NASA Contractor Report 195356

Rocketdyne Safety Algorithm

Space Shuttle Main Engine Fault Detection

Arnold M. Norman, Jr.
Rockwell International
Canoga Park, California

July 1994

Prepared for
Lewis Research Center
Under Contract NAS3–25884

National Aeronautics and
Space Administration

ABSTRACT

The Rocketdyne Safety Algorithm (RSA) has been developed to the point of use on the TTBE at MSFC on Task 4 of LeRC contract NAS3–25884. This document contains a description of the work performed, the results of the nominal test cases and a table of the resulting cutoff threshold levels, a plot of the RSA value vs. time for each nominal case, the results of the major anomaly test cases and a table of the resulting cutoff times, a plot of the RSA value vs. time for each anomaly case, a logic flow description of the algorithm, the algorithm code, and a development plan for the future efforts.

TABLE OF CONTENTS

1.0 SUMMARY AND CONCLUSION

The Rocketdyne Safety Algorithm (RSA) has been developed to the point of use in a test environment. The objective of this program is to develop the algorithm to a form which can be tested against previous engine test data sets for validation and coded for use with actual engine test data at MSFC. The real time influence coefficient engine model has been improved to the level that it works throughout the SSME sequence from start to cutoff. As a result, the RSA is functional throughout a complete test, during transients as well as steady state operating periods. The algorithm has been coded in Ada and verified to work on the SAFD hardware. However, due to timing constraints in the SAFD system, it will not be able to run concurrently with the SAFD algorithm. It has been demonstrated to work in real time on PC type processors and could be implemented on such a system at MSFC or other test facilities.

Validation tests using data from ninety two (92) actual engine tests have shown that it is not sensitive to sensor failures and will not cause inadvertent shutdowns during nominal engine runs. However, because it is a model driven algorithm, it is engine specific; and changes to the basic operating characteristics of the engine would require changes to the model. For instance, the SSME with the Pratt & Whitney ATD pump cannot be monitored with the existing algorithm without modifications. Due to this constraint, it is also difficult to utilize older engine test data for validation purposes due to changes in hardware configurations and sensor measurements.

Data from past actual SSME failure tests were used to validate the sensitivity of the algorithm to engine failures. Seventeen test cases were sufficiently similar in operation to the current configuration to provide valid comparisons. Twelve of the failures were detected significantly earlier than the existing redlines and sufficiently early to limit or prevent major damage. Of the five cases in which the algorithm detected the failure concurrently with the redlines, three were structural failures with no prior warning and the fourth was due to a fire with insufficient data available to determine if any delay occurred between the onset of the fire and the redline detection. The fifth case did exhibit a data spike about 100 seconds prior to shutdown which may be detectable by the algorithm if a different data filter is developed. It should be noted that these five cases did signal shutdown at the same time as the redlines, but are

listed as "none" in the table in Appendix A because they were not early detections.

The RSA has reached the maturity level to provide early failure detection during ground tests of the SSME. This method is readily adaptable to other engine configurations by use of engine specific models and could prevent major incidents and concomitant damage from test failures.

2.0 Introduction

The Rocketdyne Safety Algorithm was originally conceived on the LeRC contract Health Management for Rocket Engines, NAS3-25625, and is described in the Final Report, NASA CR 185223 (ref 1), and other publications concerning the RSA (references 2 through 4). In this study contract, the basic form of an algorithm was developed to detect failures in the Space Shuttle Main Engine (SSME) ground tests earlier than by other available means. The algorithm utilizes a real time model of the SSME to compare measured values of key parameters with predicted values from the model. The difference between the corresponding measured and predicted values for these key parameters are normalized, weighted, and summed. This weighted sum is then compared to an empirically determined threshold for detection of off nominal engine performance. After completion of the contract, a Rocketdyne IR&D program was conducted to develop a real time model which could be used in implementing the algorithm. An influence coefficient engine model was developed and a computer test environment was assembled for use in validating the algorithm operation. The current contract was begun after the IR&D program had successfully concluded. The objective of this program is to develop the algorithm to a form which can be tested against previous engine test data sets for validation and coded for use with actual engine test data at MSFC. The program used the model developed on IR&D and the algorithm concept previously developed on contract as starting points. Modification of the model for a robust system was the initial effort required.

3.0 Algorithm Development and Test

3.1 Algorithm Development

This Task was begun with an investigation into handling the engine startup transient, the development of sensor validation, and preliminary ideas for real time automated test case runs. The engine model available at the beginning of this Task was developed on IR&D and exhibited larger variations during the transient portions of the engine test run profile than during the steady state portions, particularly during the start transient. Although, the algorithm system was not required to operate during the transient portions for this Task, it was a desirable feature. In order to more closely predict the engine parameters during the startup transient, the Digital Transient Model was examined to see if it could lend any processing accuracy to the existing safety model algorithm. The digital transient model involves predicting engine parameters based upon valve position with no inlet conditions using an Adams-Bashforth numerical integration routine. For stability this routine requires a maximum cycle time of 5 milliseconds over which to iterate. In order to determine if this model would be feasible, the integration routine was run on eleven computer systems including SUN, PCs and Masscomp. The results showed that of the available computers, only the Masscomp 68020 -20MHz and an AT 386-33MHz (with math co-processor) could run this model if changes were made to reduce processing time. Because of this time constraint and the uncertainty that this could be implemented successfully in the available hardware within the funding constraints of this program, it was decided that the Real Time Safety Model (RTSM) developed under Rocketdyne IR&D would be used for this contract.

In an attempt to improve the RTSM, refined power level curve fit coefficients for use during start were generated to model engine performance during the first five seconds. The start transient produced spikes in the weighted sum until a settling of parameters occurred. The initial algorithm paused until 5 seconds had passed before starting to model the engine, and did not use actual observed sensor data until 17 seconds into the test. As a result, weighted sum indication of engine performance prior to 17 seconds exhibited greater variances than during the later steady state portions. To better monitor engine health during the startup transient, the algorithm was revised to utilize coefficients derived from previous engine tests. Start transient modeling now begins when the first

data point is received (0.0 to 0.2 seconds) and the H_2 inlet pressure is measured. Additionally, the first point of initialization for the model was moved from twenty seconds from engine start to five seconds from engine start in order to reduce the weighted sum and improve the performance during the first twenty seconds. This change effectively corrected the start -up transient spiking problem.

The dependent parameters monitored in the algorithm are listed below:

YNOM(i,01) = HPFT Discharge Temp A
YNOM(i,02) = HPFT Discharge Temp B
YNOM(i,03) = HPOT Discharge Temp A
YNOM(i,04) = HPOT Discharge Temp B
YNOM(i,05) = FPOV Position
YNOM(i,06) = OPOV Position
YNOM(i,07) = HPFT Speed
YNOM(i,08) = HPOT Speed
YNOM(i,09) = OPB Pc
YNOM(i,10) = FPB Pc
YNOM(i,11) = MCC Hot Gas INlet Pressure(HG IN)
YNOM(i,12) = MCC Pc

Several ideas for measurement validation were investigated for use in the algorithm. Reasonableness tests, sensor data weighting, fleetwide operating envelopes, and data trending are all possible methods for processing data.

A real time automated test system was configured using a 386 PC to take the place of an existing computer because of the speed and disk capacity to handle the one hundred nominal SSME tests to be input to the model. This PC took the validated, averaged and formatted SSME data and processed the model to generate the weighted sum as an indication of engine health.

Execution of the tests of the algorithm was automated to minimize time and labor associated with evaluation of the large number of hotfire data sets. The existing SSME hot-fire data sets were formatted to provide inputs compatible with the Real Time Safety Model software. In addition, an automated routine was prepared which accessed each hotfire data set in sequence, executed the safety algorithm in real time using that data set, and stored a time plot of

the algorithm's output (i.e. weighted sum) into an output datafile which could be reviewed off-line.

A 386 PC was integrated and tested in the real time model system to provide data storage for both the input SSME hot-fire data files and the model's weighted sum output data files. This replaced the slower 286 machine previously used in the development phase.

Other work was performed to improve the effectiveness of the safety algorithm in preparation for testing. The first initialization point was moved to 10 seconds to help insure a smooth transition from engine startup onwards. A routine for two point initialization to match model parameter curves with measured data at 65% and 100% power levels was coded and tested.

The standard deviations between measured and estimated data for the start transient were originally higher than those for mainstage data. In order to address this problem, two different arrays for start and mainstage calculation of the weighted sum were coded, which resulted in significantly better algorithm results when run on non-failure test data. This enabled the algorithm to be operational from engine start to shutdown.

Development of the necessary interface for operation on SAFD hardware was performed. Parameter calling, timing and receipt of parameters were established for the algorithm. Interface specifications for communicating with the SAFD system were received from Huntsville and reviewed. The specifications were reviewed to organize and identify what modifications must be made to the model to run on the SAFD platform such as how often the RSA algorithm is to be invoked, which CADS and which Facility data are needed for the RSA algorithm, etc. The SAFD system already has upper and lower limits established for the sensors. Additional limit checks may not be needed if they are in agreement with the established limits. An algorithm map was created which lists engine parameters and information and how often they are accessed.

A modification was made to the initialization procedure of the algorithm which increased its sensitivity. The algorithm now initializes each of the twenty two sensors used against the model rather than treating the redundant sensors as one and initializing only the resultant ten parameters. By this method, each PID is initialized to its behavior. This increased the sensitivity because the

redundant sensors normally have a small deviation from one another.

A problem caused by an artifact of the test stand calibration procedure was investigated and eliminated. After a pre-test sensor checkout, one of the channels of the HPFT speed sensor occasionally does not clear, leaving a false indication of speed. The code was changed so that, at initialization, it checks to see if the HPFT speed is less than 4000 RPM for the first 0.7 seconds. If it is, the algorithm proceeds normally; otherwise, the speed is not used until real data from the engine is received.

The primary method developed for "soft" sensor failure detection involved a "second deviation" of the sensor value. In this process, the deviation from the predicted model value of each parameter is compared with the corresponding average deviation of all other correlated engine parameters to determine if a parameter is showing a deviation much different from the others. This "second deviation" is used to weight the value of each parameter to the overall sum. The weighted sum calculation was reviewed, changed and coded for this "soft" sensor failure detection. For each parameter, the deviation from the predicted model value is subtracted from the average of all other parameter deviations; and the absolute value of the result becomes the argument to the hyperbolic secant function which returns a value between 0 and 1. This function result is used as a non-linear weight for the parameter deviation in the weighted sum calculation. The value of a specific parameter deviation will decrease in importance as it varies from the average of the other sensors. The code includes an adjustable slope for the hyperbolic secant function to change performance. In addition, strategies to place sensors into related subgroups in which sensors could be compared on a correlated basis were developed.

A Sensor Validation module was developed based on the concept that the hot gas flow path on both the fuel and oxidizer systems is considered to be a continuous series system beginning at the oxidizer preburner valve (OPOV/FPOV) and ending at the main combustion chamber pressure (MCC Pc). For example, if the OPOV were to begin to close, then the environments at all locations following in the flow path would drop. If the OPOV appears to be below the nominal position, and the OPB reads at, or above, nominal, then one of these two sensors is wrong. Similarly for the case of the OPB reading below nominal and the HPOP speed is at or above nominal.

The validation begins by first calculating the average of the vnd's (validated normalized deviations). This is done separately for the fuel and LOX sides. Each sensor's contribution is removed from the average, then its vnd is compared to the average of the remaining vnd's. The algorithm then computes vnd's for fuel, LOX, and for a total. As each sensor is considered for validation, its contribution to the vnd is subtracted from the average. Upstream sensor deviations that read lower than downstream averages are declared invalid. If MCC deviations are higher than the upstream average, then those sensors are declared invalid.

3.2 Validation Testing

Ninety of 100 nominal tests selected for this contract were run against the Rocketdyne Safety Model. Most of the ninety tests in the initial runs produced weighted sums below thirty throughout the engine test. A few had sums between thirty and fifty. Because this larger weighted sum occurs after power level changes, it was surmised that the algorithm did not wait long enough before returning to using the commanded MCC Pc rather than the measured MCC Pc [during power level changes the algorithm operates on the measured MCC Pc rather than the commanded MCC Pc]. This higher WSUM was not considered a problem since the failures previously observed have weighted sum approaching one hundred, but a more consistent value for non-failure cases is desirable in order to detect failures at the earliest possible time.

Problems were encountered in attempts to run the algorithm with the failure test cases utilized in the SAFD study. The first difficulty was due to a problem with the algorithm which was corrected by modifying the code. The weighted sum calculation was changed to limit the contribution to the weighted sum so that no single parameter could cause wsum to indicate failure. This was done by setting the maximum npd (npd is the normalized parameter deviation) = 4. The weighted sum equation was revised and is now equal to $1+npd^3$ for each parameter so that the maximum contribution by a single parameter is 65. If no PIDs are available for a particular parameter then its contribution to wsum is 1 (which is consistent with normalized parameter deviation = 0). Several test runs were performed with the modified algorithm to confirm that it operated as desired, but continued problems were encountered. The wsum general profile for the SAFD failure case data tends to run at a

higher level and with spikes and level changes not seen in nominal test cases for current tests. This was traced to the fact that the engine configuration has changed in significant ways since the SAFD failures occurred and the current engine model does not accurately predict the dependent parameters in all cases. A similar profile is encountered when the algorithm is run with Alternate Turbopump engine test data. Failure cases from the current engine configuration have the same general value and shape as nominal cases. Additionally, many parameters are missing from the old failure cases and substitute parameters must be identified and inserted. This has not always been successful. As a result, the old failure cases cannot be run and evaluated in a straight forward manner. Replacement PID's (parameter identification numbers) were selected for missing/bad PID's for use in the processing of the engine failure tests. The results of several of the runs showed successful early failure detections even with the cutoff value set at 60 which is required by the higher nominal level of the old cases.

