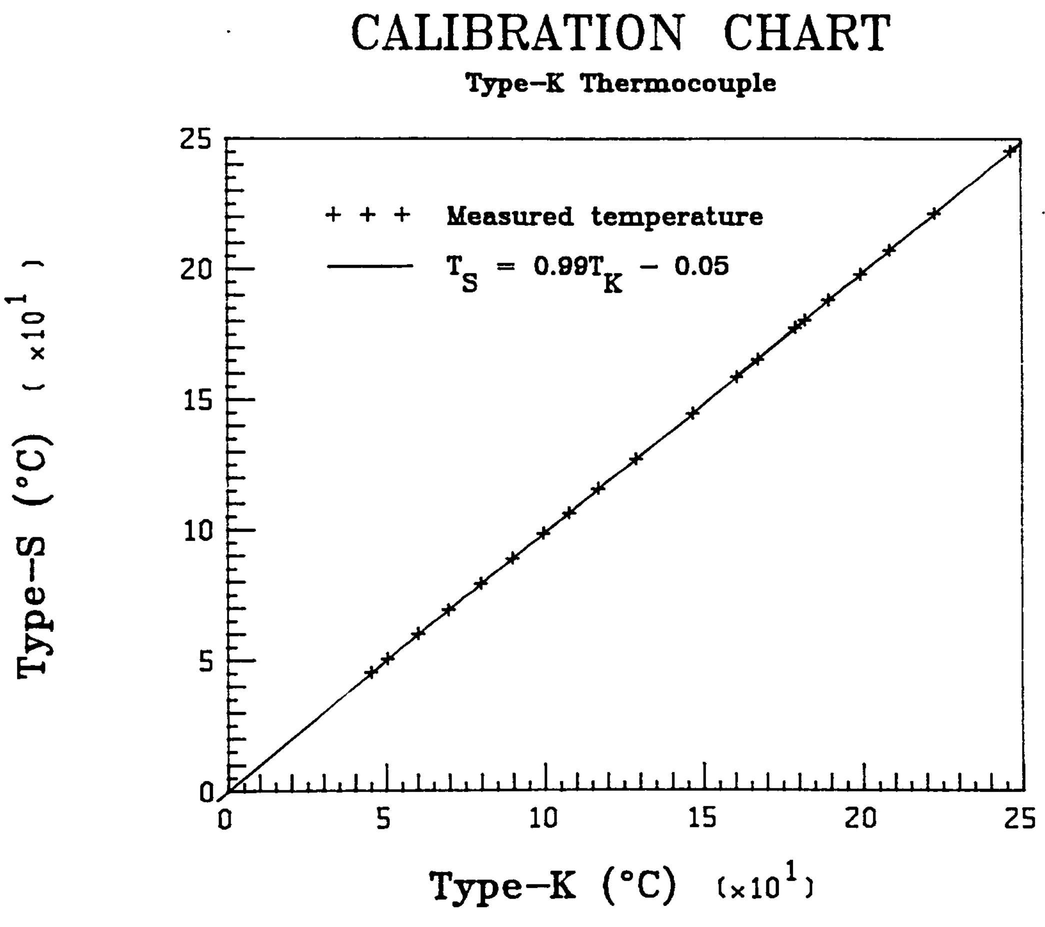


Figure 3.10: Calibration of Type-K thermocouple against Bureau of Standards Type-S thermocouple

During preliminary testing of the furnace and temperature control apparatus, temperature gradients between the Type-T thermocouple (set point) and the Type-K thermocouple (sample) were observed. A plot of the steady-state temperature between the two thermocouples is shown in Figure 3.12. The linear fitted equation

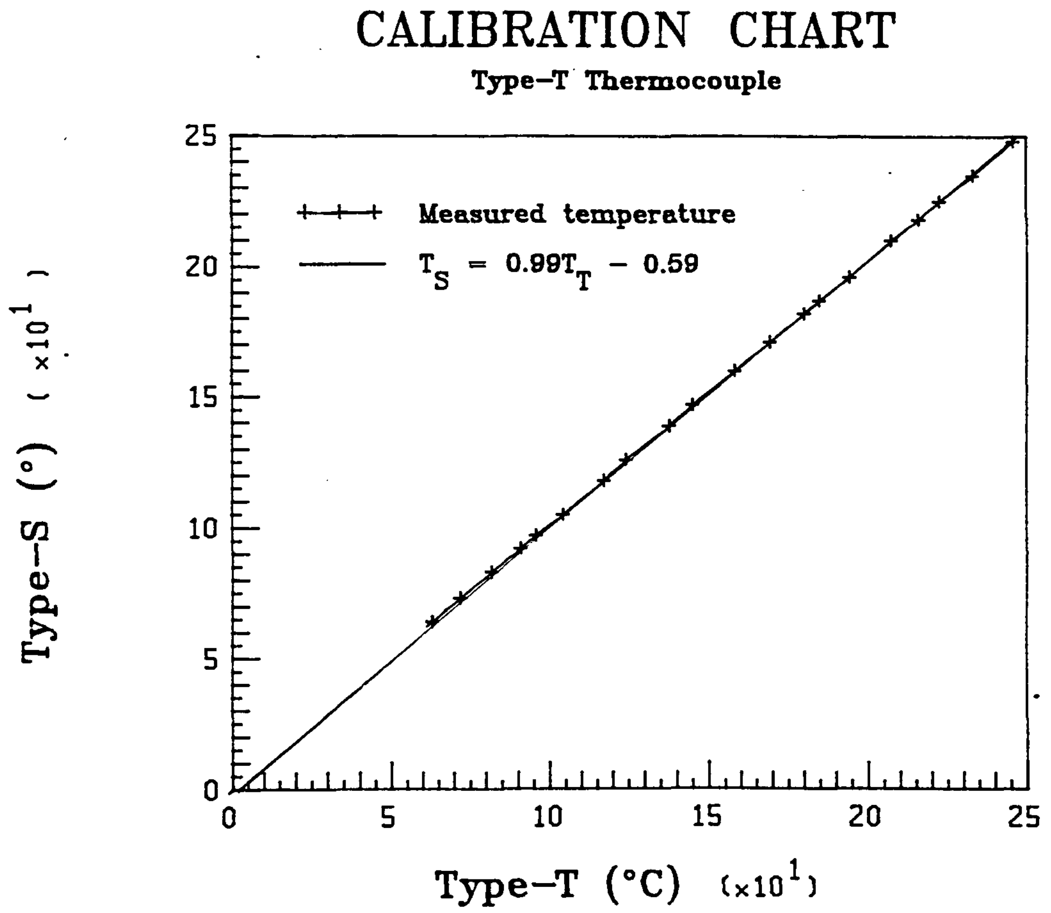


Figure 3.11: Calibration of Type-T thermocouple against Bureau of Standards Type-S thermocouple

to the data of the temperature of the thermal block and the temperature of the sample is

$$T_{setpoint} = 1.21T_{sample} - 4.67. \tag{3.6}$$

Equation 3.6 is used to estimate the applied furnace temperature necessary to raise the sample temperature to a chosen value.

Having fully described the hardware, its design and calibration, attention is next turned to the software written to both control the IS experiment as well as analyze the data.

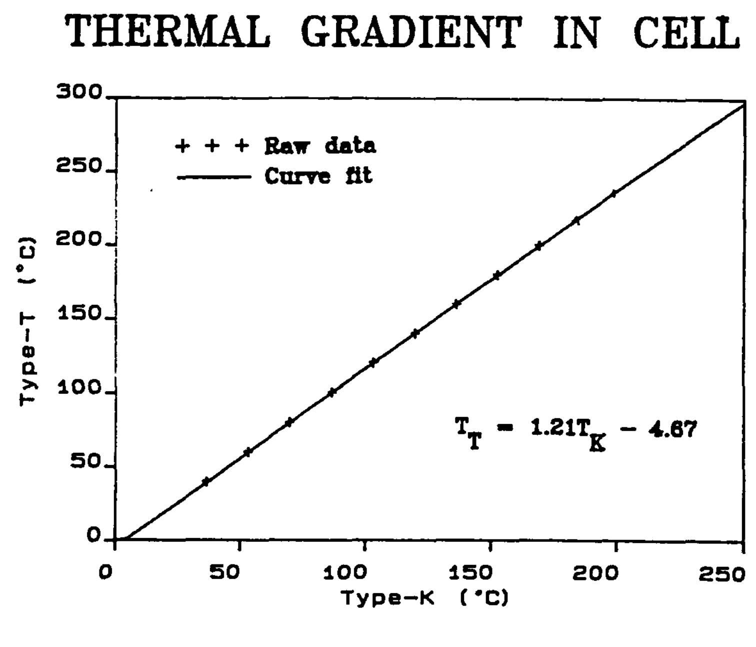


Figure 3.12: Setpoint temperature and actual temperature inside the conductivity cell

Automation Software

The computer coordinates and directs the entire system in performing the impedance spectroscopy experiment. It assigns tasks to and receives data from the various instruments and monitors the system's progress. Once data are acquired and processed, the computer is responsible for storing and/or displaying the results. The instructions (program) for the computer are written in Turbo Pascal version 5.0 and are found in Appendix D. The data-collect routine provides automated collection for the IS experiment and the data are immediately ready for analysis. A description of the software and its development in terms of controlling the system, data processing, storing data and program operation are discussed in this section.

The impedance experiment is described as a series of isothermal impedance measurements. The variables that define the impedance measuring part of the experiment are the magnitude of the applied voltage signal, the frequency range, the number of measurements during the sweep and the integration time for each spot measurement. The thermal variables are the start temperature T_{start}, final temperature T_{final} and the temperature increment T_{inc}.

System control Impedance variables are output to the Solartron through a configuration file sent by the computer or by recalling a setup with the variables already stored in one of the sixteen memory locations available on the Solartron. The thermal variables are input through the keyboard of the computer. A flow chart of the complete data collection procedure once experimental conditions are set is shown in Figure 3.13.

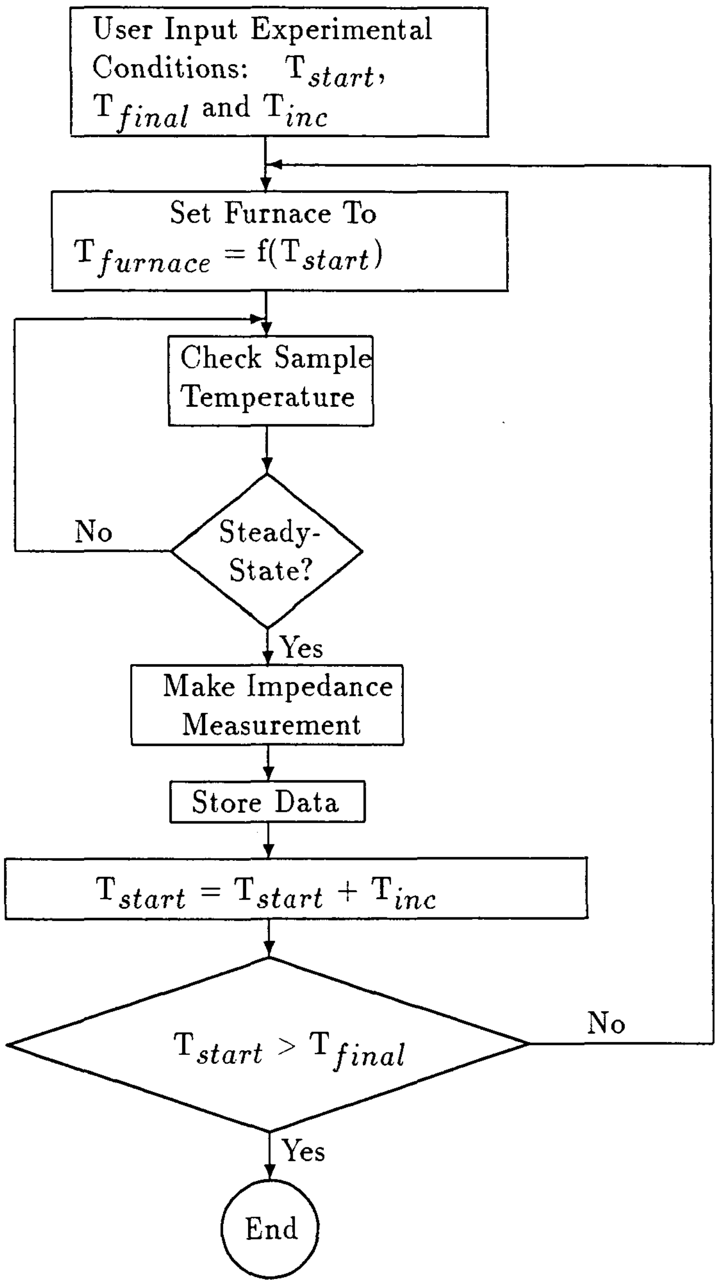

User Input Experimental Conditions: T_{start}, T_{final} and T_{inc}
Set Furnace To $T_{furnace} = f(T_{start})$
Check Sample Temperature
Steady-State?
No
Yes
Make Impedance Measurement
Store Data
$T_{start} = T_{start} + T_{inc}$
$T_{start} > T_{final}$
No
Yes
End

Figure 3.13: Flow chart of software control of the impedance experiment

The software control of this experiment using the hardware shown in Figure 3.1 is presented here. T_{start} is the initial sample temperature at which impedance measurements are to be made, however, the furnace temperature required to heat the sample to T_{start} is calculated by using Equation 3.6. The furnace set point temperature is converted to an applied voltage with the use of Equation 3.2. The D/A converter is then prompted by the computer to output the calculated voltage to set the temperature controller.

The temperature of the sample is monitored by the computer through the A/D converter connected to the Type-K thermocouple. The temperature is calculated from the quartic equation given in the Tektronix M41A3 users manual and corrected with Equation 3.5.

The computer begins checking for steady-state once the sample temperature is within three degrees of the set temperature. Steady-state, in this experiment, is defined as ten consecutive temperature measurements spaced out by N seconds that differ by no more than 0.05^{o}C from the initial measurement. N is one tenth of the total measurement time for the Solartron to sweep through the frequency range and is given by

$$N = Round(\frac{1}{10}(\# \; of \; measurements * Integration \; time)). \tag{3.7}$$

Once steady state is achieved the Solartron is prompted to make impedance measurements on the sample. In this way the deviation of the temperature during the sweep is anticipated to be no more than $\pm 0.05^{o}$C. The computer monitors the progress of the measurement through the data status byte of the Solartron. The impedance data are immediately transferred for processing from the Solartron to the computer on completion of the sweep. The Solartron are configured to always output the impedance in terms of the magnitude and the phase. The data are standardized in this manner so as to simplify the data management section of the analysis software. The raw and processed data are stored on floppy disk. The sample set point is incremented by T_{inc}, and if the new set point temperature is less than or equal to the final temperature, then the above procedure is repeated. When the set temperature exceeds T_{final}, the experiment is complete and the furnace is turned off by setting the D/A converter output voltage to zero.

Data processing Two types of data processing are required within the data collection program. The first is string manipulation of data sent and received on the GPIB bus and the other is simple to complex mathematical operations on the raw data to get the desired parameters.

Data sent from GPIB instruments contain, in addition to the numerical data of interest, string information that often contains the device address and the units of the quantity being measured. In order to store the numerical data, they must be first extracted from the data strings and this is accomplished as the data are received at the computer controller using standard string manipulation routines available in Turbo Pascal.

The other type of data manipulation in the collect program is the calculation of the dc conductivity. The dc conductivity, σ_O, is evaluated for each temperature from the raw data by converting the magnitude and phase data to real and imaginary impedance data and solving for the center and the radius of the circle. The intercept of the real axis at low frequency is determined by solving for the solution of the circle when the imaginary part is zero. The algorithm [26] is found in the source code in Appendix C and is not described here. The temperature and conductivity are then stored in a separate data file. These data files will be used for Arrhenius type plots to determine the activation energy and the high temperature limiting conductivity. By performing this data analysis at the time of collection, much time is saved in data analysis.

Data storage A sophisticated yet simple data storage system is essential for the purpose of analysis, retrieval, identification and optimizing memory use. After trying different filing systems, the one described in this section is considered to be the most effective. The following discussion assumes knowledge of the filing system in DOS as well as the rules regarding naming conventions and allowable characters for file names and directories. Additional background material can be found in the DOS manual [4].

On average, twenty isothermal impedance measurements with a density of two hundred measurements of the frequency, magnitude and phase per sweep are made during a single temperature program run. A single data file of this size would use approximately 150Kbytes and would be cumbersome to manipulate during analysis. In addition, information about the sample, the temperature during measurement and dc conductivity would not be easily accessible. An alternative strategy is therefore used where data for each frequency sweep are stored in a separate data file, identified by the temperature of the sweep.

[4]Microsoft MS-DOS Version 3.21, user's Guide, 1987. Zenith Data System Corporation, St. Joseph Michigan.

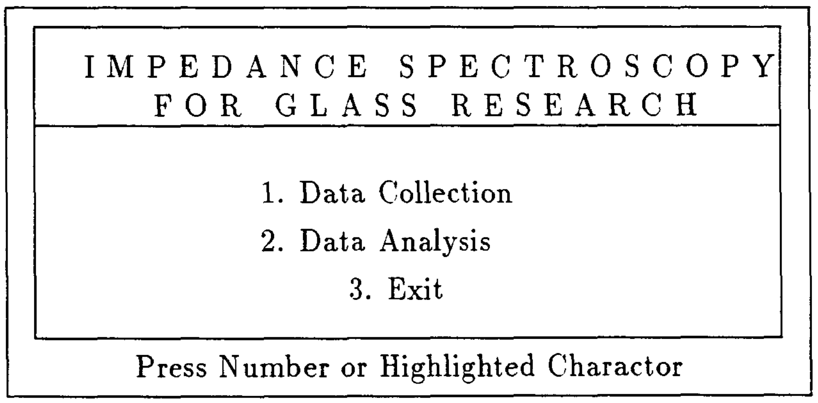
In this system, data storage is done on a high density (1.4MByte) floppy disk. Before the computer initiates a temperature program run, it first makes sure that at least 200KBytes of memory are available on the floppy disk. If not, it will prompt the user to insert a new disk until a disk with enough memory is detected.

The user is then asked to input a unique alphanumeric name with a maximum of eight characters to identify the sample. For explanation purposes, the name `GLASS1` will be used here. The computer checks the floppy disk for the directory `GLASS1`. If a directory with the same name exists, the option to overwrite or create a new directory with a different name is available. Once a unique directory is created, all files created during the experiment are stored in this directory.

The impedance data for each isothermal sweep are then stored in a file called `GLASS1.TMP` where the extension `TMP` is the set point temperature of the sample at which the sweep was conducted in degrees centigrade. The first line of the file contains the date and time, the set point temperature and the actual temperature of the sample. The impedance data are stored in three columns, one each for the frequency, impedance magnitude and the phase. The data are stored in order of increasing frequency.

The dc conductivity data are stored in a file called `GLASS.SIG` where `SIG` is short for sigma, the conductivity. This data file contains the temperature, conductivity and resistivity calculated for each isothermal sweep.

The name of the sample, the starting, final and incremental temperature, user name, date, cell constant and a one line sample description are all stored in a file called `CONTENTS.TXT`. A file called `STEADYST.ATE` contains the time of day when a system achieved steady-state for each isotherm and the maximum temperature fluctuation during the sweep. Finally, the temperature history or time versus temperature log of the complete experiment is saved under the name `TRANSIENT.RES`. All the information about the sample and the experiment is contained in these three files. All files are stored as ASCII files and are accessible to most text editors. An example of a typical directory after a data collection experiment is shown later in Figure 3.17.



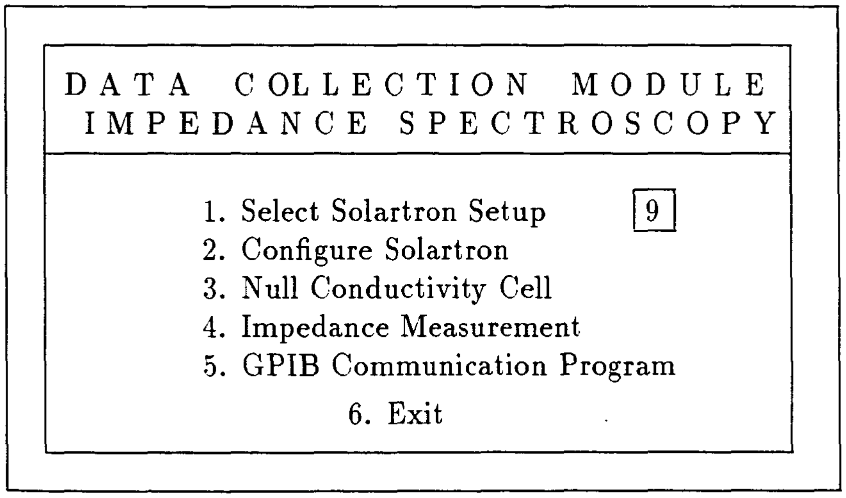
```
IMPEDANCE SPECTROSCOPY
   FOR GLASS RESEARCH

   1. Data Collection
   2. Data Analysis
        3. Exit

Press Number or Highlighted Charactor
```

Figure 3.14: Impedance spectroscopy software - Menu 1

Program operation This section is included to act as a user's guide to operation of the data-collection routine. Execution of the program `IS_WKSTN.EXE` will display a software identification screen. By pressing any key, a second screen will appear prompting the user to choose either the collection or the analysis module or exit (see Figure 3.14). Options are selected by pressing the number or the highlighted character of the selection. On choosing the data-collection option the screen will clear and then display the data-collection menu shown in Figure 3.15. In the following paragraphs, each of the options available to the user from the data-collection menu will be described.



```
DATA COLLECTION MODULE
IMPEDANCE SPECTROSCOPY

1. Select Solartron Setup      [9]
2. Configure Solartron
3. Null Conductivity Cell
4. Impedance Measurement
5. GPIB Communication Program
         6. Exit
```

Figure 3.15: Data collection menu

Select Solartron Setup: The first option gives the user the option to choose one of the nine impedance measuring setups stored in the nine memory locations on the Solartron. When selecting this option, the cursor will jump to the box for that option and wait for a number to be keyed in to select a setup. All other data-collect options are performed using the selected experimental setup here.

Configure Solartron: The second option allows the user to configure the Solartron to the specific needs of a new experiment. Selection of this option clears the screen and displays the contents of the configuration file `SOLSET.UP#` where the # mark represents the number of the currently selected memory location. Nine such configuration files exist with the same name but different last character. The displayed file can be edited and additional commands may be added as long as the following two rules are obeyed. The first ten characters of a line are reserved for output to the Solartron, the rest may be used for comments to describe the command. All new commands must occur between the clear and store setup commands. The option of saving the new configuration file or quitting without saving is available, however, the configuration file selected is stored on the Solartron and becomes the default setup.

Null Conductivity Cell: The third option allows calibration of the conductivity cell using the selected setup experimental conditions. The user is prompted to short circuit the cell for parasitic resistance and inductance evaluation. The user is then prompted to open the circuit of the conductivity cell such that measurement of any stray capacitance can be determined. On completion of this operation, the `NULL` option of the Solartron is enabled and impedance measurements using the present setup will compensate for parasitic effects. Though this is a powerful feature of the Solartron, the null calibration cannot be stored and used again when the default setup is changed and therefore the Solartron must be nulled before any data collection run.

Impedance Collection: The fourth option selects the IS experiment described above. A screen for entering the sample name, user identification, cell constant and temperature settings appears on selecting this option. Entering the same number for all the temperature settings allows a single frequency sweep to be made without a temperature program. Inputting a T_{start}, T_{final} and a T_{inc} will cause the workstation to perform as described in the last section. The input name, the date and time of logging onto the collection module are stored in a user log file called `USER.LOG`.On completion of the impedance experiment the user is logged out and returned to the main data-collect menu.

GPIB Communication: The last option is a communication program for the GPIB interface where commands can be sent to and data received from the GPIB. This option is useful when diagnosing the performance of the GPIB or of GPIB devices connected to the interface bus.

Exiting from the collection program will then display the very first menu and the option to analyze the collected or any other impedance data. The development, capabilities and operation of the analysis software is given in the next section.

Data Analysis Software

The data analysis software was developed to provide for the easy transformation of the raw data into other electrical properties in order to enhance data interpretation and to provide insight into the electrical response of the glass. Two dimensional data presentation for all impedance related functions in both the complex plane and the frequency domain is provided. In addition, a routine to evaluate and display the conduction activation energy E_a and the high temperature limiting conductivity σ_o for dc is also provide. Since in general the conductivity of a material is frequency dependent, an additional utility is provided such that the conductivity at selected frequencies can be extracted from the raw data files and a new file created in which the fixed frequency conductivity along with its corresponding temperature of measurement are stored. In this way, after all the data have been collected an Arrhenius plot of the fixed frequency conductivity can be made.

This program is written in Turbo Pascal (Version 5.0) and uses HGRAPH, a commercial package from Heartland Software[5], as its graphics driver. The analysis

[5]HGRAPH: Version 4.1 Reference Manual and User's Guide, 1987. Heartland Software, Inc., Ames, IA 50010.

package was written to be as flexible as possible with the hopes that other impedance spectroscopists may find features which are not presently available and perhaps for this reason incorporate them into their own workstation environments.

Data files that conform to the format used in this program but not obtained using the data collection program can be input to this analysis package for analysis and plotting. Publication-quality graphs of the data and results are easily user customized on the CRT and then prepared for reports by a pen plotter or a printer.

A general description of the program capabilities is given next and this is followed by a discussion of the data conversion techniques and other mathematical operations used in the program. The final section is a user's guide to the operation of the analysis software.

General Description

In order to give a global overview of the analysis software, Figure 3.16 gives the flowchart of a typical data analysis session using this package. The typical steps involved in the data analysis are

- Selection of the data file
- Choice of single or multiple data file plots
- Selection of the kind of data plot
- Display of autoscaled data plot
- Manual scaling to optimize data display
- Annotation and customizing of data legends and axis labels
- Hard copy output to printer or plotter
- Repetition on new file(s) or terminating of analysis routine

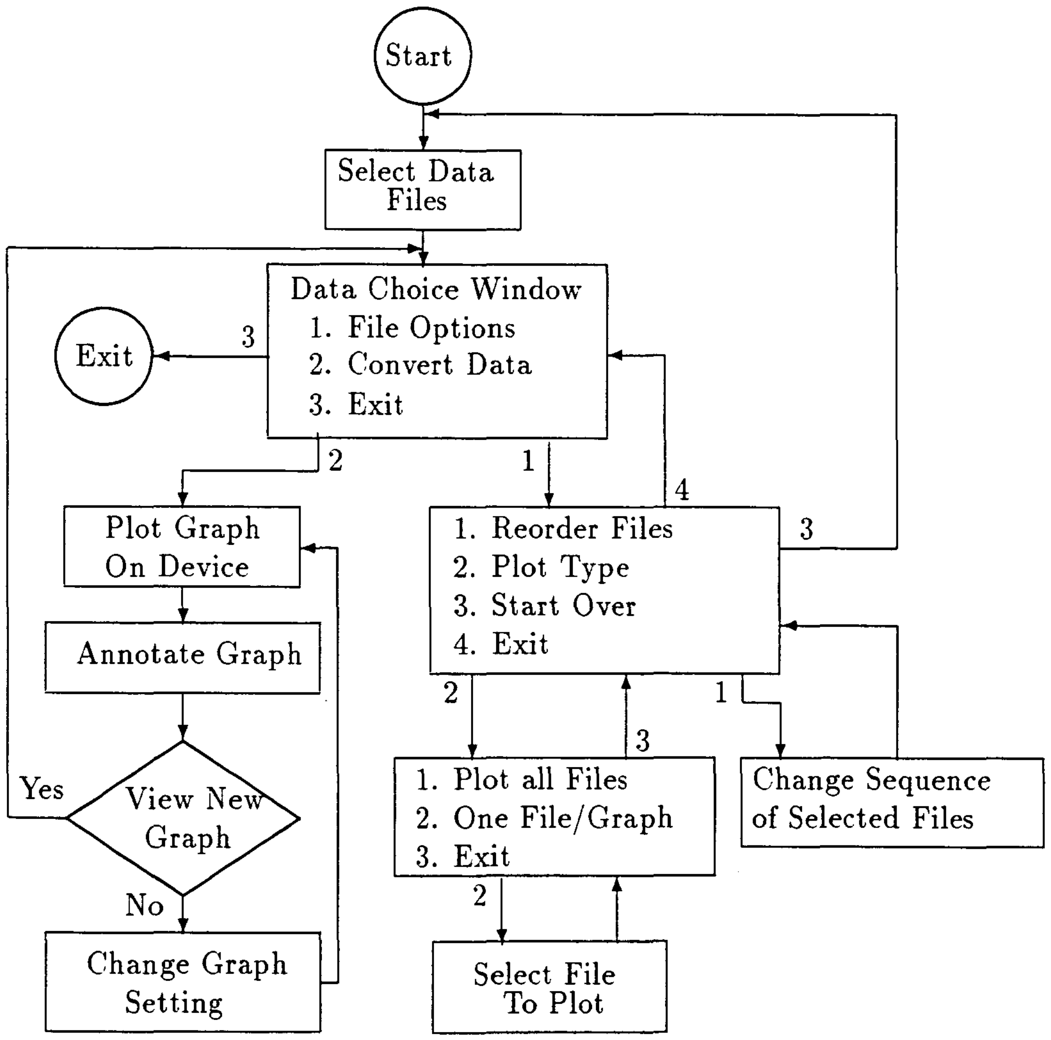


Figure 3.16: Typical data analysis session

Files to be analyzed are selected through a user friendly file manager which allows easy movement between the disk drives, directories and subdirectories. Only those files containing impedance data in terms of the frequency, magnitude and phase, or conductivity files are considered for input.

For any impedance data file, the user has up to sixteen different ways of displaying the data. Twelve of those are Z^*, A^*, M^*, ϵ^*, σ^* and ρ^* displayed either in the complex plane or in the frequency domain. Two more display options are the magnitude and phase versus log frequency and the loss tangent versus log frequency. The remaining two options are model dependent plots of the equivalent resistance and capacitance for a parallel circuit and the equivalent resistance and inductance for a series circuit both plotted versus frequency.

In all the frequency plots, the real part is plotted on the left y-axis and the imaginary part is plotted on the right y-axis. Either one of these axes may be turned on or off to display only the real part or only the imaginary part. A better view of the complex plane data can be achieved however by viewing both the real and the imaginary parts versus frequency in a three dimensional plot of the impedance function. J. MacDonald in [12, 27] has exploited this and has developed a 3-D perspective plotting package to simultaneously display the real and imaginary parts in the frequency and complex plane domains. This capability, while not part of the present software, is viewed as a powerful addition to the plotting features of the program and will be included in future revisions of the software package.

In the complex plane plots, the imaginary part is used as the y-axis and the real part is used as the x-axis. To account for the fact that many of the plots exhibit a high frequency limit where both parts go to zero, in these plots the origin is offset to negative values to allow a better visualization of the data. To make properly scaled graphs, both axes use the same scale factors and units between the tick marks. In this way, for example, the complex plane plots of the impedance come out as semicircle arcs. Without equal scaling, the arcs would be compressed or expanded, depending upon the difference in scaling between the two axes. Also for the impedance plots, the data are automatically best-fitted to the single relaxation time model (parallel RC circuit) using the best-fit parameters to draw a line through the data to show the level of either agreement or disagreement. Finally, since frequency is a parameter in the complex plane plot and is not directly plotted, frequencies are labelled at decade intervals to show the direction that frequencies change on the plot.

The Arrhenius plotting option is used to determine the conduction activation energy and the high temperature limiting conductivity. This option is only accessible to those data files with a `.SIG` extension. An error checking procedure is used both to prevent frequency files from being analyzed and to disable all options besides the Arrhenius plot for files ending with `.SIG`. The raw and the curve fit data are simultaneously plotted; the activation energy is determined from the slope and the limiting conductivity from the y-intercept of the curve fit. Standard deviations for both the activation energy and limiting conductivity are provided.

All data files plotted for the first time have default values for the titles and colors and these are defined by the program. The user name, date of data measurement, the temperature and the cell constant are extracted from the first line of the data file and from the `CONTENTS.TXT` file located in the same directory as the data file (created by the data-collect routine) and are then added to the bottom of the graph. If the CONTENTS.TXT file is not found, all variables normally extracted from this file are assigned to the value of 0 except for the cell constant which is assigned to the value of one. The user may change any of these variables with the exception of the cell constant during the course of the analysis. Help menus are available to select colors, fonts and devices to display the data.

Once a plot type is selected, the data are autoscaled to guarantee display of the data on the screen. The data range may then be changed manually or further autoscaled. The step size, the number of major and minor tick marks and the data scale factors are chosen automatically by the program. The axis types are predetermined, however, they may be changed from log to linear and vice versa.

Multiple data files may be plotted on the same graph. Axis scaling is calculated from the data for the first data file. Manual scaling is then used if the autoscaling due to the first file does not provide an optimum window to include the data from the other files. The selected files may be reordered such that the file with the widest data range is first in the list. A very common example of the use of multiple plotting is for plots of the logarithm of the conductivity versus the logarithm of the frequency. Here multiple files are plotted for the same sample with each file differing only in the temperature at which the data were calculated. Another common practice is to plot the impedance of a material at different temperatures on the same complex impedance plane plot to observe the decrease in resistance (increase of conductivity) with temperature increase.

After the graph format has been optimized for either single or multiple data file plots, the user is prompted to either annotate the graph (comment) or to input data legends for each curve plotted (legend). Each command, executed by pressing C or L, respectively, allows the user to place text information anywhere on the graph describing any relevant information the user feels appropriate.

After the final format is achieved, the user may scale the size of the graph all the way down to 0.35 of its original size and select output to either the screen, plotter, printer or a graphics data file. At this point, customized graphs can be saved for displaying or plotting at another date but they cannot be modified. This advantageous feature will be implemented in a later version of the software.

Mathematical Methods

Three types of mathematical computations are used in this program. The first is the conversion of impedance data from one form to another. The second uses iterative techniques of numerical analysis to curve fit data to a theoretical model. The third type are those which require the use of unique algorithms to solve problems within a section of the program code such as log and linear autoscaling. The first two are of interest for the spectroscopist and will be discussed here. For detail of the third type the reader is referred to the source code.

The conversion of one type of impedance data to another type requires basic knowledge of complex numbers and mathematical operations. Because most high level languages do not offer complex arithmetic, polar coordinates are used. The conversion table used to convert the raw magnitude and phase data into each of the following impedance function Z^*, A^*, M^*, ϵ^*, σ^* and ρ^* is given in Appendix C.

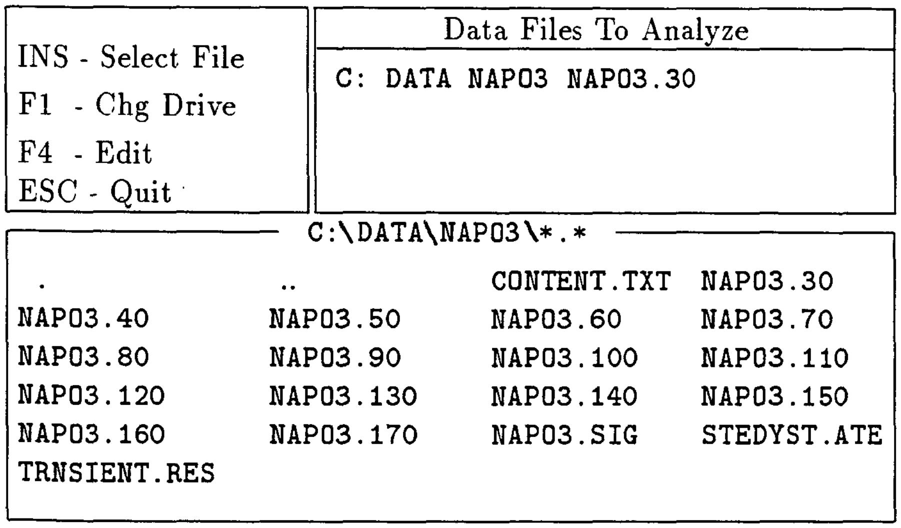
The physical and mathematical bases for these conversions are given in Appendix B.

Two iterative algorithms, one for fitting data to a straight line and the other for fitting data to a circle, are used in this program. The linear regression algorithm to solve for a straight line can be found in most numerical analysis books [25]. The algorithm to solve for the center of a circle and its radius using complex nonlinear least-squares is given by MacDonald et al. [26] who have done much statistical work on theoretical modelling of electrical impedance data to equivalent circuits. This algorithm is identical to the one used by S. Martin [6] and is incorporated in the data collection routine to calculate the dc conductivity and in the analysis program. The equations for both these algorithms are included with the impedance conversion equations in Appendix C.

User's Guide To Analysis Package

This program uses menus and keystrokes to allow movement between functions and help screens to optimize data analysis. The user is informed at each new step or keystroke of the status of the present operation and/or what is expected from the user. A first-time user should be comfortably moving through the screens and preparing quality graphs in a relatively short time. Instructions to use the program are, however, given to facilitate the operation of the routine and to allow for future optimization of the program.

The data analysis program is selected from the menu shown in Figure 3.14 during the execution of the program `IS_WKSTN.EXE`. The menu screen is replaced by a identification screen for the analysis program. On pressing the **Enter** key, the screen



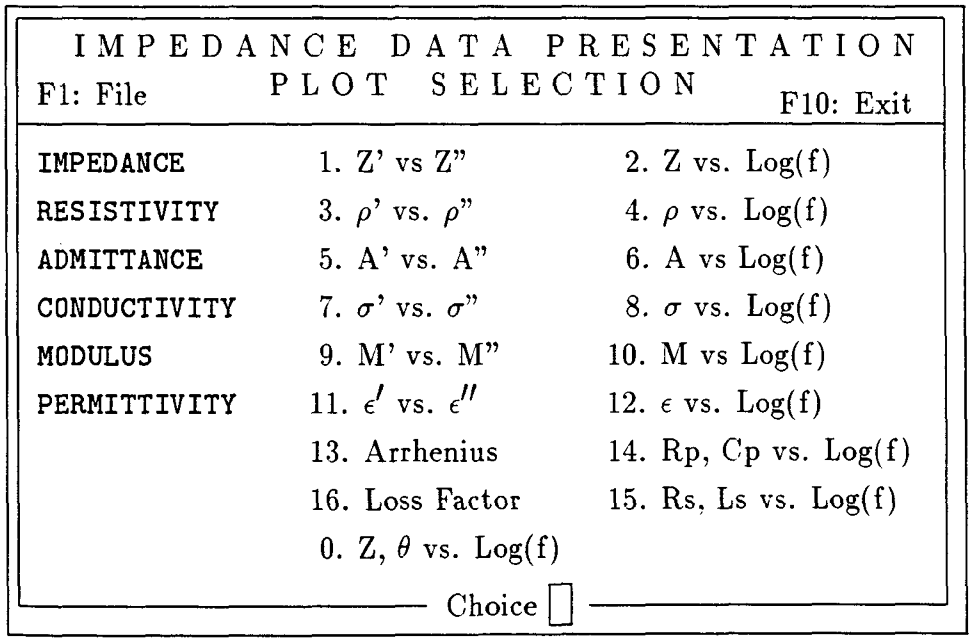
```
INS - Select File        Data Files To Analyze
F1  - Chg Drive          C: DATA NAP03 NAP03.30
F4  - Edit
ESC - Quit

                      C:\DATA\NAP03\*.*
 .            ..            CONTENT.TXT  NAP03.30
NAP03.40      NAP03.50      NAP03.60     NAP03.70
NAP03.80      NAP03.90      NAP03.100    NAP03.110
NAP03.120     NAP03.130     NAP03.140    NAP03.150
NAP03.160     NAP03.170     NAP03.SIG    STEDYST.ATE
TRNSIENT.RES
```

Figure 3.17: Data analysis: File Manager

is replaced by a file manager as shown in Figure 3.17.

File Manager: All directories are identified by a backslash before their name. To move within the file manager, the **Up**, **Down**, **Left** and **Right** arrow keys and the **Home** and **End** keys are used. To select a file, the cursor (solid colored block) is moved to the name of the file and the **Insert** key pressed; a maximum of eight files can be selected. To edit a file, the cursor is moved to the name of the file and the **F4** key pressed. A backup of this file is immediately made with `Z` as the first letter of the filename. A line editor with minimal functions may be used for the purpose of data correction and editing the `CONTENTS.TXT` file. To move to another directory, the cursor is moved to the name of that directory and the **Enter** key pressed. The files in that directory will be displayed. To change the drive or filter the files displayed, the **F1** key is pressed and a prompt at the bottom of the screen will appear. Entering the name of the new drive, the directory name and any wildcards * will select the new drive and display only the selected files. Instructions for the main keystrokes are found on the top left hand corner of the screen. To continue to the next part of the program, the **Esc** key is pressed and the data choice menu will appear.



I M P E D A N C E   D A T A   P R E S E N T A T I O N

F1: File   P L O T   S E L E C T I O N   F10: Exit

IMPEDANCE	1. Z' vs Z"	2. Z vs. Log(f)
RESISTIVITY	3. ρ' vs. ρ"	4. ρ vs. Log(f)
ADMITTANCE	5. A' vs. A"	6. A vs Log(f)
CONDUCTIVITY	7. σ' vs. σ"	8. σ vs. Log(f)
MODULUS	9. M' vs. M"	10. M vs Log(f)
PERMITTIVITY	11. ϵ' vs. ϵ''	12. ϵ vs. Log(f)
	13. Arrhenius	14. Rp, Cp vs. Log(f)
	16. Loss Factor	15. Rs, Ls vs. Log(f)
	0. Z, θ vs. Log(f)	

Choice

Figure 3.18: Data analysis: Data Choice

Data Choice: The data choice menu is shown in Figure 3.18. The filename selected for analysis is displayed at the bottom of the screen. To exit the program simply press the **F10** key. Pressing the **F1** key in this menu will cause a pop up window to appear on the top left hand corner. The options to reorder files, plot type, begin all over again or to return to the menu are available.

The first option, `Reorder Files`,allows the user to change the order of the selected files. Choosing the `Plot Type` option will cause another window to open and offer to either plot all the files on the same graph or plot one file per graph. Selecting the option `One File per Plot` will cause another window to open allowing the user to select one of the files chosen during the file manager window to be plotted. Pressing the letter **E** or the **Esc** key will take the user back to the first window. Lastly, the option `Begin All Over` will take the user back to the file manager to begin all over again.

On returning to the data choice menu, a short delay is observed as the selected file or the first file of the series of files to be plotted is input. Depending on the extension of the file, the option to plot an Arrhenius graph is enabled or disabled according to the discussion in the last section. The plot type to be displayed is chosen by keying in the number next to the plot type on the menu (see Figure 3.18) and pressing **Enter**. The chosen data type will be displayed graphically on the screen.

A prompt at the bottom of the screen to add comments (**C**) or add legends (**L**) to the graph will appear once the graph is drawn. In the legend mode, the user first selects the symbol line associated with a data curve using the function keys; **F1** represents the first data set plotted, **F2** represents the second data set and so forth. The user then uses the **arrow** key to position the cross-hair to the start of the symbol line, presses the **space bar** to pin that position down and then uses the **arrow** key to move the cross-hair to define the length of the symbol line; a final space bar defines the length. The user then types the legend for that data curve; almost all font types and numbers can be used. For a complete description the reader is referred to the HGRAPH manual. In the comment mode, the user selects one of the ten comment strings for annotation by pressing the function key as described above. The user then positions the cross-hair where the comment is to begin and presses return to pin the position down and begins typing. Again almost all fonts and characters are available. The same process can be repeated until all ten comment strings are used up. Pressing any other key will display a screen asking the user whether the present analysis of the data should be continued. Entering the **Esc** key will stop the analysis and return the user to the data choice menu. Pressing any other key will display the data input window shown in Figure 3.19 to allow changes to be made on the current graph settings.

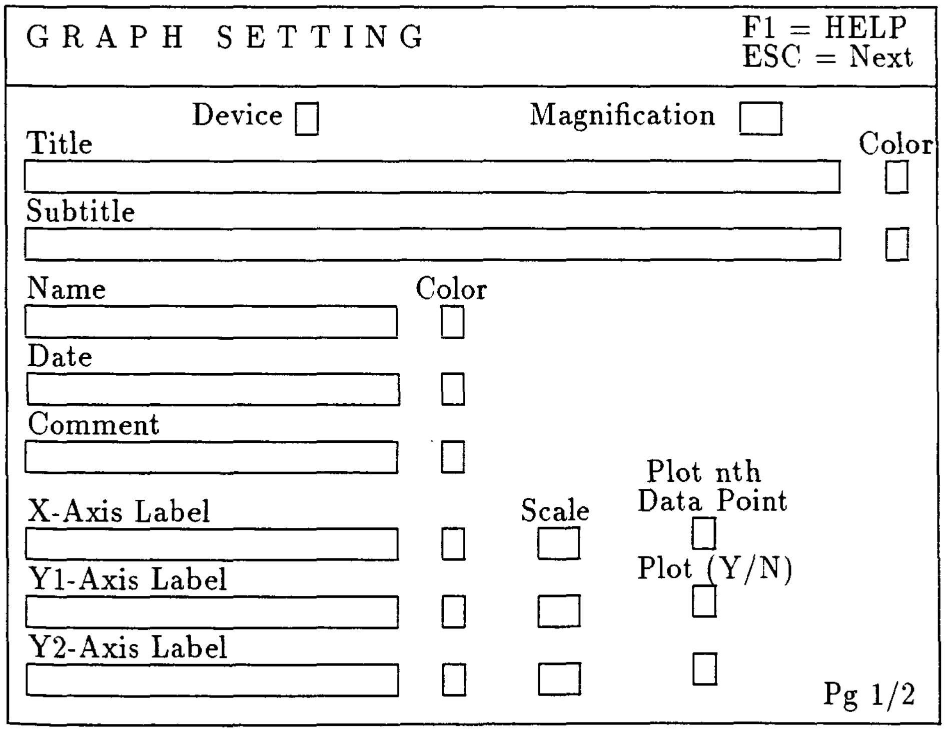


Figure 3.19: Data analysis: Graph Setting 1

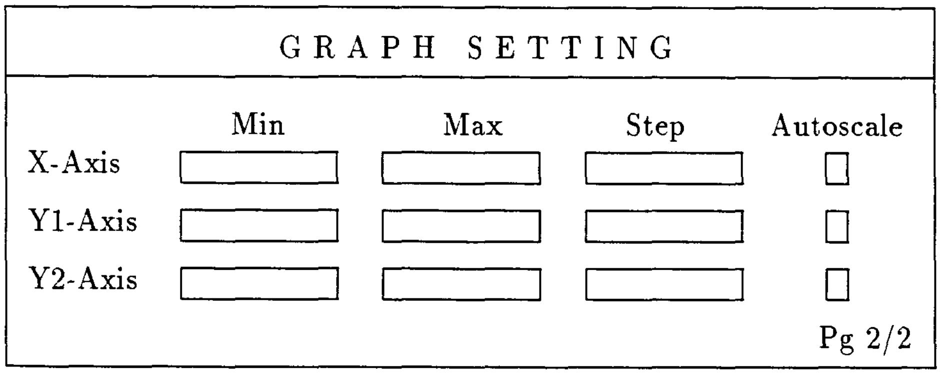
Graph Setting 1: This is one of two graph setting windows used to change the default settings of the graph. The list of devices available for output of the graph is obtained by moving the cursor into the device box and pressing the **F1** key. The devices available for plot output are the screen, the plotter, the printer (in both landscape and portrait mode) and the disk.

The title, subtitle, name, data, footnote and x-axis, y1-axis and y2-axis titles (where y1-axis is the left hand y-axis and the y2-axis is the right hand axis), can all be edited. Again, the list of available fonts and graphics string commands is accessible by moving the cursor into one of these input windows and pressing the **F1** key.

The color of any of the displayed strings on the screen is determined by a number between 0 and 10. The colors which match to these numbers are found by pressing the **F1** key when the cursor in one of the color input windows. For outputting to the plotter, the user is limited to six pen colors available on the plotter.

The axis type may be changed to log or linear by appropriately entering `Log` or `Lin` in the axis type input window. The display of both y-axes or one y-axis can be enabled or disabled by entering a letter **Y**(es) to display axis and the letter N(o) to turn off an axis. Finally, the option to display every nth data point is available by entering an integer n; for example inputting 1 will display each data point with a symbol and entering 5 will display every fifth data point. Once the user is satisfied with these graph settings, pressing the **Esc** key will save the new settings and display the second graph setup window.

Graph Setting 2: This window is used to set the axis limits by entering the minimum and maximum values for the axes in the first two boxes. The step is


GRAPH SETTING

	Min	Max	Step	Autoscale
X-Axis				☐
Y1-Axis				☐
Y2-Axis				☐

Pg 2/2

Figure 3.20: Data analysis: Graph Setting 2

the increment between two displayed numbers on the graph. For the x-axis settings, entering the letter y(es) in the autoscale window followed by the return key will result in an internal sweep of the x-data to determine the minimum and maximum values in the range. These numbers are fed into an autoscaling algorithm to determine the best minimum, maximum and step values for the x-data. Autoscaling performed on either of the y-axis data sets is done only for that x-range displayed in the x-axis settings. Manual scaling of the data can be achieved by manually entering numbers into each of the axis windows. For log scaled plots, the values input for scaling are the log of the minimum and maximum values. The step value is an integer divisor of one and tic marks are placed at powers of ten and at powers base ten of the multiples of the step. The default value is one resulting in ten tick marks per decade. Once the scaling values are input, pressing the **Esc** key will display the graph with the new settings.

The technique of analyzing multiple files is the same as that of single files. The program will automatically choose different symbols and colors for the different data files. Currently, the user cannot change the default settings for the choice of the color and symbol types.

CHAPTER 4. WORKSTATION PERFORMANCE

The workstation performance and typical results obtained using this workstation are given in this chapter. The workstation capabilities are divided into three sections. The first addresses the impedance measuring limits and accuracies of the Solartron 1260. This is followed by an analysis of the temperature control capability, and finally, this chapter is concluded by showing data obtained using this workstation for a typical fast ion conducting glass, $NaPO_3$.

Impedance Measurements

The impedance measuring capabilities of the Solartron 1260 are reported in the first part of this section. The calibration of the conductivity cell using standard resistors, capacitors and dielectrics, Teflon and quartz, are reported in the second part of this section.

Impedance limits of the Solartron 1260

As shown in Chapter 2, not only is the conductivity temperature dependent, but it is compositionally dependent as well. These two factors combine to produce conductivities which vary between 10^{-10} to 10^{-1} $\mho$/cm for glasses. Clearly, to fully characterize the conductivity, as much as possible of this range must be measurable

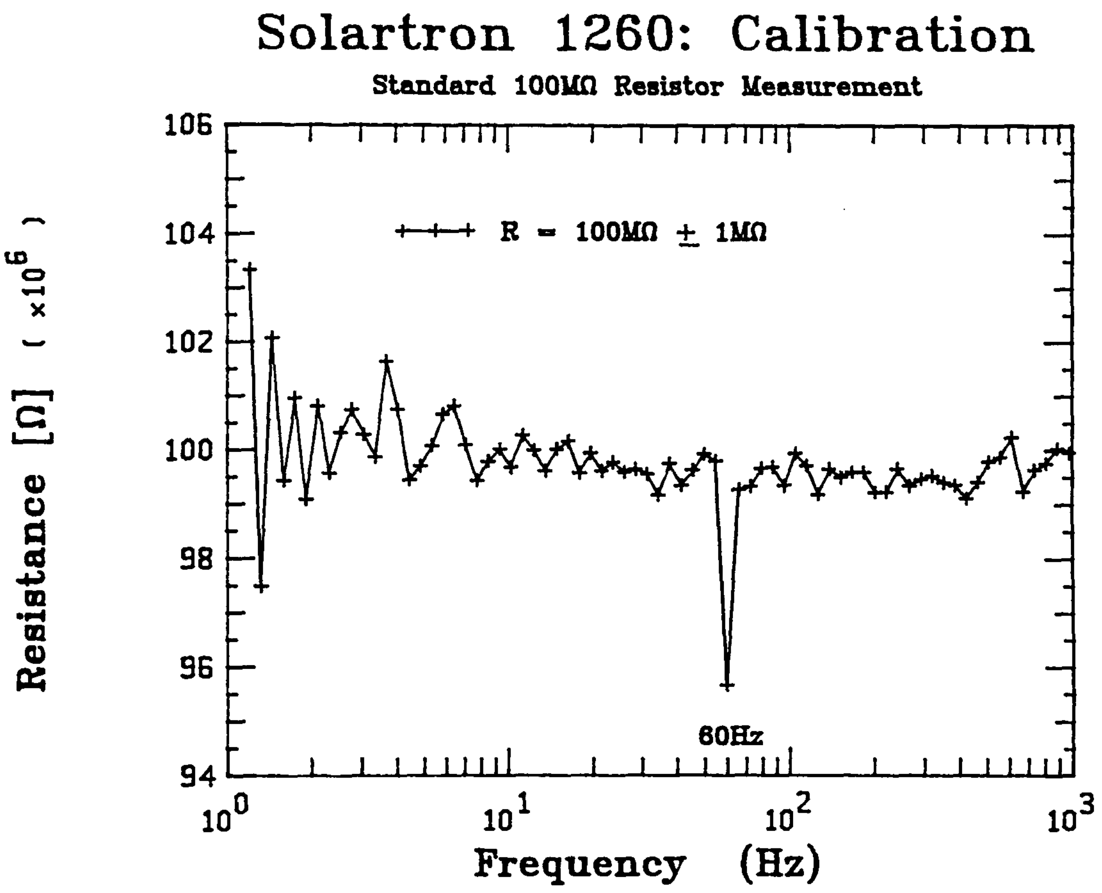


Figure 4.1: Solartron 12601: upper impedance limit test, R = 100 MΩ

using the IS workstation, and for this reason the high impedance (low conductance) measuring limit of the impedance analyzer becomes a very important quantity to determine. The upper impedance range for the Solartron is specified in the Solartron user's manual as 100 MΩ with an accuracy of ±10 MΩ. To test this capability, a 100 MΩ high precision resistor (±1%) was measured using the Solartron 12601 test module which fits directly onto the four terminals of the instrument. The results of this measurement are shown in Figure 4.1 and are within 3% of the stated resistor

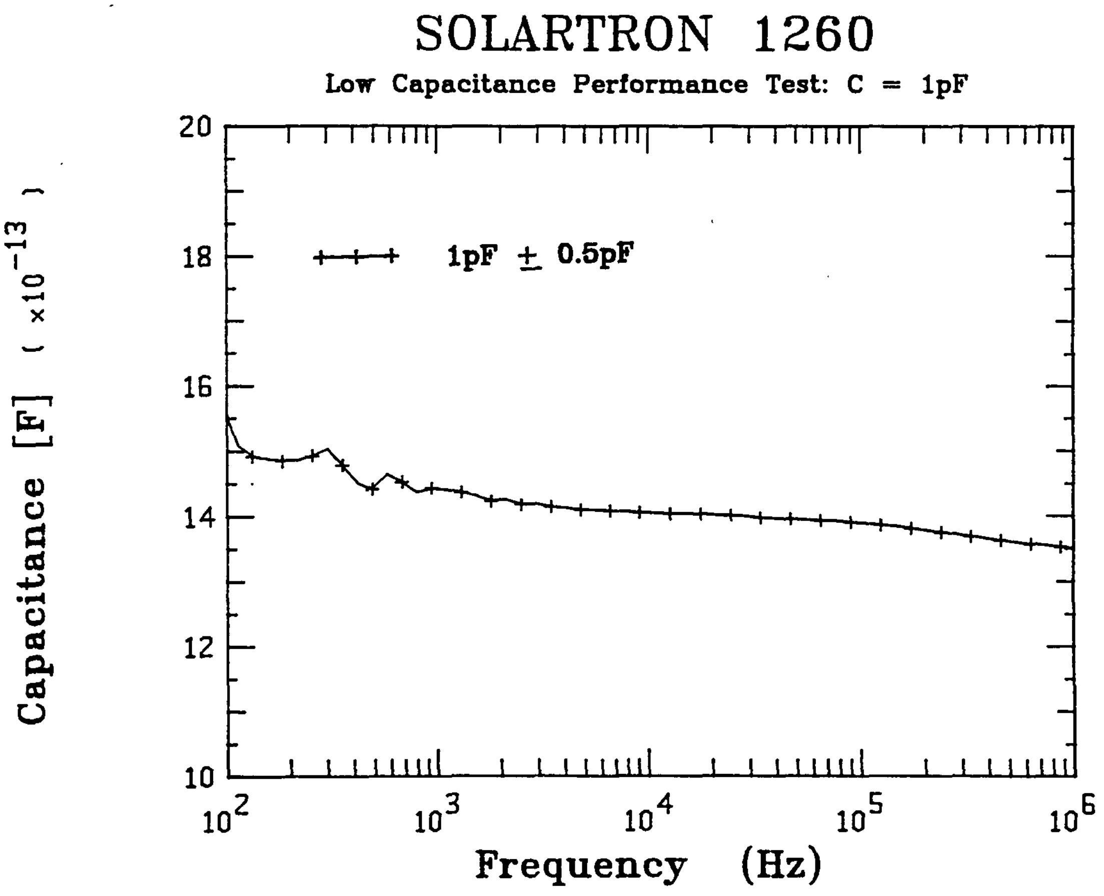


Figure 4.2: Solartron 12601: lower capacitance limit test, $C = 1$ pF

value; this error is much better than the accuracy specified ($\pm 10\%$) by the manufacturer. The spike at 60 Hz is caused by a systematic error due to inadequate shielding of the resistor from 60 Hz electrical noise.

To test the low capacitance measuring capability of the instrument, two standard capacitors, 1 pF and 22 pF, were measured. The first was to test the lower capacitance measuring limit of 1 pF specified by the manufacturer. The results of the 1 pF measurements are shown in Figure 4.2 and lie within the capacitor manufacturers specifications of ± 0.5 pF for the capacitor. The results, however, are precise over the frequency range and exhibit the characteristic decrease in capacitance due to ohmic losses in the capacitor with increase in frequency.

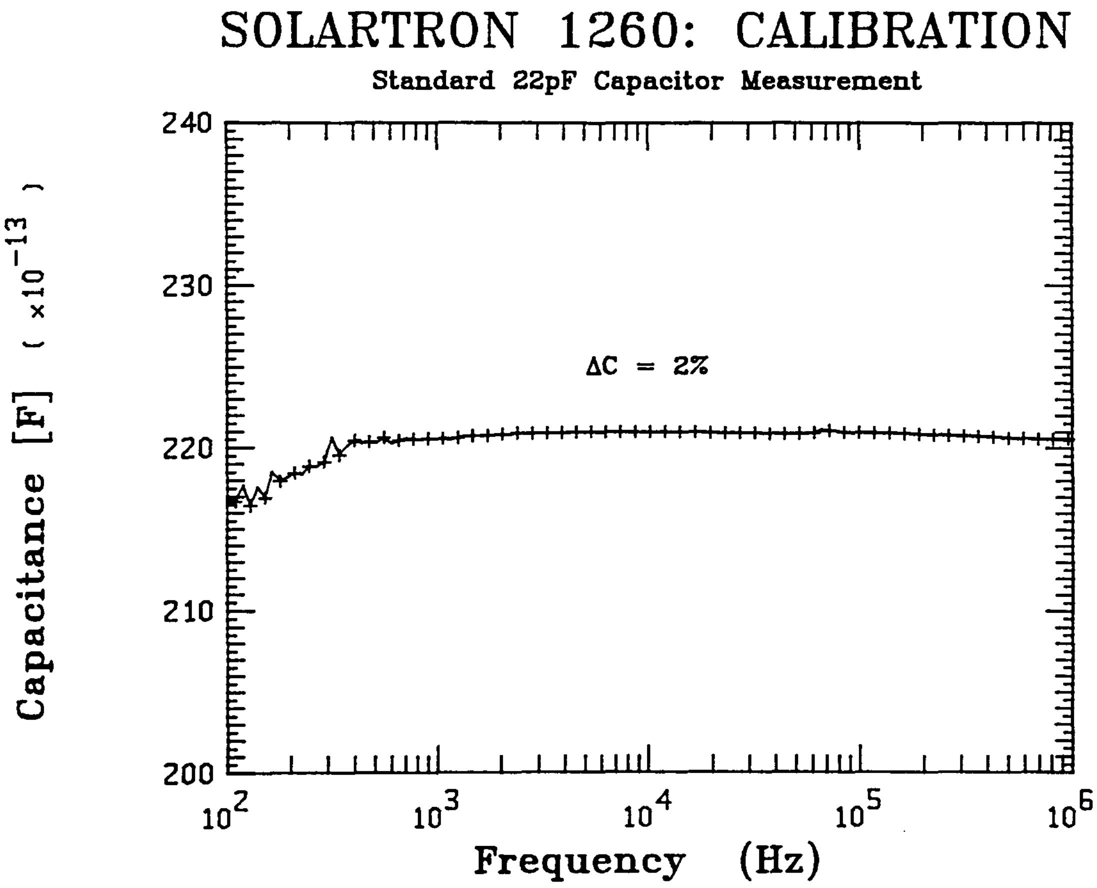


Figure 4.3: Solartron 12601: lower capacitance limit test. C = 22 pF

The relative dielectric constant of ionic conducting chalcogenide glasses vary between 10 and 30. The lowest capacitance expected to be measured by the Solartron for such glasses is approximately 9 pF, and is determined from Equation B.3 by using the largest possible cell constant (0.1/cm) and a minimum value for k' (10). A 22 pF capacitor therefore represents a median capacitance value of a chalcogenide glass specimen at high frequency and low temperature. The results of a frequency sweep of a standard 22 pF capacitor show that the measurement are within ±2% of the actual value and are shown in Figure 4.3.

The accuracy of the high impedance and capacitance test results are much better than those specified by the Solartron user's manual. Thus it has been shown that the Solartron 1260 is capable of measuring impedance and the capacitance values of ionic conducting glasses that lie at the extreme values that are expected for these glasses. As values for the glass move away from these extremes the accuracy of the measurement, will no doubt, improve considerably.

Conductivity Cell Characterization

The conductivity cell was characterized by comparing the values of standard resistors and capacitors measured in the conductivity cell and measured in the Solartron 12601 module. Once this calibration was complete, the ability of the workstation to measure dielectric properties of standard materials with sputtered circular gold electrodes prepared in the lab was tested.

Standard Elements Results of measurements of standard resistors and capacitors in the conductivity cell are described in this section. The characterization procedure began by measuring the parasitic resistance and inductance of a shorting bar first in the Solartron 12601 module and then in the conductivity cell. The resistance and inductance versus frequency is shown in Figure 4.4. The parasitic resistance in the conductivity cell is seen to be 0.3 Ω and the inductance approximately 34 nH. The resistance measured in the 12601 module was approximately 0.25 mΩ and the inductance was approximately 20 nH. From the above results, the conductivity cell leads and electrodes contribute 0.3 Ω to the measured resistance and 14 nH to the measured inductance. Both these values are relatively small compared to the magnitude of the impedance of typical glass specimens and therefore will not affect the experiment.

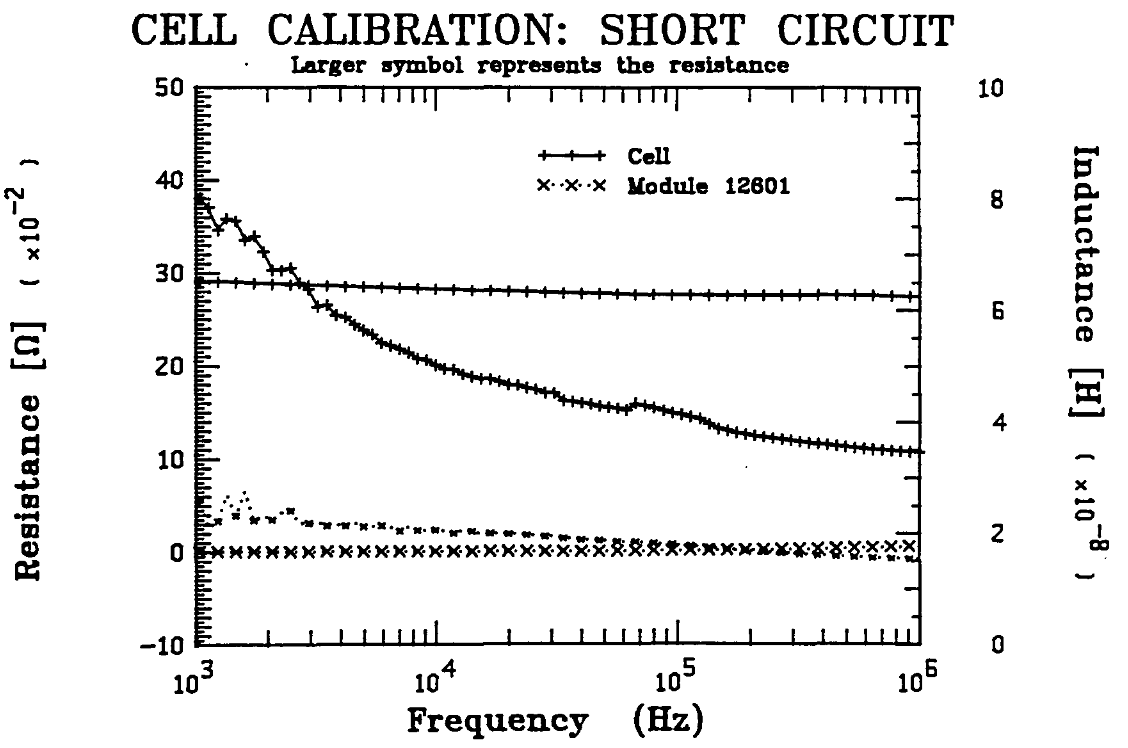


Figure 4.4: Short circuit frequency response of the Solartron 12601 and the conductivity cell

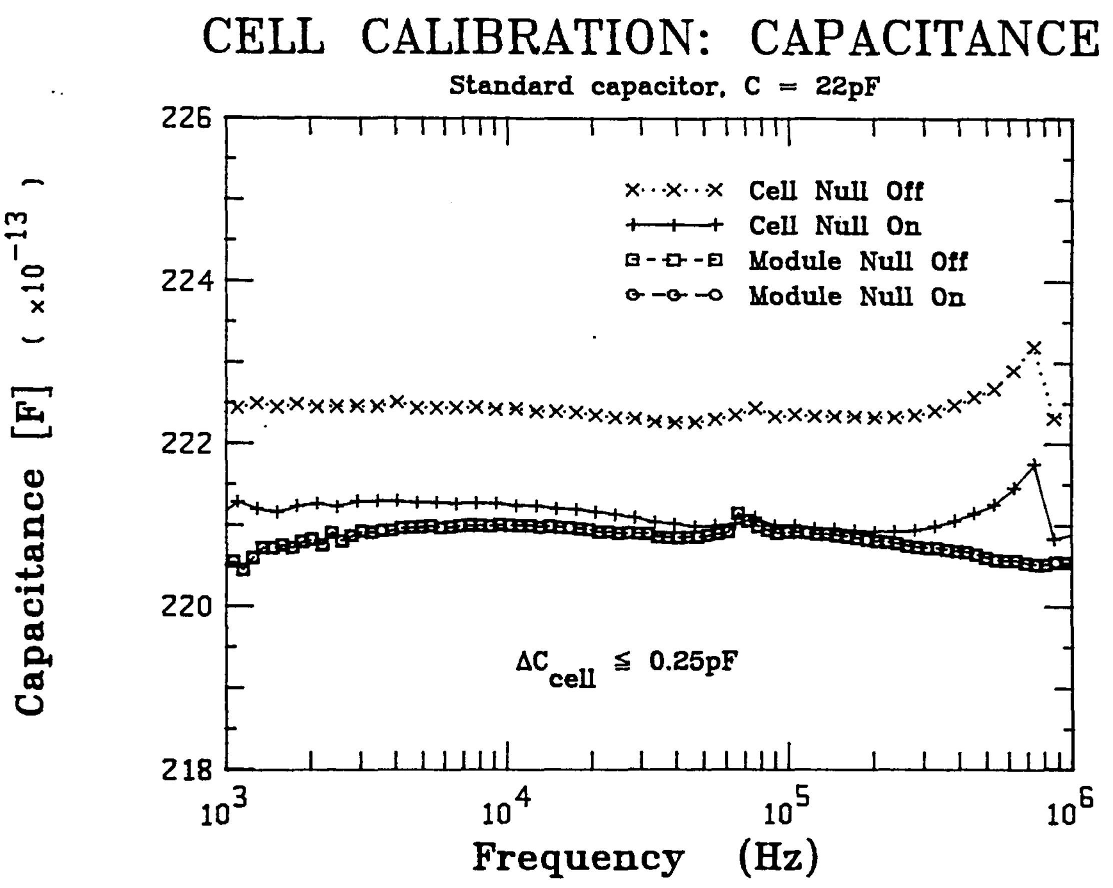


Figure 4.5: Capacitance measurements of 22 pF in the Solartron 12601 and in the conductivity cell

The parasitic (stray) capacitance of the cell was determined by measuring the capacitance of a 22 pF capacitor in the conductivity cell. The short and open circuit calibration (null) routine of in the Solartron 1260 was performed on the conductivity cell and the capacitor was measured. The same procedure was followed with the 12601 module as the sample holder instead of the conductivity cell. The capacitance measured using these four setups, calibrated (null on) and uncalibrated (null off) conductivity cell and the 12601 module, are shown plotted on in Figure 4.5. The results for the 12601 module with the null on and off are essentially identical and differ by less than 0.01 pF. The difference between the module measurements and the uncalibrated (null off) cell measurements show that the cell results are greater than the module results by approximately 0.25 pF. The calibrated (null on) cell data, however, differ by only 0.1 pF due to correction of the measurements by the nulling routine of the Solartron 1260. The stray capacitance in the cell from the above results is therefore approximately 0.25 pF and its effect on the measurements made on samples is reduced to 0.1 pF by nulling of the cell. Again it should be noted that errors due to this parasitic capacitance are negligible since typical capacitive value of glass specimens to be measured are in the range of 20 pF leading to an error of 0.5%.