Due to the algorithm modification, the failure cases and the nominal test cases had to be rerun. Several test failure runs were performed with the modified algorithm In spite of the difficulties encountered with the old test data, many of the failures are indicated by a rise in the average value of the algorithm sum. In order to rerun the nominal cases, the test data had to be transferred from data tapes to the test platform and verified to be in synchronization. Ninety-nine nominal tests were run through the Rocketdyne Safety Algorithm, but twelve of the nominal tests exhibited spiking in their weighted sum after the start and about six of these were of sufficient magnitude to cause a test termination. The cause of this problem was unknown, and a plan for isolating and correcting the anomaly was developed.

Upon investigation of the anomalous tests, one problem was discovered and corrected in the coding which caused one of the erroneous test cuts. Upon further investigation, it was determined that the remaining "anomalous cuts" were legitimate due to anomalous conditions during the tests. These tests were planned for "off nominal" conditions or utilized modified hardware and so were not terminated early or listed as abnormal. If a "normal" engine were to perform in the manner seen in any of these cases during ground testing, it would be imperative to shut it down to determine what was wrong.

After the final correction was made to the coding, the algorithm was once more applied to 17 failure cases and the remaining 92 nominal cases. Insufficient funding remained to obtain more nominal test data cases and prepare them for test in order to perform the originally desired 100 nominal test cases. The algorithm worked well on the nominal cases and the failure cases. (A summary of the nominal and failure cases and the plots of the algorithm value vs. time for each test case are presented in Appendix A.) Examination of the nominal cases showed that a two level threshold limit could effectively be applied. The originally planned limit of 60 worked satisfactorily for all cases, but it was apparent that this value could be lowered to 40 after the initial 20 seconds of testing. This is not unexpected since the model predicts actual performance better after the initial calibration checks. This lower limit provides the potential for earlier anomaly detection. When applied to the failure cases, the algorithm detected the anomalies significantly earlier than the failures in 12 of the 17 cases. Of the five tests failures which were not detected earlier than the existing redlines, three were structural failures with no forewarnings. One was due to a fire in the HPOTP, but data are not available on the speed of propagation or what triggered the controller to shutdown. The fifth is detectable in a spike over 100 seconds early if the current data filtering technique is not applied, which indicates that an improvement in the filtering method could lead to improved sensitivity. The results of the tests indicate that the algorithm is very successful in detecting failures significantly earlier than with traditional methods.

The "SAFD version" of this algorithm was successfully tested on a Sun sparcstation at Rocketdyne. A disk with this version and the PC version were provided to HSL to verify that the algorithm interfaces correctly and operates within expected execution speed for the SAFD platform. Tests of the algorithm on the SAFD hardware indicated that it works satisfactorily; but due to SAFD system operating time, it may not be able to work concurrently with the SAFD algorithm. This cannot be verified unless actual tests are performed. It will work alone on the SAFD hardware or on a dedicated PC at the facility. A future increase in the operating speed of the SAFD hardware would also enable it to run concurrently with the SAFD algorithm. The RSA code listing is presented in Appendix B.

4.0 ALGORITHM DESCRIPTION

The Rocketdyne Safety Algorithm is presented here in a logic flowpath. Section 4.1 will describe some of the features in the logic diagrams and the general flow of calculations, section 4.2 presents the logic diagrams, and section 4.3 contains the values of variables used in the algorithm including the influence coefficients.

4.1 General Flow of Calculations

- The modules outlined in this document and used in the logic flow diagrams only provide logical separation according to the algorithm specifications.
- Using a predetermined array of influence coefficients and initial nominal values, estimates of the dependent variables are derived from polynomial equations
- Consistency of redundant measurements is tested against maximum and minimum limits
- Anomalous measurements are deleted from final processing or limited in their contribution to the weighted sum calculations

Variables

ADT[ ]	Algorithm Data Table value (measured)
COEF[i][m][n]	Influence coefficients
D[m]	"Normal" parameter deviations
DELTAX[n]	% deltas between nominal & measured independent variable values
DELTAY[m]	The sum of INFLU(m,n) - DELTAX(n)
DEPNOM[m]	Nominal dependent values
DMAX	Maximum number of standard deviations allowed between "good" measured and estimated values
EST[m]	Estimated nominal engine values
INDNOM[n]	Nominal independent values
INDVAL[n]	Measured independent parameter values
INFLU[m][n]	Independent parameter influences on each dependent parameter
ND	Normalized measurement deviations
NPD[c]	Normalized parameter deviations
PCHK[j]	Maximum delta between the two closest values
P L	Normalized power level from MCC pressure reference
V[m]	Array of flags denoting validity of parameters

VAL[j] Average of two closest values from ADT for sets of 3 measurements

VMAX[b] Values for one standard deviation between deltas of normalized, redundant measurements(for consistency check)

VND[ ] Validated, normalized deviations of each measurement (O thru 21)

XNOM Model independent parameter initial conditions

YNOM Model dependent parameter initial conditions

Indices

i = array index (O - 3)
j = subscript for sets of three independent variable measurements
m = dependent variable subscript (O - 21)
n = independent variable subscript (O - 6)
a = subscript for redundant pairs of dependent parameters (0-10)
c = normalized parameter deviation subscript (O - 10)

Module C

This module validates the independent parameters:

LH2 Flow	LOX Flow
LH2 Inlet Pressure	LOX Inlet Pressure
LH2 Inlet Temperature	LOX Inlet Temperature
LH2 repressurization flow	LOX repressurization flow

For these parameters, this module will determine if the sensors for a particular measurement agree within defined limits. There are three values for each parameter. The two closest values are selected and averaged to obtain the validated model input for each parameter. If the two closest values differ by more than a predetermined amount, that variable is set to zero. This has the effect of disqualifying that parameter. The averaged, validated parameters are entered into a new array called INDVAL[n].

Module D

By using the independent parameter influences, the dependent variables can be estimated more accurately than from thrust alone With power level dependence taken into account, deviations of a particular measurement from expected levels will probably be detected earlier and with greater confidence. Influence

coefficients can be adjusted as more operational experience is accumulated.

A series of calculations in this module lead to estimated values for the 11 redundant pairs of dependent parameters:

HPFTP Turbine Discharge Temp
HPOTP Turbine Discharge Temp
FPOV Position
OPOV Position
HPFTP Shaft Speed
HPOTP Shaft Speed
OPB Chamber Pressure
FPB Chamber Pressure
MCC Hot Gas Injection Pressure
MCC Chamber Pressure, Channel A
MCC Chamber Pressure, Channel B

The DEPNOM[m] array and the INDNOM[n] array are calculated.

Next, the difference between the independent nominal values, INDNOM[n], and the independent measured values, INDVAL[n], is divided by INDNOM[n]. This results in the DELTAX[n] array which are the percent deltas between nominal and measured independent values.

A group of third order power level equations is then used to calculate the INFLU[m][n] matrix which are estimates of the percent change in each dependent parameter caused by a 1% perturbation in each parameter.

The total influence on each dependent parameter due to the actual perturbations in each independent parameter is calculated by summing the products of the influ[m][n] matrix and the deltax[n] array. These values are saved as the DELTAY[m] array - the percent changes in each dependent parameter.

The final estimates for dependent parameter values, EST[m], are determined by adjusting the nominal dependent values, DEPNOM[m], by the percent changes due to independent parameter perturbations, DELTAY[m]. The EST[m] array is calculated by multiplying DEPNOM[m] by one plus DELTAY[m].

Module F

This is the final module for eliminating erroneous dependent measurements that are out of their expected range when correlation with other measurements is not confirmed. Redundant measurements that do not agree within specific limits are also eliminated from further use or limited in their contribution to the weighted sum calculations.

The output for this module is the validated, normalized deviation of each measurement (all redundant measurements included). There are twenty-two (22) measurements.

An array of twenty-two flags is initialized to keep track of which measurements are valid. Each flag, V[m], is set to one or zero. If any measurement does not exist (ADT[m] = 0.0) or has been found unreasonable in previous modules then the corresponding flag, V[m], is set to zero for the remainder of the test.

Next, the normalized measurement deviations are calculated by taking the difference between the measurement (from the Algorithm Data Table) and the estimate, EST[m], derived in Module D, and dividing by the normal parameter deviation values, D[m].

The next step is to test the validity of the normalized deviations just calculated. First, an array of maximum differences allowed between redundant measurement deviations is initialized with predetermined values, VMAX[a]. The minimum average deviation of valid measurements that indicates an anomaly is set to the predetermined value.

The absolute value of the difference between each redundant pair of normalized parameter deviations, |[ND(a1) - ND(a2)]|, is calculated and compared to three times the standard deviation for this difference, DMAX * VMAX[a]. If this difference is less than three standard deviations, the redundant parameters are consistent.

Module G

In this module, the validated normalized deviations for each of the twenty-two (22) measurements are combined to yield the final, normalized parameter deviations, NPD[c]:

NPD[O]	HPFTP Discharge Temperature
NPD[1]	HPOTP Discharge Temperature
NPD[2]	FPOV Position
NPD[3]	OPOV Position
NPD[4]	HPFTP Shaft Speed
NP D[5]	HPOTP Shaft Speed
NPD[6]	OPB Chamber Pressure
NPD[7]	FPB Chamber Pressure
NPD[8]	MCC Hot Gas Injection Pressure
NPD[9]	MCC Chamber Pressure, Channel A
NPD[10]	MCC Chamber Pressure, Channel B

Influence Coefficients

Influence coefficients are an approximation of output from the Power Balance Model (the Power Balance Model is a steady state computer model of the SSME capable of modeling effects of variations in independent parameters [e.g. engine oxidizer inlet pressure] on dependent parameters [e.g. HPOTP turbine discharge temperature]). They are based upon the assumption that effects on dependent parameters of perturbations in independent parameters are linear over ranges of interest. Influence coefficients are used in the Rocketdyne Safety Algorithm to estimate values for the 10 dependent parameters listed above. An estimate for each adjusted, nominal dependent parameter value, EST[m], is used as a baseline value to compare against the measured value of the dependent parameter.

Influence coefficients consist of three components:

- third order power level curve fit to the nominal value of each dependent parameter:

$$DepNom = A_0 + A_1*PL + A_2*PL^2 + A_3*PL^3$$

- third order power level curve fit to the nominal value of each independent parameter:

$$IndNom = B_0 + B_1*PL + B_2*PL^2 + B_3*PL^3$$

- third order power level curve fits to the percent change in a dependent parameter due to a 1% perturbation in an independent parameter:

$$Influ = C_0 + C_1*PL + C_2*PL^2 + C_3*PL^3$$

Influence coefficients are used as follows:

- calculate the percent delta between measured and nominal independent parameters:

  DELTAX = (IndMeasured - IndNom)/IndNom

- calculate the total effect on the dependent parameter due to the variation of each independent parameter from its nominal value:

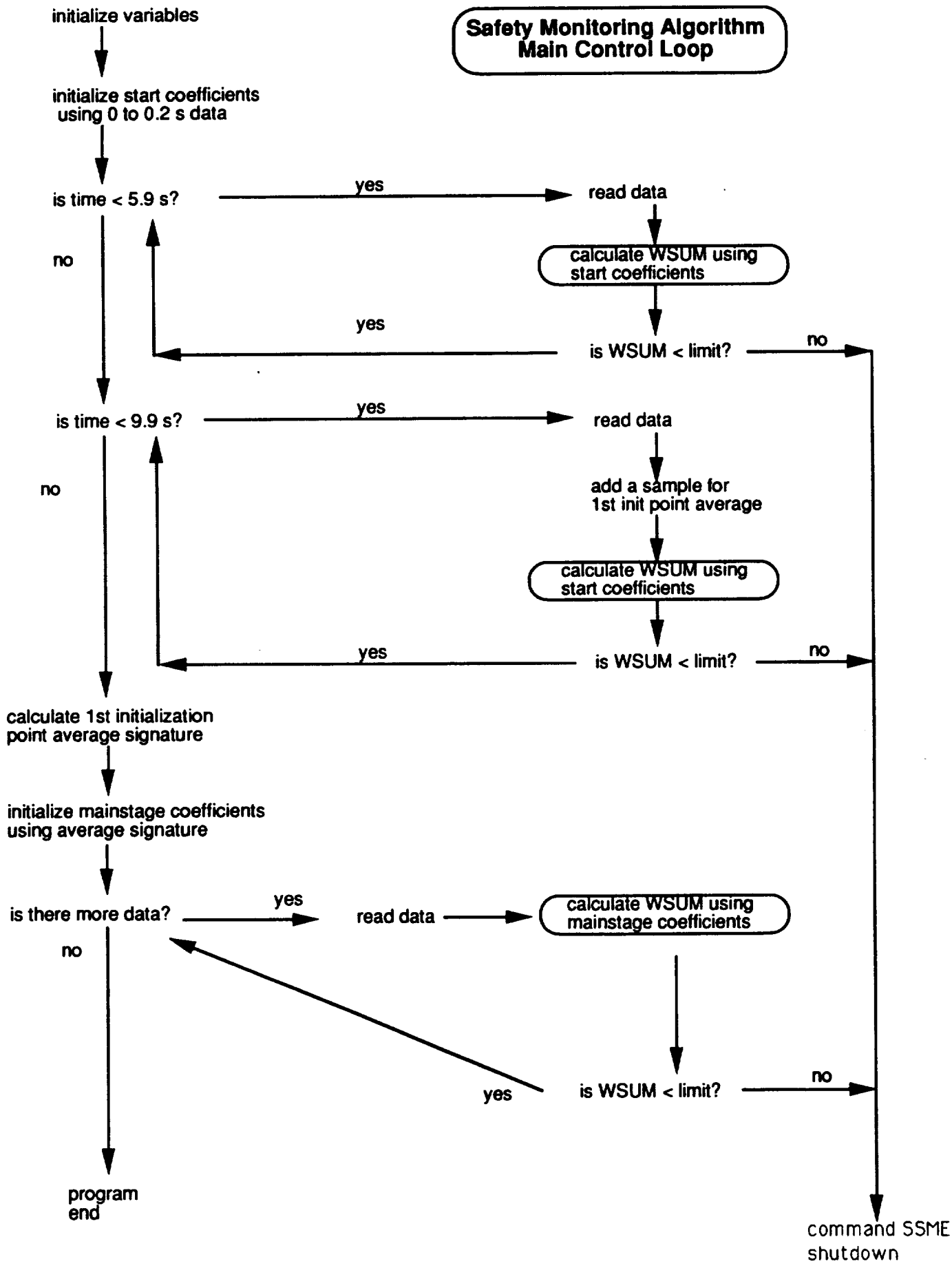
$$DELTAY = \sum_{n=0}^{6} influ * DELTAX[n]$$

- adjust the nominal dependent value for the variations in the independent parameters:

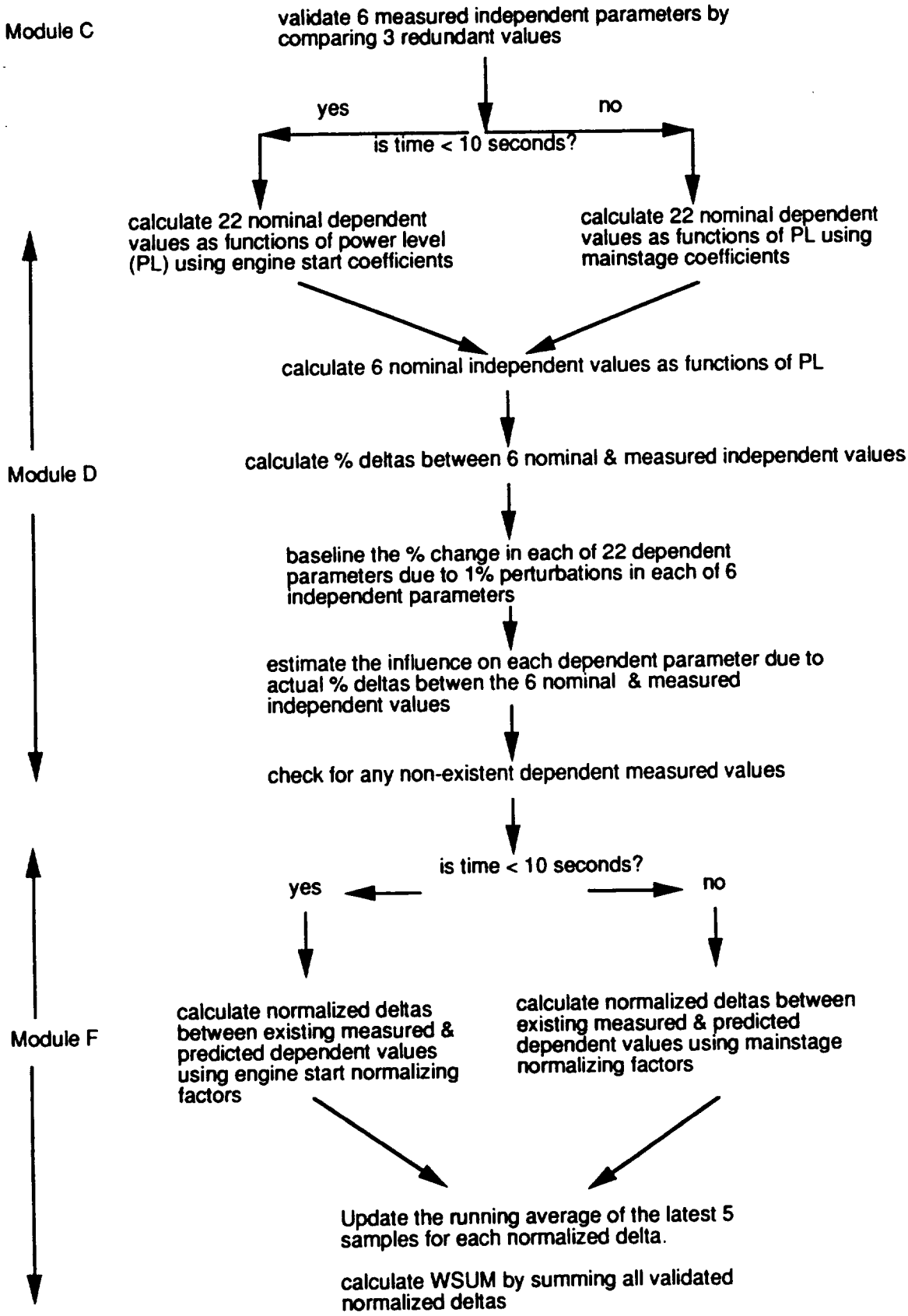
  EST = DepNom * (1+DELTAY)

4.2 Logic Flow Diagrams

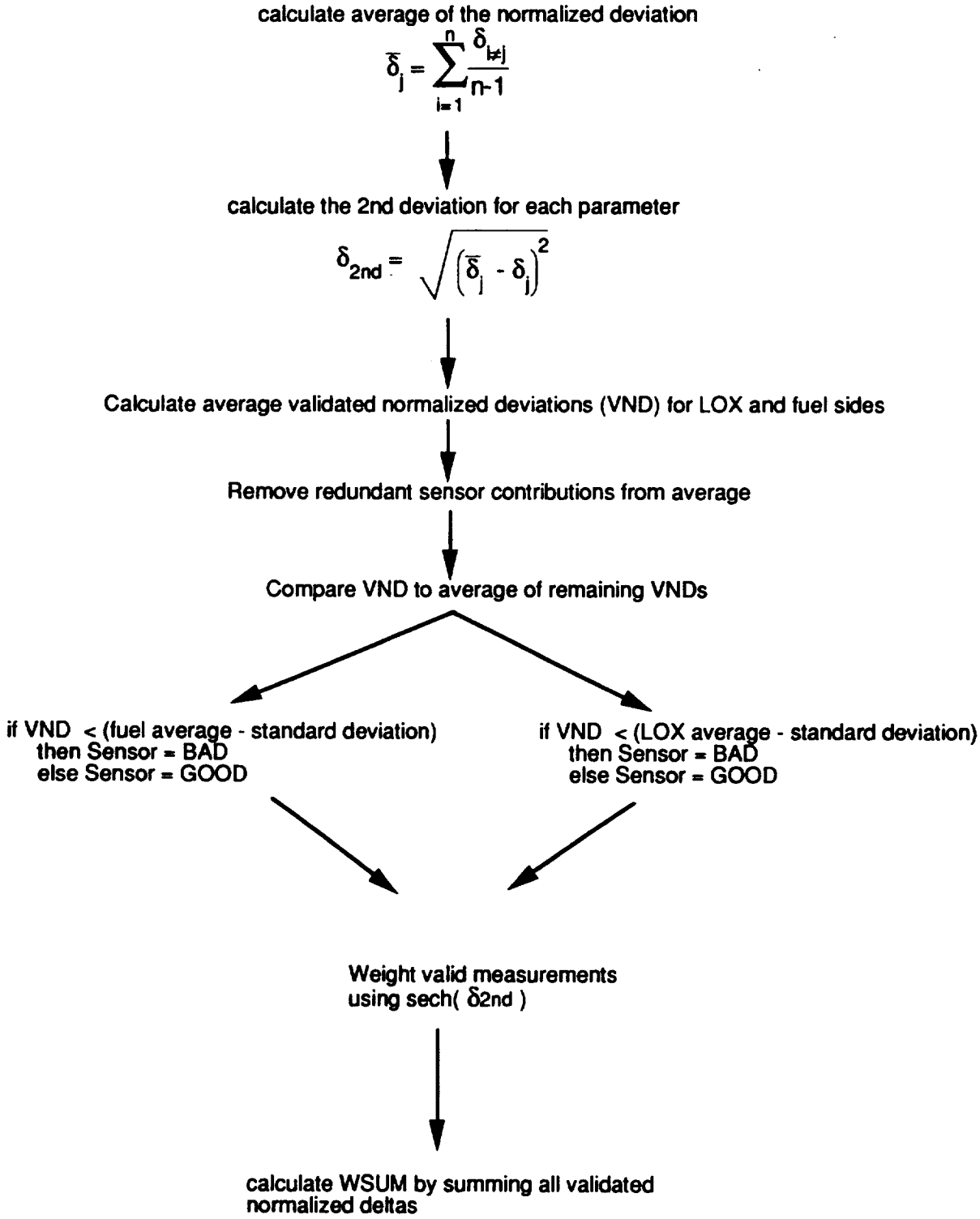
Attached below are the logic diagrams for the Rocketdyne Safety Algorithm described above.


Safety Monitoring Algorithm
Main Control Loop
initialize variables
initialize start coefficients
using 0 to 0.2 s data
is time < 5.9 s?
yes
read data
calculate WSUM using
start coefficients
is WSUM < limit?
yes
no
no
is time < 9.9 s?
yes
read data
add a sample for
1st init point average
calculate WSUM using
start coefficients
is WSUM < limit?
yes
no
no
calculate 1st initialization
point average signature
initialize mainstage coefficients
using average signature
is there more data?
yes
read data
calculate WSUM using
mainstage coefficients
no
is WSUM < limit?
yes
no
program
end
command SSME
shutdown

Safety Monitoring Algorithm
Weighted Sum Calculation



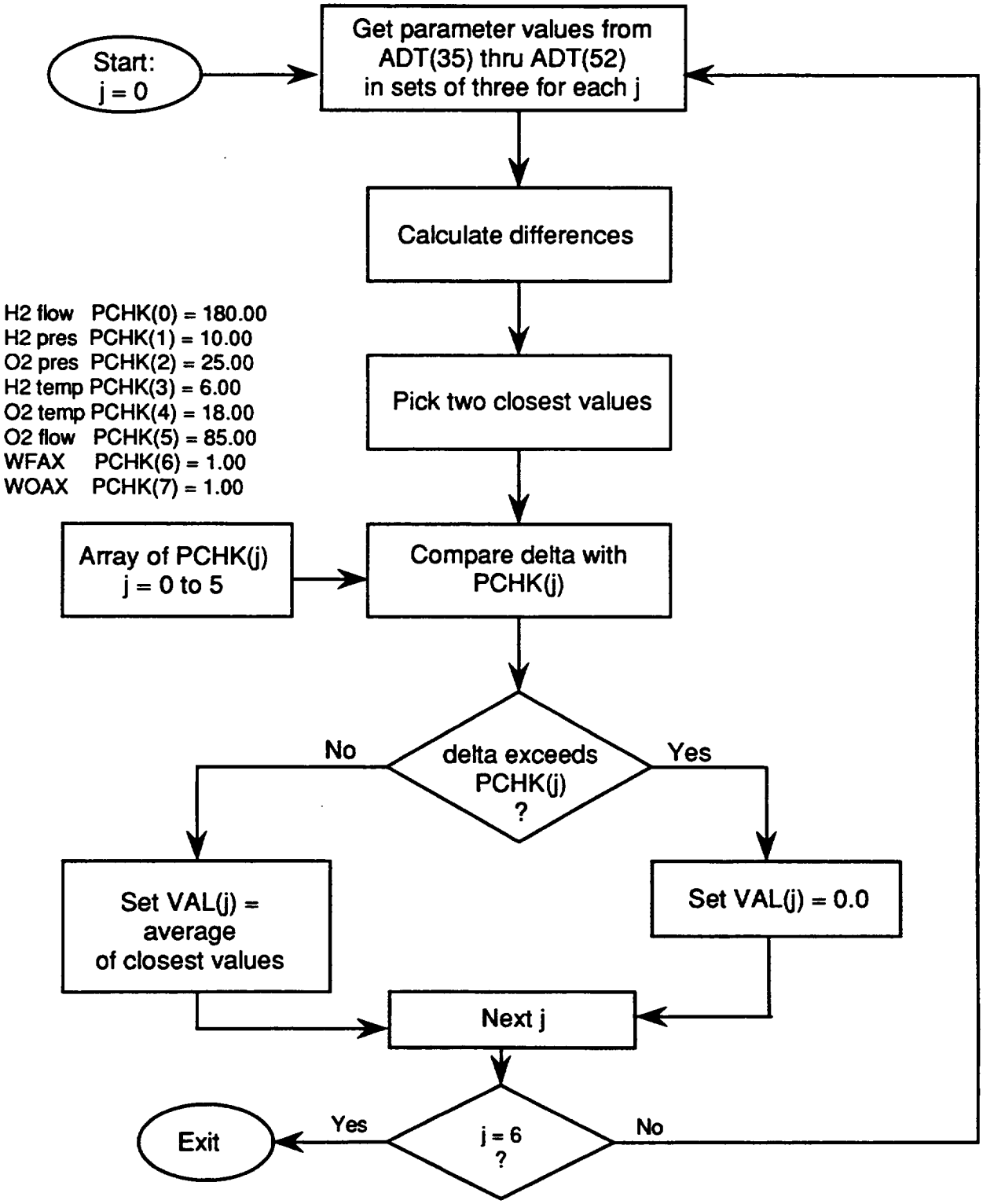
Safety Monitoring Algorithm
Sensor Validation