Next, a series of resistors were measured in the nulled conductivity cell and the nulled Solartron 12601. This experiment was performed to determine the effectiveness of the nulling calibration for different impedance magnitudes. For low impedance measurements, R = 10 Ω and R = 100 Ω, the cell measurements are systematically less than the 12601 measurements by 0.4 Ω and are shown in Figure 4.6 and Figure 4.7. An error of 5% is, therefore, expected for low values of resistance but this error is expected to decrease as the magnitude of the impedance increases. This improved accuracy was confirmed by measuring the resistance of a 10.2 kΩ resistor over the complete frequency range. Figure 4.8 shows that the results are essentially identical up to 10^5 Hz. The nonlinear behavior of the resistor at high frequency is a typical response exemplifying the difficulty of making accurate impedance measurements at even higher frequencies.

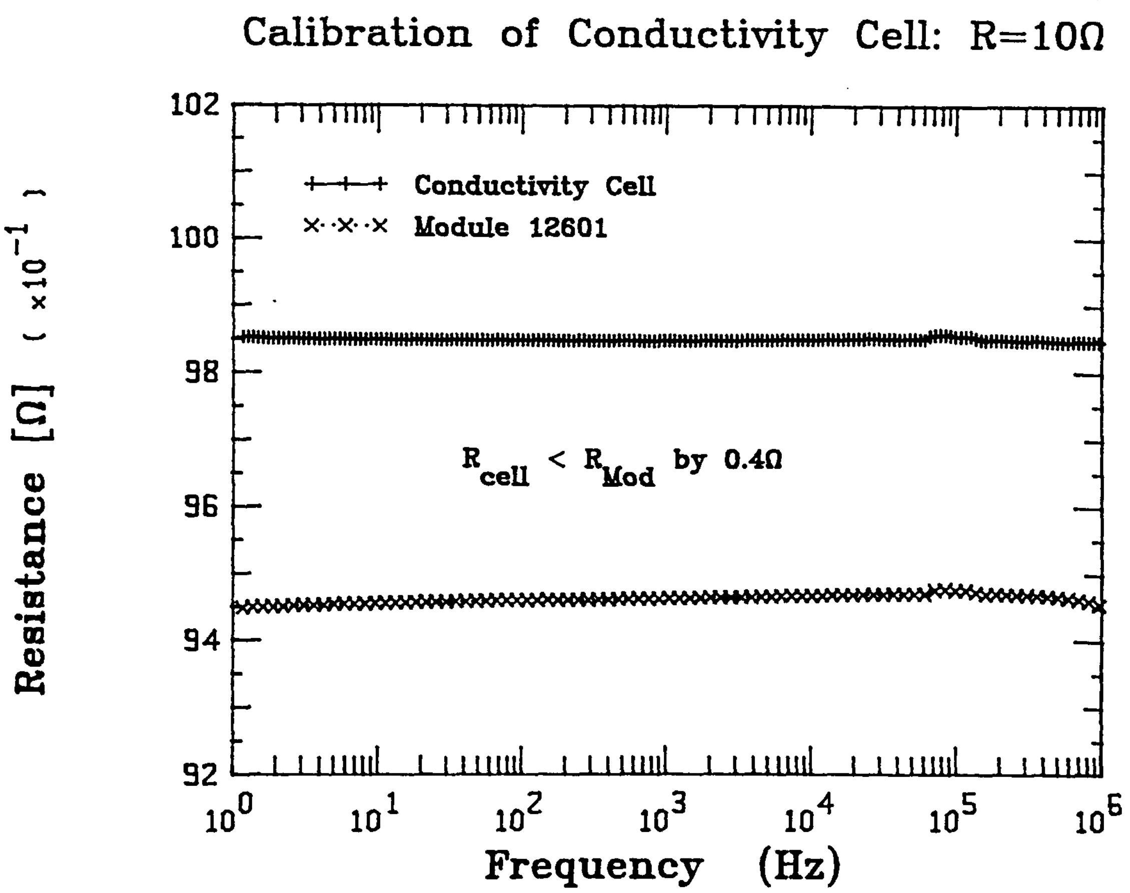


Figure 4.6: Resistance measurement of a 10Ω resistor in the Solartron 12601 and in the conductivity cell

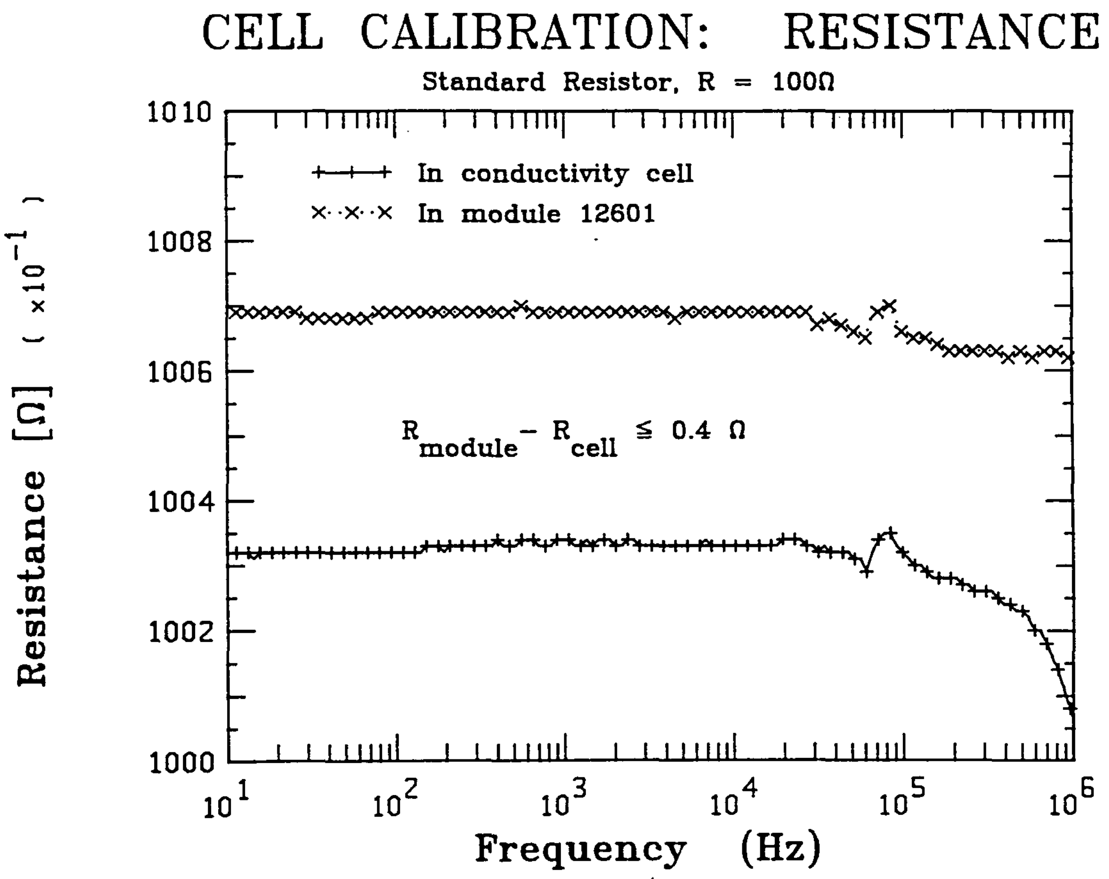


Figure 4.7: Resistance measurements of a 100 Ω resistor in the Solartron 12601 and in the conductivity cell

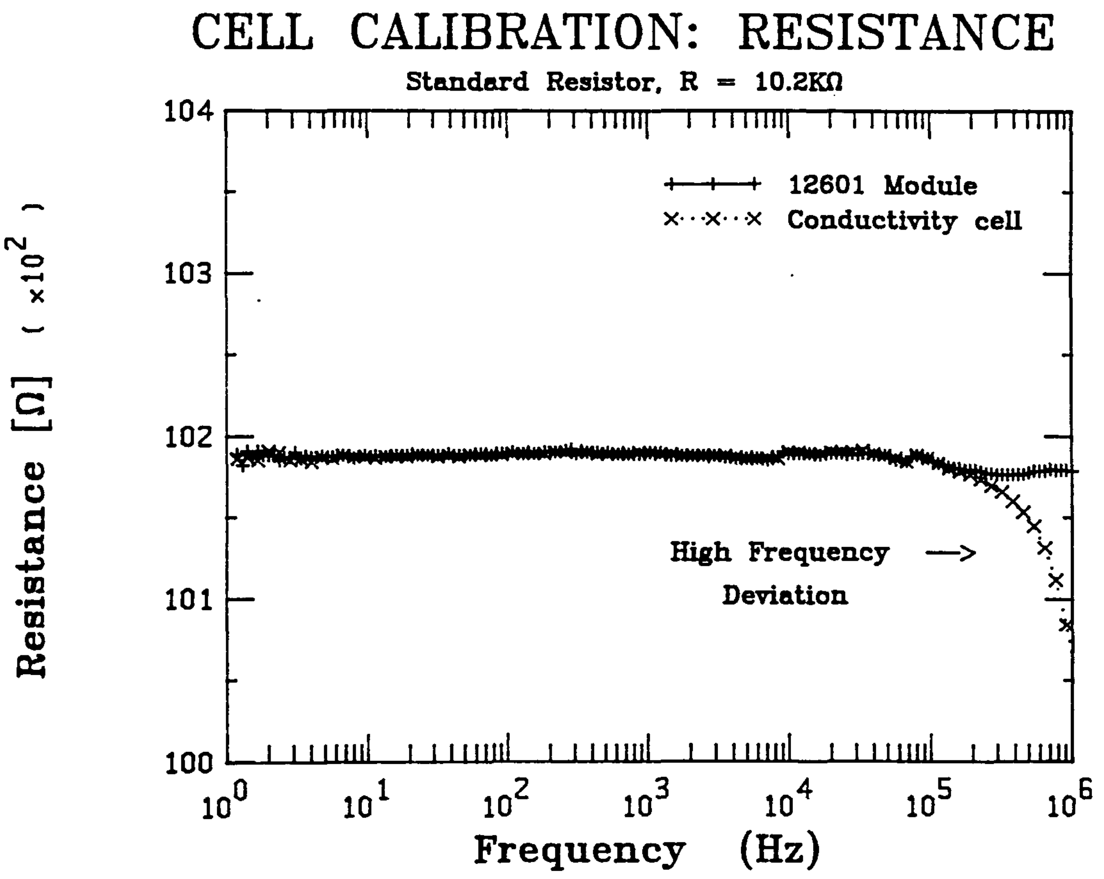


Figure 4.8: Resistance measurements of a 10.2 kΩ resistor in the Solartron 12601 and in the conductivity cell

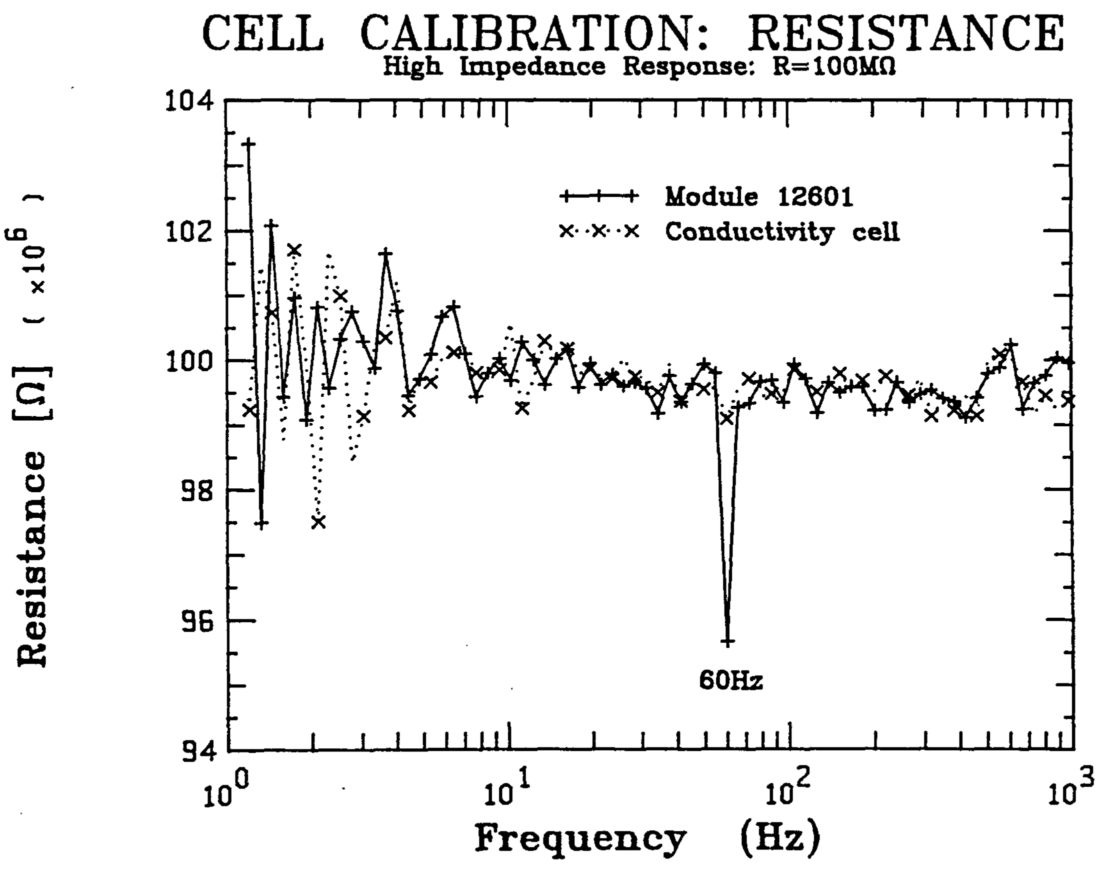


Figure 4.9: Resistance measurements of a 100 MΩ resistor in the Solartron 12601 and in the conductivity cell

To check for current leakage paths in the conductivity cell, a 100 MΩ resistor was measured in the conductivity cell with the nulling operation on and the results compared to those obtained using the 12601 module. The data from the 12601 module have already been shown in Figure 4.1. All leakage paths with a resistance less than a 100 MΩ, due to bad connections or inadequate electrode isolation from the casing of the grounded cell, should affect the measurement of the 100 MΩ resistor.

The conductivity cell data are plotted along with the data obtained using the 12601 module in Figure 4.9. The graph shows a cell response identical to the module, except for improvement of the data at 60 Hz, due to complete shielding of electrical noise by the brass container of the cell.

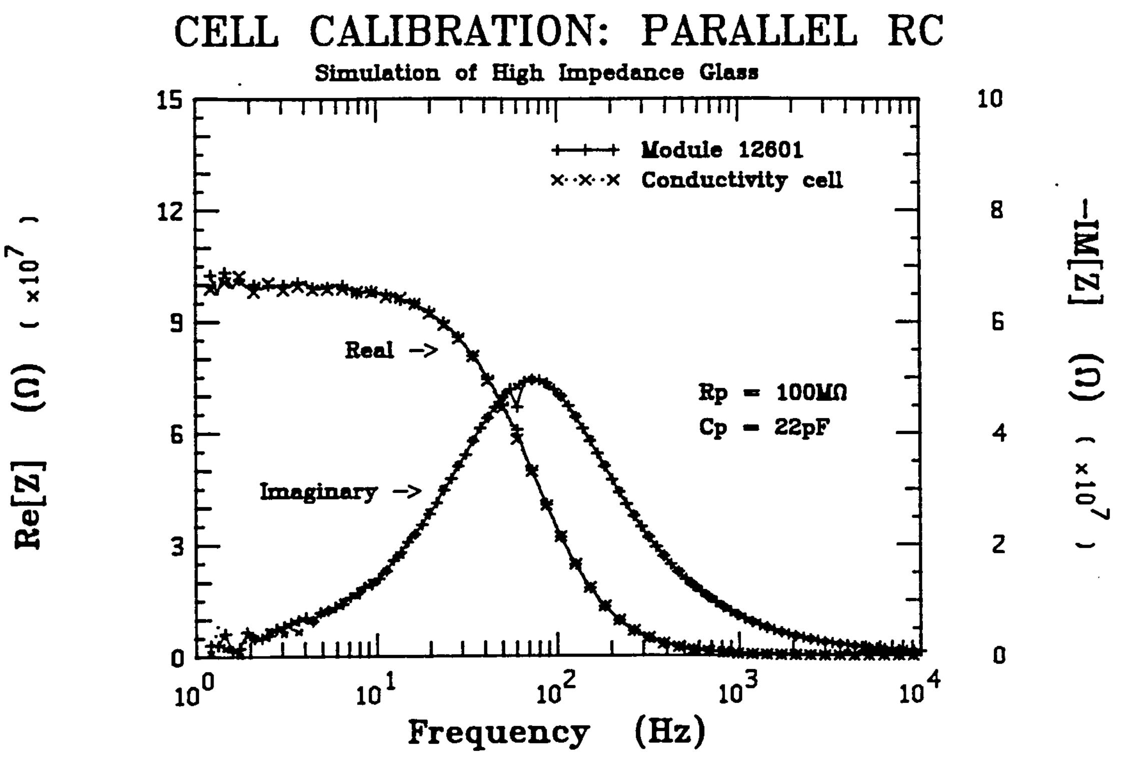


Figure 4.10: Complex impedance measurements of a parallel 100 MΩ resistor and a 22 pF capacitor in the conductivity cell

Finally, the parallel RC circuit with a 100 MΩ resistor and a 22 pF capacitor used to choose the Solartron over the HP4194A impedance analyzer (see Chapter 3) was measured in the conductivity cell. The data are shown in Figure 4.10 and show that the same results are obtained in both the conductivity cell and in the Solartron 12601 module.

The above characterization experiments have shown that the conductivity cell can be used for making accurate and precise measurements on ionically conducting glasses. Parasitic line resistance, inductance and stray capacitance are small enough that the nulling routine of the Solartron 1260 can compensate for their contribution during the measurement. The high impedance measurements show that the impedance measuring range for this workstation using the conductivity cell is the same as that defined for the Solartron 1260. The accuracy for all measurements using the nulling operation on the conductivity cell are within 5% of the actual value and are in close agreement to the accuracy reported in the Solartron manual using the 12601 module.

Standard Dielectrics To further test the capability of the conductivity cell and Solartron 1260, capacitors made from high purity silica glass and Teflon wafers as the dielectric and gold sputtered circular electrodes were examined. Due to the extremely high resistance of each of these dielectrics, accurate measurement of the dielectric constant of these materials is considered as a very stringent test of the performance of the impedance workstation. Impedance data for each capacitor were collected using the data collection software, and their relative dielectric constant were plotted versus frequency using the data analysis routine. The experimental values for the relative dielectric constants for SiO_2 glass are shown in Figure 4.11 and for Teflon in Figure 4.12 and the data show virtually no frequency dependence and are in

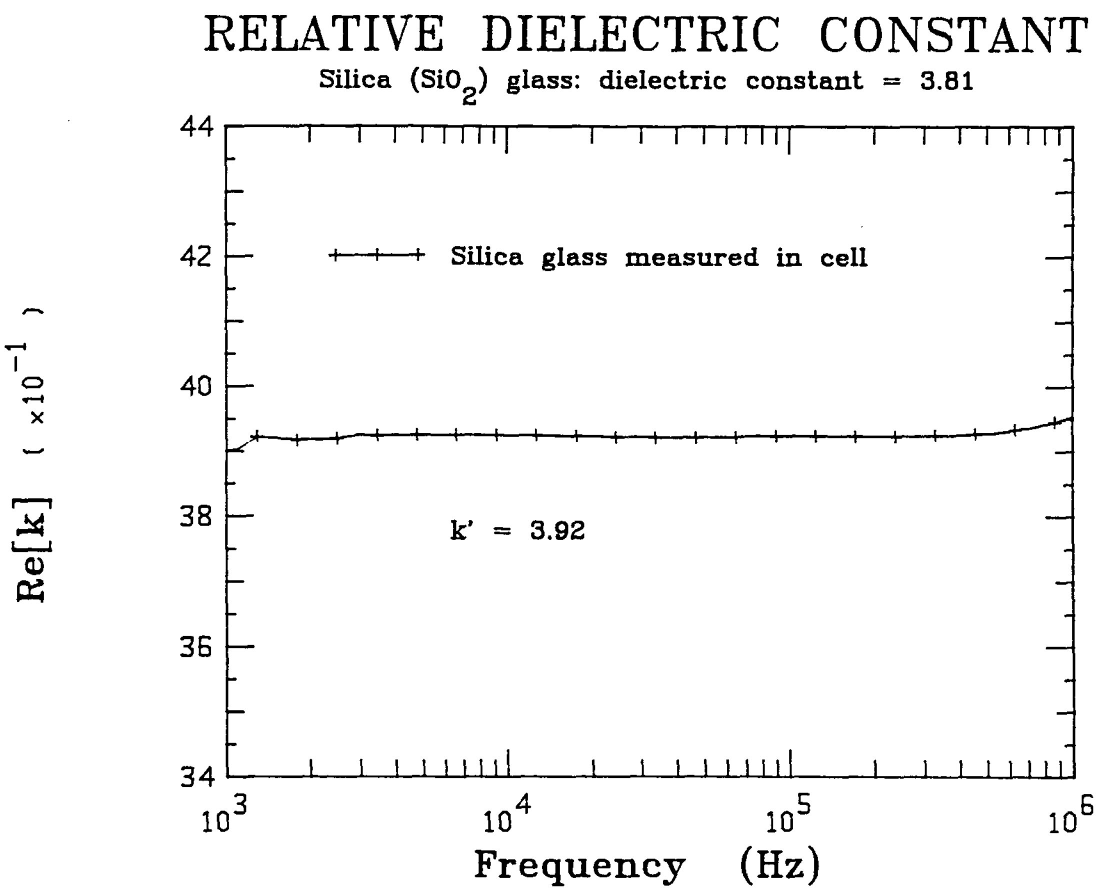


Figure 4.11: Dielectric measurement of high purity silica glass

excellent agreement with the literature values for these materials [15]. The reported relative dielectric constant for silica glass is 3.81 while the measured value is 3.92, which is within 3% of the reported value. The reported relative dielectric constant for Teflon is 2.10 [15] and the measured values for two different Teflon capacitors were 2.16 and 2.18. The deviation from the reported value for the worst case is less than 4%. The results for the relative dielectric constants for these standard materials are in excellent agreement with the standard reported values and are within the error

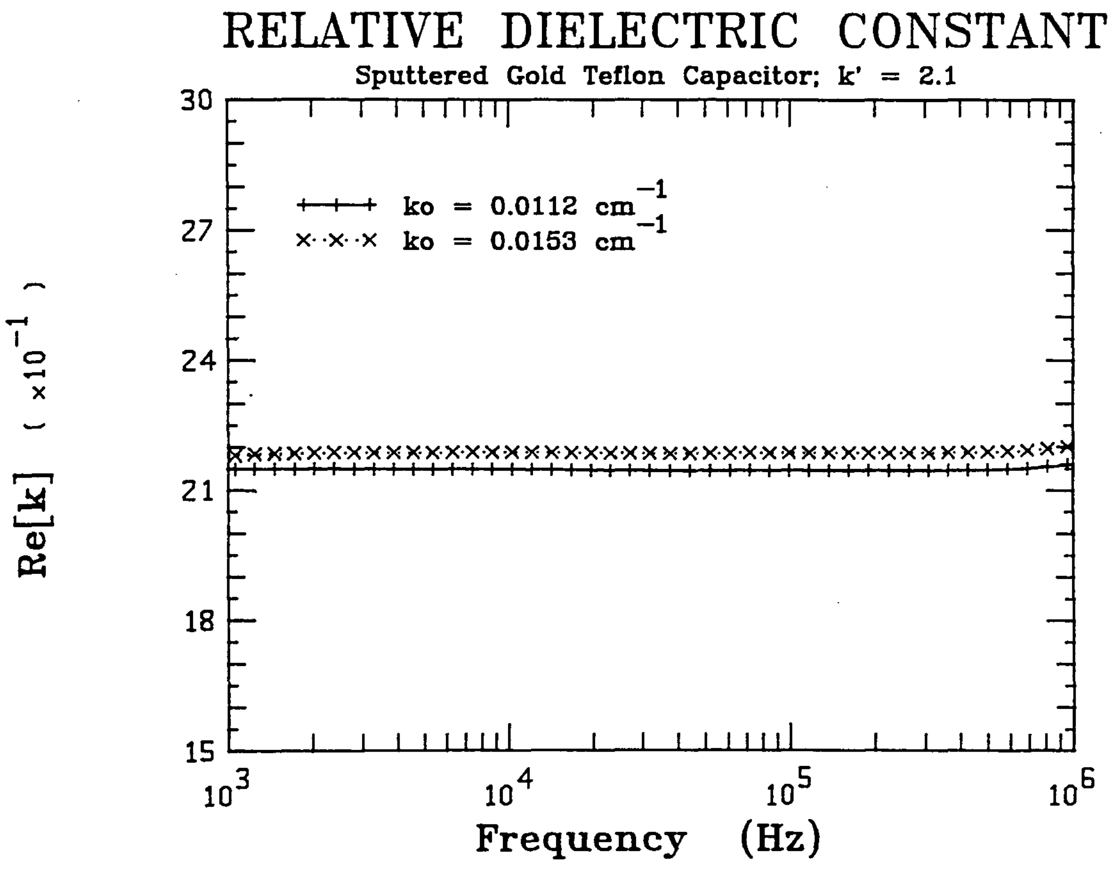


Figure 4.12: Dielectric measurement of Teflon

margin of ±5% determined in the calibration section.

The above experiment confirm the accuracy of the workstation for low capacitance measurements, since the capacitance in each of these tests was less than 2 pF, and the accuracy of the measured dielectric constants are within 5% of the reported values.

To summarize the characterization of the conductivity cell, the series resistance and inductance and the parallel capacitance due to the leads and electrodes in the conductivity cell are 0.3 Ω, 14 nH and 0.25 pF, respectively. The series and parallel resonance frequencies for this combination of circuit elements is much greater than the upper measuring frequency capability of the Solartron. Using the nulling operation, low capacitance and high impedance dielectrics can be measured with an accuracy of better than ±5% at the extremes of the measuring range. Relative dielectric constant values of standard materials using sputtered gold electrodes prepared in the manner described in Chapter 3 have been measured to an accuracy of 5%. Since fast ion conducting glass samples will be prepared for impedance measurements in the same manner as the dielectrics used above, and the typical impedances of these glass samples will be well within the measuring range of the Solartron, the impedance analyzer and the conductivity cell performance described above show that the IS workstation is well suited to the electrical characterization of these glasses.

Temperature Control

A typical time-temperature history during an IS data collection experiment is shown in Figure 4.13. The setpoints for this run were changed in steps of ten, starting at 30^{o}C, and steady state was determined as a temperature deviation of less than 0.05^{o}C over 8 minutes.

The approach to steady state temperature response within the conductivity cell is displayed in Figure 4.14 and is a blow up of the temperature profile in Figure 4.13 between 60 and 70^{o}C. Temperature stability is achieved within 80 minutes for this experiment and impedance measurements are made within 2.5^{o}C of the user specified temperature. For most experimental setups, the period to achieve steady state is less

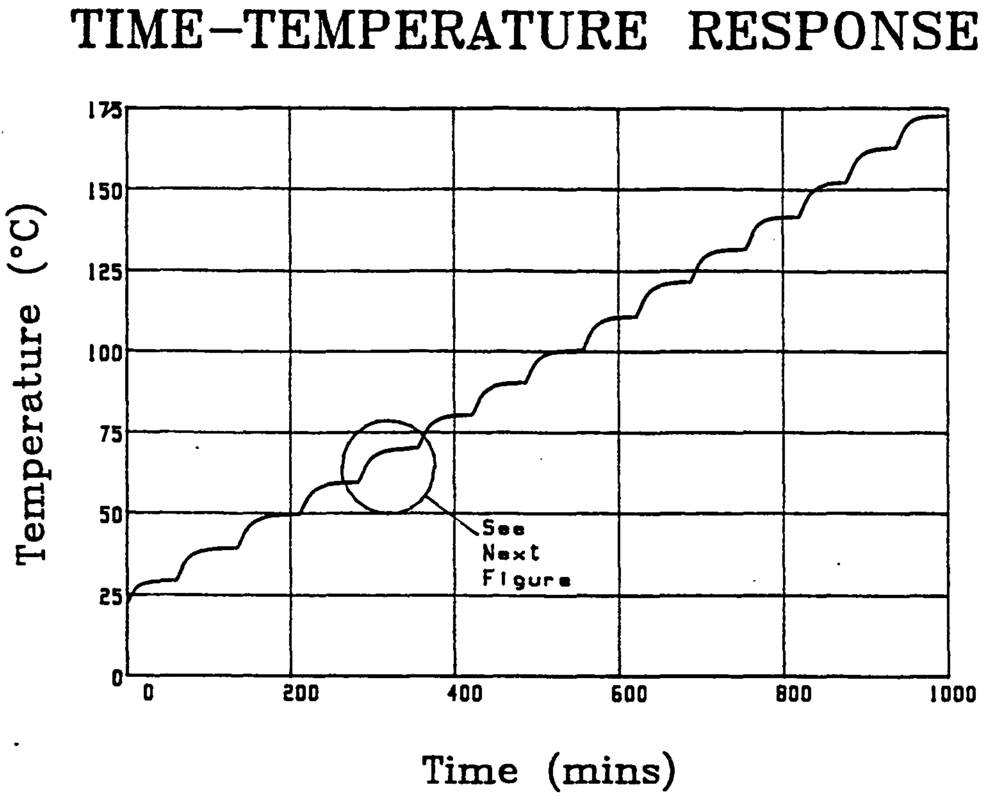


Figure 4.13: Time-temperature history during a data-collection experiment

than 8 minutes, and most impedance measurements therefore are made in less than 50 minutes.

Results for the FIC glass, sodium metaphosphate

To test the complete performance of the workstation, a sodium phosphate glass, $NaPO_3$, which is a typical fast ion conducting glass, was prepared and electrical measurements were performed using the data collection routine. The measurements were analyzed using the data analysis routine and are shown in this section.

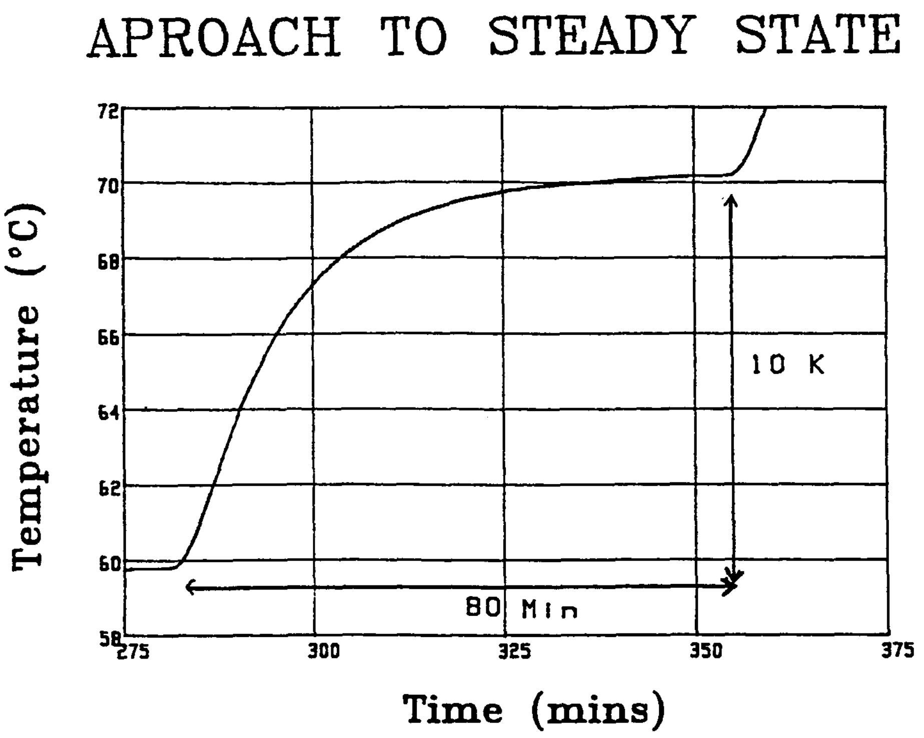


Figure 4.14: Approach to steady state temperature response of conductivity cell ramped from 60^oC to 70^oC

Sodium metaphosphate glass was prepared by heating 99% pure sodium metaphosphate crystal, purchased from Fisher Scientific, in an open platinum crucible over a blue flame until it was a liquid. It was than heated in a muffle furnace at 1100^oC for 30 minutes. The melt was quenched into a flat disk between two smooth surfaced graphite thermal blocks heated to 250^oC ($\sim T_g$) and allowed to cool at a rate of 0.5^oC/min. The quenching blocks produce relatively flat smooth disks 2 cm in diameter and 2 mm in thickness.

Gold electrodes were sputtered onto the disk as described in Chapter 3. The area of the electrode surface was determined by measuring its diameter with a caliper and the thickness of the glass was determined using a micrometer. The sample was hermetically sealed into the cell in a helium glove box.

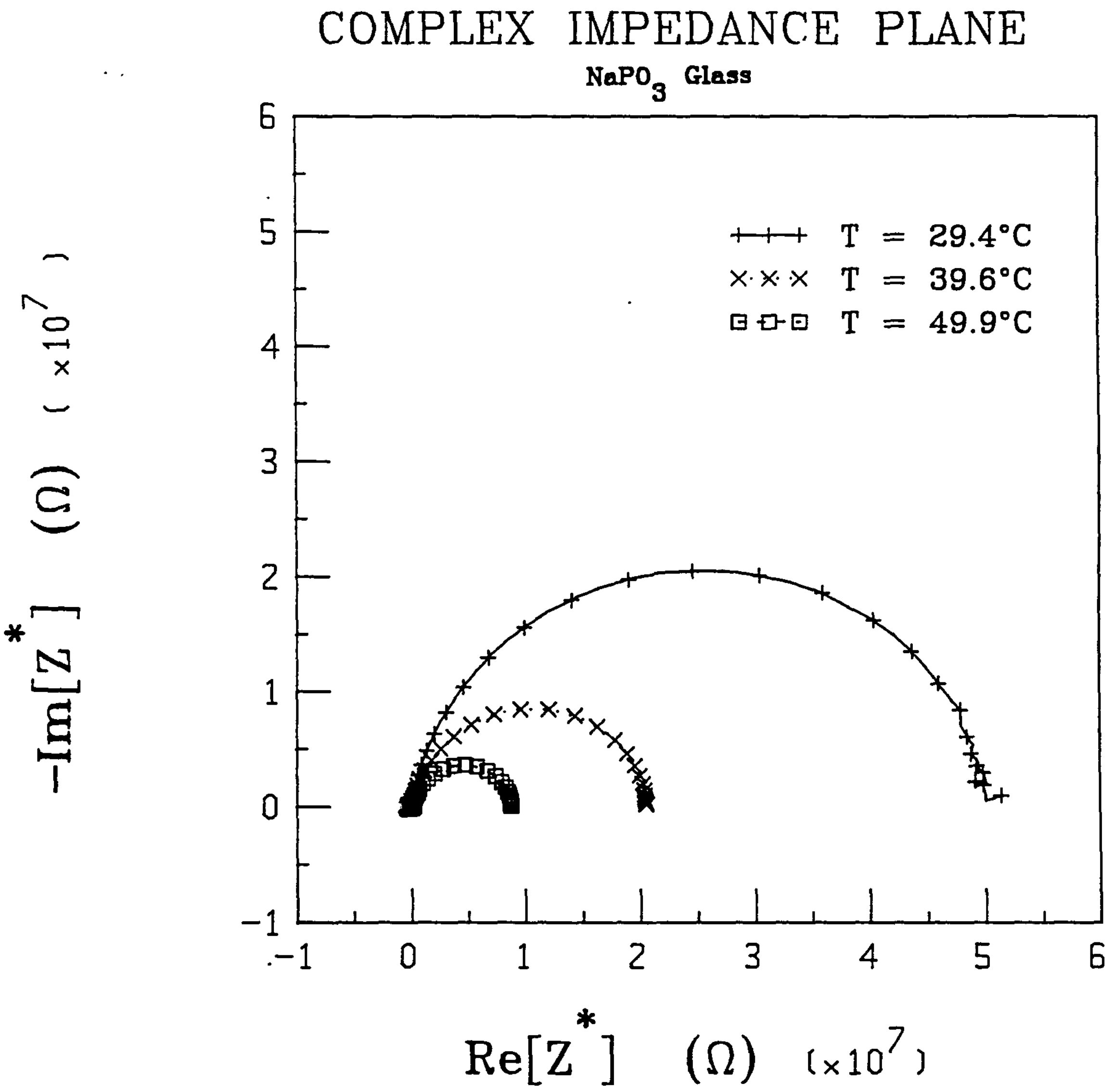


Figure 4.15: Complex impedance plane plots at different temperatures for $NaPO_3$ glass

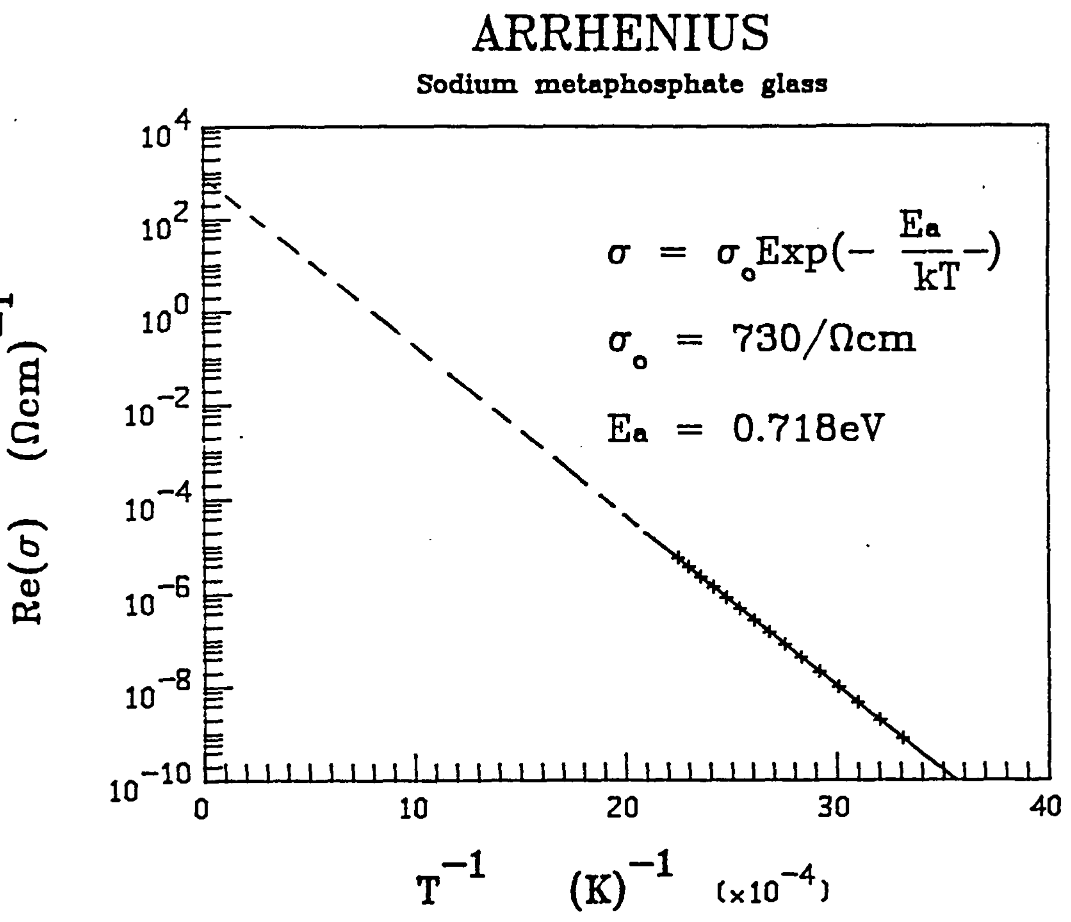


Figure 4.16: Arrhenius plot of conductivity of $NaPO_3$ glass. The dashed line is extended above Tg to show the limiting high conductivity

Impedance measurements to determine the conductivity of this glass were performed over the complete frequency range (1Hz to 1MHz) and the temperature was ramped in steps of 10^o starting at 30^oC and ending at 170^oC. The dramatic change in resistivity with temperature is shown in Figure 4.15. The typical depressed arcs decrease in radius as the temperature increases. The real dc conductivity determined from the complex plane are plotted versus temperature in an Arrhenius plot in Fig-

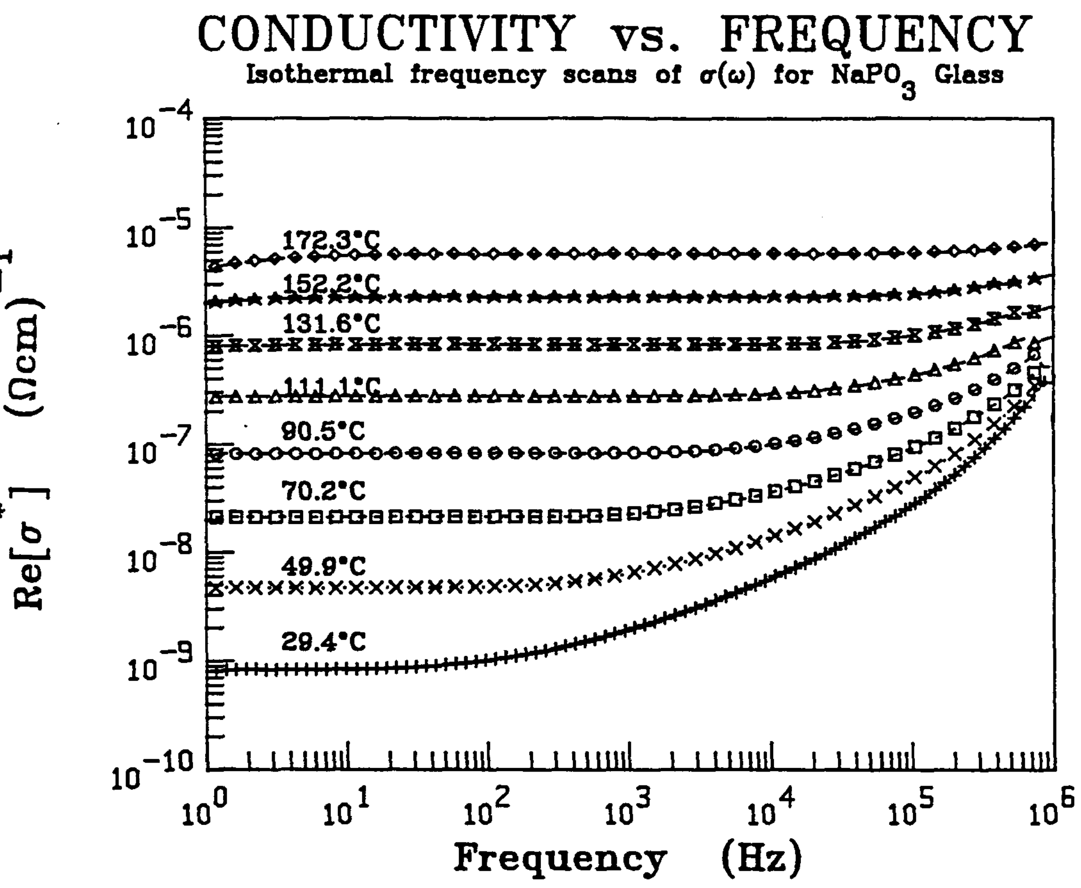


Figure 4.17: Isothermal frequency scans of $\sigma(\omega)$ for $NaPO_3$

The above experiment shows that the activation energy, the limiting high temperature conductivity and the relative dielectric constant of the glass can be measured accurately using this workstation. The value for the activation energy for $NaPO_3$ is in excellent agreement with that reported by Martin [6]. The value for the limiting conductivity is slightly larger but is still in good agreement with that reported by Martin [6]. The relative dielectric constant was measured to within five percent of the value reported by Martin [6].

To show that the conductivity cell seal is hermetic. the following experiment was performed in a glove box. Two silicon sulfide glass pieces were weighed on an A&D digital balance to a resolution of 1mg. One piece was sealed in a glass jar and stored in the glove box, while the other was sealed in the conductivity cell, taken out of the glove box, and heated for 12 hours at 200^{o}C in the thermal block. The cell was air cooled to room temperature and brought back into the glove box. The two glass pieces

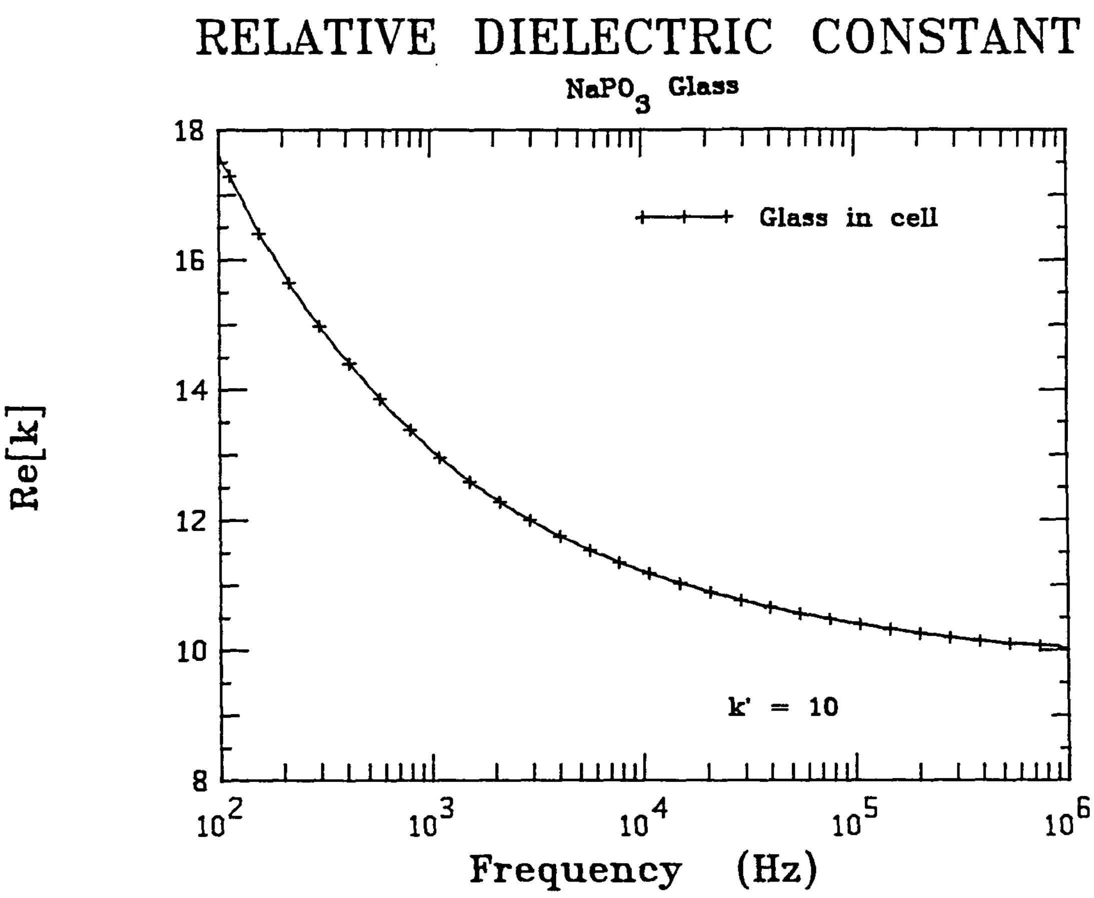


Figure 4.18: Frequency dependence of the real part of the relative dielectric constant for NaPO$_3$ glass

were reweighed to determine if any weight change occurred due to surface reaction with the atmosphere in the cell or the glove box. The initial and final weights of both samples were identical indicating that the cell was indeed hermetically sealed. This test was performed using silicon sulfide glass because this glass is extremely hygroscopic and decomposes rapidly in air to give H_2S gas and SiO_2 and is typical of the fast ion conducting glasses to be studied using this workstation.

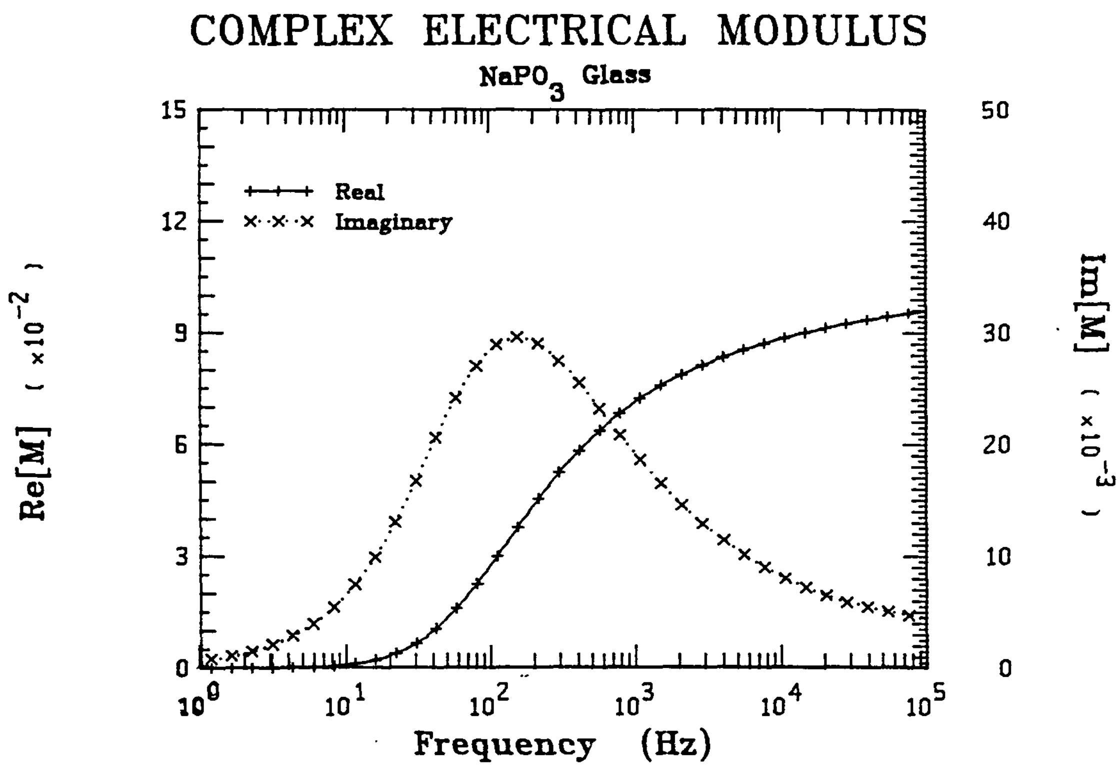


Figure 4.19: Frequency dependence of the electrical modulus for $NaPO_3$ glass at 30^oC

CHAPTER 5. CONCLUSIONS

The impedance workstation described in this thesis is capable of making convenient, fast, accurate and precise electrical impedance measurements over wide impedance and frequency ranges. Experimental variables for temperature control and impedance measurements are easy to set through user friendly interfaces between the computer and the user. The hardware performance of the workstation is summarized below.

- **Impedance Ranges**
  - Resistance : 1 Ω to 100 MΩ
  - Capacitance : 1 pF to 10 mF
- **Frequency**
  - Frequency range: 1 Hz to 1 MHz
  - Maximum number of frequencies scanned per sweep = 50000
- **Temperature**
  - Range : 25 to 230^oC
  - Isothermal stability : $\pm 0.05^oC$
  - Maximum time to stabilize : 80 minutes
- **Atmosphere**
  - Helium or any other inert gas

- **Sample size**
  - Maximum diameter = 4cm
  - Maximum thickness = 0.5cm
- **Maximum error in measurements**
  - Resistance = ±5%
  - Capacitance = ±5%

The workstation through the robust menu driven data collection software allows automated calibration of the conductivity cell, easy input of operator selected experimental parameters and unattended operation of the complete impedance experiment.

Data analysis is facilitated through graphical display of all the impedance related functions. Conduction activation energy and the limiting conductivity are calculated and are made available to the user through an Arrhenius plot. Finally, graphs can be customized on the CRT and prepared in publication-quality for reports by a plotter or printer.

Improvements in this impedance spectroscopy workstation are, nevertheless, possible. The frequency window may be widened by one more decade by increasing the upper frequency range to 10 MHz. This is a difficult task since signal leads act as transmission lines and the effect of line inductance and parasitic capacitance is dominant at high frequency. Sub-ambient temperature capability should be added, especially, for the study of highly conductive materials for which, at room temperature and above, only the resistive part of the spectra can presently be seen. Finally, a routine to perform 3-D graphics along with the standard graphics already present would greatly enhance the data presentation and analysis routines.

BIBLIOGRAPHY

[1] Baurel, J. E. 1969. J. Phys. Chem. 30: 341-351.

[2] Hodge, I. M., M. D. Ingram and A. R. West. 1976. J. Electroanal. Chem. 74: 341-351.

[3] Macedo, P. B., C. T. Moynihan and R. Bose. 1972. Phys. Chem. Glasses. 13: 171-175.

[4] Ravine D. and J. L . Souquet. 1974. J. Chim. Phys. 71:693-701.

[5] Cole, K. S. and R. H. Cole. 1941. J. Chem. Phys. 9: 341-351.

[6] Martin, S. W. 1986. Ph.D. Dissertation. Purdue University, Indiana.

[7] Kulkarni, A. R., H. S Maiti and A. Paul. 1984. Bull. Mater. Sci. 6: 201-221.

[8] Tuller, H. L. and M. W. Barsoum. 1985. J. Non-Cryst. Solids. 73: 331-341.

[9] Pye, L. D., H. J. Stevens and W. C. LaCourse. 1972. Introduction To Glass Science. Plenum, New York.

[10] Morey, G. W. 1950. The Properties of Glass. Reinhold Publishing Corporation, New York.

[11] Serway, R. A. 1982. Physics: For Scientists and Engineers. CBS College Publishing, New York.

[12] MacDonald, J. R. 1987. Impedance Spectroscopy Emphasizing Solid Materials and Systems. John Wiley and Sons, Inc., New York.

[13] Nilsson, J. W. 1983. Electric Circuits. Addison-Wesley, Cambridge, Mass.

[14] Churchill, R. V. and J. W. Brown. 1984. Complex Variables and Applications. McGraw-Hill Book Company, New York.

[15] Buchanan, R. C. 1986. Ceramic Materials For Electronics. Marcel Dekker,Inc., New York.

[16] Isard, J. O. 1988. Solid States Ionics. 31: 187-196.

[17] Elliot, S. R. 1988. Solid State Ionics. 27: 131-149.

[18] Weingarten, I. R. 1955. Annual Report of the Conference on Electrical Insulation. 53.

[19] Morse, C. T. 1973. J. Phys. E. 7: 657-662.

[20] Klein, R. M. and D. A. Ploof. 1977. Ceramic Bulletin. 57(6): 582-586.

[21] Engstrom, H. and J. C. Wang. 1980. Solid State Ionics. 1: 441-459.

[22] Staudt, U. and G. Schon. 1980. Solid Statè Ionics. 2: 175-183.

[23] Boukamp, B. A. 1984. Solid State Ionics. 11: 339-346.

[24] Weast, R. C. 1977. Handbook of Physics and Chemistry. CRC Press Inc., Cleveland.

[25] Press, W. H., B. P. Flannery, S. A. Teulosky and W. T. Vetterling. 1986. Numerical Recipes The Art of Scientific Computing. Cambridge University Press, New York.

[26] MacDonald, J. R., J. Schoonman and A. P. Lehnen. 1981. J. Electroanal. Chem. 131:77-95.

[27] Hurt, R. L. and J. R. MacDonald. 1986. Solid State Ionics. 20: 111-124.

[28] Kingery, W. D., H. K. Bowen and D. R. Uhlman. 1976. Introduction to Ceramics. John Wiley and Sons Inc., New York.

ACKNOWLEDGEMENTS

I would like to thank my major professor Dr. Steve W. Martin for tolerating my numerous mistakes and persistent questioning, and for his consistent help and advice, especially, during the writing of this thesis.

Thanks are also due Dr. Micheal Berard, Dr. Glenn E. Fanslow, Dr. Edwin C. Jones and Dr. David M. Martin for serving on my graduate committee.

Sincere thanks are due to John Rundle for suggesting improvements in the design of the conductivity cell and for daily help even in the most trivial of problems.

A special thanks to Jim Hudgens and Mark Wagner for preparation of glasses while my hand was broken, to Joe Kincs for general help in the laboratory, and to Dave Juedes for help in the drawings of some of the figures in this thesis.

I am grateful for the moral support and constant encouragement throughout the last year and during the writing of my thesis to two very good people, my brother Dipak Patel and Heidi Williams.

Most of all, I wish to thank my parents, Mr. Naginbhai Patel and Mrs. Urmilaben Patel who have been a source of encouragement throughout my education; I could not have accomplished so much without their love and support.

Acknowledgement is also gratefully made to the National Science Foundation and Iowa State University for the financial support of this work.

APPENDIX A. DEFINITIONS

VARIABLE	UNITS	DEFINITION
A	cm^2	Electrode area normal to electric field
B	℧	Conductance
C	F	Capacitance of cell with solid dielectric
C_o	F	Capacitance of cell - vacuum dielectric
d	cm	distance between electrodes
E	V/cm	Electric field strength
E_a	eV	Conduction activation energy
G	℧	Susceptance
I	C/s (A)	$\delta Q/\delta t$: current
k_o	cm^{-1}	Cell constant of dielectric: d/A
k^*, k', k''	cm^{-1}	Complex relative dielectric constant
M^*, M', M''	cm/F	Complex, real and imaginary modulus
P	FV/cm^2	Electric polarization
Q	C	Total charge
R	Ω	Resistance

VARIABLE	UNITS	DEFINITION
T_{start}	^oC	Starting temperature of sample
T_{final}	^oC	Final temperature of sample
T_{inc}	^oC	Starting temperature of sample
X	Ω	Reactance
Y^*, Y', Y''	$\mho$	Complex, real and imaginary admittance
Z^*, Z', Z''	Ω	Complex, real and imaginary impedance
$\tan\delta$	unitless	Loss tangent: ϵ''/ϵ'
ϵ_o	F/cm	Dielectric permittivity of free space
ϵ^*, ϵ', ϵ''	F/cm	Complex, real, imaginary dielectric constant
ρ^*, ρ', ρ'',	Ωcm	Complex, real, imaginary resistivity
σ_o	$\mho$ /cm	High temperature limiting conductivity
σ^*, σ',σ'',	$\mho$ /cm	Complex, real, imaginary conductivity

APPENDIX B. DIELECTRIC THEORY

The capacitance, C, is defined as the ratio of the magnitude of the electrical charge stored on two conducting plates separated by a nonconducting material to the magnitude of the potential difference between them:

$$C = \frac{Q}{V}. \tag{B.1}$$

The unit of capacitance is coulombs per volt and is called farads (F) after Michael Faraday. The capacitance of a vacuum capacitor is determined purely by its geometry; for parallel plate capacitors, the capacitance is proportional to the area A of its plates and inversely proportional to the plate separation d [11]. The proportionality constant is called the dielectric permittivity and is denoted by ϵ and has the units of F/cm. In a vacuum, the capacitance is given by

$$C_O = \epsilon_O \frac{A}{d}, \tag{B.2}$$

where the subscript implies free space, ϵ_O is called the absolute permittivity of free space (8.854 x 10^{-14} F/cm) and the ratio d/A is called the cell constant k_O. The capacitance of a capacitor with the same geometry but with a nonconducting material inserted between the plates is given by

$$C = \epsilon \frac{A}{d}, \tag{B.3}$$

where ϵ is the dielectric permittivity of the material. The ratio of the capacitance with a nonconducting material between the conductors to the capacitance with free space between the conductors is the same as the ratio of the two dielectric constants and is called the relative dielectric constant k':

$$k' = \frac{C}{C_o} = \frac{\epsilon}{\epsilon_o}. \tag{B.4}$$

The relative dielectric constant is greater than one for all materials. If a sinusoidal voltage $V^* = V_o e^{j\omega t}$ is applied to a capacitor then Equation B.1 can be rewritten as:

$$Q = CV_o e^{j\omega t}. \tag{B.5}$$

The current during discharge of the capacitor can be determined by taking the first derivative of Equation B.5 with respect to time and rearranging Equation B.4 and substituting for C to give:

$$I^* = \frac{\delta Q}{\delta t} = j\omega k' C_o V^*. \tag{B.6}$$

By rearranging Equation B.6, the impedance of the capacitor is

$$Z_C = \frac{V^*}{I^*} = \frac{1}{j\omega k' C_o}. \tag{B.7}$$

The current is an imaginary number implying that it leads the applied voltage by 90^o. No real current is measured through the capacitor and therefore no heating losses are observed. In real capacitors, however, such ideal behavior is not observed. Electric heating is detected and the measured phase shift is less than 90 degrees. This can be mathematically corrected by substituting k' with $k^* = k' - jk''$, a complex number. The current is then given by

$$I^* = \omega k'' C_o V^* + j\omega k' C_o V^*. \tag{B.8}$$

where k'' in the above equation is called the relative loss factor because it gives rise to a conduction current I_R. k' is called the relative dielectric constant and is proportional to the charge storing ability of the capacitor. Figure B.1 shows the graphical interpretation of these phenomena.

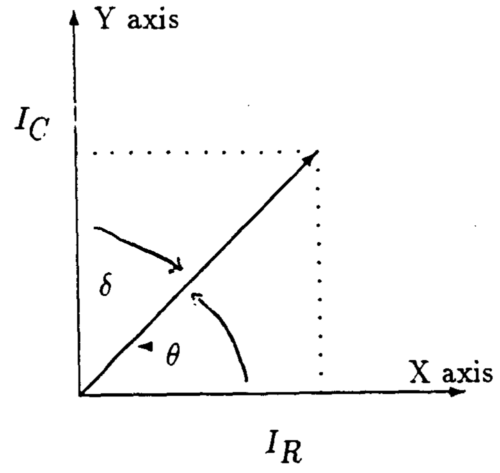


Figure B.1: Charging and loss current for a capacitor

From the magnitude of these currents, a quantity called the dissipation factor can be defined as:

$$tan(\delta) = \frac{I_R}{I_C} = \frac{k''}{k'} \tag{B.9}$$

where δ, I_R and I_C is defined in Figure B.1. This is a measure of the conduction current relative to that of the displacement current or the ratio of the ohmic loss in the material to the energy stored. A material is said to be a good conductor if $k'' \gg k'$ and a good insulator if $k' \gg k''$. The reciprocal of this number is called the quality factor Q and is a frequently used parameter during design of electronic devices and circuits. Good insulators usually have a dissipation factor less than 0.001 and a relative dielectric constant less than 30. Materials used in capacitors have k' value greater than 30 and dissipation factors less then 0.001 and are called dielectrics [15].

The values of k' and k'' can be calculated from the impedance data and these equations are derived here. Equation B.8 is divided through by the voltage, to give the admittance. The admittance of a dielectric material is then given by

$$A^* = \omega k'' C_O + j\omega k' C_O. \tag{B.10}$$

Dividing both sides of Equation B.10 by ωC_O and multiplying by $-j$ yields the result:

$$k^* = k' - jk'' = -j\frac{A^*}{\omega C_O} = -j\frac{1}{\omega C_O Z^*}. \tag{B.11}$$

It has been shown from the derivation above that all the impedance related functions used in IS can be derived from two material properties - the relative dielectric constant k' and the relative loss factor k''.

So far the dielectric permittivity has been described mathematically, a macroscopic description of the behavior of dielectric materials in the presence of an electric fields is presented next.

Polarization A dielectric reacts to an electric field because it contains charge carriers and dipoles that can be displaced. A dipole consists of two equal and opposite charges separated by a distance. The phenomenon of displacement of dipoles is called polarization which is the ability of a material to neutralize part of an applied field. The voltage in a capacitor containing a dielectric is given by

$$V = \frac{Q/k'}{C_O} \tag{B.12}$$

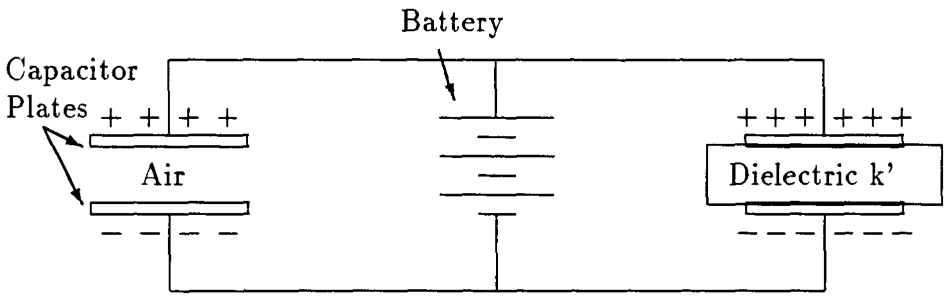


Figure B.2: Charge stored on capacitor with dielectric is increased by k'

where only a fraction $(1/k')$ of the total charge Q sets up an electric field E to give the potential difference V. The rest of the charge is neutralized by the polarization of the dielectric. This means, for the same applied voltage, the charge stored in a capacitor with a dielectric is greater than a capacitor with free space (see Figure B.2).

Charges on the plates of a capacitor are cancelled by dipoles induced by the electric field E. The electric flux density or electric displacement D is given by (1) the charge density established by the applied electric field without the dielectric and (2) the additional charge density equal to the neutralized charge due to the electric field induced by the dipoles. The polarization is a function of the electric field and therefore the dielectric displacement can be written as shown:

$$D = \epsilon_O E + P = \epsilon_O k^* E. \qquad \text{(B.13)}$$

The unit of D is charge per unit area. The total electrical displacement D in a material is related to the applied field E by the complex relative dielectric permittivity k^*. By rewriting the Equation B.13, the polarization can be expressed as

$$P = \epsilon_O k^* E - \epsilon_O E = \epsilon_O (k^* - 1) E. \qquad \text{(B.14)}$$

The polarization P has the units of surface charge per unit area, but it can be equated to the number of dipoles per unit volume and is defined as

$$P = N\mu \tag{B.15}$$

where N is the number of dipoles per unit volume and μ is the average dipole moment of the dipoles. The average dipole moment is a function of the local electric field E$'$ and a proportionality factor α. The electric field locally acting is not the same as the applied electric field due to the polarization of the surrounding medium. α is called the polarizability, which is a true material property, and is a measure of the dipole moment per unit of local field strength. Dipoles of different nature and environment are likely to be present in a dielectric and therefore a more appropriate expression for the polarization is

$$P = \sum_{i=1}^{n} N_i \alpha_i E_i' \tag{B.16}$$

where the subscript i represents each set of dipoles. If it is assumed that the local electric field E$'$ is the same for all the dipoles then E$'$, as first calculated by Mosotti in [28], is given by

$$E' = E + \frac{P}{3\epsilon_o} = \frac{E}{3}(k^* + 2). \tag{B.17}$$

Substituting Equation B.17 and Equation B.14 into Equation B.16 and rearranging the terms gives the final result which is better known as the Clausius-Mosotti equation [28]:

$$\frac{k^* - 1}{k^* + 2} = \frac{1}{3\epsilon_o} \sum_{i=1}^{n} N_i \alpha_i. \tag{B.18}$$

This equation describes the relationship between the relative dielectric permittivity of a material, the number of polarizable species N_i and the in polarizability, α_i.

The four major types of polarization in solids are (1) electronic α_e (2) molecular, atomic or ionic α_a (3) orientation α_d and (4) space charge polarization α_i. Each dipole in the presence of a slowly varying electric field will keep up with the field and contribute to a pure capacitance. The current therefore leads the applied field by 90^o. As the frequency of the changing field increases, some types of dipoles may not be able to keep up and the current will lag by an angle δ implying that a component of the current is in phase with the applied field. This is due to an inertia-to-charge movement for orientation and space charge type polarization. The frequency at which maximum loss occurs is called the relaxation frequency. As the frequency increases even further, the natural absorption frequency of the atomic bonds and electrons will be reached and further losses will be observed. This frequency is labeled the resonant frequency. Each of the four classes of polarizable species is described next.

Electronic Polarization This occurs in all materials because they consist of ions which are surrounded by electron clouds. The center of gravity of the negative electron cloud in relation to the positive atom nucleus shifts in the presence of a varying field. As electrons are very light they can respond to alternating fields quickly and contribute to the dielectric constant at frequencies well into the optical spectrum - 10^{15} Hz.

Molecular, Atomic and Ionic Polarization This physical process occurs in the infrared region ($10^{12} - 10^{13}$ Hz). Bonds between atoms are stretched by applied electric fields to give rise to a dipole. Resonance losses occur at a frequency that depends upon bond strength between the ions and the reduced pair mass of the atom pair. The infrared absorption will be broad if there are several types of ions and a distribution of bond strengths.