Module C #1
Validate Independent Parameters

Method: Average of two closest of three values

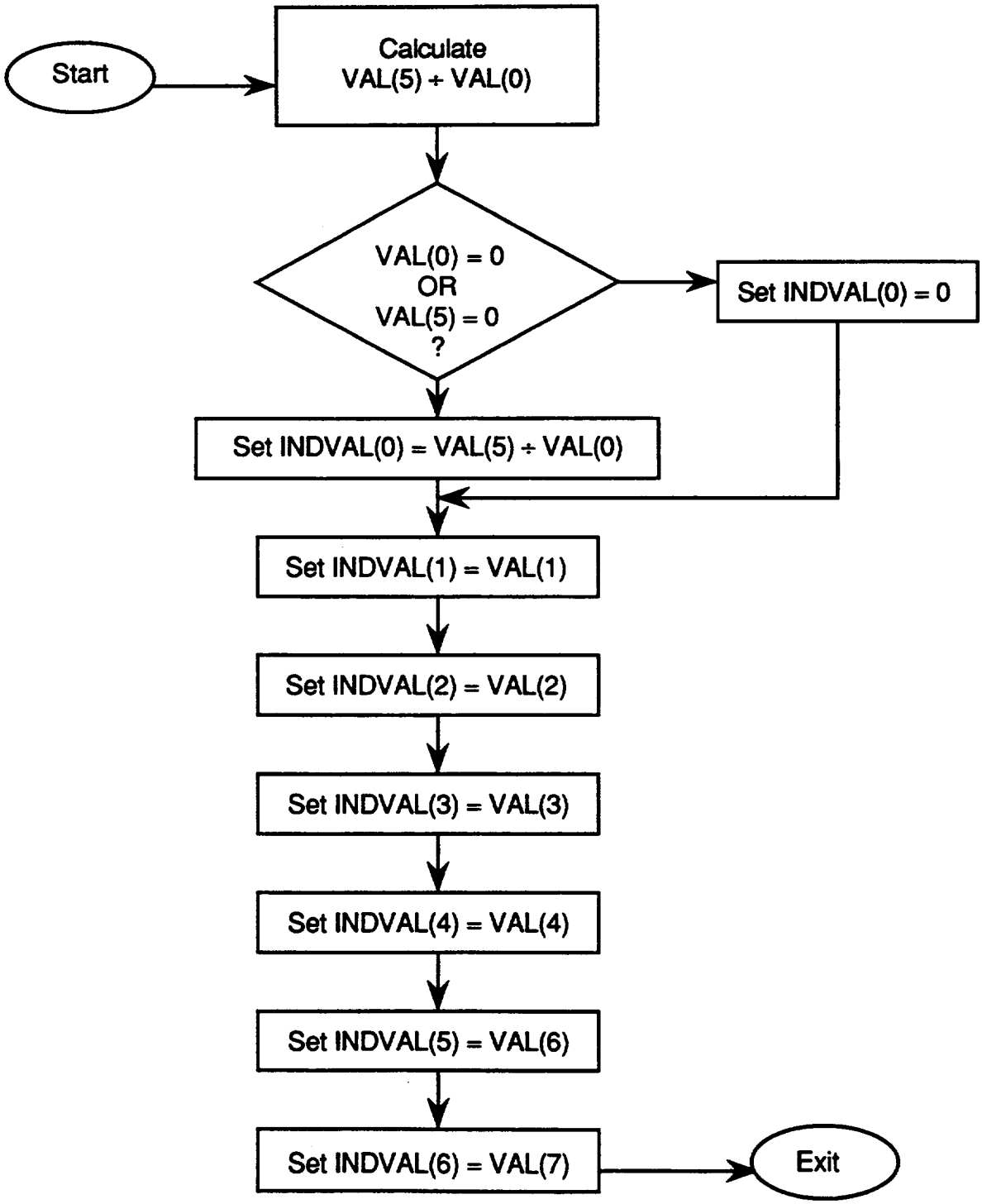
- Values set to zero if two closest differ by more than a preselected value
- Define an array of validated, Model Independent Variables: INDVAL(0) thru INDVAL(5)



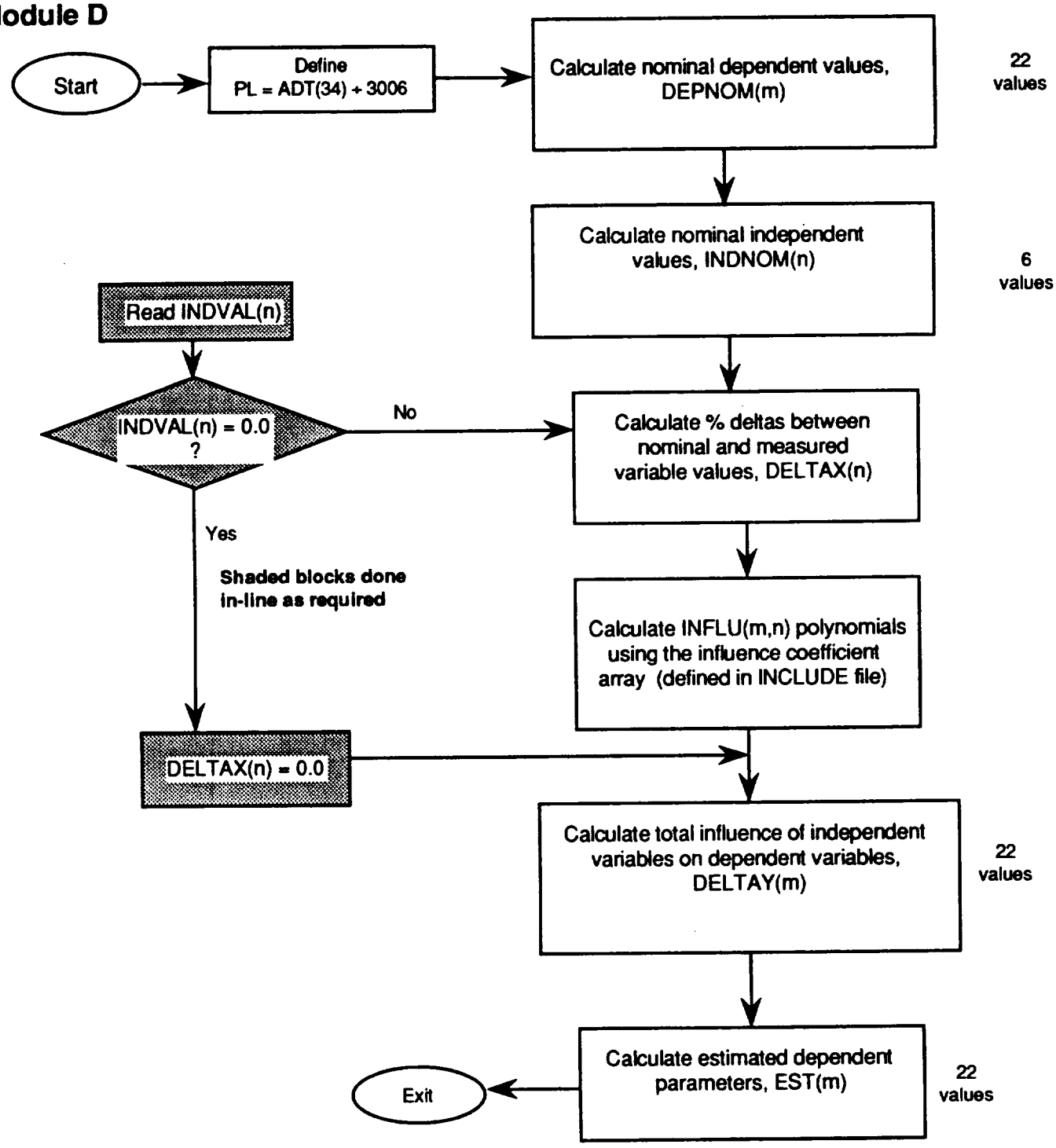
Module C #2
Validate Independent Parameters

Method: Average of two closest of three values

- Values set to zero if two closest differ by more than a preselected value
- Define an array of validated, Model Independent Variables: INDVAL(0) thru INDVAL(7)



Module D



$$DEPNOM(m) = YNOM(0,m) + YNOM(1,m) * PL + YNOM(2,m) * PL^2 + YNOM(3,m) * PL^3$$

$$INDNOM(n) = XNOM(0,n) + XNOM(1,n) * PL + XNOM(2,n) * PL^2 + XNOM(3,n) * PL^3$$

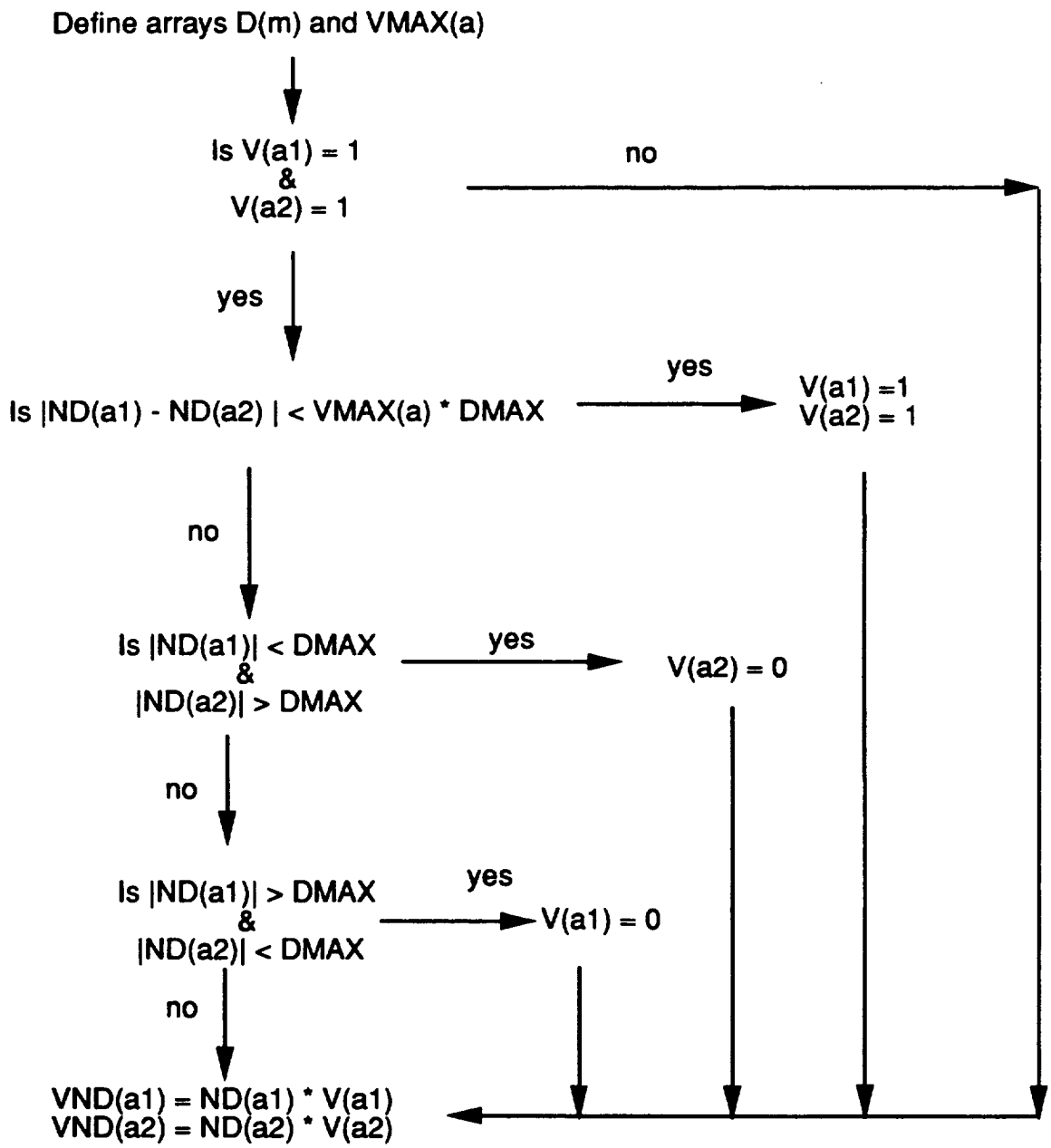
$$DELTAX(n) = [INDVAL(n) - INDNOM(n)] + INDNOM(n)$$

$$INFLU(m,n) = COEF(0,m,n) + COEF(1,m,n) * PL + COEF(2,m,n) * PL^2 + COEF(3,m,n) * PL^3$$

$$DELTAY(m) = \sum_{n=0}^{6} [INFLU(m,n) * DELTAX(n)]$$

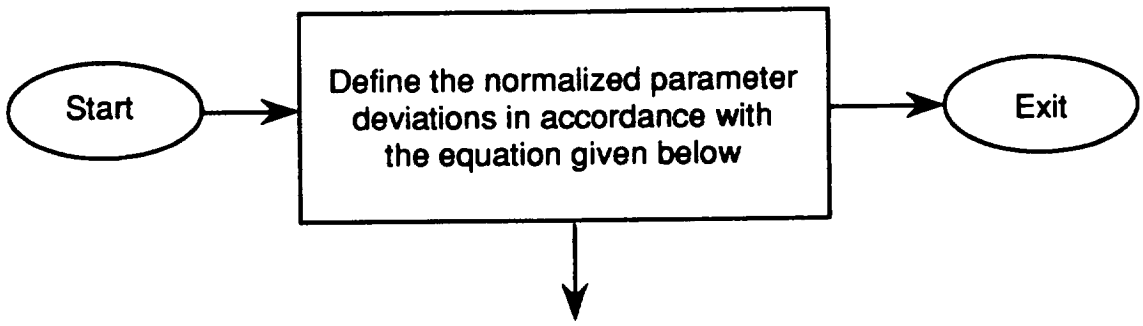
$$EST(m) = DEPNOM(m) * [1 + DELTAY(m)]$$