Orientation Polarization This type of polarization is due to permanent molecular or ionic or induced dipoles. They orient with the oscillating field with the help of thermal energy. Molecules containing permanent dipoles may be rotated against a restoring force about an equilibrium position. These dipoles have a relaxation frequency of approximately 10^{11} Hz. Rotation of dipoles between two equivalent equilibrium positions is another mechanism. Such oscillations occur continuously as thermally activated ions hop in the direction favored by the electric field. It is these ions which contribute to the dc conductivity. The frequency at which the hopping ion can no longer keep up with the alternating field is called the relaxation frequency and this may be observed in the impedance part of the electromagnetic spectrum.

Space Charge Polarization Space charge occurs when mobile charge carriers are impeded by a physical barrier that inhibits charge migration. The charges are not supplied or discharged at an electrode. This charges contribute to a large capacitance but only at frequencies less then 10Hz.

The relation between the relative dielectric permittivity and the polarizability can now be written in terms of the species described above as

$$\frac{k^* - 1}{k^* + 2} = \frac{1}{3\epsilon_o} N_e\alpha_e + N_a\alpha_a + N_d\alpha_d + N_i\alpha_i. \tag{B.19}$$

Low frequency measurements of the relative dielectric permittivity is a measure of all the polarizable species. As frequency increases the relative dielectric permittivity

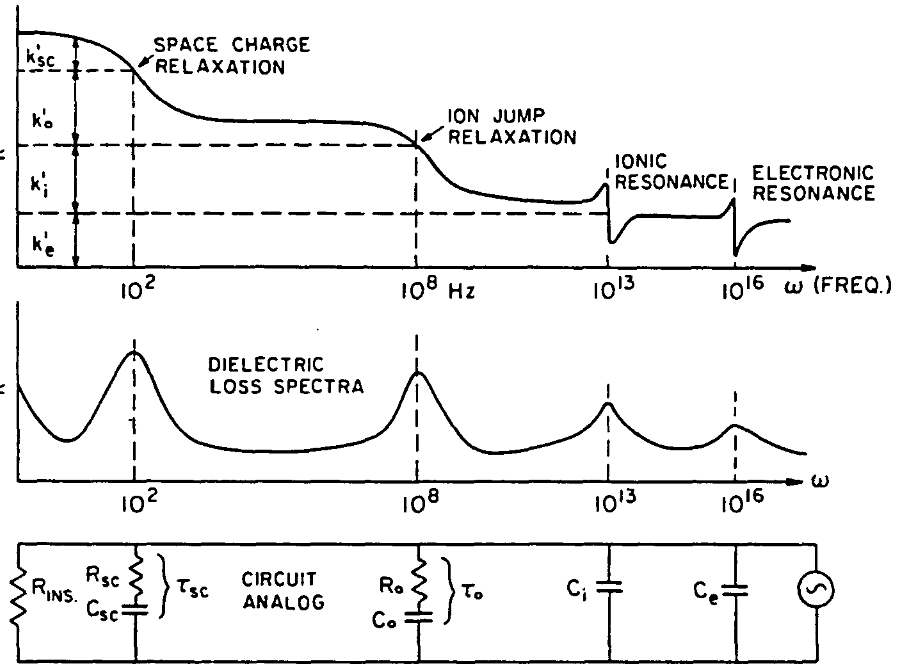


Figure B.3: Dielectric dispersion

decreases as the relaxation or resonant frequency is passed for the various species. At the upper extreme of the frequency range during impedance measurements, k^* is only a function of the electronic and atomic polarization. The behavior of k' and k'' versus frequency is shown in Figure B.3 along with an equivalent circuit representation [15]. Observe the inclusion of the series resistance in the space charge and ion jump relaxation circuits, this is to represent the actual work done to move the dipole. The relaxation frequency is given by $1/(2\pi RC)$ and the resonant frequency is the natural absorption frequency of the electronic or bond dipole.

For an in-depth discussion the reader is referred to one of the following references [9, 15, 28].

APPENDIX C. MATHEMATICAL METHODS

Data Conversion

Data are stored in terms of the magnitude and phase of the impedance and to determine the other impedance related functions, the data have to be converted from the basic form to the new. The conversions are given here and are based on the definitions given in Chapter 2 and Appendix B.

Impedance

$$Z' = \mid Z \mid cos\theta \tag{C.1}$$

$$Z'' = \mid Z \mid sin\theta \tag{C.2}$$

Resistivity

$$\rho' = \frac{\mid Z \mid}{k_O} cos\theta \tag{C.3}$$

$$\rho'' = \frac{\mid Z \mid}{k_O} sin\theta \tag{C.4}$$

Admittance

$$A' = \frac{1}{\mid Z \mid} cos(-\theta) \tag{C.5}$$

$$A'' = \frac{1}{\mid Z \mid} sin(-\theta) \tag{C.6}$$

Conductivity

$$\sigma' = \frac{k_O}{\mid Z \mid} cos(-\theta) \tag{C.7}$$

$$\sigma'' = \frac{k_o}{\mid Z \mid} sin(-\theta) \tag{C.8}$$

Relative Dielectric Permittivity

σ'

$$k' = \frac{k_o}{\omega\epsilon_o \mid Z \mid} cos(-\theta) \tag{C.9}$$

$$k'' = \frac{k_o}{\omega\epsilon_o \mid Z \mid} sin(-\theta) \tag{C.10}$$

Modulus

$$M' = \frac{\omega\epsilon_o \mid Z \mid}{k_o} sin\theta \tag{C.11}$$

$$M'' = \frac{\omega\epsilon_o \mid Z \mid}{k_o} cos\theta \tag{C.12}$$

Loss Tangent

$$tan\delta = \frac{cos(-\theta)}{sin(-\theta)} \tag{C.13}$$

R$_p$ - C$_p$

$$R_p = \frac{\mid Z \mid}{cos\theta} \tag{C.14}$$

$$C_p = -\frac{sin\theta}{\omega \mid Z \mid} \tag{C.15}$$

R$_s$ - L$_s$

$$R_s = \mid Z \mid cos\theta \tag{C.16}$$

$$L_s = \frac{\mid Z \mid sin\theta}{\omega} \tag{C.17}$$

Least-Squares Algorithms

The algorithm to determine a and b for the equation of a best-fit straight line

$$y = bx + c \tag{C.18}$$

is determined by minimizing the sums of the squares of the deviations. The Pascal algorithm to do this is given below in [25]. The subroutine returns the values for a, b, and their standard deviations siga and sigb, the chi-square and the goodness of the fit q when the data set x and y are input. A thorough discussion of the algorithm is given in reference [25].

```
PROCEDURE fit(x,y: glndata; ndata: integer; sig: glndata;
         VAR a,b,siga,sigb,chi2,q: real);
(* Programs using routine FIT must define the type
TYPE
   glndata = ARRAY [1..ndata] OF real;
in the main routine.   *)
VAR
   i: integer;
   t,sy,sxoss,sx,st2,ss,sigdat: real;
BEGIN
   sx := 0.0;
   sy := 0.0;
   st2 := 0.0;
   b := 0.0;
   FOR i := 1 to ndata DO BEGIN          { Accumulate sums }
      sx := sx+x[i];
      sy := sy+y[i]
   END;
   ss := ndata
   sxoss := sx/ss;
   FOR i := 1 to ndata DO BEGIN
      t := x[i]-sxoss;
      st2 := st2+t*t;
      b := b+t*y[i]
   END
   b := b/st2;            { Solve for a,b, siga and sigb }
   a := (sy-sx*b)/ss;
   siga := sqrt((1.0+sx*sx/(ss*st2))/ss);
   sigb := sqrt(1.0/st2);
   chi2 := 0.0;
   FOR i := 1 to ndata DO BEGIN              { calculate chi-square }
      chi2 := chi2+sqr(y[i]-a-b*x[i])
   END;
   q := 1.0;
   sigdat := sqrt(chi2/(ndata-2));
   siga := siga*sigdat;
   sigb := sigb*sigdat
   q := gammq(0.5*(ndata-2),0.5*chi2)
END;
```

The matrix algorithm for the determination of the best-fit semi-circle is given by J. MacDonald et al. [26] and the software algorithm is given by S. Martin [6]. The loci of a circle are defined by the following equation

$$r^2 = (x - x_O)^2 + (y - y_O)^2. \tag{C.19}$$

The algorithm to calculate x_O, y_O and r is given in a terms of summations. The x-axis shift is given by

$$x_O = \frac{(\Sigma y^3 + \Sigma y x^2)\Sigma y x^2 - (\Sigma x y^2 + \Sigma x^3)\Sigma y^2}{\Sigma (xy)^2 - \Sigma x^2 y^2} \tag{C.20}$$

and the y-axis shift is

$$y_O = \frac{(\Sigma x^3 + \Sigma x y^2)\Sigma y x - (\Sigma x y^2 + \Sigma y^3)\Sigma x^2}{\Sigma (xy)^2 - \Sigma x^2 y^2} \tag{C.21}$$

and r is calculated using the center of the circle with the equation given below

$$r = \sqrt{x_O^2 + y_O^2}. \tag{C.22}$$

The x-axis intercept is calculated by solving for x when y is equal to zero and is given below:

$$x = \sqrt{r^2 - y_o^2} + x_o. \tag{C.23}$$

APPENDIX D. PASCAL SOURCE CODE

The source code for the data-collection and data-analysis package is written in Turbo Pascal Version 5.0. The data-collection module, in addition, uses a Pascal source library provided by BBS ELECTRONICS INC. to drive their GPIB-1000 IO-card. The data-analysis module uses HGRAPH Pascal routines to display graphics and plot to different devices. Since, these are not standard routines, a description of those routines used in each program is summarized before presenting the source code.

Throughout the programming environment, extensive use of screen files for introducing the software, data input windows and menus have been used to create a user friendly interface. This screen files were made by using a software package from the PC-SIG shareware library called BOX.EXE[1] and the Pascal source code to display the files within a Turbo Pascal environment. A list of the screen files used in this program are shown below.

CLR_HELP.SCR	C_MENU_1.SCR	D_CHOICE.SCR
GRP_SET1.SCR	GRP_SET2.SCR	IMPED1.SCR
INTRO.SCR	INTROGRP.SCR	IS_WKSTN.LOG
IS_WKSTN.SCR	THERMAL.SCR	

The source code is broken into four sections, the first is for the package which

[1]Nescatunga Software, Box 5942, Katy, TX 77450.

links the two modules together. The next two sections contain the data-collection and the data-analysis source code, respectively. Some procedures are used by both modules and are stored in libraries called Turbo Pascal units and the source code for these units are found in the last section.

IsWkstn.Pas

Both the data-collection and the data-analysis programs are independent of each other and are stored in their executionable form. The two programs are linked together by this source code.

```
program is_wkstn;
{$M $4000,0,0}
uses
  getfiles,dos,crt;
var
 ch : char;

 procedure write_errors(stng:string);
   begin
     gotoxy(24,0);
     write(stng);
     delay(3000);
     ch := 'e';
   end;

  begin
    repeat
      Screen_file('is_wkstn.scr');
      ch := readkey;
      swapvectors;
      case ch of
        '1', 'c', 'C' : begin
                          ch := '1';
                          Exec('dec10.exe','')
```

```
                    end;
      '2', 'A', 'a' : begin
                       ch := '2';
                       Exec('conv18.exe','')
                     end;
      '3', 'e', 'E' : ch := '3';
    end;
    clrscr;
    swapvectors;
    if (ch = '1') or (ch='2') then
      case doserror of
        2 : write_errors('File '+ch+' Not Found');
        8 : write_errors('Not Enough Memory !' );
      end;
     swapvectors;
  until ch = '3';
End.
```

Data-Collection Code

Automation of the workstation is done using the IEEE-488 parallel interface using a GPIB-1000 interface card and software from BBS ELECTRONICS INC. Only five procedures, to initialize or abort, to send or receive data and to serial poll the status byte of an instrument, are used in this program. The format for each procedure is described here to help make the source code more readable, but for complete discussion the reader is referred to the BBS **READ.ME** file.

$$enter(instring, device\#) \tag{D.1}$$

Procedure D.1 is used to receive data from the GPIB, where the data are received from the instrument with the GPIB address device# and is stored as a string in the data variable instring.

$$output(outstring, device\#) \tag{D.2}$$

Data are output from the computer to an instrument using procedure D.2, where the outstring is the data sent and device# is the address of the instrument. Finally, the serial polling of the status byte of an instrument is used to determine whether the Solartron has completed a series of measurements. A function is used to perform this task and is given below.

$$spoll(device\#) \tag{D.3}$$

To initialize the GPIB, the procedure *INIT* is used and to abort an operation *ABORTIO* is used.

```
{         D A T A  C O L L E C T  M O  D U L E


 -------------------------------------------------------
 GPIB Addr.     Instrument              Type
-------------------------------------------------------

   28        Impedance Bridge        SOLARTRON 1260
   15        A/D converter           HP3478A
   23(2)     D/A Converter           TEK 50M20
-------------------------------------------------------        }

{$R-}    {Range checking off}
{$B+}    {Boolean complete evaluation on}
{$S+}    {Stack checking on}
{$I+}    {I/O checking on}
{$N-}    {No numeric coprocessor}
{$M 65500,16384,655360} {Turbo 3 default stack and heap}
{$V-}

Uses
 getfiles, edtool, Crt, Dos;
{$I IEEE488.PAS}    {include the gpib driver}

type
      strng                     = string[255];
      strng1                    = string[14];

const
      bridge      : integer=28;   { Impedance Bridge = Adr(28) }
      multimeter  : integer=15;   { Digital multimeter         }
      multifunc   : integer=23;   { Multifunction Interface    }
      crlf        : string[2]=#13#10;

var
      init_temp,final_temp,step,i,
                          data_points,integ_time         :integer;
      instring                                           :bus_string;
      done                                               :boolean;
```

```
     file_name                                           :string[80];
     outfile, infile , Arhenius, trans, logfile, sst     :text;
     temperature,volt,seconds,zero,constant,start, final:real;
     ans , Sol_setup, ch                                 :char;


{==========================================================
 This subroutine initializes all the IEEE interfaceable devices, the
     initial setup can be modified in the CONST block.
 ----------------------------------------------------------}

procedure initialize_all_devices;
const

    meter_setup      :string[30]='INIT;DCV;CALC ON;AVE 10;MODE TRIG';
    multifunc_setup  :string[10]='INIT;';
    d_a_setup        :string[20]='SEL 2;DT OFF';

begin
     zero:=1;
     sol_setup := '9';
     integ_time := 0;
     data_points := 0;
     init;              {This are GPIB commands to initialize the}
     abortio;                            {cable}
     output(meter_setup+crlf,multimeter);
     output(multifunc_setup+crlf,multifunc);
     output(d_a_setup+crlf,multifunc);
     output(crlf,multifunc);
end;

{==========================================================
          Uses GetTime procedure to return a time string
 ----------------------------------------------------------}

function time:strng1;
var
     hour_ss, minute_ss, second_ss, sec100  :word;
     temp1,temp2                            :strng1;
```

```
begin
     GetTime(hour_ss, minute_ss, second_ss, sec100);
     str(hour_ss, temp1);
     str(minute_ss, temp2);
     if minute_ss < 10 then
        temp2 := concat('0', temp2);
     temp1 := concat(temp1, ':', temp2);
     str(second_ss, temp2);
     if second_ss < 10 then
        temp2 := concat('0', temp2);
     time := concat(temp1, ':', temp2);
end;

{=====================================================================
     Procedure creates a directory where all the data files for a
    particular sample are going to be stored for each temperature.
 --------------------------------------------------------------------}
procedure create_data_storage_area;
var
   diskspace          :real;

begin
  repeat
    write_data_situation(red,white,'Checking If Enough Space !');
    diskspace := DiskSize(1);
    if diskspace < 1e5 then
      begin
        Beep; Beep;
        write_data_situation(red,white,
        'Less Then 100 KBytes of Memory, PUT NEW DISK in Drive A: ');
        delay(4000);
        write_data_situation(red,white,'Press SPACE BAR when ready');
        repeat until ReadKey = Chr(32);
     end;
     write_data_situation(black,black,'');
     until diskspace > 1e5;
end;

{=====================================================================
            Makes filenames for each isothermal temperature
 -------------------------------------------------------------------}
procedure create_isothermal_filename(suffix:integer);
var
  new_file_name    :strng;
  disk_space       :real;
  error_chk        :integer;

begin
   str(suffix,new_file_name);
   new_file_name:='a:'+file_name+'.'+new_file_name;
   open_file_to_write(outfile,new_file_name,error_chk);
end;

{===================================================================
  Calculates the temperature using the quartic equation given in the
              manual for the thermocouple type K
 -------------------------------------------------------------------}
procedure calculate_type_k(volt:real;var temperature:real);

const
        a0              : real=0;               { M41A3 manual   }
        a1              : real=2.4383248e-2 ;   { T > 0          }
        a2              : real=9.7830251e-9 ;
        a3              : real=3.6276965e-12;
        a4              : real=-2.5756438e-16;
        b0              : real=-1.4999;     { bx - calibrated fifth }
        b1              : real=1.11998;     {      order fit        }
        b2              : real=-2.419e-3;
        b3              : real=1.6655e-5;
        b4              : real=-4.644e-8;
        b5              : real=4.3487e-11;
        c0              : real=-1.4999;         { T > 0 }
        c1              : real=2.3783697e-2 ;
        c2              : real=-2.4382217e-6 ;
        c3              : real=-6.8203073e-10 ;
        c4              : real=-9.4854031e-14;
begin
     volt:=volt*1e6;
     if volt > 0 then
         temperature:= a0+(a1+(a2+(a3+a4*volt)*volt)*volt)*volt
     else
         temperature:= c0+(c1+(c2+(c3+c4*volt)*volt)*volt)*volt ;
     if temperature > 30 then
        temperature:=b0+(b1+(b2+(b3+(b4+b5*temperature)*temperature)*
                      temperature)*temperature)*temperature; end;

{=====================================================================
     Use the HP3478A multimeter to read the voltage for the type K
  -------------------------------------------------------------------}
procedure get_type_k(var temperature:real);

var
    in_string           :strng;
    volt                :real;
    j                   :integer;

begin
 repeat
   output('DCV',15);
   enter(in_string,15);
   j := pos(#13,in_string);
   delete(in_string,j,5);
   val(in_string,volt,j);
   calculate_type_k(volt,temperature);
 until temperature > -240;
end;

{=====================================================================
Calculates the temperature for  the temperature controller in order to
    get the true set temperature - coefficients form least square fit
---------------------------------------------------------------------}
procedure calculate_type_T(var temperature:real);

const
   b0             :real=-12.47;      { bx - coefficients to convert }
   b1             :real=1.52332;     {      temperature to offset   }
   b2             :real=-7.732e-3;   {      for the controller      }
  b3             :real=5.4641e-5;
  b4             :real=-1.828e-8;
  b5             :real=2.3275e-10;

begin
  temperature:=b0+(b1+(b2+(b3+(b4+b5*temperature)*temperature)*
                     temperature)*temperature)*temperature;
end;


{=====================================================================
     Check if Solartron is done making a sweep or a measurement
---------------------------------------------------------------------}
procedure check_if_operation_completed(status_byte : integer);
 var
   k    : integer;
   temp : real;
 begin
   final := start;                        {*Temperature at sweep begin *}
   repeat
     k := spoll(bridge);                  { Check if sweep is complete }
     gotoxy(30,25);
     write('Please Wait !! ',k);
     if (init_temp <> final_temp ) and (k = 2) then {thermal prog ?}
       begin
         get_type_k(temp);
         if abs(temp - start) > abs(final - start) then
              final := temp
       end;
  until k = status_byte ;                 { srq+sweep+measure = 70  }
 end;

{=====================================================================
     Procedure performs the ZERO-OPEN compensation function in the
     Impedance Bridge to subtract parasitic capacitance within the
                         conductivity cell.
---------------------------------------------------------------------}
procedure send_data(data: bus_string);    { simplifies data xmit }
    begin
      output(data+crlf,bridge);
      delay(1000);
    end;

{===================================================================
                    Option 2: Initalize Solartron
 -------------------------------------------------------------------}
procedure Initialize_Solartron(ch:char);
 var
   i  :integer;
   command : string[10];
   comment : string[67];

   begin
     clrscr;
     writeln;
     writeln('Edit Configuration File (y/n) ?');
     readln(ans);
     repeat
       if (ans = 'y') or (ans = 'Y') then edit_file('sol_set.up'+ch);
       send_data('RS'+ch);
       Open_file_to_Read(infile,'Sol_Set.Up'+ch,i);
       textbackground(blue);
       textcolor(white);
       clrscr;
    writeln('PLEASE WAIT !  COMMANDS SENT TO SOLARTRON AS PRINTED !');
     writeln;
     readln(infile,comment);
     writeln(comment); writeln;
     readln(infile);
     readln(infile,command,comment);
     while not eof(infile) do
      begin
        writeln(command,comment);
        trim(command);
        send_data(command);
        if pos('SF',command) <> 0 then
          begin
            delete(command,1,2);
            val(command,data_points,i)
        end;
      readln(infile,command,comment);
    end;
    close(infile);
    writeln;
    writeln('Reset configuration on Solartron (y/n) ?');
    readln(ans);
    until (ans = 'n') or (ans = 'N');
end;

{=====================================================================
                 Option 3: Calibrate Conductivity Cell
 ---------------------------------------------------------------------}
procedure zero_open(number :char); var ans  : char;
 begin
    ClrScr;
    send_data('rs'+number);        { Recall setup number             }
    send_data('*sre4');            { Enable SRQ - sweep complete     }
    send_data('nl2');              { Evaluate Null                   }
    writeln('Please  S H O R T  circuit the Conductivity Cell !!!');
    writeln;writeln('Press <Enter>  to continue !! ');
    readln;
    send_data('cp');                   { Pause/Cont}
    check_if_operation_completed(70);
    send_data('*sre4');            { Enable SRQ - sweep complete}
    GotoXY(6,0);
    writeln('O P E N  Circuit Conductivity Cell !!!');
    writeln;writeln('Press <ENTER> when ready !! ');
    readln;
    send_data('cp');               { Pause/Cont                     }
    check_if_operation_completed(70);
    writeln;
    send_data('nl1');                  { Turn Null On               }
    send_data('cs'+number);            { Clear setup #              }
    send_data('ss'+number);
    writeln('Auto Integration on (y/n) ? ');
    readln(ans);
    if (ans = 'y') or (ans='Y') then
      send_data('au1');
    writeln('normalize complete');
 end;


{======================================================================
Magnitude and phase results measured over the frequency spectrum at a
             constant temperature using the Solartron 1260
 ---------------------------------------------------------------------}
procedure impedance_measure;
var
     j ,err  : integer;

 {--------------------------------}
  Break Input String: F, Z, theta
 {--------------------------------}
  procedure clean_imped_data(data:bus_string;var freq,mag,phase:real);

    var
      cut_data     : bus_string;
      i            : integer;

    begin
       cut_data := COPY(DATA,1,14);
       if pos(',',cut_data) = 14
         then
           begin
             data := ','+data;
             delete(cut_data,14,1);
           end;
       val(copy(cut_data,1,14),FREQ,i);
       CUT_DATA := COPY(DATA,16,11);
       val(copy(data,16,11),mag,i);
       CUT_DATA := COPY(DATA,28,11);
       val(copy(data,28,11),phase,i)
    end;

 {--------------------------------}
   Prompt Solartron to Measure
 {--------------------------------}
  procedure read_data;
  var
     j                               : integer;

  begin
    send_data('OP2,0');    { Turn output to display       }
    send_data('*sre4');    { Enable SRQ - sweep complete  }
    send_data('RE');       { Begin sweep                  }
    check_if_operation_completed(70);
  end;

{--------------------------------}
 Write raw & calc. data to Floppy
{--------------------------------}
procedure write_to_file;
  var
     data_in                         : bus_string;
     j , err                         : integer;
     magnitude, phase ,frequency     : array [1..200] of real;

  {--------------------------------}
   Non-Linear Least-Squares Routine
  {--------------------------------}
 procedure calculate_arhenius_data;
   const
    pi : real=3.141593;
   var
       lsq                                 : array [1..7] of real;
       j,k,l,One_MHz                       : integer;
       xc, yc, r, sigma, x1, x2, scale     : real;
       Re , Im                             : array [1..200] of real;

   begin
     for k := 1 to data_points do
       begin
         Re[k]      :=  Magnitude[k]*Cos(phase[k]*pi/180);
         Im[k]      := -Magnitude[k]*Sin(phase[k]*pi/180);
       end;
     k := data_points - 1;
     while frequency[k] > 1e6 do k := k - 1;
    One_MHz := k ;
    while  ((Re[k] > Re[k+1]) and (Im[k] > Im[k+1]))  do k:=k-1;
    while  ((Re[k] > Re[k+1]) and (Im[k] < Im[k+1]))  do k:=k-1;
    if ( One_MHz - k ) > 5 then
     begin
        for j:=1 to 7 do lsq[j] := 0;
        l := k;
        x1 := Re[k]; x2 := Re[k];
        for j := k to One_MHz do max_min(j,x1,x2,Re[j]);
        if x2 > 1e5 then scale := 1/x2 else scale := 1;
        for j := k to One_MHz do
         begin
            Re[j]  := scale * Re[j] ;
            Im[j]  := scale * Im[j] ;
            lsq[1] := lsq[1] + Re[j]*Re[j];       {SUM  Re^2  }
            lsq[2] := lsq[2] + Re[j]*Re[j]*Re[j]; {SUM  Re^3  }
            lsq[3] := lsq[3] + Re[j]*Im[j]*Im[j]; {SUM  ReIm^2 }
            lsq[5] := lsq[5] + Im[j]*Im[j];       {SUM  Im^2  }
            lsq[6] := lsq[6] + Im[j]*Im[j]*Im[j]; {SUM  Im^3  }
            lsq[7] := lsq[7] + Re[j]*Re[j]*Im[j]; {SUM  ImRe^2 }
         end;
         yc := lsq[4]*lsq[4] - lsq[1]*lsq[5];
                                         { yc =(ReY)^2 - (X^2 Y^2)}
     xc := ((lsq[6]+lsq[7])*lsq[4] -(lsq[3]+lsq[2])*lsq[5])/yc;
     yc := ((lsq[3]+lsq[2])*lsq[4]-(lsq[6]+lsq[7])*lsq[1])/(-yc);
     R  := 0.5*scale*sqrt(xc*xc + yc*yc);
     yc := yc*scale/2;
     xc := xc*scale/2;
     sigma := constant/(sqrt( r*r - yc*yc ) + xc);
     append(arhenius);
     writeln(arhenius, temperature, sigma, 1/sigma);
     close(arhenius);
   end;
end;

  begin
    send_data('CZ1');       { Data output in form of Z and theta }
    send_data('fd0');       { Goto beginning of data file  }
    send_data('op2,1');     { Turn on output to GPIB       }
    repeat
       send_data('FO') ;     { Goto beginning of data file  }
       j := spoll(bridge) ;
    until (j = 16) or (j = 22);  { Means Measurement AVailable  }
    FOR J := 1 TO data_points DO
     BEGIN
       enter(data_in,bridge);    { Read  Data }
       clean_imped_data(data_in,frequency[j],magnitude[j],phase[j]);
       if frequency[j] <> 0.0 then
         writeln(outfile,frequency[j]:14,' ',
                             magnitude[j]:11,' ',phase[j]:8:3);
     end;
    close(outfile);
    send_data('OP2,0');
    calculate_Arhenius_data;
  end;


 {--------------------------------}
 begin
   send_data('bk');                { Interrupt all operations}
   read_data;
   write_to_file;
   str(init_temp,instring);
   instring := 'a:\nulloff\'+file_name+'.'+instring;
   open_file_to_write(outfile,instring,err);
   send_data('NLO');
   write_to_file;
   send_data('NL1');
 end;

{==================================================================
  Procedure gets data for the temperature range and step between each
                      impedance measurement.
 ------------------------------------------------------------------}
procedure get_initial_data(var init_temp,final_temp,step:integer);

type
  data_string                     = string[80];
```

```
var
  name, composition, cell_constant, comments, data : data_string;
   year, month, day, Dayofweek                       : word;
   returnCode, error_chk                             : integer;
   position                                          : byte;
   directory                                         : strng;
   f                                                 : file;

            {================================================
                   User input window to enter data
             -----------------------------------------------}
procedure data_box(var position:byte; x,y,size:byte; data_type: char;
                    var data:data_string; var intgr_string: integer);
 begin
   GotoXY (x,y);
   if data_type = 'S' then ptoolent(data,data_type,size,0,ReturnCode)
     else ptoolent(intgr_string,data_type,size,0,ReturnCode);
      case returncode of
        72     : if position > 1 then position := position - 1
                   else beep;
        1,2,80 : position := position + 1;
      end;

    end;
   {====================================================
     Read Solartron Config. file for Integ t, and #pts
   ----------------------------------------------------}
    procedure get_solartron_data(ch : char);
     var
       infile : text;
       command : string[10];
       comment : string[67];
     begin
            Open_file_to_Read(infile,'Sol_Set.Up'+ch,i);
            readln(infile,comment);
            readln(infile);
            readln(infile,command,comment);
            while not eof(infile) do
              begin
```

```
                trim(command);
                if pos('SF',command) <> 0 then
                   begin
                     delete(command,1,2);
                     val(command,data_points,i)
                   end;
                if pos('IS',command) <> 0 then
                   begin
                     delete(command,1,2);
                     val(command,integ_time,i)
                   end;
                 readln(infile,command,comment);
             end;
           if data_points = 0 then
             begin
               data_points := 200;
               send_data('SF200')
             end;
           if integ_time = 0 then
             begin
               integ_time := 1;
               send_data('IS1')
             end;
           close(infile);
      end;
begin
  screen_file('imped1.pck');
  create_data_storage_area;
  name := ''; composition := ''; cell_constant := ''; comments := '';
  file_name:= ''; init_temp := 0; final_temp := 0; step := 0;
  TextBackground(Black);
  TextColor(Yellow);
  position := 1;
  repeat
    case position of
      1:  begin
            data_box(position,19,7,14,'S',file_name,error_chk);
            repeat
              directory := 'a:\'+file_name;
```

```
{$I-}
ChDir(directory);
{$I+}
error_chk := IOResult;
if error_chk = 0 then
  begin
    write_data_situation(red,white,
     'DIRECTORY exists!  OVERWRITE ? ');
    data := readkey;
    write(data);
    write_data_situation(red,white,'');
    TextBackground(Black);
    TextColor(Yellow);
    if (data = 'y') or (data = 'Y') then
      begin
        open_file_to_read(infile,'a:Contents.txt',
                                         error_chk);
        If error_chk = 0 then
         begin
           Readln(infile,name);
           delete(name,1,7);
           readln(infile,composition);
           readln(infile,composition);
           delete(composition,1,14);
           readln(infile,cell_constant);
           delete(cell_constant,1,22);
           readln(infile);
           readln(infile,comments);
           delete(comments,1,15);
           val(comments,init_temp,returncode);
           readln(infile,comments);
           val(comments,final_temp,returncode);
           readln(infile,comments);
           delete(comments,1,15);
           val(comments,step,returncode);
           readln(infile,comments);
           delete(comments,1,11);
           close(infile);
          end else
```

```
              write_data_situation(Red,white,
                           'Contents.Txt Not Found ! ');
              error_chk := 1;
             end
        else
         data_box(position,19,7,14,'S',file_name,error_chk);
       end;
     write_data_situation(red,white,'');
     TextBackground(Black);
     TextColor(Yellow);
   until error_chk <> 0;
   if not ((data = 'y') or (data = 'Y')) then
     begin
       MkDir(directory);
       ChDir(directory)
     end;
   ChDir('c:');
  end;
2: begin
    write_data_situation(black,yellow,
       'Name of Person Setting Conditions');
    data_box(position,56,7,20,'S',name,error_chk)
  end;
3: begin
    write_data_situation(black,yellow,
       'Sample Description, Experimental Conditions');
    data_box(position,22,9,54,'S',composition,error_chk)
  end;

4:  begin
     write_data_situation(black,yellow,
     'Ratio of thickness of sample to electrode area - d/A');
     data_box(position,24,11,8,'S',cell_constant,error_chk)
  end;

5: begin
    write_data_situation(black,yellow,
       'Starting Temperature of Experiment in Centegrade');
    data_box(position,30,13,3,'I',data,Init_temp)
  end;
```

```
    6:  begin
          write_data_situation(black,yellow,
             'Final Temperature of Experiment in Centegrade');
          data_box(position,59,13,3,'I',data,final_temp)
        end;
    7:  begin
          write_data_situation(black,yellow,
             'Temperature Step between Impedance Measurements');
          data_box(position,74,13,3,'I',data,step)
        end;
    8:  begin
         write_data_situation(black,yellow,'General Comment Area');
         data_box(position,19,16,58,'S',comments,error_chk)
        end;
  end;
until position = 9;
write_data_situation(red,white,
   'Please Wait - Storing Initial Data in "Contents.Txt" !!');
open_file_to_write(outfile,'a:contents.txt',error_chk);
open_file_to_write(arhenius,'a:'+file_name+'.SIG',error_chk);
writeln(arhenius,'Arhenius Data-Set  1'); writeln(arhenius);
close(arhenius);
writeln(outfile,'USER : ',name);
GetDate(year,month,day,dayofweek);
writeln(outfile,'Date : ',month,'/',day,'/',year);
trim(composition);
writeln(outfile,'COMPOSITION : ',composition);
writeln(outfile,'Cell Constant (d/A) : ',cell_constant);
trim(cell_constant);
val(cell_constant,constant,error_chk);
writeln(outfile);
writeln(outfile,'Initial Temp = ',init_temp);
writeln(outfile,'Final Temp   = ',final_temp);
writeln(outfile,'Step         = ',step);
writeln(outfile,'COMMENTS : ',comments);
close(outfile);
get_solartron_data(sol_setup);
assign(logfile,'is_wkstn.log');
append(logfile);
```

```
   delete(composition,30,100);
   writeln(logfile,name:20,composition,'   ',
                              month,'/',day,'/',year,'   ',time);
   close(logfile);
   ClrScr;
end;

{====================================================================
   Checks if the the temperature in the cell is at the setpoint and
                   not fluctuating - steady state.
 --------------------------------------------------------------------}
procedure check_steady_state(set_temp,step:integer);

var
   data, volt_read                                          :bus_string;
   last_temperature, types,
   typek, old_steady_state,setvoltage                             :real;
   flag, thermo_relay, period, counter, temp_difference      :integer;
   time_counter                                              :integer;


begin
  GotoXY (20,4); write(SET_TEMP);
  temp_difference := 4;
  append(trans);
  flag:=0;
  period := round(integ_time*data_points/20);  {delay between measure}
  get_type_k(old_steady_state);
  time_counter := 1;
  repeat                                  { Wait for Thermal Mass to }
    get_type_k(temperature);                     { heat up !}
    GotoXY(65,2);  write(time:8);           { * Transient Response * }
    GotoXY(19,6);  write(temperature:7:2);
    if time_counter = 10 then
     begin
       writeln(trans,time:8,temperature:7:2);
       time_counter := 1
     end;
    time_counter := time_counter + 1 ;
  until ((temperature - old_steady_state) > 2)
                      or (abs(temperature - set_temp) < 2) ;
  last_temperature := temperature;

  repeat                                    { * Steady State Response * }
    get_type_k(temperature);
    if abs(temperature-last_temperature) < 0.05 then
        flag := flag + 1
    else
      begin
        for counter := 1 to 10 do
          begin
            GotoXY(12,(counter+12)); write('       ');
            GotoXY(24,(counter+12)); write('     ');
          end; {for}
        GotoXY(65,2); write(time:8);
        flag := 1;
        last_temperature := temperature
      end;   {else}
    GotoXY (12,FLAG+12); write(temperature:7:2);
    GotoXY(24,FLAG+12); write(ABS(temperature-LAST_temperature):4:3);
    for counter := 1 to period do
      begin
        get_type_k(temperature);                           { heat up !}
        GotoXY(65,2);  write(time:8);
        GotoXY(19,6);  write(temperature:7:2);
        Delay(1000);                                { changed 1 seconds}
      end;
    writeln(trans,time:8,temperature:8:2);
  until (flag = 10);
  gotoXY(24,24);  write(temperature:7:2);
  WRITEln(OUTFILE,time:8,'    ',set_temp,'    ',temperature:10:3);
  close(trans);
end;


{=====================================================================
  Combines the procedures check_steady_state and impedance_measure to
  step through the temperature range and make impedance measurements.
 --------------------------------------------------------------------}
```

```
procedure make_measurement;

var
     setvoltage, temp_offset                 :real;
     setvoltage_str                          :strng;
     time_ss                                 :array[1..100] of strng1;
     temp_ss                                 :array[1..100] of real;
     hour_ss, minute_ss, second_ss, sec100   :word;
     step_number,j,k                         :integer;
     sst                                     :text;

begin
   open_file_to_write(sst,'a:steadyst.ate',j);
   open_file_to_write(trans,'a:trnsient.res',j);
   close(trans);
   writeln(sst,'  Start          End          dT  ');
   repeat
      screen_file('thermal.scr');
      TextBackground(Black);
      TextColor(White);
      step_number:= 0;                { dT in cell and setpoint }
      temp_offset := 0.86202+(0.88468+(7.4616e-3+(-81.76e-6+
                   (421.72e-9 - 816.4e-12* init_temp)*init_temp)*
                                init_temp)*init_temp)*init_temp;
      setvoltage := ( temp_offset * 0.01 ) + 2.5 ;
      str(setvoltage:6:3, setvoltage_str);
      setvoltage_str:= concat('SEL 2;VOLT',setvoltage_str,';',crlf);
      output(setvoltage_str,multifunc);
      output(crlf,multifunc);
      create_isothermal_filename(init_temp);
      for  j:=1 to step_number do
        begin
          GotoXY(38,6+j);
          write(time_ss[j]:8);
          gotoxy(48,6+j); write(temp_ss[j]:7:2);
         end;
      if ( init_temp <> final_temp )  or ( init_temp <> 20 ) then
          check_steady_state(init_temp,step);
      get_type_k(start);                              {*}
      impedance_measure;
      writeln(sst,start:10:3,final:10:3,(start-final):10:3);
      step_number := step_number + 1 ;
      init_temp := init_temp + step ;
      get_type_k(temp_ss[step_number]);
      time_ss[step_number] := time;
      delay(2000)
   until init_temp > final_temp;
   close(sst);                                     {*}
   setvoltage := 0 ;
   str(setvoltage:6:3, setvoltage_str);
   setvoltage_str:= concat('SEL 2;VOLT',setvoltage_str,';',crlf);
   output(setvoltage_str,multifunc);
   output(crlf,multifunc);
   append(logfile);
   writeln(logfile,'Ended':50,time);
   close(logfile);
   if final_temp = init_temp then
    begin
      erase(sst);
      erase(trans);
      erase(arhenius)
    end;
end;

{=====================================================================
 Option 6: GPIB Communication, this is a simplified version of a user
                    friendly communication program
 ---------------------------------------------------------------------}
procedure hpterm;

var
     data, CUT_DATA                  : bus_string;
     device, solartron, flag, j      : integer;
     TEMP                            :STRING[5];
     TMP                             :CHAR;
     MAGNITUDE, PHASE, FREQUENCY     :REAL;

 procedure clear_all;
```

```
   begin
     ABORTIO;
     clear(16);
     clear(30);
     clear(24);
     clear(23);
     device := 24;
   end;

begin
  solartron := 28;
  flag:=0;
  clrscr;
  repeat
     write('HPTERM > ');
     read(device,data);
     case device of
       -1 : clear_all;
       -2 : begin
              device := status;
              writeln('Error check : ', device);
              device := errcheck;
              writeln('Error check : ', device)
             end;
       -3 : begin
              write('HPTERM spoll> ');
              read(device,data);
              device := spoll(device);
              writeln('the serial poll is : ',device);
            end;
       end;
     if device = 1 then
       flag:=1
     else
       begin                         {#10 : LF}
        case device of               {#13 : CR}
           16, 28 : Temp := crlf;
           23     : Temp := ';'+#10;
           24     : TEMP := ';'+crlf;
```

```
          end;
          output(data+temp,device);
          writeln;
          writeln('Sent > ',data+temp);
          enter(DATA,DEVICE);
          WRITELN('Recieved > ',data);
        end;
   until flag=1;
End;

{============================
The Main Body of the Program
 ----------------------------} begin
   DirectVideo := False;          { Will not write directly to Screen }
   clrscr;
   screen_file('intro.pck');
   initialize_all_devices;
   readln;
   clrscr;
   repeat
      Screen_file('c_menu_1.scr');
      init_temp := 20; final_temp:= 20; step := 20;
      textbackground(lightgray); textcolor(blue);
      gotoXY(56,10); write(sol_setup);
      gotoxy(40,23);
      ch := Readkey;
      write(' '+ch+' ');
      case ch of
        '1','s','S'  :  begin
                          repeat
                            gotoXY(56,10);
                            ch := ReadKey;
                            write(ch);
                          until (ch > #48) and (ch < #58);
                          sol_setup := ch ;
                          if ch = '6' then ch := '1' {Stop Prog*}
                        end;
        '2','c','C'  :  Initialize_solartron(sol_setup);
        '3','n','N'  :  zero_open(sol_setup);

        '4','t','T'  :  begin
                          get_initial_data(init_temp,final_temp,step);
                          make_measurement;
                          clrscr; writeln('Operation Successful')
                        end;

        '5','g','G'  :  HPTERM;
      end;
    until (ch = 'e') or (ch = 'E') or ( ch = '6');
    textbackground(Blue); textcolor(white);
    clrscr;
End.
```

Data-Analyze Code

The flow chart in Figure 3.16 describes the Data-Analyze source code. Non-standard Turbo Pascal procedures for file management and graphics are either found in the units described in the next section or in the HGRAPH routines. The HGRAPH routines used in this program are summarized here and the interested reader is referred to the HGRAPH user's manual.

```
{
              HGRAPH Routines Called by Data-Analyze Program
              ----------------------------------------------

                    INIPLT(Unit, Rotation, Scalefactor)

The first routine INIPLT, initializes the graphics device defined by
Unit and the second and third arguments control the angle of Rotation
of the plot and the Scalefactor.

                    GRAPHBOUNDARY(x1, x2, y1, y2);

The routine GRAPHBOUNDARY defines the viewport area of the device to
display the graph where (x1,y1) define the lower left corner and
(x2,y2) define the upper right corner.

                         SCALE(x1, x2, y1, y2)

This SCALE routine defines the transformation of input data values to
the device coordinates.  (x1,y1) is mapped onto the lower left
corner of the GraphBoundary and (x2,y2) is mapped to the top right
corner of the GraphBoundary.

AXIS(xtic,xsubtic,xfrmt,xtitle,xsize,ytic,ysubtic,yfrmt,ytitle,ysixe)

Various procedures with the format shown above are used to draw a set
of axes.  xtic defines the interval for each displayed tick mark and
number.  The xsubtic controls the number of subtics displayed.  The
format of the axis numbers is controlled by xfrmt.  The size of the
```

axis label and the label are defined by xsize and xtitle. The arguments for y have the same function as the x arguements.

```
              SMOOTH(X ,Y ,N ,COL ,SYM ,SIZE ,NUM ,LINE)
            POLYLINE(X ,Y ,N ,COL ,SYM ,SIZE ,NUM ,LINE)
```

The procedure SMOOTH draws raw data and a spline curve through the data while POLYLINE draws the data but connects each point with a straight line. X and Y are real arrays with N data points, and the COLor, SYMbol, SIZE and LINEtype are controlled by inputing integers between 1-10. NUM controls the number of data points displayed on the device.

```
         JUSTIFYSTRING(X, Y, CAPTION, ANGLE, SIZE, XJUS, YJUS)
```

X and Y define the location on the plotting device to write a CAPTION controlled by SIZE and ANGLE and left, right, top, bottom and centered to the define location with XJUS and YJUS.