MODULE F
Calculate validated, normalized deviations



Module G

Define Normalized Parameter Deviations



$$NPD(a) = (|VND(a1)| + |VND(a2)|) / (V(a1) + V(a2))$$

4.3 Constants and Influence Coefficients

Below are the variable constants, data structures and influence coefficients used in the algorithm.

	i = 0	i = 1	i = 2	i = 3
XNOM(i,0)	6.0	0.0	0.0	0.0
XNOM(i,1)	30.0	0.0	0.0	0.0
XNOM(i,2)	100.0	0.0	0.0	0.0
XNOM(i,3)	37.0	0.0	0.0	0.0
XNOM(i,4)	164.0	0.0	0.0	0.0
XNOM(i,5)	5.97545E-02	0.5719	6.801E-02	0.0
XNOM(i,6)	0.202924	1.679	-2.849E-01	0.0
YNOM_START				
YNOM_START(i,0)		0.000	520.0420	1777.250
0.000				
YNOM_START(i,1)		0.000	520.0420	1777.250
0.000				
YNOM_START(i,2)		0.000	-537.0478	2449.364
0.000				
YNOM_START(i,3)		0.000	-537.0478	2449.364
0.000				
YNOM_START(i,4)		0.000	0.000	0.000
0.000				
YNOM_START(i,5)		0.000	0.000	0.000
0.000				
YNOM_START(i,6)		0.000	0.000	0.000
0.000				
YNOM_START(i,7)		0.000	0.000	0.000
0.000				
YNOM_START(i,8)		0.000	93179.35	-124648.3
63325.11				
YNOM_START(i,9)		0.000	93179.35	-124648.3
63325.11				
YNOM_START(i,10)		0.000	0.000	0.000
0.000				
YNOM_START(i,11)		0.000	0.000	0.000
0.000				
YNOM_START(i,12)		0.000	4604.870	187.0750
554.5640				
YNOM_START(i,13)		0.000	4604.870	187.0750
554.5640				
YNOM_START(i,14)		0.000	4952.407	-1274.236
1205.108				
YNOM_START(i,15)		0.000	4952.407	-1274.236
1205.108				
YNOM_START(i,16)		0.000	3003.940	196.7420
0.000				
YNOM_START(i,17)		0.000	3003.940	196.7420
0.000				

YNOM_START(i,18)	0.000	3006.00	0.0000	
0.000				
YNOM_START(i,19)	0.000	3006.00	0.000	
0.000				
YNOM_START(i,20)	0.000	3006.00	0.000	
0.000				
YNOM_START(i,21)	0.000	3006.00	0.000	
0.000				

YNOM(i,0)	1684.000	-746.336	721.239	0.000
YNOM(i,1)	1684.000	-746.336	721.239	0.000
YNOM(i,2)	478.000	714.638	110.014	0.000
YNOM(i,3)	478.000	714.638	110.014	0.000
YNOM(i,4)	76.830	-25.4172	29.1003	0.000
YNOM(i,5)	72.590	-63.8407	57.6847	0.000
YNOM(i,6)	17204.000	13012.3	4720.79	0.000
YNOM(i,7)	4884.000	23191.7	-836.435	0.000
YNOM(i,8)	-160.000	4604.87	187.075	554.564
YNOM(i,9)	-493.000	8396.85	-4697.71	2445.17
YNOM(i,10)	132.000	3003.94	196.742	0.000
YNOM(i,11)	0.000	3006	0.000	0.000

COEF(i,0,1)	6.755810E-02	3.13551E-01	-6.33715E-01	0.00000E+00
COEF(i,0,2)	-1.70377E-02	1.42833E-02	-4.50418E-03	0.00000E+00
COEF(i,0,3)	1.339000E-02	-3.66619E-03	-3.06278E-03	0.00000E+00
COEF(i,0,4)	4.452700E-01	-5.47878E-01	3.84827E-01	0.00000E+00
COEF(i,0,5)	7.812990E-02	-3.78262E-01	1.77979E-01	0.00000E+00
COEF(i,0,6)	2.482170E-03	-6.37037E-03	2.89615E-03	0.00000E+00
COEF(i,0,7)	-6.22494E-04	-3.36372E-04	3.61408E-04	0.00000E+00

COEF(i,1,1)	6.755810E-02	3.13551E-01	-6.33715E-01	0.00000E+00
COEF(i,1,2)	-1.70377E-02	1.42833E-02	-4.50418E-03	0.00000E+00
COEF(i,1,3)	1.339000E-02	-3.66619E-03	-3.06278E-03	0.00000E+00
COEF(i,1,4)	4.452700E-01	-5.47878E-01	3.84827E-01	0.00000E+00
COEF(i,1,5)	7.812990E-02	-3.78262E-01	1.77979E-01	0.00000E+00
COEF(i,1,6)	2.482170E-03	-6.37037E-03	2.89615E-03	0.00000E+00
COEF(i,1,7)	-6.22494E-04	-3.36372E-04	3.61408E-04	0.00000E+00

COEF(i,2,1)	-1.79474E-01	5.309600E+00	-1.78281E+00	0.00000E+00
COEF(i,2,2)	-9.45672E-04	1.754860E-02	-1.05341E-02	0.00000E+00
COEF(i,2,3)	-3.15037E-02	-4.35284E-02	4.25821E-02	0.00000E+00
COEF(i,2,4)	3.09397E-02	-2.52552E-01	0.00000E+00	0.00000E+00
COEF(i,2,5)	-4.63672E-01	1.84179E+00	-7.88768E-01	0.00000E+00
COEF(i,2,6)	-5.18391E-03	1.99238E-02	-8.76068E-03	0.00000E+00
COEF(i,2,7)	1.87290E-03	3.37052E-03	-2.36554E-03	0.00000E+00

COEF(i,3,1)	-1.79474E-01	5.309600E+00	-1.78281E+00	0.00000E+00
COEF(i,3,2)	-9.45672E-04	1.754860E-02	-1.05341E-02	0.00000E+00
COEF(i,3,3)	-3.15037E-02	-4.35284E-02	4.25821E-02	0.00000E+00
COEF(i,3,4)	3.09397E-02	-2.52552E-01	0.00000E+00	0.00000E+00
COEF(i,3,5)	-4.63672E-01	1.84179E+00	-7.88768E-01	0.00000E+00
COEF(i,3,6)	-5.18391E-03	1.99238E-02	-8.76068E-03	0.00000E+00
COEF(i,3,7)	1.87290E-03	3.37052E-03	-2.36554E-03	0.00000E+00

COEF(i,4,1)	-3.27388E+00	8.50257E+00	-6.35848E+00	0.00000E+00
COEF(i,4,2)	-2.72697E-02	6.61322E-02	-4.57526E-02	0.00000E+00
COEF(i,4,3)	4.14552E-02	-8.71833E-02	5.334435E-02	0.00000E+00
COEF(i,4,4)	1.65703E+00	-4.27165E+00	2.88507E+00	0.00000E+00
COEF(i,4,5)	-7.63844E-02	2.17812E-01	-1.79975E-01	0.00000E+00
COEF(i,4,6)	-1.04716E-02	2.78641E-02	-2.02631E-02	0.00000E+00
COEF(i,4,7)	-2.40354E-03	6.10207E-03	-4.52982E-03	0.00000E+00
COEF(i,5,1)	-3.27388E+00	8.50257E+00	-6.35848E+00	0.00000E+00
COEF(i,5,2)	-2.72697E-02	6.61322E-02	-4.57526E-02	0.00000E+00
COEF(i,5,3)	4.14552E-02	-8.71833E-02	5.334435E-02	0.00000E+00
COEF(i,5,4)	1.65703E+00	-4.27165E+00	2.88507E+00	0.00000E+00
COEF(i,5,5)	-7.63844E-02	2.17812E-01	-1.79975E-01	0.00000E+00
COEF(i,5,6)	-1.04716E-02	2.78641E-02	-2.02631E-02	0.00000E+00
COEF(i,5,7)	-2.40354E-03	6.10207E-03	-4.52982E-03	0.00000E+00
COEF(i,6,1)	1.33173E+00	-3.68307E+00	3.04542E+00	0.00000E+00
COEF(i,6,2)	6.04503E-04	0.00000E+00	0.00000E+00	0.00000E+00
COEF(i,6,3)	1.30019E-02	-2.74375E-02	0.00000E+00	0.00000E+00
COEF(i,6,4)	2.20889E-02	-6.10283E-02	0.00000E+00	0.00000E+00
COEF(i,6,5)	8.40340E-01	-2.36836E+00	1.88330E+00	0.00000E+00
COEF(i,6,6)	-1.19188E-03	2.29577E-03	0.00000E+00	0.00000E+00
COEF(i,6,7)	-9.63561E-04	2.13802E-03	0.00000E+00	0.00000E+00
COEF(i,7,1)	1.33173E+00	-3.68307E+00	3.04542E+00	0.00000E+00
COEF(i,7,2)	6.04503E-04	0.00000E+00	0.00000E+00	0.00000E+00
COEF(i,7,3)	1.30019E-02	-2.74375E-02	0.00000E+00	0.00000E+00
COEF(i,7,4)	2.20889E-02	-6.10283E-02	0.00000E+00	0.00000E+00
COEF(i,7,5)	8.40340E-01	-2.36836E+00	1.88330E+00	0.00000E+00
COEF(i,7,6)	-1.19188E-03	2.29577E-03	0.00000E+00	0.00000E+00
COEF(i,7,7)	-9.63561E-04	2.13802E-03	0.00000E+00	0.00000E+00
COEF(i,8,1)	-6.31739E-01	1.12647E+00	-8.22106E-01	0.00000E+00
COEF(i,8,2)	-1.17272E-02	1.26742E-02	-5.26788E-03	0.00000E+00
COEF(i,8,3)	1.19841E-03	-4.42708E-03	2.67544E-03	0.00000E+00
COEF(i,8,4)	3.06835E-01	-3.73406E-01	2.76127E-01	0.00000E+00
COEF(i,8,5)	-2.97835E-02	8.02084E-02	-4.00085E-02	0.00000E+00
COEF(i,8,6)	-1.32908E-04	0.00000E+00	0.00000E+00	0.00000E+00
COEF(i,8,7)	-1.11418E-04	3.62329E-04	-2.01927E-04	0.00000E+00
COEF(i,9,1)	-6.31739E-01	1.12647E+00	-8.22106E-01	0.00000E+00
COEF(i,9,2)	-1.17272E-02	1.26742E-02	-5.26788E-03	0.00000E+00
COEF(i,9,3)	1.19841E-03	-4.42708E-03	2.67544E-03	0.00000E+00
COEF(i,9,4)	3.06835E-01	-3.73406E-01	2.76127E-01	0.00000E+00
COEF(i,9,5)	-2.97835E-02	8.02084E-02	-4.00085E-02	0.00000E+00
COEF(i,9,6)	-1.32908E-04	0.00000E+00	0.00000E+00	0.00000E+00
COEF(i,9,7)	-1.11418E-04	3.62329E-04	-2.01927E-04	0.00000E+00
COEF(i,10,1)	-1.53861E-01	7.19883E-01	-3.08034E-01	0.00000E+00
COEF(i,10,2)	7.03175E-05	0.00000E+00	0.00000E+00	0.00000E+00
COEF(i,10,3)	-5.88139E-02	9.10559E-02	-4.02095E-02	0.00000E+00
COEF(i,10,4)	-4.07650E-03	7.66015E-03	-5.73094E-03	0.00000E+00
COEF(i,10,5)	1.13350E-01	3.59439E-01	-1.50146E-01	0.00000E+00

COEF(i,10,6)	-9.45481E-04	4.14947E+00	-1.81101E-03	0.00000E+00
COEF(i,10,7)	8.458920E-04	0.00000E+00	0.00000E+00	0.00000E+00
COEF(i,10,1)	-1.53861E-01	7.19883E-01	-3.08034E-01	0.00000E+00
COEF(i,10,2)	7.03175E-05	0.00000E+00	0.00000E+00	0.00000E+00
COEF(i,10,3)	-5.88139E-02	9.10559E-02	-4.02095E-02	0.00000E+00
COEF(i,10,4)	-4.07650E-03	7.66015E-03	-5.73094E-03	0.00000E+00
COEF(i,10,5)	1.13350E-01	3.59439E-01	-1.50146E-01	0.00000E+00
COEF(i,10,6)	-9.45481E-04	4.14947E+00	-1.81101E-03	0.00000E+00
COEF(i,10,7)	8.458920E-04	0.00000E+00	0.00000E+00	0.00000E+00
COEF(i,11,1)	-1.06276E-02	4.20930E-02	0.00000E+00	0.00000E+00
COEF(i,11,2)	-3.09662E-04	-3.51086E-04	2.46367E-04	0.00000E+00
COEF(i,11,3)	-1.11283-02	5.70060E-03	0.00000E+00	0.00000E+00
COEF(i,11,4)	7.83933E-03	-2.56483E-04	1.02111E-02	0.00000E+00
COEF(i,11,5)	-2.95686E-02	2.10816-E01	-7.71641E-02	0.00000E+00
COEF(i,11,6)	-4.82056E-04	1.94540E-03	-7.86290E-04	0.00000E+00
COEF(i,11,7)	6.05011E-04	-1.20095E-04	0.00000E+00	0.00000E+00
COEF(i,12,1)	-1.06276E-02	4.20930E-02	0.00000E+00	0.00000E+00
COEF(i,12,2)	-3.09662E-04	-3.51086E-04	2.46367E-04	0.00000E+00
COEF(i,12,3)	-1.11283-02	5.70060E-03	0.00000E+00	0.00000E+00
COEF(i,12,4)	7.83933E-03	-2.56483E-04	1.02111E-02	0.00000E+00
COEF(i,12,5)	-2.95686E-02	2.10816-E01	-7.71641E-02	0.00000E+00
COEF(i,12,6)	-4.82056E-04	1.94540E-03	-7.86290E-04	0.00000E+00
COEF(i,12,7)	6.05011E-04	-1.20095E-04	0.00000E+00	0.00000E+00
COEF(i,13,1)	-4.17196E-01	3.95345E-01	-3.26333E-01	0.00000E+00
COEF(i,13,2)	-3.58206E-03	2.19911E-03	-6.03413E-04	0.00000E+00
COEF(i,13,3)	1.29300E-03	-4.85527E-03	2.59230E-03	0.00000E+00
COEF(i,13,4)	1.15425E-01	-1.52808E-01	1.16439E-01	0.00000E+00
COEF(i,13,5)	-2.13535E-02	5.46972E-02	-1.49099E-02	0.00000E+00
COEF(i,13,6)	-1.19488E-04	0.00000E+00	0.00000E+00	0.00000E+00
COEF(i,13,7)	-8.58883E-05	3.00620E-04	-1.28705E-04	0.00000E+00
COEF(i,14,1)	-4.17196E-01	3.95345E-01	-3.26333E-01	0.00000E+00
COEF(i,14,2)	-3.58206E-03	2.19911E-03	-6.03413E-04	0.00000E+00
COEF(i,14,3)	1.29300E-03	-4.85527E-03	2.59230E-03	0.00000E+00
COEF(i,14,4)	1.15425E-01	-1.52808E-01	1.16439E-01	0.00000E+00
COEF(i,14,5)	-2.13535E-02	5.46972E-02	-1.49099E-02	0.00000E+00
COEF(i,14,6)	-1.19488E-04	0.00000E+00	0.00000E+00	0.00000E+00
COEF(i,14,7)	-8.58883E-05	3.00620E-04	-1.28705E-04	0.00000E+00
COEF(i,15,1)	-2.66681E-02	9.92401E-03	-2.27310E-02	0.00000E+00
COEF(i,15,2)	-2.37199E-04	0.00000E+00	0.00000E+00	0.00000E+00
COEF(i,15,3)	-3.22731E-04	0.00000E+00	0.00000E+00	0.00000E+00
COEF(i,15,4)	1.00202E-02	-1.39743E-02	1.31286E-02	0.00000E+00
COEF(i,15,5)	1.17278E-03	-3.27962E-05	4.38514E-03	0.00000E+00
COEF(i,15,6)	2.65487E-06	-1.72772E-04	3.49690E-05	0.00000E+00
COEF(i,15,7)	2.85558E-05	-1.72770E-05	1.68002E-05	0.00000E+00
COEF(i,16,1)	-2.66681E-02	9.92401E-03	-2.27310E-02	0.00000E+00
COEF(i,16,2)	-2.37199E-04	0.00000E+00	0.00000E+00	0.00000E+00
COEF(i,16,3)	-3.22731E-04	0.00000E+00	0.00000E+00	0.00000E+00

COEF(i,16,4)	1.00202E-02	-1.39743E-02	1.31286E-02	0.00000E+00
COEF(i,16,5)	1.17278E-03	-3.27962E-05	4.38514E-03	0.00000E+00
COEF(i,16,6)	2.65487E-06	-1.72772E-04	3.49690E-05	0.00000E+00
COEF(i,16,7)	2.85558E-05	-1.72770E-05	1.68002E-05	0.00000E+00
COEF(i,17,1)	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00
COEF(i,17,2)	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00
COEF(i,17,3)	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00
COEF(i,17,4)	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00
COEF(i,17,5)	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00
COEF(i,17,6)	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00
COEF(i,17,7)	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00

5.0 Development Plan

5.1 Study Plan

There are several follow on development tasks which should be undertaken. These tasks range from further development of the algorithm itself to a more complementary integration with the SAFD platform. These tasks include, but are not limited to, failure detection techniques, improve startup modeling, investigation for new dependent parameters, and sensor validation.

Recommendations

Despite the success of the Rocketdyne Safety Algorithm, several areas of the algorithm need to be addressed and refined.

- sensor validation
- accounting for failure of PID 129 or 130 (MCC PC A1 & A2) and calculating power level
- making wsum a function
- dealing with measured pc during transients

There are indications that the accuracy of predictions declines as engine power level varies from the initialization point. A potential solution is to make wsum a function of how far away the current power level is from what the power level was during the initialization point. The wsum would be scaled down in cases of extreme power level variation to account for less accurate predictions.

PIDs 129 and 130 (MCC PC A1 & A2) are used to calculate power level during power level transients. There is currently no way to replace one or both of these PIDs in the power level calculation if

these PIDs were to fail. A methodology to do this should be incorporated into the algorithm.

New Dependent Parameters

Currently 11 pairs of redundant parameter measurements are used by the algorithm to determine dependent parameter values. It will be a worthwhile task to investigate additional parameters which may be useful as dependent parameters. Prime candidates would be other PIDs which are redundant measurements since they can be used to help validate each other. PIDs which fall into this category include:

LPFTP and LPOTP speeds
LPFP, HPFP, LPOP, HPOP and PBP discharge pressures
HPFP and HPOP inlet pressures
HPFP inlet temperature
PBP discharge temperature
MCC oxidizer injection temperature

New dependent parameters should not be limited to redundant PIDs, however. Several single measurement PIDs are very indicative of engine condition. The addition of these PIDs to the set of dependent parameters should be investigated as well.

Sensor Validation

For a full treatment of sensor validation, the algorithm should not only look at multiple sensors and the comparison of their individual values, but also the interaction of different sensors. This sensor validation would involve looking at flow paths through the engine and determining the series of sensors which are linked in behavior so that it can be determined if an individual sensor has gone awry. For example, the hot gas flow path on both the fuel and oxidizer systems is considered to be a continuous series system beginning at the oxidizer preburner valve (OPOV/FPOV) and ending at the main combustion chamber pressure (MCC Pc). If the OPOV were to be closed off, then the environments at all locations following in the flow path would drop to zero. It follows that if, instead of closing it completely, the OPOV position were but reduced, then the following environments would be correspondingly reduced. If the OPOV appears to be below the nominal position, and the OPB reads at or above nominal, then one of these two sensors is wrong. Similarly for the case of the OPB reading below nominal and the HPOP speed at or

above nominal. This is an example of a rule that would be developed and coded into the algorithm.

Startup modeling

The current power level based curves used to predict parameter values during engine startup are fits to data from a very limited number of tests. Three areas have been identified as having potential for improvement in startup modeling.

1)Use the DTM to generate nominal SSME startup data, and generate new curve fits from these data. The resulting curves should be more representative of a nominal engine.

2)Base at least part of the startup modeling on commanded positions of propellant valves. This should improve the prediction of MCC and preburner primes and other start fluctuations.

3)Use start envelopes to define upper and lower bounds of parameter values. The envelopes will be wider at certain operating times when confidence in predictions declines.

Engine Diagnostics

Three areas where the diagnostic capabilities of the algorithm could be further developed are the baseline WSUM, data trending and failure pattern recognition.

In the current code, an engine shutdown command is issued when the wsum value exceeds a threshold value. This value could be the result of numerous parameters deviating from nominal operation or just a few parameters. It would be beneficial if there were a "yellow line" or region wherein the weighted sum value was above that of nominal engine operation but below the shutdown limit. In this manner, the engineers would know that the engine is performing out of norm but not in a manner representing critical failure.

Failure detection exists in the Rocketdyne Safety Algorithm as a flag indicating failure when the weighted sum value exceeds the threshold value. This is the simplest way to detect failure and in the case of catastrophic failure should command engine shutdown before engine damage. However, a more sophisticated approach would involve looking at the data for trends. For example, a slope-average algorithm might be employed to detect slow failures. In this case, the slope of the averaged data would be examined rather than the averaged data. In this manner, the RSA would be more sensitive to subtle changes in slope and could track increasing slope-averages for trends which could indicate engine failure.

The failure pattern recognition task would involve studying the details of all the dependent parameters and their variations for various failures.

Algorithm Validation

Once new dependent parameters, sensor validation, improved start-up modeling and engine diagnostics have been incorporated into the this algorithm, it must be validated by monitoring nominal tests and a variety of types of failure tests. This task would involve running the algorithm with previously transferred tests and transferring the more recent SSME tests available to Rocketdyne.

Feasibility Study of Pratt & Whitney Pump

This sub task would involve studying and evaluating the P&W pump performance data to determine a methodology for incorporating the P&W pump into the model.