Data collect source code

```
{              -----------------------------------------
               SOURCE CODE FOR THE DATA ANALYSIS MODULE
               -----------------------------------------      }


{$R-}    {Range checking off}
{$B-}    {Boolean complete evaluation on}
{$S+}    {Stack checking on}
{$I+}    {I/O checking on}
{$N-}    {No numeric coprocessor}
{$M 65500,16384,655360}

Program analyses(input,output);

Uses
  hgrglb,        {HGRAPH units}
  hgrlow,
  hgrlin,
  hgraxi,
  hgrstr,
  hgrlgn,
  edtool,        {Units described in next section}
  getfiles,
  math,
  crt,
  dos;

type
  fnt                = string[2];
  strng              = string[14];
  strng22            = string[22];
  pltstring          = string[120];
  step_array         = array [1..8] of real;
  count_array        = array [1..10] of integer;
  comment_array      = array [1..10] of pltstring;
  legends            = array [1..10] of legend;

const
  permittivity     : real=8.854e-14;
  bld              : fnt=^F+'6';           {define fonts}
  Ibld             : fnt=^F+'7';
  cmplx            : fnt=^F+'4';
  drft             : fnt=^F+'0';
  dplx             : fnt=^F+'2';
  grk              : fnt=^F+'A';
  scrpt            : fnt=^F+'8';
  up               : Char=^U;
  dn               : char=^D;

VAR
      infile, outfile                                         : TEXT;
      i, j , xmark,  ymark , tau                             : INTEGER;
      counter,file_counter, files_processed, DEVICE_FLAG,
      num_point                                               : byte;
      in_name, out_name, files_to_fix                        : STRNG;
      dummy                                                  : strng22;
      date, legend, temperature, name, title, xtitle,
      ytitle, ytitle2, subtitle, resistance, capacitance,
      cell_const, y1legend,  y2legend, ploty1, ploty2      : pltstring;
      data_label                                       : comment_array ;
      freq, mag, phase, lfreq, x, y , yc_fit, xc_fit      : data_array;
      xfreq, yfreq                                        : step_array;
      cell_constant, plot_scale                                  : real;
      choice                                                     : word;
      files_to_analyze                                : file_to_analyze;
      ch, plot_all_files, plot_labels                            : char;
      mnx, mny, mny2, mxx, mxy, mxy2, xstp, ystp, y2stp          : real;
      xi, yi                                             : count_array;
      data_legend                                            : legends;

{==================================================================
          Read contents.txt file and data file for raw data
{-----------------------------------------------------------------}
PROCEDURE getfile_names(var counter:byte);
var
    x , y            :integer;
    x1, x2, y1, y2   :Byte;
    first            :string[17];
    D                :Dirstr;
    N                :Namestr;
    E                :Extstr;
    ans              :char;


begin
{$I-}
  repeat
    files_to_Analyze[9] := files_to_Analyze[counter];
       FSplit ( files_to_Analyze[counter], D, N, E );
       assign (infile,d+'contents.txt');    {Read Contents file}
       reset (infile);
       if IORESULT = 0 then
          begin
            readln(infile,name);
            readln(infile,date);
            readln(infile,subtitle);
```

```
        read(infile,dummy);
        readln(infile,cell_constant);
        Close(infile);
        subtitle:=concat(cmplx+subtitle);
        name:=concat(cmplx+name);
        date:=concat(cmplx+date);
        if cell_constant <> 1 then
         begin
           str(cell_constant:6:4,cell_const);
           cell_const:=concat(cmplx+'Ko = '+cell_const+' /cm')
         end
        else cell_const :='';
      end
   else
      begin
        subtitle := bld; name := cmplx;
        date := cmplx; cell_const := '1';
        cell_constant := 1;
      end;
                            {Read Raw Data File}
open_file_to_read(infile,files_to_analyze[9],x);
reset(infile);
readln(infile,first,freq[1]);
readln(infile);
x := IOResult ;
str(freq[1]:5:2,N);
temperature := '';
if freq[1] <> 1 then
   temperature := concat(cmplx+'Temp: '+N+' '+grk+#39+cmplx+'C') ;
i := 0;
repeat
  inc (i) ;
  readln(infile, freq[i], mag[i], phase[i]);
  x := IOResult ;
until eof(infile) or (x <> 0);
if (x <> 0) or (i = 1) or (freq[1] = 0) then
  begin
    for y := counter to 5 do
        files_to_analyze[y] := files_to_analyze[y+1];
        files_to_analyze[9] := 'INVALID DATA FILE ' +
                                                files_to_analyze[9] ;
        Beep; Beep;
        write_data_situation(red, white, files_to_analyze[9]);
        delay(1000);
        dec (file_counter);
      End;
    close(infile);
    if counter = file_counter then counter := counter - 1 ;
    until ((x = 0) and (i > 1)) or (counter = 0);
    {$I+}
  end;
{======================================================================
                    Gets the raw data and plots it
{----------------------------------------------------------------------}
procedure draw_graphs;
var
  max_x, min_x, min_y, max_y, min_y2, max_y2,
  xstep, ystep, y2step, Ko, minix, maxix                      : real;

{======================================================================
          Routines draw to plot data on linear and log plots
{----------------------------------------------------------------------}
procedure plot_it( j:integer; title, xtitle, ytitle, ytitle2, name,
                   subtitle,date, temperature : pltstring ;
                   x, y1, y2 : data_array ; i : integer           );
type
   string4                                                   =string[4];
var
   xtype, ytype, y2type                                      :pltstring;
   angle                                                     : integer;
   title_col, subtitle_col, name_col, date_col,
   temperature_col, xtitle_col, ytitle_col, ytitle2_col,
   device, data_col, data_symbol, line_type, xsubtic,
   ysubtic, y2subtic                                             :byte;
   chr                                                           :char;


{-----------------------------}
{ Linear X and Linear Y plot  }
{-----------------------------}
procedure Plot_LinY_LinX;
 begin
   if (choice = 1) or (choice = 3) or (choice = 9) then
    begin
      GRAPHBOUNDARY(2550,7450,1100,6000);
      SCALE(min_x,max_x,min_y,max_y);
      if plot_all_files = 'N' then
        begin
           smooth(xc_fit,yc_fit,150,1,10,0,0,0);
           Polyline(xfreq,yfreq,8,1,0,4,num_point,9);
           xmark := iscrx(xfreq[tau]) ;
           ymark := iscry(yfreq[tau]) + 150 ;
           justifystring(xmark,ymark,Dummy,0,2,center,base)
        end;
    end
   else
    begin
      GRAPHBOUNDARY(2000,8000,1100,6000);
      SCALE(min_x,max_x,min_y,max_y);
    end;
   AXIS(xstep,xsubtic,'8.6',xtitle,3,ystep,ysubtic,'8.6',ytitle,3);
   TAXIS(xstep,xsubtic,'0','',3,ystep,ysubtic,'0','',3);}
   polyline(x,y1,i,data_col,data_symbol,2,num_point,line_type);
 end;

{-----------------------------}
{ Log X and Linear Y1 Plot    }
{-----------------------------}
procedure Plot_LinY_logX(ydata : data_array;i: integer;
                          min_y, max_y, ystep : real;
                          ysubtic : byte; ytitle : pltstring);

 begin
   GRAPHBOUNDARY(1800,8200,1200,6100);
   SCALE(min_x,max_x,min_y,max_y);
   if choice = 16 then
     LXAXIS(xstep,xsubtic,'7',xtitle,3,ystep,ysubtic,'8.0',ytitle,3)
   else
```

```
    LXAXIS(xstep,xsubtic,'7',xtitle,3,ystep,ysubtic,'8.6',ytitle,3);
   TLXAXIS(xstep,xsubtic,'0','',3,-ystep,ysubtic,'7',ytitle,3);
   LXAXIS2(xstep,xsubtic,'7',xtitle,3,ystep,ysubtic,'0','',3);
   polyline(x,ydata,i,data_col,data_symbol,2,num_point,line_type);
 end;

{-----------------------------}
{ Log Y and Linear X Plot     }
{-----------------------------}
procedure Plot_LogY_LinX;
begin
   GRAPHBOUNDARY(1800,8200,1200,6100);
   SCALE(min_x,max_x,min_y,max_y);
   LAXIS(xstep,xsubtic,'8.6',xtitle,3,ystep,ysubtic,'7',ytitle,3);
   if choice = 13 then
     begin
       SCALE(min_x,max_x,min_y,max_y);
       smooth(xc_fit,yc_fit,150,1,10,0,0,0);
       line_type := 9
     end;
   polyline(x,y1,i,data_col,data_symbol,2,num_point,line_type);
 end;

{-----------------------------}
{ Log X and Log Y1 Plot       }
{-----------------------------}
procedure Plot_LogY_logX(ydata : data_array;i: integer;
                         min_y, max_y,ystep : real;
                         ysubtic : byte; ytitle : pltstring);

     begin
        GRAPHBOUNDARY(1800,8200,1200,6100);
        SCALE(min_x,max_x,min_y,max_y);
        LLAXIS(xstep,xsubtic,'7',xtitle,3,ystep,ysubtic,'7',ytitle,3);
        polyline(x,y1,i,data_col,data_symbol,2,num_point,line_type);
      end;

{-------------------------------------------
     Log X and Log Y1 and log Y2 Plot
-------------------------------------------}
procedure Plot_LogY2_logy1_logX;
var
  l : integer ;
 begin
   GRAPHBOUNDARY(1800,8200,1200,6100);
   SCALE(min_x,max_x,min_y,max_y);
   LLAXIS(xstep,xsubtic,'7',xtitle,3,ystep,ysubtic,'7',ytitle,3);
   polyline(x,y1,i,data_col,data_symbol,2,num_point,line_type);
   GRAPHBOUNDARY(1800,8200,1200,6100);
   SCALE(min_x,max_x,min_y2,max_y2);
   LogY2Axis(xstep,xsubtic,'7','',3,y2step,y2subtic,'7',ytitle2,3);
   if plot_all_files = 'N'  then
     polyline(x,y2,i,data_col+1,data_symbol+1,2,num_point,line_type+1)
   else
     polyline(x,y2,i,data_col,data_symbol,num_point,1,line_type);
 end;

{-----------------------------}
{ Lin X and Linear Y1 and Y2  }
{-----------------------------}
procedure Plot_LinY2_LinY1_LinX;

 begin
   GRAPHBOUNDARY(2000,8000,1100,6000);
   SCALE(min_x,max_x,min_y,max_y);
   AXIS(xstep,xsubtic,'8.6',xtitle,3,ystep,ysubtic,'8.6',ytitle,3);
   smooth(x,y1,i,data_col,data_symbol,2,num_point,line_type);
   polyline(x,y1,i, 2, 2, 1, 1, 2);            {Draw data curve}
   SCALE(min_x,max_x,min_y2,max_y2);
   smooth(x,y2,i,data_col+1,data_symbol+1,2,num_point,line_type+1);
  end;

{-----------------------------}
{ Log X and Linear Y1 and Y2  }
{-----------------------------}
procedure Plot_LinY2_LinY1_LogX;

begin
```

```
   GRAPHBOUNDARY(1800,8200,1200,6100);
   SCALE(min_x,max_x,min_y,max_y);
   LXAXIS(xstep,xsubtic,'7',xtitle,3,ystep,ysubtic,'8.6',ytitle,3);
   TLXAXIS(xstep,xsubtic,'0','',3,-ystep,ysubtic,'7',ytitle,3);
   polyline(x,y1,i,data_col,data_symbol,2,num_point,line_type);
   SCALE(min_x,max_x,min_y2,max_y2);
  LXAXIS2(xstep,xsubtic,'7',xtitle,3,y2step,y2subtic,'8.6',ytitle2,3);
   if plot_all_files = 'N'  then
     polyline(x,y2,i,data_col+1,data_symbol+1,2,num_point,line_type+1)
   else
       polyline(x,y2,i,data_col,data_symbol,num_point,1,line_type);
 end;

{---------------------------------------------------}
{ Write pre-set Graph Settings for  Window  1
{---------------------------------------------------}
procedure write_default_data;

begin
     screen_file('grp_set1.scr');
     TextBackground(red);
     TextColor(white);
     GotoXY(30,6);writeln(device);
     GotoXY(60,6);writeln(plot_scale:3:2);
     GotoXY(5,8); trim(title); writeln(title);
     GotoXY(77,8); writeln(title_col);
     GotoXY(5,10); trim(subtitle); writeln(subtitle);
     GotoXY(77,10); writeln(subtitle_col);
     GotoXY(5,13); trim(name); writeln(name);
     GotoXY(35,13); writeln(name_col);
     GotoXY(5,15); trim(date); writeln(date);
     GotoXY(35,15); writeln(date_col);
     GotoXY(5,17); trim(temperature); writeln(temperature);
     GotoXY(35,17); writeln(temperature_col);
     GotoXY(5,20); writeln(xtitle);
     GotoXY(35,20); writeln(xtitle_col);
     GotoXY(44,20); writeln(xtype);
     GotoXY(55,20);  writeln(num_point);
     GotoXY(5,22); writeln(ytitle);
     GotoXY(35,22); writeln(ytitle_col);
     GotoXY(44,22); writeln(ytype);
     GotoXY(55,22); writeln(ploty1);
     GotoXY(5,24); writeln(ytitle2);
     GotoXY(35,24); writeln(ytitle2_col);
     GotoXY(44,24); writeln(y2type);
     GotoXY(55,24); writeln(ploty2);
end;


{-----------------------------------------------------}
{ Write pre-set Graph Settings for User Window 2
{-----------------------------------------------------}
procedure write_default_data2;
begin
   screen_file('grp_set2.scr');
   GotoXY(21,9); writeln(min_x:10);
   GotoXy(34,9); writeln(max_x:10);
   GotoXY(47,9); writeln(xstep:10);
   GotoXY(21,11); writeln(min_y:10);
   GotoXY(34,11); writeln(max_y:10);
   GotoXY(47,11); writeln(ystep:10);
   GotoXY(21,13); writeln(min_y2:10);
   GotoXY(34,13); writeln(max_y2:10);
   GotoXY(47,13); writeln(y2step:10);
end;

{---------------------------------------------------------------------
 Simple line editor to input data for the input windows for the graph
  settings, insert/overwrite modes, delete and backspace are used for
   deleting, right and left arrow move within an input window and up
 arrow moves back to the last window and enter moves to the next.  F1
                          shows the help menu.
 --------------------------------------------------------------------}


procedure get_new_setting(x,y,lenth:integer;var new_string:pltstring);
        {screen location ( , )  }

var
```

```
  tmp_string                    : pltstring;
  new_location                  : byte;
  i,j,insert_mode               : integer;

begin
    GotoXY(74,4); writeln('{OVR}');
    Gotoxy(x,y);
    i := 1;
    new_location := 0;
    tmp_string := '';
    ch := ReadKey;
    insert_mode := -1;
    While (ch <> #13) and (ch <> #27) and ((i-1) < lenth) and
                               (new_location = 0) do  {carrige return}
      begin
         while ch = #0 do
             begin
               ch := ReadKey;
               case ch of
{INS}            #82 : begin
                          insert_mode := -1*insert_mode;
                          GotoXY(74,4);
                          If insert_mode = 1 then writeln('{INS}')
                             Else Writeln('{OVR}');
                        end;
{del}            #83 : begin
                         delete(new_string,0,i+1);
                         new_string :=
                              concat(tmp_string,new_string);
                         for j :=length(new_string) to (lenth-1) do
                              begin  GotoXY(x+j,y); write(' '); end;
                        end;
{F1 = Help}      #59 : begin
                          if lenth <> 10 then
                          begin
                            if lenth < 5 then
                                screen_file('clr_help.scr')
                             else screen_file('fnt_help.scr');
                            repeat ch := ReadKey until ch <> #0;
```

```
                              write_default_data;
                             end;
                           end;
{Up,Dn arrow}          #72,#88 : begin
                                  new_location := 1
                                 end;
{Rt arrow}             #77 : begin
                             tmp_string := new_string;
                             delete(tmp_string,i+1,lenth);
                             delete(new_string,0,i+1);
                             new_string :=
                                   concat(tmp_string,new_string);
                             j := length(new_string);
                             If (i <= lenth)  and (i <= j) then
                               i := i + 1
                           end;
{Lt arrow}              #75 : begin
                              if i > 1 then i := i - 1 ;
                              delete(tmp_string,i,lenth);
                              delete(new_string,0,i);
                              new_string :=
                                  concat(tmp_string,new_string);
                            end;
                        end;
                        GotoXY(x,y);
                        write(new_string);
                        GotoXY(x+i-1,y);
                        if new_location = 0 then ch := ReadKey;
                        if ch = #13 then new_location := 1;
                      end;
          if new_location = 0 then
           begin
            if insert_mode = 1 then
               begin
                 if ch <> #8 then                      {backspace}
                    begin
                      tmp_string := concat(tmp_string,ch);
                      delete(new_string,lenth,1);
                      delete(new_string,0,i);
            If I <> lenth then
               i := i + 1
          end
        else
          begin
            if i > 1 then i := i - 1 ;
            delete(tmp_string,i,lenth);
            delete(new_string,0,i+1)
          end;
     end
  else
     begin
       if ch <> #8 then                               {backspace}
         begin


            i := i + 1;
            tmp_string := concat(tmp_string,ch);
            delete(new_string,0,i)
         end
       else
         begin
            if i > 1 then i := i - 1 ;
            delete(tmp_string,i,lenth);
            delete(new_string,0,i+1)
         end;
      end;
    new_string := concat(tmp_string,new_string);
    GotoXY(x,y);
    write(new_string);
    for j :=length(new_string) to (lenth-1) do
      write(' ');
    GotoXY(x+i-1,y);
    ch := ReadKey;
  end;
 end;
 if ch = #0 then ch := readKey;
 trim(new_string);
end;
```

```
{-----------------------------------------------------------------------
 Labels the graph for all defined variables through the Graph Setting
                     windows and Graphic window
 ----------------------------------------------------------------------}
procedure label_graph
var
 count:integer;
begin
     if (plot_all_files = 'N') then
      begin
        coltyp(title_col);
        justifystring(5000,6800,title,0,4,center,center);
        coltyp(subtitle_col);
        justifystring(5000,6400,subtitle,0,2,center,center)
      end
      else
    if (plot_labels = 'Y') then
      begin
        coltyp(name_col);
        justifystring(7200,500,name,0,2,left,base);
        coltyp(date_col);
        justifystring(7200,300,date,0,2,left,base);
        coltyp(temperature_col);
        justifystring(7200,100,temperature,0,2,left,base);
        justifystring(100,500,Resistance,0,2,left,base);
        justifystring(100,300,Capacitance,0,2,left,base);
        justifystring(100,100,Cell_const,0,2,left,base)
    end;
    for count:= 1 to 10 do
       justifystring(xi[count],yi[count],data_label[count],
                                               0,2,left,center);
    for count := 1 to 8 do
       reclegend(data_legend[count]);
end;

{-----------------------------------------------------------------------
    Determines the Plot Type string to display on the Graph Setting
                  Window from the graph type integer
 ----------------------------------------------------------------------}
```

```
procedure determine_plot_type_string(j:integer);

begin
   if (j mod 2) = 1 then xtype := 'Log'  else xtype := 'Lin';
   if ((j div 2) mod 2) = 1 then ytype := 'Log'  else ytype := 'Lin';
   if j > 6 then y2type:='Log'
      else if j > 3 then y2type := 'Lin'
           else y2type := '';
   if j = 8 then
     begin
        xtype :='Log' ;
        ytype := 'Log';
        y2type := 'Log'
     end;
end;

{----------------------------------------------------------------------
   Determines the Plot Type from the selected integer to display on
                       the Graph Setting Window
 ---------------------------------------------------------------------}
procedure determine_plot_type(var j:integer);
begin
   j:=0;
   if xtype = 'Log' then j:=j+1;
   if ytype = 'Log' then j:=J+2;
   if y2type ='Log' then j:=J+5;
   if y2type = 'Lin' then  j:=J+4;
end;

{----------------------------------------------------------------------
            Get user data for the two Graph Setting Windows
 ---------------------------------------------------------------------}
procedure get_user_data;
var
   position   : integer;
   temp       : pltstring;
   chk        : integer;

   procedure get_color(x,y,lenth:integer; var string_col:byte);
```

```
  var
    colour   :pltstring;
    i        : integer;
  begin
     str(string_col,colour);
     repeat
       get_new_setting(x,y,lenth,colour);
       val(colour,string_col,i)
     until i = 0;
  end;
  procedure get_number(x,y,lenth:integer; var max_min:real);
  var
    colour   :pltstring;
    i        : integer;
  begin
     str(max_min:10,colour);
     repeat
       get_new_setting(x,y,lenth,colour);
       val(colour,max_min,i)
     until i = 0;
 end;

{-------------------------------------------------------------------
                 Autoscale function used in Window 2
 -------------------------------------------------------------------}
procedure quick_autoscale(x,y:integer; xrange, x_array : data_array;
                        axis_type:pltstring; var x1,x2,xstep :real);
  var
    tmp                   : pltstring;
    l, xmin_loc, xmax_loc : integer;
    num_x_min, num_x_max  : real;
  begin
    tmp := '';
    get_new_setting(x,y,1,tmp);
    if (tmp = 'y') or (tmp = 'Y') then
      begin
        xmin_loc := 1;
        xmax_loc := i;
        if position > 3 then { Selects X range to autoscale Y }
```

```
            begin
              num_x_min := min_x;
              num_x_max := max_x;
              while xrange[xmin_loc] < num_x_min do
                xmin_loc := xmin_loc + 1;
              xmax_loc := xmin_loc;
              while ( xrange[xmax_loc] < num_x_max ) and
                                                     ( xmax_loc < i) do
                xmax_loc := xmax_loc + 1 ;
             end;
        x1 :=  x_array[xmin_loc];
        x2 := x1 ;
        for l := xmin_loc to xmax_loc do
             max_min(l,x1,x2,x_array[l]);
        if axis_type = 'Log' then
            log_autoscale(minix,maxix,x1,x2,xstep)
           else  autoscale(x1,x2,xstep);
        write_default_data2
       end;
   end;

begin  { get user data }
   position:=0;
   write_default_data;
   repeat
     case position of
         0: get_color(30,6,1,device);
         1: begin
              str(plot_scale:3:2,temp);
              repeat
                get_new_setting(60,6,3,temp);
                val(temp,plot_scale,chk)
              until chk = 0;
            end;
         2: get_new_setting(5,8,55,title);
         3: get_color(77,8,1,title_col);
         4: get_new_setting(5,10,70,subtitle);
         5: get_color(77,10,1,subtitle_col);
         6: get_new_setting(5,13,22,name);
         7: get_color(35,13,1,name_col);
     8: get_new_setting(5,15,22,date);
     9: get_color(35,15,1,date_col);
    10: get_new_setting(5,17,22,temperature);
    11: get_color(35,17,1,temperature_col);
    12: get_new_setting(5,20,22,xtitle);
    13: get_color(35,20,1,xtitle_col);
    14: get_new_setting(44,20,3,xtype);
    15: get_color(55,20,1,num_point);
    16: get_new_setting(5,22,22,ytitle);
    17: get_color(35,22,1,ytitle_col);
    18: get_new_setting(44,22,3,ytype);
    19: get_new_setting(55,22,1,ploty1);
    20: get_new_setting(5,24,22,ytitle2);
    21: get_color(35,24,1,ytitle_col);
    22: get_new_setting(44,24,3,y2type);
    23: get_new_setting(55,24,1,ploty2);
 end;
 if ch = #72 then position := position - 1
   else position := position + 1;
 if position < 0 then position := 0;
until ch = #27;
position:=0;
write_default_data2;
repeat
 case position of
     0: get_number(21,9,10,min_x);
     1: get_number(34,9,10,max_x);
     2: get_number(47,9,10,xstep);
     3: quick_autoscale(68,9,x,x,xtype,min_x,max_x,xstep);
     4: get_number(21,11,10,min_y);
     5: get_number(34,11,10,max_y);
     6: get_number(47,11,10,ystep);
     7: quick_autoscale(68,11,x,y1,ytype,min_y,max_y,ystep);
     8: get_number(21,13,10,min_y2);
     9: get_number(34,13,10,max_y2);
    10: get_number(47,13,10,y2step);
    11: quick_autoscale(68,13,x,y2,y2type,min_y2,max_y2,y2step);
   end;
 if ch = #72 then position := position - 1
```

```
       else position := position + 1;
     if ch = #59 then Beep;           {Option to use Annotation}
     if position < 0 then position := 0;
    until ch = #27;
end;

{-------------------------------------------------------------------
           Default colors to use for displaying first plot
 ------------------------------------------------------------------}
procedure initial_label_colors;
begin
  determine_plot_type_string(j);
  title_col := 1 ; subtitle_col := 4 ; name_col := 1; date_col := 1;
  temperature_col := 1 ; xtitle_col := 0; ytitle_col := 0 ;
  ytitle2_col := 0;  data_col := files_processed+2 ;
  data_symbol := files_processed - 1;
   line_type := files_processed - 1;
end;

{-------------------------------------------------------------------
      If axis type is changed, convert data to log or linear form
 ------------------------------------------------------------------}
procedure correct_data;

procedure convert_to_log(x: data_array; var logx:data_array);
var
  l       : integer;
begin
    for l:= 1 to i do                   {i = # of data points}
      logx[i] := 0.4342945*ln(x[i]);
end;
begin
     case j of
        1,5: convert_to_log(freq,x);
        2: convert_to_log(mag,y1);
        3: begin
              convert_to_log(mag,y1);;
              convert_to_log(freq,x);
           end;
        6: begin
             convert_to_log(mag,y1);;
             convert_to_log(phase,y2);
           end;
        7:  begin
             convert_to_log(mag,y1);;
             convert_to_log(phase,y2);
             convert_to_log(freq,x);;
            end;
         end; {case of j}

end;

{----------------------------------------------------------------------
     Draw the graph to the device by calling one of the plot type
                      procedures described above
 ---------------------------------------------------------------------}
procedure draw_to_device;

 function subtic(step:real):byte;
 begin
   if (step >= 1) or (step <= 0) then     { 10^(trunc(log(step)))}
     subtic := round(step/exp(trunc(0.43443*ln(abs(step)))*ln(10)))
   else
     subtic := round(step/exp(trunc(0.43443*ln(abs(step))-1)*ln(10)))
                                         { 10^(trunc(log(step)-1))}
  end;

begin
   xsubtic := subtic(xstep);
   ysubtic := subtic(ystep);
   y2subtic:= subtic(y2step);
   GRAPHBOUNDARY(2000,8000,1400,5600);
    case j of {lsb  msb}
      0: Plot_LinY_LinX ;
      1: Plot_LinY_logX(y1,i,min_y,max_y,ystep,ysubtic,ytitle);
      2: Plot_LogY_LinX ;
      3: Plot_LogY_LogX(y1,i,min_y,max_y,ystep,ysubtic,ytitle) ;
      4: Plot_LinY2_LinY1_LinX ;
      5: begin
```

```
          if (ploty1='N') or (ploty1='n') then
           Plot_LinY_logX(y2,i,min_y2,max_y2,y2step,y2subtic,ytitle2)
          else if (ploty2='N') or (ploty2='n') then
             Plot_LinY_logX(y1,i,min_y,max_y,ystep,ysubtic,ytitle)
          else
             Plot_LinY2_LinY1_LogX ;
           end;
        8: begin
            if (ploty1='N') or (ploty1='n') then
                 Plot_LogY_logX(y2,i,min_y2,max_y2,
                                             y2step,y2subtic,ytitle2)
            else
               if (ploty2='N') or (ploty2='n') then
                  Plot_Logy_LogX(y1,i,min_y,max_y,
                                                ystep,ysubtic,ytitle)
               else
                  Plot_LogY2_LogY1_LogX ;
           end;
     end;
end;

{-----------------------------------------------------------------
        Writes messages of different colors on the Graphics window
 ----------------------------------------------------------------}
procedure center_cmd(color:integer; d_label :pltstring);
begin
   coltyp(color);
   justifystring(5000,100,d_label,0,2,center,center);
end;

{-----------------------------------------------------------------
                         Customize Graphics Screen
 ----------------------------------------------------------------}
 procedure customize_screen;
  var
    count, symb  : integer;
    chr          : char;
  begin
    if device = 0 then
```

```
      begin
        center_cmd(-1,' < PLEASE WAIT > ');
        center_cmd(1,'< "C" - COMMENT   "L" - LEGEND > ');
        chr := readkey;
        while (chr='l') or (chr='L') or (chr='C') or (chr='c') do
        begin
          center_cmd(-1,'< "C" - COMMENT   "L" - LEGEND > ');
          if (chr = 'C') or (chr = 'c') then
            begin
              repeat
                center_cmd(1,' < PRESS F1 - F10 TO LABEL > ');
                chr := readkey;
                if chr = #0 then
                  begin
                    chr := Readkey;
                    if (chr > #58) and (chr < #69) then
                      begin
                        count := ord(chr) - 58;
{HGRAPH routine}        readstring(xi[count],yi[count],
                                          data_label[count],0,2)
                      end
                  end;
                center_cmd(-1,' < PRESS F1 - F10 TO LABEL > ');
              until (chr < #59) or (chr > #69);
            end
          else if (chr = 'L') or (chr = 'l') then
            begin
              repeat
                center_cmd(1,' < SELECT LEGEND ( F1 - F10 ) > ');
                chr := readkey;
                if chr = #0 then
                  begin
                    chr := Readkey;
                    if (chr > #58) and (chr < #69) then
                      begin
                        count := ord(chr) - 58;
                        if plot_all_files = 'N' then symb := -2
                          else symb := count;
{HGRAPH routine to}     conlegend(data_legend[count],count+2,
{draw legends}                                           count-1,2,1,count-1);
                      end
                  end;
                center_cmd(-1,' < SELECT LEGEND ( F1 - F10 ) > ');
              until (chr < #59) or (chr > #68);
            end;
          center_cmd(1,'< "C" - COMMENT   "L" - LEGEND > ');
          chr := readkey
        end;
        center_cmd(-1,'< "C" - COMMENT   "L" - LEGEND > ');
     end;
   end;

{----------------------------------------------------------------------
Message sent on completion of graphics section: option to continue !
 ----------------------------------------------------------------------}
procedure endplt_redefined(var ch :char);
  begin
    if device = 0 then
      begin
       center_cmd(-1,' < PLEASE WAIT > ');
       center_cmd(3,' < PRESS ANY KEY TO CONTINUE > ')
      end;
    ENDPLT;                                              {Terminate HGraph}
    textcolor(white);
    GotoXY(8,15);
    writeln('<ESC> to Re-Calculate Data to View in Another Form');
    GotoXY(10,20);
    writeln('Press <ANY OTHER KEY> to CONTINUE Present Analysis');
    ch := readkey;
  end;

begin     {Main Plotting Routine Body}
 initial_label_colors;
 if plot_all_files = 'N' then         {True = Single Plot}
  begin
    DEVICE := 0;
    INIPLT(device, normal, plot_scale);        {Initialize HGraph}
    draw_to_device;
  Label_Graph(title,name,subtitle,date,temperature);
  customize_screen;
  ENDPLT_redefined(ch);                         {Terminate HGraph}
  while  ch <> #27 do                    {change graph settings}
    begin
      device := 0;
      Get_user_data;
      determine_plot_type(j);
      if choice = 39 then correct_data;
      INIPLT(device, normal, plot_scale);       {Initialize HGraph}
      draw_to_device;
      Label_Graph(title,name,subtitle,date,temperature);
      customize_screen;
      EndPlt_redefined(ch);
    end;
end
else if plot_all_files = 'Y' then          {Multiple plot}
   begin
    device := device_flag;
    if files_processed < file_counter then
     begin
        if files_processed = 1 then
           begin
             if ch = #27 then
               begin
                 min_x := mnx ; min_y := mny ; min_y2 := mny2 ;
                 max_x := mxx ; max_y := mxy ; max_y2 := mxy2 ;
                 xstep := xstp ; ystep := ystp ; y2step := y2stp
               end;
             if ch = #13 then
                begin
                  mnx := min_x ; mny := min_y ; mny2 := min_y2 ;
                  mxx := max_x ; mxy := max_y ; mxy2 := max_y2 ;
                  xstp := xstep ; ystp := ystep ; y2stp := y2step ;
                end;
             INIPLT(device,normal, plot_scale);
             if device = 0 then center_cmd(3,' < PLEASE WAIT > ');
             Label_Graph(title,name,subtitle,date,temperature);
             draw_to_device
     end
  else
     begin
       SCALE(mnx,mxx,mny,mxy);

       if choice = 13 then
         begin
           smooth(xc_fit,yc_fit,150,data_col,0,0,0,line_type);
           polyline(x,y1,i,data_col,data_symbol,2,num_point,9)
         end
       else
       if (ploty1<>'N') and (ploty1 <> 'n') then
         polyline(x,y1,i,data_col,data_symbol,2,2,line_type)
       else
         begin
           SCALE(mnx,mxx,mny2,mxy2);
           polyline(x,y2,i,data_col,data_symbol,2,
                                            num_point,line_type)
         end;
       if (((choice mod 2 ) = 0) or (choice = 15))
                  and (choice <> 16)
                  and ((ploty1 ='y') or (ploty1 = 'Y'))
                  and ((ploty2 ='y') or (ploty2 = 'Y')) then
         begin
            SCALE(mnx,mxx,mny2,mxy2);
            polyline(x,y2,i,data_col,data_symbol,1,
                                            num_point,line_type)
         end;
     end;
end else
begin
  customize_screen;
  ENDPLT_REDEFINED(CH);
  if ch <> #27 then
     begin
       device := 0;
       files_processed := 0 ;
       min_x := mnx ; min_y := mny ; min_y2 := mny2 ;
       max_x := mxx ; max_y := mxy ; max_y2 := mxy2 ;
       xstep := xstp ; ystep := ystp ; y2step := y2stp;
```

```
              Get_user_data;
              device_flag := device;
              mnx := min_x ; mny := min_y ; mny2 := min_y2 ;
              mxx := max_x ; mxy := max_y ; mxy2 := max_y2 ;
              xstp := xstep ; ystp := ystep ; y2stp := y2step ;
              determine_plot_type(j);
              if choice = 39 then correct_data;
            end;

       end;
   end;
end;                 {End of Plot it}

{----------------------------------------------------------------------
   The following routines are to calculate the different forms of
  presenting the magnitude, phase and frequency data of a material.
 ---------------------------------------------------------------------}

{----------------------------------------------------------------------
 Determines the curve fit for the raw data and the best curve fit for
                  the semicircle and RC curve fit.
 ---------------------------------------------------------------------}
procedure semi_circle_fit(x,y:data_array;i:integer;
                          var xc_fit, yc_fit : data_array;
                          var tau : integer);
var
  lsq                                         : array [1..7] of real;
  j,k,l,One_MHz                               : integer;
  xc, yc, r, x_intercept, x1, x2, cap , scale  : real;

begin
  scale := 1;
  dummy := '';
  k := I-1;
  if max_y > 1e6 then scale := max_y ;
  while freq[K] > 1e6 do k := k - 1;
  One_MHz := k ;
  while  ((x[k] > x[k+1]) and (y[k] > y[k+1]))  do k:=k-1;
  while  ((x[k] > x[k+1]) and (y[k] < y[k+1]))  do k:=k-1;
```

```
if ( One_MHz - k ) > 5 then
   begin
      for j:=1 to 7 do
        begin
          lsq[j] := 0;
          xfreq[j] := 0 ;
          yfreq[j] := 0 ;
        end;
      l := k + 1;
      x1 := y[k]/scale;
      for j:= k+1 to One_MHz do
        begin
          x[j]   := x[j]/scale;
          y[j]   := y[j]/scale;
          lsq[1] := lsq[1] + x[j]*x[j];            {SUM  X^2  }
          lsq[2] := lsq[2] + x[j]*x[j]*x[j];       {SUM  X^3  }
          lsq[3] := lsq[3] + x[j]*y[j]*y[j];       {SUM  XY^2 }
          lsq[4] := lsq[4] + x[j]*y[j];            {SUM  XY   }
          lsq[5] := lsq[5] + y[j]*y[j];            {SUM  Y^2  }
          lsq[6] := lsq[6] + y[j]*y[j]*y[j];       {SUM  Y^3  }
          lsq[7] := lsq[7] + x[j]*x[j]*y[j];       {SUM  YX^2 }
          if y[j] > x1 then
            begin
              x1 := y[j];
              l := j
            end;
        end;
      yc := lsq[4]*lsq[4] - lsq[1]*lsq[5];     {(XY)^2 - (X^2 Y^2)}
      xc := ((lsq[6]+lsq[7])*lsq[4] -(lsq[3]+lsq[2])*lsq[5])/yc;
      yc := ((lsq[3]+lsq[2])*lsq[4]-(lsq[6]+lsq[7])*lsq[1])/(-yc);
      r  := 0.5*scale*sqrt(xc*xc + yc*yc);
      yc := yc*scale/2;
      xc := xc*scale/2;
      xfreq[8] := xc;
      yfreq[8] := -yc;
      x_intercept := sqrt( r*r - yc*yc ) + xc;
      if max_y < x_intercept then max_y := x_intercept;
      if min_y > -yc then min_y := -yc;
      x[l] := scale*x[l];
```

```
yc_fit[102] :=  sqrt(r*r - exp(2*ln(abs(x[1]-xc)))) + yc;
cap  := yc_fit[102]/(2*pi*freq[1]*
       (x[1]*x[1] + yc_fit[102]*yc_fit[102]));  {Z"/(w*|Z|^2}
str(x_intercept:9,resistance);
str(cap:9,capacitance);
lsq[2] := 1/(2*pi*x_intercept*cap);     {relaxation frequency}
lsq[1] := 1e8;                         {used for frequency steps}
for j := 1 to 150 do
 begin
   lsq[1] := lsq[1] - 0.333*
                 exp(2.30258*trunc(0.4342945*Ln(lsq[1])));
   yc_fit[j] := 1 + exp(2*
                      ln(abs(x_intercept*cap*2*pi*lsq[1])));
   xc_fit[j] := x_intercept/yc_fit[j] ;
   yc_fit[j] := (2*pi*lsq[1]*x_intercept*x_intercept*cap)
                                                 /yc_fit[j];
 end;
lsq[1] := 1;                           {used for frequency steps}
for j := 1 to 7 do
  begin
    lsq[1] := 10*lsq[1];
    yfreq[j] := 1+exp(2*ln(abs(x_intercept*cap*2*pi*lsq[1])));
    xfreq[j] := x_intercept/yfreq[j] ;
    yfreq[j] := (2*pi*lsq[1]*x_intercept*x_intercept*cap)
                                                 /yfreq[j];
    if lsq[1] > lsq[2] then                begin
         tau := j;
         if abs(yfreq[j] - y[1]) > abs(yfreq[j-1] - y[1]) then
            tau := j - 1 ;
         case tau of
            1: dummy :=  '10Hz'   ;
            2: dummy :=  '100Hz'  ;
            3: dummy :=  '1KHz'   ;
            4: dummy :=  '10KHz'   ;
            5: dummy :=  '100KHz' ;
            6: dummy :=  '1MHz'   ;
            7: dummy :=  '10MHz'  ;
         end;
         lsq[2] := 1e10;
```

```
              end;
           end;
           if choice < 3 then
             begin
               resistance  := cmplx+'R  ='+resistance+grk+' W'+cmplx;
               capacitance := cmplx+'C  ='+capacitance+cmplx+' F'
              end
           else
             begin
               resistance  := grk+'r'+cmplx+'  ='+
                                        resistance+grk+' W'+cmplx+'/cm';
                   capacitance := grk+'e'+cmplx+'  ='+
                                                capacitance+cmplx+' F'
                  end;
      end
    else
      begin
           for j:=1 to i do
            begin
             yc_fit[j]  := y[j] ;
             xc_fit[j]  := x[j]
            end ;
           x_intercept := x[i]
      end;
end;


{----------------------------------------------------------------------
       Takes the log of the frequency and the max and min values
 ---------------------------------------------------------------------}
procedure all_max_min_and_Log_freq;
begin
     lfreq[j]:= 0.4342945*ln(freq[j]);
     max_min(j,min_y,max_y,x[j]);
     max_min(j,minix,maxix,freq[j]);
end;

{------------------------------------------- }
{ Z' vs. -Z" data calculation and axis labels }
{------------------------------------------- }
```

```
procedure Re_Im_Impedance;

begin
     for j:= 1 to i do                         {i = # of data points}
        begin
          x[j] := mag[j]*cos(phase[j]*pi/180);       { Z' }
          y[j] := -mag[j]*sin(phase[j]*pi/180);      { Z" }
          all_max_min_and_Log_freq;
        end;
     min_y := 0;
     semi_circle_fit(x,y,i,xc_fit,yc_fit,tau);
     autoscale(min_y,max_y,ystep);
     y2step := ystep;
     if choice = 1 then
      begin
        min_x   := min_y;
        max_x   := max_y;
        xstep   := ystep;
        title := bld+'COMPLEX IMPEDANCE';
        xtitle := bld+'Re[Z'+up+'*'+dn+']  ('+grk+'W'+bld+')';
        ytitle := bld+'-Im[Z'+up+'*'+dn+']  ('+grk+'W'+bld+')';
          plot_it(0,title,xtitle,ytitle,ytitle2,name,subtitle,
                                          date,temperature,x,y,yc_fit,j)
        end
     else
      begin
           log_autoscale(minix,maxix,min_x,max_x,xstep);
           min_y2 := min_y;
           max_y2 := max_y;
           title   := bld+'COMPLEX IMPEDANCE FREQUENCY RESPONSE';
           xtitle  :=bld+'Frequency  (Hz)';
           ytitle  := bld+'Re[Z]  ('+grk+'W'+bld+')';
           ytitle2 := bld+'-IM[Z]  ('+grk+'W'+bld+')';
           plot_it(5,title,xtitle,ytitle,ytitle2,name,subtitle,
                                            date,temperature,lfreq,x,y,j);
      end;
end;

{------------------------------------------------ }
```

```
{ rho' vs. rho" data calculation and axis labels }
{----------------------------------------------- }
procedure Re_Im_Resistivity;

begin
  for j:= 1 to i do                  {i = # of data points}
    begin
      x[j] := (mag[j]*cos(phase[j]*pi/180))/cell_constant;
      y[j] := -(mag[j]*sin(phase[j]*pi/180))/cell_constant;
      all_max_min_and_Log_freq;
    end;
  semi_circle_fit(x,y,i,xc_fit,yc_fit,tau);
  min_y := 0;
  autoscale(min_y,max_y,ystep);
  y2step := ystep;
  if choice = 3 then
    begin
      min_x   := min_y;
      max_x   := max_y;
      xstep   := ystep;
      title := bld+'COMPLEX RESISTIVITY';
      xtitle := bld+'Re['+grk+'r'+bld+']  (' +grk+'W'+bld+'/cm)';
      ytitle := bld+'-Im['+grk+'r'+ bld +']  (' +grk+'W'+bld+'/cm)';
      plot_it(0,title,xtitle,ytitle,ytitle2,name,subtitle,
                                     date,temperature,x,y,yc_fit,j)
    end
  else
    begin
      log_autoscale(minix,maxix,min_x,max_x,xstep);
      min_y2 := min_y;
      max_y2 := max_y;
      title   := bld+'COMPLEX RESISTIVITY FREQUENCY RESPONSE';
      xtitle  :=bld+'Frequency  (Hz)';
      ytitle := bld+'Re['+grk+'r'+bld+ ']  (' +grk+'W'+bld+'/cm)';
      ytitle2:= bld+'-Im['+grk+'r'+ bld+']  (' +grk+'W'+bld+'/cm)';
      plot_it(5,title,xtitle,ytitle,ytitle2,name,subtitle,
                                    date,temperature,lfreq,x,y,j);
  end;
end;            {Resistivity}
```

```
{------------------------------------------ }
{ A' vs. A" data calculation and axis labels  }
{------------------------------------------ }
procedure Re_Im_Admittance;

begin    {Admittance}
  for j:= 1 to i do                    {i = # of data points}
     begin
       x[j] := cos(-phase[j]*pi/180)/mag[j];       { A' }
       y[j] := sin(-phase[j]*pi/180)/mag[j];       { A" }
       all_max_min_and_Log_freq;
       max_min(j,min_y2,max_y2,y[j]);
     end;
  autoscale(min_y, max_y, ystep);
  autoscale(min_y2, max_y2, y2step);
  if choice = 5 then
begin
     min_x   := min_y;
     max_x   := max_y;
     xstep   := ystep;
     min_y   := min_y2;
     max_y   := max_y2;
     ystep  := y2step;
     title := bld+'COMPLEX ADMITTANCE';
     xtitle := bld+'Re[A]  (' +grk+'W'+bld+'/cm)';
     ytitle := bld+'-Im[A]  (' +grk+'W'+bld+'/cm)';
     plot_it(0,title,xtitle,ytitle,ytitle2,name,subtitle,
                                   date,temperature,x,y,mag,j)
   end
else
   begin
     title := bld+'COMPLEX ADMITTANCE FREQUENCY RESPONSE';
     xtitle  :=bld+'Frequency  (Hz)';
     ytitle := bld+'Re[A]  (' +grk+'W'+bld+'/cm)';
     ytitle2 := bld+'-Im[A]  (' +grk+'W'+bld+'/cm)';
     plot_it(5,title,xtitle,ytitle,ytitle2,name,subtitle,
                                   date,temperature,lfreq,x,y,j);
   end;
```

```
end;

{--------------------------------------------------- }
{ Sigma' vs. Sigma" data calculation and axis labels  }
{--------------------------------------------------- }
Procedure Re_Im_Conductance;      { |A| <phase  = (1/|Z|) <-phase  }

begin    {Conductance}
  for j:= 1 to i do                    {i = # of data points}
   begin
     x[j] := 0.4342945*ln(cos(-phase[j]*pi/180)*cell_constant/mag[j]);
     y[j] := 0.4342945*ln(sin(-phase[j]*pi/180)*cell_constant/mag[j]);
       all_max_min_and_Log_freq;
       max_min(j,min_y2,max_y2,y[j]);
     end;
     min_y := trunc(min_y);
     max_y := trunc(max_y + 1);
     ystep := 1;
     min_y2 := trunc(min_y2);
     max_y2 := trunc(max_y2 + 1);
     y2step := 1;
  if choice = 7 then
    begin
      min_x   := min_y;
      max_x   := max_y;
      xstep   := ystep;
      min_y   := min_y2;
      max_y   := max_y2;
      ystep   := y2step;
      title := bld+'COMPLEX CONDUCTIVITY';
      xtitle := bld+'Re['+grk+'s'+up+bld+'*'+dn+']
                                          (' +grk+'W'+bld+'/cm)';
      ytitle := bld+'-Im['+grk+'s'+up+bld+'*'+dn+']
                                          (' +grk+'W'+bld+'/cm)';
      plot_it(0,title,xtitle,ytitle,ytitle2,name,subtitle,
                                        date,temperature,x,y,mag,j)
   end

  else
    begin
```

```
      log_autoscale(minix,maxix,min_x,max_x,xstep);
      title := bld+'COMPLEX CONDUCTIVITY FREQUENCY RESPONSE';
      xtitle  :=bld+'Frequency  (Hz)';
      ytitle := bld+'Re['+grk+'s'+up+bld+'*'+dn+']
                                             (' +grk+'W'+bld+'/cm)';
      ytitle2 := bld+'-Im['+grk+'s'+up+bld+'*'+dn+']
                                             (' +grk+'W'+bld+'/cm)';
      plot_it(8,title,xtitle,ytitle,ytitle2,name,subtitle,
                                         date,temperature,lfreq,x,y,j);
    end;
end;

{-----------------------}
{ Log sigma versus 1/T   }
{-----------------------}
procedure Arrhenius;
 var
   a,b,siga,sigb,chi2 :real;

begin    {Log sigma vs. 1/T}
for j:= 1 to i do                    {i = # of data points}
    begin
      x[j]     := 1/(freq[j]+273);             {1/T}
      y[j]     := 0.4342945*ln(mag[j]);     {Log sigma}
      max_min(j,min_x,max_x,x[j]);
      max_min(j,min_y2,max_y2,mag[j]);
    end;
   log_autoscale(min_y2,max_y2,min_y,max_y,ystep);
   min_y2 := 1/max_x - 273; max_y2 := 1/min_x - 273;
   autoscale(min_y2,max_y2,y2step);
   autoscale(min_x, max_x, xstep);
   fit(x,y,i,a,b,siga,sigb,chi2);
   siga := abs( min_x - max_x )/150;
   xc_fit[1] := min_x - siga;
   yc_fit[1] := a + b*xc_fit[1];
   for j := 2 to 150 do
     begin
       xc_fit[j] := xc_fit[j-1] + siga;
       yc_fit[j] := a + b*xc_fit[j]
    end;
   b := -b*2.303*8.62e-5;           { Eact =   eV}
   a := exp(a*2.3026);
   str(b:7:3,resistance);
   str(a:7,capacitance);
   resistance  := cmplx+'Eact = '+resistance+' eV';
   capacitance := grk+'s'+cmplx+'o  = '+
                       capacitance+' /('+grk+'W'+cmplx+'cm)' ;
   name :=resistance;
   date := capacitance;
   title:= bld+'ARRHENIUS';
   xtitle := bld+'T'+up+'-1'+dn+'   (K)'+up+'-1'+dn;
   ytitle := bld+'Re('+grk+'s'+bld+')  (' +grk+'W'+bld+'cm)'+up+'-1';
   ytitle2 := bld+'( C)';
   plot_it(2,title,xtitle,ytitle,ytitle2,name,subtitle,
                                   date,temperature,x,y,freq,i);
  end;   {Re_Conductivity_LFreq}

{-------------------------------------------}
{ M', M" vs. Log frequency and axis labels  }
{-------------------------------------------}
Procedure Re_Im_Modulus_LFreq;

begin    {Modulus}
  Ko:=2*pi*permittivity/cell_constant;
  for j:= 1 to i do                   {i = # of data points}
    begin
      x[j]    := -Ko*freq[j]*mag[j]*sin(phase[j]*pi/180); { weoZ"/K }
      y[j]    := Ko*freq[j]*mag[j]*cos(phase[j]*pi/180);  { weoZ'/K }
      all_max_min_and_Log_freq;
      max_min(j,min_y2,max_y2,y[j]);
    end;
   autoscale(min_y, max_y, ystep);
   autoscale(min_y2, max_y2, y2step);
   semi_circle_fit(x,y,i,xc_fit,yc_fit,tau);
   if choice = 9 then
    begin
     min_x   := min_y;
     max_x   := max_y;
    xstep   := ystep;
    title := bld+'ELECTRICAL MODULUS';
    xtitle  :=bld+'Frequency  (Hz)';
    Xtitle  := bld+'Re[M]';
    ytitle := bld+'Im[M]';
       plot_it(0,title,xtitle,ytitle,ytitle2,name,subtitle,
                                   date,temperature,x,y,mag,j)
  end
 else
   begin
    log_autoscale(minix,maxix,min_x,max_x,xstep);
    title := bld+'COMPLEX MODULUS FREQUENCY RESPONSE';
    xtitle  :=bld+'Frequency  (Hz)';
    ytitle  := bld+'Re[M]';
    ytitle2 := bld+'Im[M]';
    plot_it(5,title,xtitle,ytitle,ytitle2,name,subtitle,
                                     date,temperature,lfreq,x,y,j);
 end;
end;  {Re_Im_Modulus_LFreq}

{-------------------------------------------}
{ E', E" vs. Log frequency and axis labels  }
{          Dielectric Constant              }
{-------------------------------------------}
procedure Re_Im_Dielectric_Lfreq;

begin
  Ko:= 2*pi*permittivity*1e15/cell_constant;    {scaled by 1E15 to }
  for j:= 1 to i do                         { decrease round off error}
    begin
      x[j] := (1e15*sin(-phase[j]*pi/180))/(mag[j]*Ko*freq[j]);
      y[j] := (1e15*cos(-phase[j]*pi/180))/(mag[j]*Ko*freq[j]);
      all_max_min_and_Log_freq;
      max_min(j,min_y2,max_y2,y[j]);
    end;
  autoscale(min_y, max_y, ystep);
  autoscale(min_y2, max_y2, y2step);
  if choice = 11 then
   begin
```

```
     min_x   := min_y;
     max_x   := max_y;
     xstep   := ystep;
     title   := bld+'RELATIVE DIELECTRIC PERMITTIVITY';
     xtitle  := bld+'Re[k]';
     ytitle := bld+'Im[k]';
     plot_it(0,title,xtitle,ytitle,ytitle2,name,subtitle,
                                       date,temperature,x,y,MAG,j)
   end
else
   begin
    log_autoscale(minix,maxix,min_x,max_x,xstep);
    title   := bld+'RELATIVE DIELECTRIC PERMITTIVITY';
    xtitle  := bld+'Frequency  (Hz)';
    ytitle  := bld+'Re[k]';
    ytitle2 := bld+'Im[k]';
    plot_it(5,title,xtitle,ytitle,ytitle2,name,subtitle,
                                       date,temperature,lfreq,x,y,j);
   end;
end;

{-----------------------------------------------}
{ Loss factor vs. Log frequency and axis labels  }
{-----------------------------------------------}
procedure loss_factor;

begin
  Ko:= 2*pi*permittivity*1e15/cell_constant;    {scaled by 1E15 to }
  for j:= 1 to i do                            { decrease round off error}
    begin
      x[j] := -cos(phase[j]*pi/180)/sin(phase[j]*pi/180);
      y[j] := x[j];
      {x[j] := 180*arctan(y[j]/x[j])/pi;}  {to solve for loss angle }
      all_max_min_and_Log_freq;
      max_min(j,min_y2,max_y2,y[j]);
    end;
  autoscale(min_y, max_y, ystep);
  autoscale(min_y2, max_y2, y2step);
  min_x   := min_y;
  max_x   := max_y;
  xstep   := ystep;
  title   := bld+'LOSS TANGENT';
  xtitle  := bld+'Frequency  (Hz)';
  ytitle  := bld+'tan('+grk+'d  '+bld+')';
{    ytitle  := grk+'d';}
  ytitle2 := 'Try this';
  log_autoscale(minix,maxix,min_x,max_x,xstep);
  plot_it(1,title,xtitle,ytitle,ytitle2,name,subtitle,
                          date,temperature,lfreq,x,y,j);
 end;


{-----------------------------------------------}
{ Mag, Phase vs. Log frequency and axis labels  }
{-----------------------------------------------}
Procedure Mag_Phase_Lfreq;

begin     { freq vs. magnitude, phase}
  for j:= 1 to i do                  {i = # of data points}
    begin
      x[j] := mag[j];
      y[j] := phase[j];
      all_max_min_and_Log_freq;
      max_min(j,min_y2,max_y2,y[j]);
   end;
  autoscale(min_y, max_y, ystep);
  autoscale(min_y2, max_y2, y2step);
  log_autoscale(minix,maxix,min_x,max_x,xstep);
  title:= bld+'MAGNITUDE PHASE RESPONSE';
  xtitle :=bld+'Frequency  (Hz)';
  ytitle2 := bld+'PHASE  ['+grk+chr(39)+bld+']';
  ytitle :=bld+'MAGNITUDE  (' +grk+'W'+bld+')';;
  plot_it(5,title,xtitle,ytitle,ytitle2,name,subtitle,
                                     date,temperature,lfreq,x,y,j);
end;

{-------------------------------------------}
{ Rp, Cp vs. Log frequency and axis labels  }
{-------------------------------------------}
```

```
Procedure Rp_Cp;

begin
  for j:= 1 to i do                    {i = # of data points}
    begin
      x[j]    := mag[j]/cos(phase[j]*pi/180);
      y[j]    := -sin(phase[j]*pi/180)/(2*pi*freq[j]*mag[j]);
      all_max_min_and_Log_freq;
      max_min(j,min_y2,max_y2,y[j]);
     end;
    autoscale(min_y, max_y, ystep);
    autoscale(min_y2, max_y2, y2step);
    log_autoscale(minix,maxix,min_x,max_x,xstep);
    title := bld+'FREQUENCY RESPONSE';
    xtitle  :=bld+'Frequency  (Hz)';
    ytitle  := bld+'Resistance ['+grk+'W'+bld+']';
    ytitle2 := bld+'Capacitance [F]';
    plot_it(5,title,xtitle,ytitle,ytitle2,name,subtitle,
                                   date,temperature,lfreq,x,y,j);
end;  {Re_Im_Modulus_LFreq}

{-------------------------------------------}
{ Rs, Ls vs. Log frequency and axis labels  }
{-------------------------------------------}
Procedure Rs_Ls;         { RL in series }

begin
   for j:= 1 to i do                    {i = # of data points}
     begin
       x[j]    := mag[j]*cos(phase[j]*pi/180);
       y[j]    := mag[j]*sin(phase[j]*pi/180)/(2*pi*freq[j]);
       all_max_min_and_Log_freq;
       max_min(j,min_y2,max_y2,y[j]);
     end;
    autoscale(min_y, max_y, ystep);
    autoscale(min_y2, max_y2, y2step);
    log_autoscale(minix,maxix,min_x,max_x,xstep);
    title := bld+'FREQUENCY RESPONSE';
    xtitle  :=bld+'Frequency  (Hz)';
    ytitle  := bld+'Resistance ['+grk+'W'+bld+']';
    ytitle2 := bld+'Inductance [H]';
    plot_it(5,title,xtitle,ytitle,ytitle2,name,subtitle,
                                        date,temperature,lfreq,x,y,j);
end;  {Re_Im_Modulus_LFreq}

{-----------------------------------}
{ General purpose plot of any type  }
{-----------------------------------}
Procedure Gen_Plot;

begin
    for j:= 1 to i do                    {i = # of data points}
      begin
        max_min(j,min_x,max_x,freq[j]);
        max_min(j,min_y,max_y,mag[j]);
        max_min(j,min_y2,max_y2,phase[j]);
      end;
     autoscale(min_x, max_x, xstep);
     autoscale(min_y, max_y, ystep);
     autoscale(min_y2,max_y2,y2step);
     title   := bld+'Graph';
     xtitle  :=bld+'X Axis';
     ytitle  := bld+'Y Axis';
     plot_it(0,title,xtitle,ytitle,ytitle2,name,subtitle,
                               date,temperature,freq,mag,phase,i);
end;
{--------------------------------------------------------
     Initialize device and plot types and data arrays
 --------------------------------------------------------}
procedure init_labels;
var
 i : integer;
 begin
   device := 1 ;
   ploty1:='y';
   ploty2:='y';
   For i := 1 to 10 do
    begin
```

```
      xi[i] := 0; y[i] := 0;
      data_label[i] := '';
      data_legend[i].icol := -1;
      data_legend[i].title := ''
    end;
 end;


begin                                 {M A I N   P R O G R A M}
  num_point := 1;
 device_flag := 0;
 plot_all_files := 'N';
 counter := 1;
 d_choice_window;

 Getfile_names(counter);
 repeat
   init_labels;
    files_processed := 1;
    resistance := '';
    Repeat
      Textbackground(lightgray);
      if plot_all_files = 'Y' then
        files_to_Analyze[9] := 'All Files !';
      write_data_situation(Lightgray, blue,
                                  'Using '+ files_to_analyze[9]);
      GotoXY(42,23);
      TextBackground(red);
      TextColor(WHITE);
      ch := ReadKey;
      if ch = #0 then
        begin
          ch := ReadKey;
          if ch = #59 then
             begin
               redo_filenames(plot_all_files,counter,file_counter,
                                                     files_to_analyze);
               write_data_situation(Red, WHite,'Please Wait !
                      Reading Data File '+files_to_analyze[counter]);
               getfile_names(counter);
```

```
            end;
          if ch = #68 then resistance := '20';
        end
      else if ch <> #13 then
        begin
          if ch = #8 then
            begin
              write(ch); write(' ');
              delete(resistance,length(resistance),1);
            end
          else
            resistance := resistance + ch;
          GotoXY(42,23);
          write(resistance);
        end;
    until (ch = #13) or (resistance = '20');
    val(resistance,choice,j);
    repeat
      if (plot_all_files = 'Y')
                         and (files_processed < file_counter) then
          Getfile_names(files_processed)
      else if plot_all_files = 'Y' then
        begin
          counter := 1;
          Getfile_names(counter)
        end;
      ytitle2:= '';
      resistance := '';
      capacitance := '';
      if (choice = 13) and
(copy(files_to_analyze[9],length(files_to_analyze[9])-2,3) <> 'SIG')
      then
         begin
           write_data_situation(Red, white, 'Not Arhenius File');
           choice := 21
         end;
      case choice of  { converts the raw freq, mag and phase as
                          selected through data choice window }
          1,2: Re_Im_Impedance;

          3,4: Re_Im_Resistivity;

          5,6: Re_Im_Admittance;

          7,8: Re_Im_Conductance;

         9,10: Re_Im_Modulus_LFreq;

        11,12: Re_Im_Dielectric_Lfreq;

           13: Arrhenius ;

           14: Rp_Cp;

           15: Rs_Ls;

           16: loss_factor;

            0: Mag_Phase_Lfreq;

           17: Gen_Plot;

      end; {case}
      files_processed := files_processed + 1;
   until ( plot_all_files = 'N' ) or
                            ( files_processed = file_counter + 1 );
    d_choice_window;
    init_labels;

  until choice=20;
 end;
BEGIN  { main program }
   plot_labels := 'N';
   plot_scale := 1.0;
   screen_file('introgrp.scr');
   readln;
   get_files(files_to_analyze,file_counter);
```

```
    repeat
       draw_graphs;
    until choice = 20;
    TextBackground(Blue);
    textcolor(white);
    clrscr;
END.
```