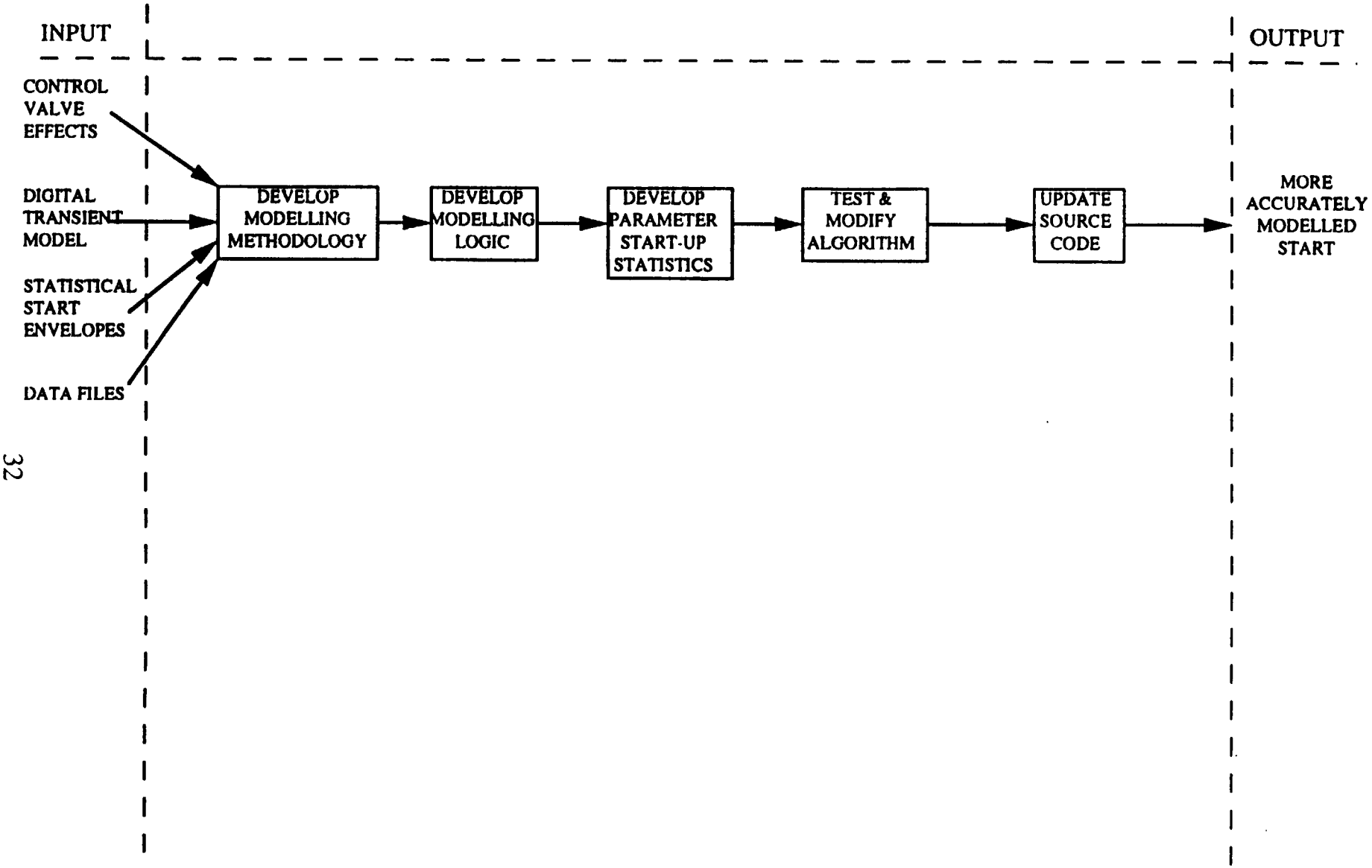
Work Breakdown Structure

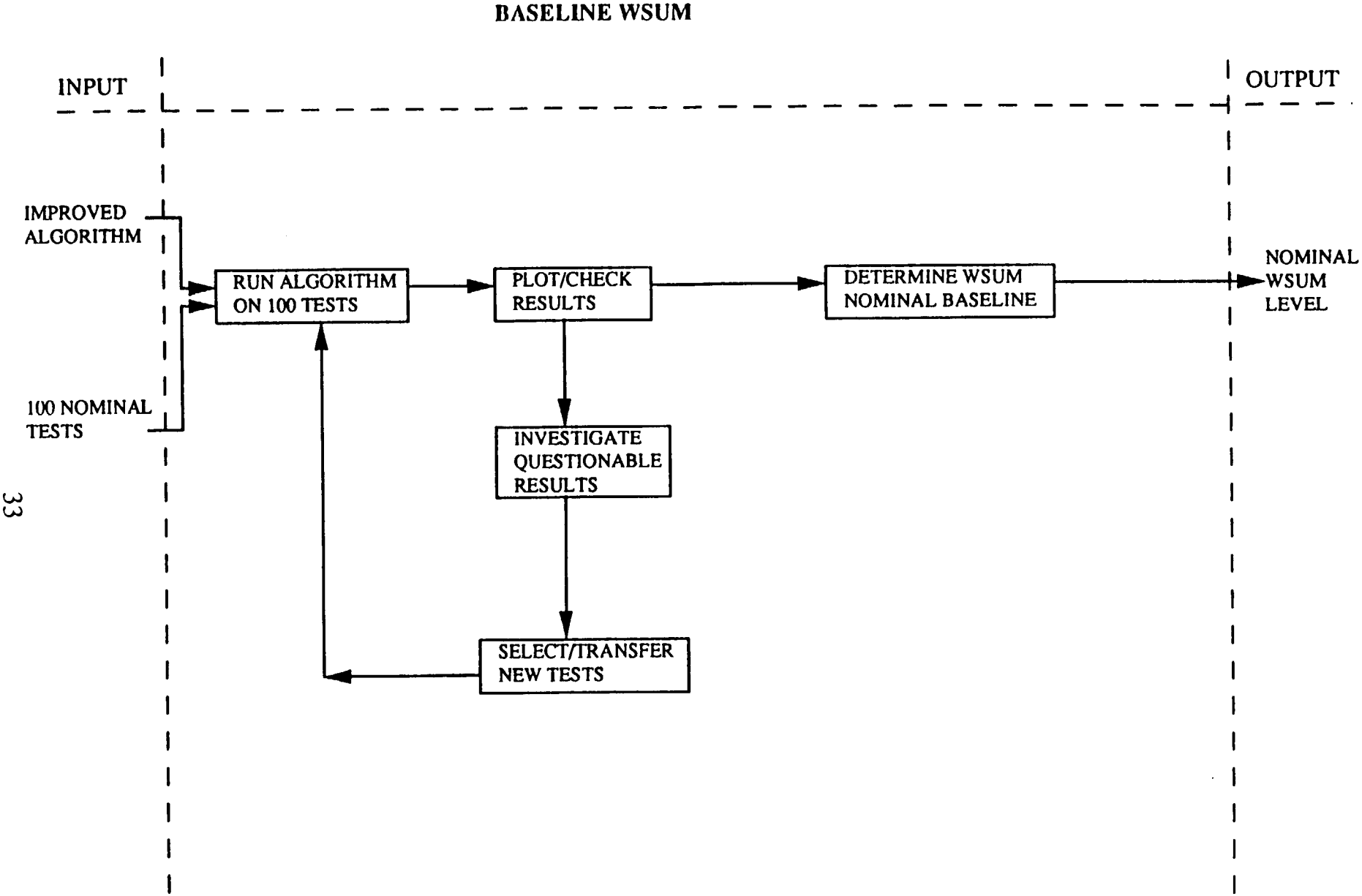
The proposed effort is broken down into the following WBS elements:

1.0 Improve Algorithm Robustness
1.1 Develop new dependent parameters
1.2 Develop more robust sensor validation methodology
1.3 Improve engine startup model
2.0 Develop engine diagnostics module
2.1 Determine baseline WSUM
2.2 Develop data trending methodology
2.3 Develop failure pattern recognition module
3.0 Perform P&W pump feasibility study
4.0 Incorporate changes into RSA
4.1 Modify algorithm interface & code
4.2 Perform algorithm validation testing
5.0 Task Management
5.1 Program management
5.2 Travel
5.3 Reporting

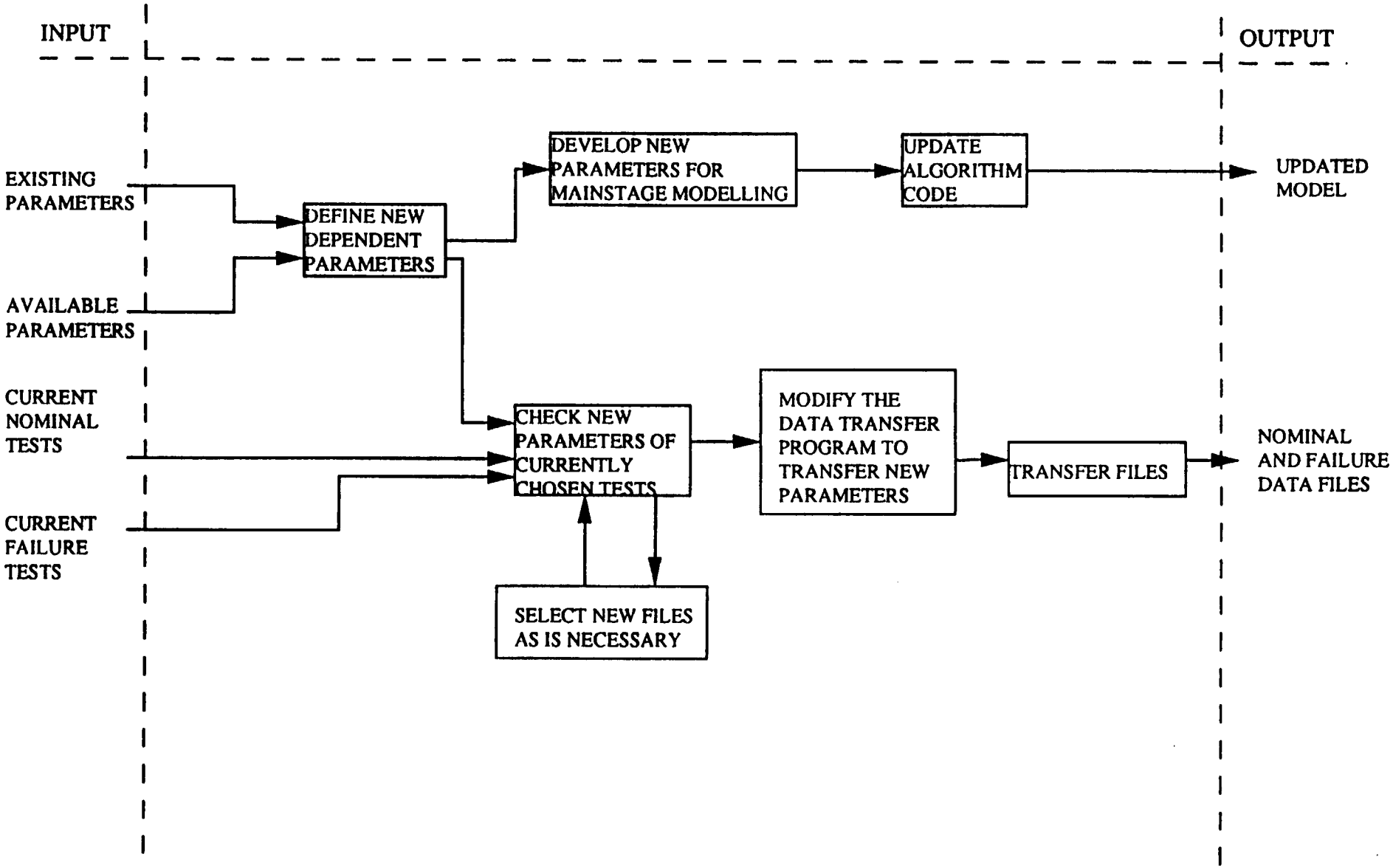
The cost of the complete program described above including travel, manpower, and material is estimated at $250,000. The logic flow diagrams for the specific tasks and a schedule follow.

IMPROVE STARTUP MODELLING

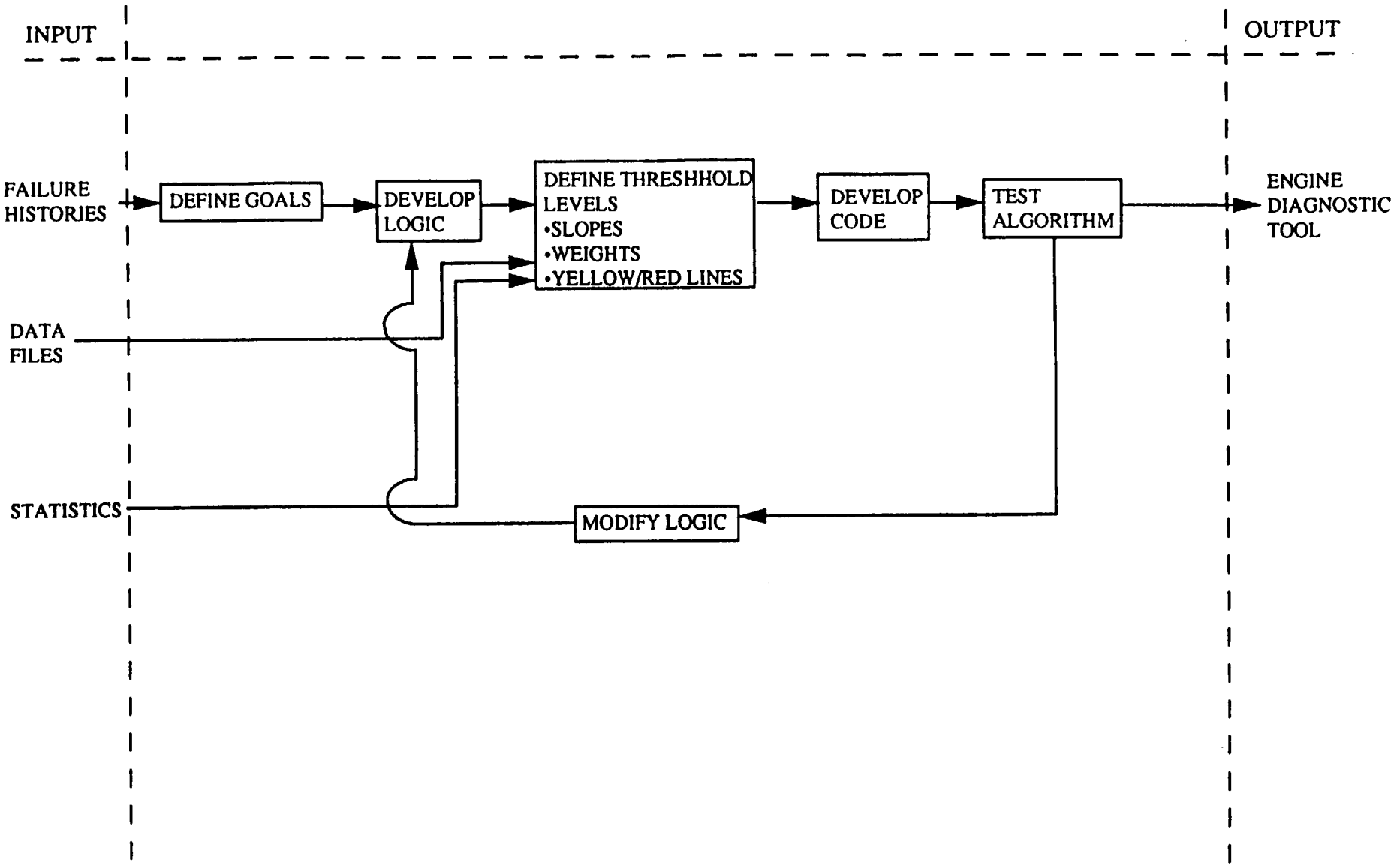



BASELINE WSUM
INPUT
OUTPUT
IMPROVED ALGORITHM
100 NOMINAL TESTS
RUN ALGORITHM ON 100 TESTS
PLOT/CHECK RESULTS
DETERMINE WSUM NOMINAL BASELINE
NOMINAL WSUM LEVEL
INVESTIGATE QUESTIONABLE RESULTS
SELECT/TRANSFER NEW TESTS