Pascal Units

GetFiles.Pas

This unit contains the routines to read, edit and write files, display introduction, help and user input screen files created by BOX.EXE, and select files as described by the file manager. All the window routines for the data choice window are also included in this unit.

```
{$R-}    {Range checking off}
{$B-}    {Boolean complete evaluation on}
{$S+}    {Stack checking on}
{$I-}    {I/O checking on}
{$N-}    {No numeric coprocessor}
UNIT GETFILES;
{-------------------------------------------------------------------}
interface

Uses
  crt,
  dos,
  edtool;

Type

 file_to_analyze   = Array [1..9] of pathstr;
 data_array        = Array [1..300] of real;

procedure get_files(var file_names: file_to_analyze;
                                              var file_count:byte);

procedure Draw_box(x1,y1,x2,y2 :byte);  {begin of drawing box}

procedure  screen_file(scrfile:pathstr);

procedure open_file_to_read(var infile:text; file_name:pathstr;
                                           var error_chk: integer);
```

```
procedure open_file_to_write(var outfile:text; file_name:pathstr;
                                                var error_chk: integer);

procedure edit_file(file_name: pathstr);

procedure write_data_situation(text_color, back_color : byte ;
                                                write_string : pathstr);

Procedure d_choice_window;

procedure redo_filenames(var plot_all_files:char;var counter,
           file_counter:byte;var files_to_analyze : file_to_analyze);

{------------------------------------------------------------------}
Implementation

var
  files_to_analyze : file_to_analyze;
  file_counter : byte;
  file_name : pathstr;

{===================================================================}
   This subroutine displays packed video memory files created using
   BOX.EXE, Nescatunga Software, Box 5942, Katy, TX 77450.
{-------------------------------------------------------------------}
procedure  screen_file(scrfile:pathstr);

type
   pack        = record
                  packnm : Byte;  {run length}
                  packch : char;  {repeated character}
                  packat : Byte;  {repeated attribute}
                 end;
   map         = record
                  scrch : char;
                  scrat : Byte;
                  end;
   screen      = array[1..25,1..80] of map;
   anystr      = string[80];

var
    filevarm   : file;
    packbuf    : array[1..2000] of pack;
    cs         : screen Absolute $b800:0000;
    ms         : screen Absolute $b000:0000;
    loadscr    : screen;
    filenm     : anystr;
    color      : boolean;
{======================================================}
procedure checkcolor;
begin
     if (Mem[0000:1040] and 48) <> 48
        then color := true
        else color := false;
end;

{======================================================}
procedure loadpacked(pfile:anystr);
{                                                      }
{ This procedure loads a Packed Format screen created }
{ by BOX.  The Packed format utilizes a run-length    }
{ encoding scheme that must be unpacked.  Each record }
{ in a Packed Format file is three bytes long. Byte 1 }
{ is the run length, i.e. the number of characters to }
{ repeat.  Byte 2 is the character to repeat and      }
{ byte 3 is the attribute of the character.           }
{                                                      }
var
    ii,jj,sloc,sx,sy,numrec  : integer;

begin
   sloc := 1;              {SLoc is location on screen}
   Assign(filevarm,pfile);
   {$I-} reset(filevarm); {$i+}
   if IOResult = 0 then          {found good file name}
     begin
        blockread(filevarm,packbuf,48,numrec);
        jj := 0;
        while sloc < 2001 do
        begin
          jj := jj + 1;
          for ii := 1 to packbuf[jj].packnm do
           begin
            sy := (sloc-1) div 80 + 1;       {row}
            sx := (sloc-1) mod 80 + 1;       {column}
            loadscr[sy,sx].scrch := packbuf[jj].packch;
            loadscr[sy,sx].scrat := packbuf[jj].packat;
            sloc := sloc + 1;
           end;
        end;
        if color then cs := loadscr
                 else ms := loadscr;
        close(filevarm);
   end
   else                             {couldn't find file}
     begin
       GotoXY(1,24);
       write('ERROR - Could not find file ',
                                pfile,' Copy file in directory');
     end;
end;


{======================================================}
begin {Main Routine}
  checkcolor;
  ClrScr;
  loadpacked(scrfile);
end;

{======================================================================
Opens a file to Read and returns and returns an error message if file
             is not found, prevents program from crashing!
{--------------------------------------------------------------------}
procedure open_file_to_read(var infile:text; file_name:pathstr;
                                             var error_chk : integer);
  begin
    assign(infile,file_name);                {Open Data File}
    {$I-}
    reset(infile);
    {$I+}
    error_chk := IOResult;
    If IOResult <> 0 then
      write_data_situation(red,white,'Error reading File '+file_name);
  end;

{=====================================================================
Opens a file to Write and returns and returns an error message if file
             is not found, prevents program from crashing!
{---------------------------------------------------------------------}

procedure open_file_to_write(var outfile:text; file_name:pathstr;
                                              var error_chk: integer);

  begin
    assign(outfile,file_name);               {Open Data File}
    {$I-} rewrite(outfile);
    error_chk := IOResult; {$I+}
    If IOResult <> 0 then
      write_data_situation(red,white,'Error reading File '+file_name);

  end;

{=====================================================================
A low level editor for files containing less than 500 lines using the
   edit commands in the Unit: EDTOOLS.PAS.  A backup of the file is
                   always made if the file is saved.
{---------------------------------------------------------------------}
PROCEDURE edit_file(file_name:pathstr);

type

  lines             = string[80];
  page              = array [1..500] of lines;
```

```
var
   x                          : integer;
   D                          : Dirstr;
   N                          : Namestr;
   E                          : Extstr;
   ans                        : char;
   infile, outfile            : text;
   last , line_number , k, position      : integer;
   line                       : page;

   {---------------------------------------------
         Displays 19 lines on screen at a time
   {---------------------------------------------}
  procedure write_19_lines(line_number, last: integer);
   var
     i,j,k  :integer;

    begin
       textbackground(blue);
       textcolor(lightgray);
       gotoxy(1,1);
       clrscr;
       i := 0;
       while (line_number + i <= last) and ( i < 19) do
         begin
            gotoxy(1,i+1);
            write(line[line_number + i]);
            i := i + 1 ;
         end;
    end;

begin
   for k := 1 to 500 do line[k] := '';
   textbackground(black);
   window(1,1,80,25);
   clrscr;
   open_file_to_read(infile,file_name,k);        {Open Data FIle}
   fsplit(file_name,d,n,e);
   file_name := fexpand(file_name);
```

```
open_file_to_write(outfile,d+'z'+n+e,k);
last := 0;
repeat
    last := last + 1 ;
    readln(infile,line[last]);
    writeln(outfile,line[last]);
until eof(infile);

close(outfile);
close(infile);
textcolor(white);
textbackground(red);
gotoXY(2,25);
write(' F1 - INS ');
gotoXY(22,25);
write(' F4 - DEL ');
gotoXY(42,25);
write(' F9 - SAVE ');
gotoxy(68,25); write(' F10 - QUIT ');
gotoXY((40-(length('Edit : '+file_name) div 2)-1),1);
write(' Edit : ',file_name,' ');
window(1,3,80,23);
ans := ' ';
line_number := 1;
position := 1;
k := 1;
write_19_lines(line_number,last);
while ans <> #27  do                                {esc to quit}
  begin
    textbackground(blue);
    textcolor(WHITE);
    gotoxy(1,position) ;
    ptoolent(line[line_number],'S',80,0,k);
    textbackground(blue);
    textcolor(lightgray);
    gotoXY(1,position); write(line[line_number]);
    case k of
        1, 80   : begin    {cr or dn arrow}
                    if line_number < last then
                      begin
```

```
                line_number := line_number + 1;
                if position < 19 then
                  position := position + 1
                else
                  begin
                   write_19_lines(line_number-18,last);
                   position := 19
                  end;
               end;
          end;
72        : begin    {up arrow}
             if line_number > 1 then
                line_number := line_number - 1;
             if position > 1 then
                position := position - 1
             else
              begin
               write_19_lines(line_number,last);
               position := 1
             end;
           end;
79,81      : begin          {79 = END}           {Pgdn}
             if (line_number+18 < last) and (k=81) then
               begin
                 line_number := Line_number + 18 ;
                 write_19_lines(line_number,last);
                 line_number := line_number + position - 1
               end
              else
                begin        {End}
                  line_number := last;
                  position := 19;
                  write_19_lines(line_number-18,last)
                end;
           end;
73, 71      : begin        {Pgup}
              if (line_number - (17 + position)  > 0) and
                                         (k = 73) then
               begin
```

```
                              line_number := Line_number-(17+position);
                              write_19_lines(line_number,last);
                              line_number := line_number + position - 1
                            end
                           else
                            begin         {Home}
                              line_number := 1;
                              position := 1;
                              write_19_lines(line_number,last);
                            end;
                         end;

{F1 - INS} 59       :  begin
                            for k := last downto line_number  do
                              line[k+1] := line[k];
                            last := last + 1 ;
                            line[line_number] := '';
                          write_19_lines(line_number-position+1,last);
                         end;
{F4 - Del} 62       :  begin
                            for k := line_number to last do
                              line[k] := line[k+1];
                            line[last] := '';
                            last := last - 1 ;
                          write_19_lines(line_number-position+1,last);
                         end;
           67,68        :  begin            {F9 - save, F10 - quit}
                            textbackground(red);
                            textcolor(white);
                            if  k = 68 then
                             begin
                                gotoXY(25,position);
                                Write(' ESC : Quit without Saving  ');
                                ans := ReadKey;
                              end;
                             If (k = 67) or (ans = #67) then
                              begin
                                gotoXY(25,position);
                                write(' Please Wait - Saving File ! ');
                              open_file_to_write(outfile,file_name,k);
                               for k := 1 to last do
                                 writeln(outfile,line[k]);
                               close(outfile);
                              if ans = #67 then ans := #27;
                             end;
                        end;
        end ;
      end;
      window(1,1,80,25);
      clrscr;
  end;


{=====================================================================
     Subroutine draws double lined boxes for the pop up windows in
               the data-choice window in Data-Analysis.
{---------------------------------------------------------------------}
Procedure Draw_box(x1,y1,x2,y2 :byte);
     var          {top left corner, bottom right corner}
      i  : byte;
     begin
       GotoXY(x1,y1);
       write(#213);
       for i:= 2 to (x2-1) do write(#205);
       write(#184);
       for i:= 2 to (y2-1) do
        begin
          GotoXY(x1,i); write(#179);
          GotoXY(x2,i);write(#179);
        end;
       GotoXY(x1,y2);
       write(#212);
       for i:= 2 to (x2-1) do write(#205);
       write(#190);
     end;

{=====================================================================
   Displays an informative message at the bottom of the data-choice
                    window e.g. INVALID DATA FILE
```

```
{---------------------------------------------------------------------}
procedure write_data_situation(text_color, back_color : byte ;
                                           write_string : pathstr);
   var
     x1, x2, y1, y2  : Byte;
     k               : integer;

     begin
        x1 := lo(windmin); y1 := hi(windmin);
        x2 := lo(windmax); y2 := hi(windmax);
        window(1,1,80,25);
        TextBackGround(Black);
        GotoXY(1,25);
        For k := 1 to 75 Do Write(' ');
        TextBackground(Text_color);
        Textcolor(Back_color);
        GotoXY(40-(length(write_string) div 2),25) ;
        Write(' '+write_string+' ') ;
        window(x1,y1,x2,y2) ;
end;

{=====================================================================
  File manager window is actually drawn on the screen and no a screen
                            file using box
{---------------------------------------------------------------------}
Procedure first_screen;
var j : byte;
 begin
    highvideo;
    TEXTBACKGROUND(blue);
    clrscr;
    textcolor(YELLOW);
    WINDOW(25,1,80,9);
    draw_box(1,1,55,9);
    GotoXY(17,2); write('DATA FILES TO ANALYZE');
    draw_box(1,3,55,9);
    textcolor(white);
    Window(26,4,79,8);
    for j:= 1 to file_counter do
    begin
         if j < 5 then gotoxy(2,j) else gotoxy(28,j-4);
         write(files_to_analyze[j]) ;
    end;
    textcolor(YELLOW);
    WINDOW(2,1,24,9);
    draw_box(1,1,22,9);
    textcolor(lightgray);
    GotoXY(9,1);write(' HELP ');
    GOtoXY(4,4);write('F1  - Chg Drive');
    GotoXY(4,6);write('F4  - Edit');
    GotoXY(4,3);write('INS - Select File');
    GotoXY(4,7);write('ESC - Quit');
 end;

{=====================================================================
    File manager routines to search and sort the directories and to
               move and select files in the file manager
{---------------------------------------------------------------------}
{$S-}
const
  MaxDirSize = 512;

type
  DirPtr   = ^DirRec;     {pointer of type all file data}
  DirRec   = record
               Attr: Byte;
               Time: Longint;
               Size: Longint;
               Name: string[12];
             end;
  DirList  = array[0..MaxDirSize - 1] of DirPtr;

var
  WideDir: Boolean;
  Count: Integer;
  position  : longint;
  Path: PathStr;
  Dir: DirList;
```

```
function NumStr(N, D: Integer): String;
begin
  NumStr[0] := Chr(D);
  while D > 0 do
  begin
    NumStr[D] := Chr(N mod 10 + Ord('0'));
    N := N div 10;
    Dec(D);
  end;
end;


procedure QuickSort(L, R: Integer);
var
  I, J: Integer;
  X, Y: DirPtr;
begin
  I := L;
  J := R;
  X := Dir[(L + R) div 2];
  repeat
    while (Dir[I]^.Name <  X^.Name) do Inc(I);
    while (X^.Name < Dir[J]^.Name) do Dec(J);
    if I <= J then
    begin
      Y := Dir[I];
      Dir[I] := Dir[J];
      Dir[J] := Y;
      Inc(I);
      Dec(J);
    end;
  until I > J;
  if L < J then QuickSort(L, J);
  if I < R then QuickSort(I, R);
end;

procedure GetCommand;
var
  I,J: Integer;
  Attr: Word;
  S: PathStr;
  D: DirStr;
  N: NameStr;
  E: ExtStr;
  F: File;
begin
  WideDir := TRUE;
  Path := FExpand(Path);
  if Path[Length(Path)] <> '\' then
  begin
    Assign(F, Path);
    GetFAttr(F, Attr);
    if (DosError = 0) and (Attr and Directory <> 0) then
      Path := Path + '\';
  end;
  FSplit(Path, D, N, E);
  if N = '' then N := '*';
  if E = '' then E := '.*';
  Path := D + N + E;
end;

procedure FindFiles;
var
  F: SearchRec;
begin
  Count := 0;
  FindFirst(Path, ReadOnly + Directory + Archive, F);
  while (DosError = 0) and (Count < MaxDirSize) do
  begin
    if F.Attr = $10 then F.Name := '$' + F.Name + '\' ;
    GetMem(Dir[Count], Length(F.Name) + 10);
    Move(F.Attr, Dir[Count]^, Length(F.Name) + 10);
    Inc(Count);
    FindNext(F);
  end;
end;

procedure SortFiles;
```

```
begin
  if Count <> 0 then
    QuickSort(0, Count - 1);
end;

procedure PrintFiles(var position:longint);
var
  I, P, j, x, y: Integer;
begin
     TEXTBACKGROUND(lightgray);
     textcolor(black);
     WINDOW(2,11,79,24);
     ClrScr;
     WINDOW(2,11,80,24);
     draw_box(1,1,78,14);
     WINDOW(2,11,79,24);
     J := 1;
     y := 2;
     x := 0;
     for I := Position to Count-1 do        { begin of file writing }
     with Dir[I]^ do
       begin
         if not ((y = 13) and (j = 5)) then
            begin
                 if J < 5 then j := j + 1
                 else
                   begin
                        y := y + 1;
                        J := 1
                   end;
                 x := 15 * (j-1) + 3 ;
                 if pos('$',name) = 1 then delete(name,1,1);
                 GotoXY(x,y);  Write(Name);
            end;
         end;
     textbackground(red);
     textcolor(white);
     I := 40 - ( length(Path) div 2 );
     GotoXY(i-1,1); Write(' ',Path,' ');
```

```
end;


Procedure move_around;
 var
   I,x,y          :  integer;
   ch             :  char;
   j              :  byte;
   temp           :  string[1];
   S              :  PathStr;
   D              :  DirStr;
   N              :  NameStr;
   E              :  ExtStr;
   F              :  File;

 procedure highlight_region(x,y :integer;bg,tc:byte);
     begin
       TEXTBACKGROUND(bg);
       textcolor(tc);
       window(x+1,y+10,x+13,y+10);
       clrscr;
       write(Dir[i]^.name);
     end;

   procedure find_directory;
      begin
        GetCommand;
        FindFiles;
        SortFiles;
        position := 0;
        i := 0;
        x := 18; y := 2;
        printfiles(position);
        highlight_region(x,y,black,white);
      end;

begin
    file_counter:= 1;
    find_directory;
    repeat
```

```
      WINDOW(2,11,79,24);
      ch := ReadKey;
      if ch = #0 then
        begin
          highlight_region(x,y,lightgray,black);
          ch := readKey;
          case ch of
{F1}         #59 : begin
                     FSplit(Path, D, N, E);
                     WINDOW(2,11,79,25);
                     textbackground(red);
                     textcolor(White);
                     GotoXY(2,15);
                     for i := 1 to 76 do write(' ');
                     GotoXY(2,15);
                     write(' New Drive or File Selection : ');
                     readln(path);
                     if Pos(':',Path) <> 2 then Path := D + Path ;
                     textbackground(blue);
                     GotoXY(1,15);
                     for i := 1 to 78 do write(' ');
                     Find_directory;
                   end;
{F4}         #62 : begin
                     FSplit(Path, D, N, E);
                     files_to_analyze[file_counter]:= d+Dir[i]^.Name;
                     edit_file(files_to_analyze[file_counter]);
                     first_screen;
                     find_directory;
                   end;

{Home}       #71 : Find_Directory;

{End}        #79 : begin
                     if count < 59 then beep    {45}
                     else begin
                       if (count-59) mod 5 <> 0 then
                         position := ( ( ( count - 59 ) div 5)+1)*5
                       else position := ( ( count - 59 ) div 5 ) * 5;
```

```
                      i := position + 54 ;
                      x := 3;
                      y := 13;
                      printfiles(position);
                     end;
                   end;
{up}         #72 : begin
                     if i < 5 then  beep
                     else
                       begin
                         i := i - 5 ;
                         if y = 2 then
                            begin
                             position := position - 5;
                             printfiles(position)
                            end
                         else y := y - 1
                       end;
                   end;
{left}       #75 :  begin
                     if i > 0 then
                       begin
                         i := i - 1 ;
                         if (i mod 5) = 3  then
                           begin
                             x := 63 ;
                             if y = 2 then
                              begin
                                position := position - 5;
                                printfiles(position)
                              end
                             else y := y - 1 ;
                           end else
                             x := x - 15;
                       end
                       else  beep;
                   end;
{right}      #77 : begin
                     if (i < count-1) then
```

```
                    begin
                      i := i + 1 ;
                      if (i mod 5) = 4  then
                        begin
                          x := 3;
                          if y = 13 then
                           begin
                             position := position + 5;
                             printfiles(position)
                           end
                          else y := y + 1;
                        end else
                          x := x + 15;
                     end
                     else  beep;
                end;
{down}      #80 :   begin
                      if i + 6 > count then beep
                      else
                        begin
                          i := i + 5 ;
                          if y = 13 then
                           begin
                                 position := position + 5;
                                 printfiles(position)
                           end
                           else y := y + 1;
                        end;
                    end;
{Ins}        #82 :   begin
                        If Pos('\',Dir[i]^. Name) = 0 then
                        begin
                        FSplit(Path, D, N, E);
                       files_to_analyze[file_counter]:=d+Dir[i]^.Name;
                        first_screen;
                        file_counter := file_counter + 1 ;
                      end else beep;
                     end;
             end; {case}
```

```
            highlight_region(x,y,black,white);
         end
      else
         begin
            case ch of
               #13 : begin
                        FSplit(Path, D, N, E);
                        if Dir[i]^.Name = '.\' then
                           Path := copy(Path,1,3)
                        else if Dir[i]^.Name = '..\' then
                           begin
                              position := pos('.',path) - 2;
                              delete(path,position,position+14);
                              position := length(path);
                              Repeat
                                 e := copy(path,position,1);
                                 if (e <> '\') and  (e <> ':') then
                                   delete(path,position,1)
                                 else
                                  if  e = '\' then  e := ':' ;
                                 position := position - 1 ;
                              until  e = ':'
                           end
                           else if Pos('\',Dir[i]^. Name) <> 0 then
                                Path := D + Dir[i]^.Name + '*.*' ;
                           find_directory;

           end; {case}


           end;

      highlight_region(x,y,black,white);
   end;
   until ( ch = #27) or (file_counter > 8);         {esc}
end;

{==================================================================
   FIle-Manager subroutine which calls routines to find directories,
          move around in the file manager and select files.
```

```
{---------------------------------------------------------------------}
procedure get_files(var file_names: file_to_analyze;
                                                var file_count:byte);
begin
  window(1,1,80,25); CLrScr;
  file_counter := 0;
  first_screen;
  path := 'c:';
  move_around;
  file_names := files_to_analyze;
  file_count := file_counter;
  window(1,1,80,25); Clrscr;
end;

{=====================================================================
                 Begin routines for Data Choice Window
{---------------------------------------------------------------------}


{=====================================================================
      Writes first letter of string in different color for menus
{---------------------------------------------------------------------}
procedure write_command(key_color : byte; commandkey : pltstring;
                   org_color : byte; rest_of_string : pltstring);
  begin
    textcolor(key_color);
    write(commandkey);
    textcolor(org_color);
    write(rest_of_string)
  end;

{=====================================================================
                Displays data choice screen memory file
{---------------------------------------------------------------------}
Procedure d_choice_window;
begin
   window(1,1,80,25);
   TextBackground(red);
   TextColor(white);
   screen_file('d_choice.scr');
   GotoXY(42,23);
end;

{=====================================================================
                       Pop up File Option window
 ---------------------------------------------------------------------}
procedure redo_filenames(var plot_all_files:char;var counter,
            file_counter:byte;var files_to_analyze : file_to_analyze);

var
    tmp, name_length, vert_length    :integer;
    chr                              :string[1];
    ch                               :char;
    count                          : byte;

  {--------------------------------------
      Draw window for pop up menu
   --------------------------------------}
  procedure draw_window(x1,y1,x2,y2:byte);
   begin
     textbackground(lightgray);
     textcolor(black);
     window(x1,y1,x2,y2);
     Clrscr;
     window(x1,y1,x2,y2+1);
     draw_box(1,1,x2-x1+1,y2-y1+1);
     draw_box(1,1,x2-x1+1,3);
     gotoXY(1,3); write(#198);
     gotoXY(x2-x1+1,3); write(#181);
    end;

  {--------------------------------------
     Write options in first pop up menu
   --------------------------------------}
  PROCEDURE file_option_window;
   begin
     draw_window(1,1,20,11);
     GotoXY(5,2); write('FILE OPTION');
   GotoXY(3,4); write_command(red,'R',black,'e-order Files');
   GotoXY(3,6); write_command(red,'P',black,'lot Type');
   GotoXY(3,8); write_command(red,'B',black,'egin All Over');
   GotoXY(3,10); write_command(red,'E',black,'xit');
   write_data_situation(Lightgray, blue,
          'Press High Lighted Charactor For Option');
 end;

{--------------------------------------
     Write List of selected files
 --------------------------------------}
procedure list_dir_window(x1,y1,x2,y2:byte);
 var count : byte;
  begin
     draw_window(x1,y1,x2,y2);
     For count:= 1 to file_counter-1 do
      begin
         GotoXY(6,count+3);
         write(files_to_analyze[count]);
      end;
      textcolor(red);
      For count := 1 to file_counter-1 do
        begin
           GotoXY(3,count+3); write(count,'.')
        end;
      textcolor(black);
  end;


{--------------------------------------
          Menu for Plot Type
 --------------------------------------}
 procedure plot_type_window(x:char);
  begin
    draw_window(19,6,40,14);
    GotoXY(8,2); write('PLOT TYPE');
    GotoXY(3,4); write_command(red,'A',black,'ll On One Graph');
    GotoXY(3,6); write_command(red,'O',black,'ne File One Plot');
    GotoXY(3,8); write_command(red,'E',black,'xit');
    textcolor(white); textBackground(black);
    case x of
      'Y':  begin GotoXY(3,4); write('All On One Graph')  end;
      'N':  begin GotoXY(3,6); write('One File One Plot') end;
    end;
  end;

{-------------------------------------
        Menu for Reorder files
 -------------------------------------}
 procedure reorder_files;
  var
    temp_list : file_to_analyze;
    ans       : char;
    j,k,l     : integer;

  begin
    write_data_situation(White, Red,
      'Enter New Number Order          or              K - Kill');
    l := 1;
    temp_list[9] := files_to_analyze[9];
    repeat
      GotoXY(4,l+4); ch := ReadKey;
      if ch = #0 then ch := ReadKey ;
      if (ch > #48 ) and (ch < #58) then
        begin
          write(ch);
          val(ch,j,k);
          temp_list[j] := files_to_analyze[l];
          l := l + 1;
        end;
     if ch = #72 then l := l - 1 ;
     if ch = #80 then l := l + 1 ;
     if (ch = 'k') or (ch = 'K') then begin
        file_counter := file_counter - 1;
        l := l + 1
      end;
     if l = file_counter then
      begin
        Beep;
```

```
          if (ch = 'f') or (ch = 'F') then
            write_data_situation(White, Red,
             'Press "D" if Done else Press "R" to ReOrder Files')
          else
            write_data_situation(Red, White,
            'Make Sure Files Sequential ! Press "F" if Finished');
        end;
     until (ch = 'f') or (ch ='F') ;
     files_to_analyze := temp_list;
     counter := 1;
  end;

begin  {main routine for re_do_filenames}
 Repeat
      File_Option_Window;
      ch := ReadKey;
  case ch of
    'r','R' :
        begin
          name_length := 20;
          for tmp := 1 to file_counter-1 do
          if name_length < length(files_to_analyze[tmp]) then
             name_length := length(files_to_analyze[tmp]);
          list_dir_window(18,4,25+name_length,file_counter+9);
          GotoXY(((name_length+8) div 2)-6,2);
          write('RE-ORDER FILES');
          GotoXY(3,file_counter+4);
          write_command(red,'R',black,'e-Order Files');
          GotoXY(3,file_counter+5);
          write_command(red,'D',black,'one');
          repeat
            ch := ReadKey;
            if (ch = 'r') or (ch = 'R') then reorder_files;
            list_dir_window(18,4,25+name_length,file_counter+9);
            GotoXY(((name_length+8) div 2)-6,2);
            write('RE-ORDER FILES');
            GotoXY(3,file_counter+4);
            write_command(red,'R',black,'e-Order Files');
            GotoXY(3,file_counter+5);
          write_command(red,'D',black,'one');
          if (ch = 'd') or (ch = 'D') or (ch = #27) then
            begin
              d_choice_window;
              file_option_window
            end
         until (ch = 'd') or (ch = 'D') or (ch = #27)
      end;
'b','B' :
      begin
          for count := 1 to 8 do
             files_to_analyze[counter] := '';
          get_files(files_to_analyze,file_counter);
          counter := 1;
          plot_all_files:='N'
      end;

'p','P' :
      begin
           repeat
           plot_type_window(plot_all_files);
           ch := ReadKey;
           if ch = #0 then ch := ReadKey;
           if (ch = 'A') or (ch = 'a') or (ch = #72) then
             begin
               plot_all_files := 'Y';
               plot_type_window('Y');
               write_data_situation(Lightgray, Red,
                'All Selected Files Plotted On The Same Graph');
             end;
           if (ch ='O') or (ch = 'o') or (ch = #80) then
             begin
               if (ch = #80) then
                 begin
                   plot_all_files := 'N';
                   plot_type_window('N');
                   write_data_situation(Lightgray, Red,
                     'Press "O" to Select File to Plot')
                 end
               else begin
```

```
 write_data_situation(Lightgray, Red,
  'HighLight File To Plot Then Enter "E" to Exit');
 plot_all_files := 'N';
 plot_type_window('N');
 if (counter < 0) or (counter > file_counter) then
     counter := 1;
 name_length := 20;
 for tmp := 1 to file_counter-1 do
  if name_length < length(files_to_analyze[tmp]) then
     name_length := length(files_to_analyze[tmp]);
list_dir_window(38,11,45+name_length,file_counter+15);
 GotoXY(((name_length+8) div 2)-9,2);
 write('SELECT FILE TO PLOT');
 GotoXY(((name_length+8) div 2)-2,file_counter+4);
 write_command(red,'E',black,'xit');
 repeat
   TextBackGround(black);
   TextColor(white);
   GotoXY(6,counter+3);
   write(files_to_analyze[counter]);
   ch := ReadKey;
   if (ch = #0) then ch := ReadKey;
   if ((ch > #48) and (ch < #57))
         or (ch = #72) or (ch =#80) then
     begin
       TextColor(Black);
       TextBackGround(lightgray);
       GotoXY(6,counter+3);
       write(files_to_analyze[counter]);
       TextBackGround(black);
       TextColor(white);
       if (counter > 1) and (ch = #72)
          then counter := counter - 1;
       if (counter < file_counter-1) and (ch = #80)
          then counter := counter + 1;
       if ((ch > #48) and (ch < #57)) then
        begin
           val(ch,counter,tmp);
           if (tmp = 0) and
```

```
                              (counter < file_counter) then
                         begin
                          GotoXY(6,counter+3);
                          write(files_to_analyze[counter])
                         end
                      end;
                   end;
               until (ch = #27) or (ch = 'e') or (ch ='E');
               ch := 'a';
               write_data_situation(Red, WHite,
                'Reading Data File '+files_to_analyze[counter]);
               d_choice_window;
               file_option_window;
           end;
                end;
                until (ch='e') or (ch = 'E') or ( ch = #27);
                d_choice_window;
                ch := 'a';
             end;
      end;
   until (ch = 'e') or (ch = 'E') or (ch = 'B') or
                                             (ch = 'b') or (ch = #27);
   d_choice_window;
 end;

begin
end.
```

Math.Pas

This unit contains the algorithms to determine the autoscaling routine used to plot the data in the data analysis routine and the linear least-squares routine to determine the Arrhenius variables.

```
{$R-}    {Range checking off}
{$B+}    {Boolean complete evaluation on}
{$S+}    {Stack checking on}
{$I+}    {I/O checking on}
{$N-}    {No numeric coprocessor}


UNIT MATH;
{-------------------------------------------------------------------}
interface

Uses
  crt,
  dos;

procedure max_min(j :integer;var  x1,x2:real; x:real);

procedure autoscale(var min, max, step : real);

procedure log_autoscale(x1,x2 : real; var min, max, step : real);

procedure fit(x,y: data_array; ndata: integer;
                                VAR a,b,siga,sigb,chi2: real);

{-------------------------------------------------------------------}
Implementation

{--------------------------------------------------------------------
    Determines whether x1 is the minimum and and x2 is the maximum
           compared to x and is assigned to it if it is not
 -------------------------------------------------------------------}
procedure max_min(j :integer;var  x1,x2:real; x:real);
```

```
 begin
    If j = 1 then
       begin
          x2 :=  x;
          x1 :=  x;
        end;
    if x > x2 then
        x2 := x
    else
        if x < x1 then
             x1 := x;
 end;

{-------------------------------------------------------------------
                          Autoscaling Routine
  Given two different numbers, this sunroutine determines the maximum
  and minimum and the step between the two to give axis divisions of
                 1,2,3,4,5 and 10 during plotting }
 -------------------------------------------------------------------}
procedure autoscale(var min, max, step : real);
var
   multiplier    : real;
   flag          : boolean;
begin
   flag := false ;
   max := 1.05*max;
   repeat
     step := abs (( max - min ) / 5) ;   { Assume 5 divisions }
     if (step >= 1) or (step <= 0) then  { 10^(trunc(log(step)))}
          multiplier := exp(trunc(0.43443*ln(step))*ln(10))
     else
          multiplier := exp(trunc(0.43443*ln(step)-1)*ln(10));
                                     { 10^(trunc(log(step)-1))}
     step := trunc(step/multiplier) + 1;
     if step > 5 then
        step := 10;
     step := multiplier * step ;
     if min >= 0 then
        min := trunc(min/step) * step
     else                            { shift axis -ve }
        min := ( ( trunc ( min / step ) - 1 ) * step );
     max := min + ( 5 * step ) ;
     if  abs(200 * step /(max + min))  < 1  then
       begin          { checks for very small fluctuation in }
          min := min / 2 ;         {  data }
          max := max * 2
       end
     else
          flag := true ;
   until flag = true;
end;


{--------------------------------------- }
{ Log Autoscale the axis to give divisions }
{----------------------------------------}
procedure log_autoscale(x1,x2 : real; var min, max, step : real);
var
 x : real;

begin
    min := trunc(0.43443*ln(x1));
    min := min - 1;
    max := trunc(0.43443*ln(x2));
    if max > 0 then max := max + 1 ;
    step := 1
end;


{--------------------------------------- }
{        Y = mX + b linear fit           }
{----------------------------------------}

PROCEDURE fit(x,y: data_array; ndata: integer;
                              VAR a,b,siga,sigb,chi2: real);
VAR
   i: integer;
   wt,t,sy,sxoss,sx,st2,ss,sigdat: real;
BEGIN
```

```
   sx := 0.0;
   sy := 0.0;
   st2 := 0.0;
   b := 0.0;
   FOR i := 1 to ndata DO
     BEGIN
       sx := sx+x[i];
       sy := sy+y[i]
     END;
   ss := ndata;
   sxoss := sx/ss;
   FOR i := 1 to ndata DO
     BEGIN
        t := x[i]-sxoss;
        st2 := st2+t*t;
        b := b+t*y[i]
      END;
   b := b/st2;
   a := (sy-sx*b)/ss;
   siga := sqrt((1.0+sx*sx/(ss*st2))/ss);
   sigb := sqrt(1.0/st2);
   chi2 := 0.0;
   FOR i := 1 to ndata DO chi2 := chi2+sqr(y[i]-a-b*x[i]);
   sigdat := sqrt(chi2/(ndata-2));
   siga := siga*sigdat;
   sigb := sigb*sigdat
END;

begin
End.
```