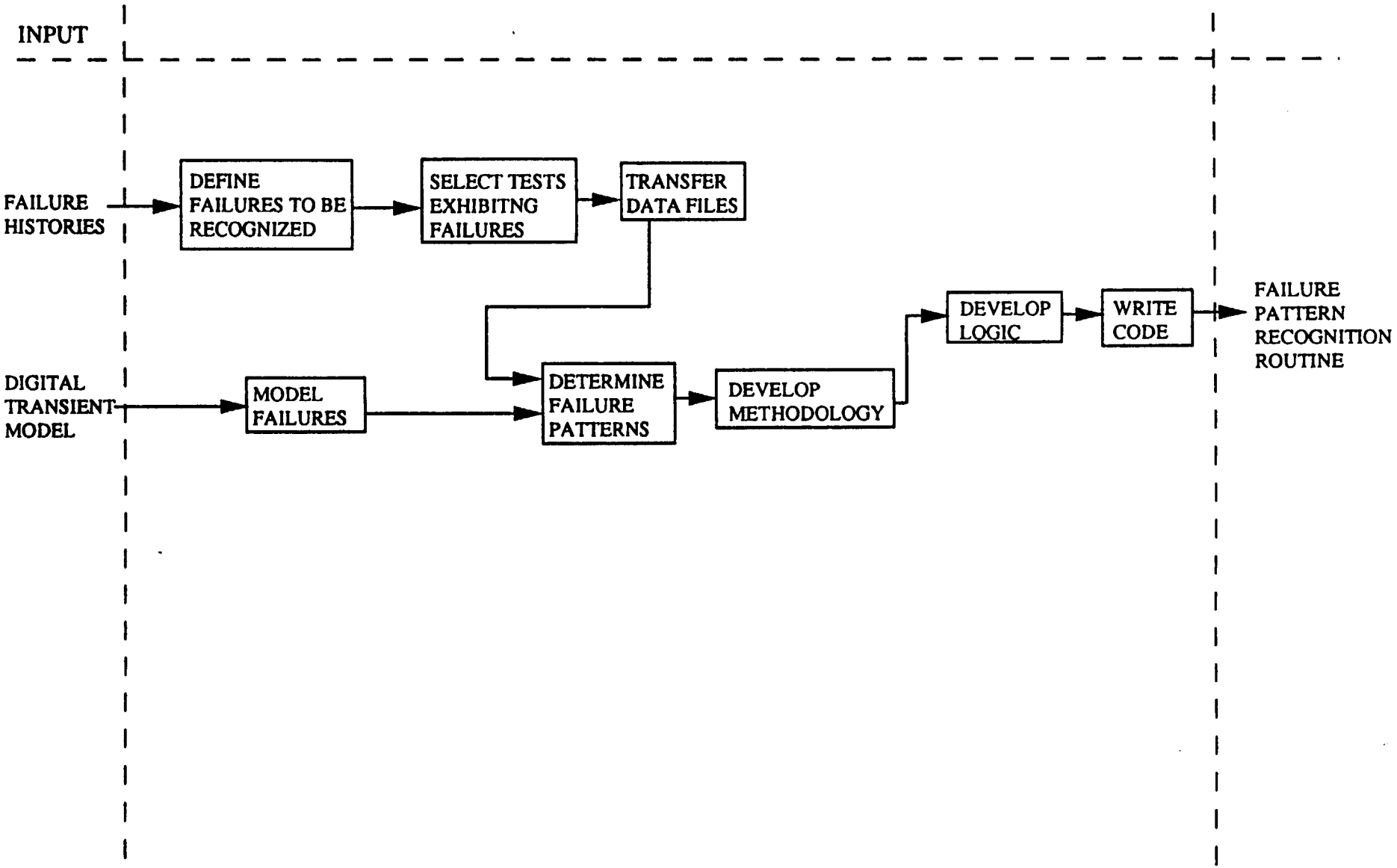
IDENTIFICATION OF NEW DEPENDENT PARAMETERS



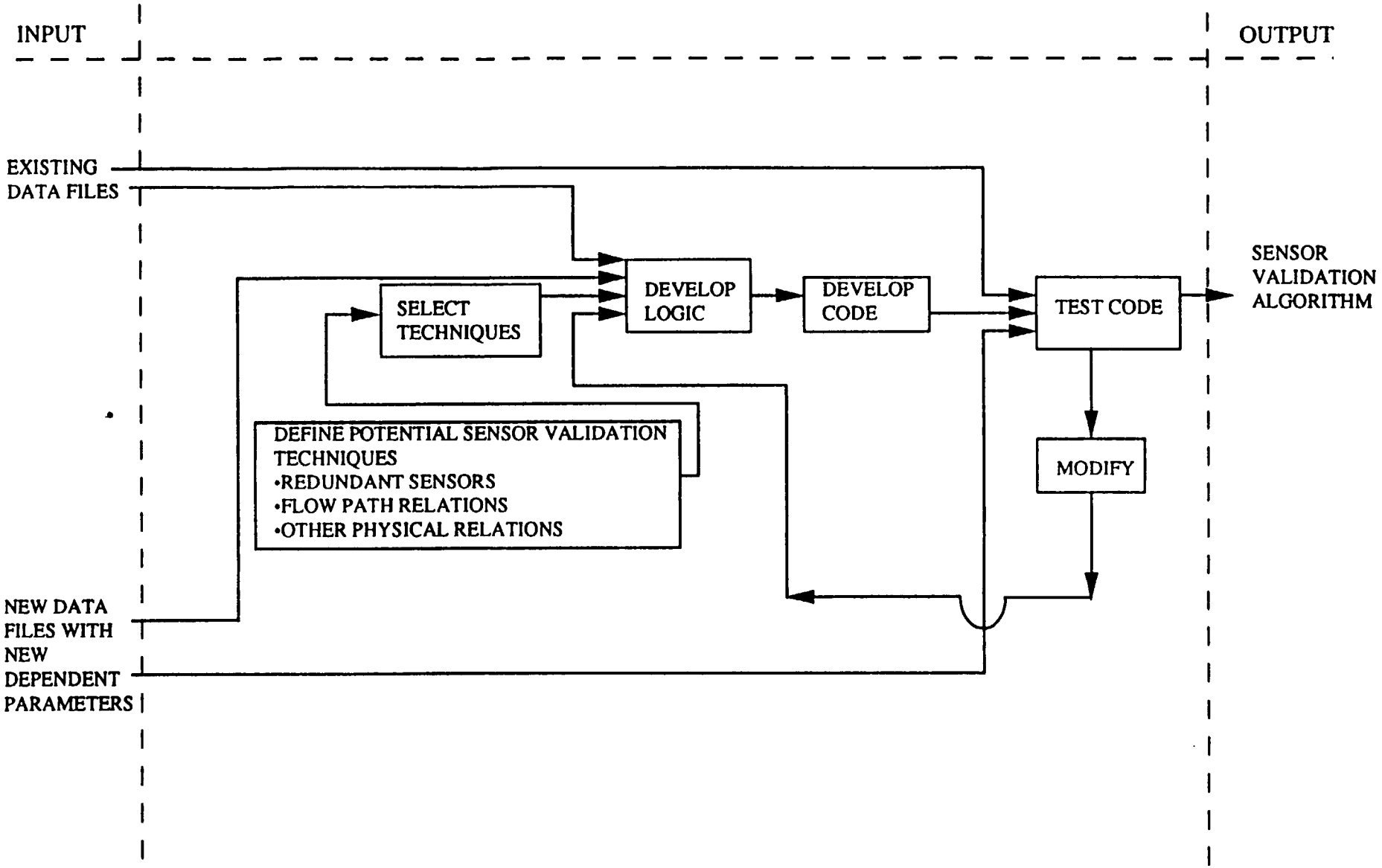
DATA TRENDING



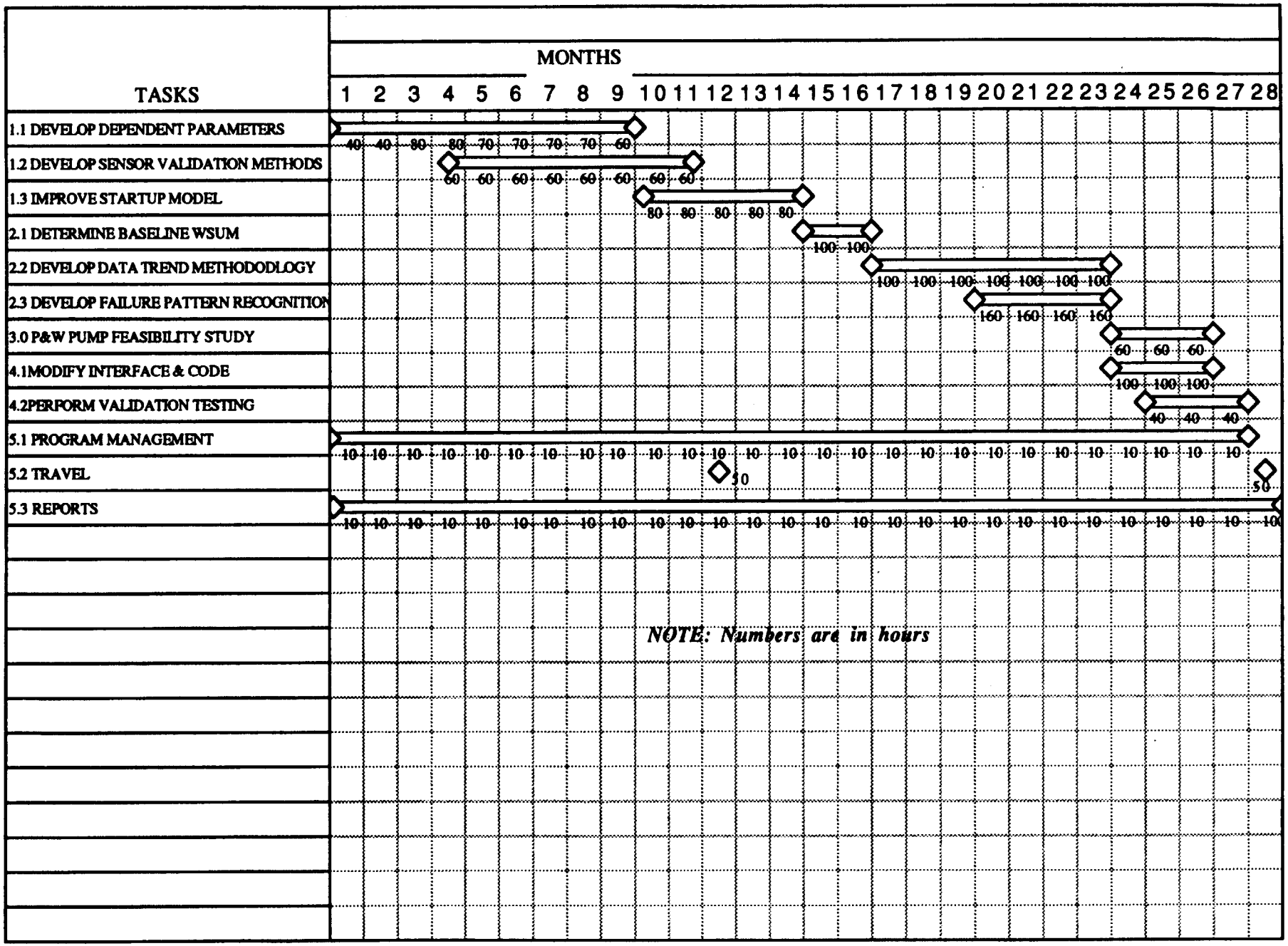
FAILURE PATTERN RECOGNITION



SENSOR VALIDATION



PROPOSED RSA IMPROVEMENT PLAN



Thursday, March 24, 1994

References

1. Health Management System for Rocket Engines, Final Report, NASA CR 185223, 15 January, 1990

2. Development of a Real-time Model Based Safety Monitoring Algorithm for the SSME, AIAA92-3165, A.Norman, J.Maram, P.Coleman, M.D'Valentine, & A.Steffens, 28th AIAA/SAE/ASME/ASEE Joint Propulsion Conference, July 6-8,1992

3. Health Management System for Rocket Engines, E. Nemeth, J.Maram, A.Norman, 2nd Annual Health Monitoring Conference for Space Propulsion Systems, Cincinnati, Ohio, November, 1990

4. Development of a Health Monitoring Algorithm, AIAA-90-1991, E.Nemeth, A.M.Norman. Jr., 26th AIAA/SAE/ASME/ASEE Joint Propulsion Conference

APPENDIX A
RSA VALIDATION TEST RESULTS

FAILURE TEST LIST

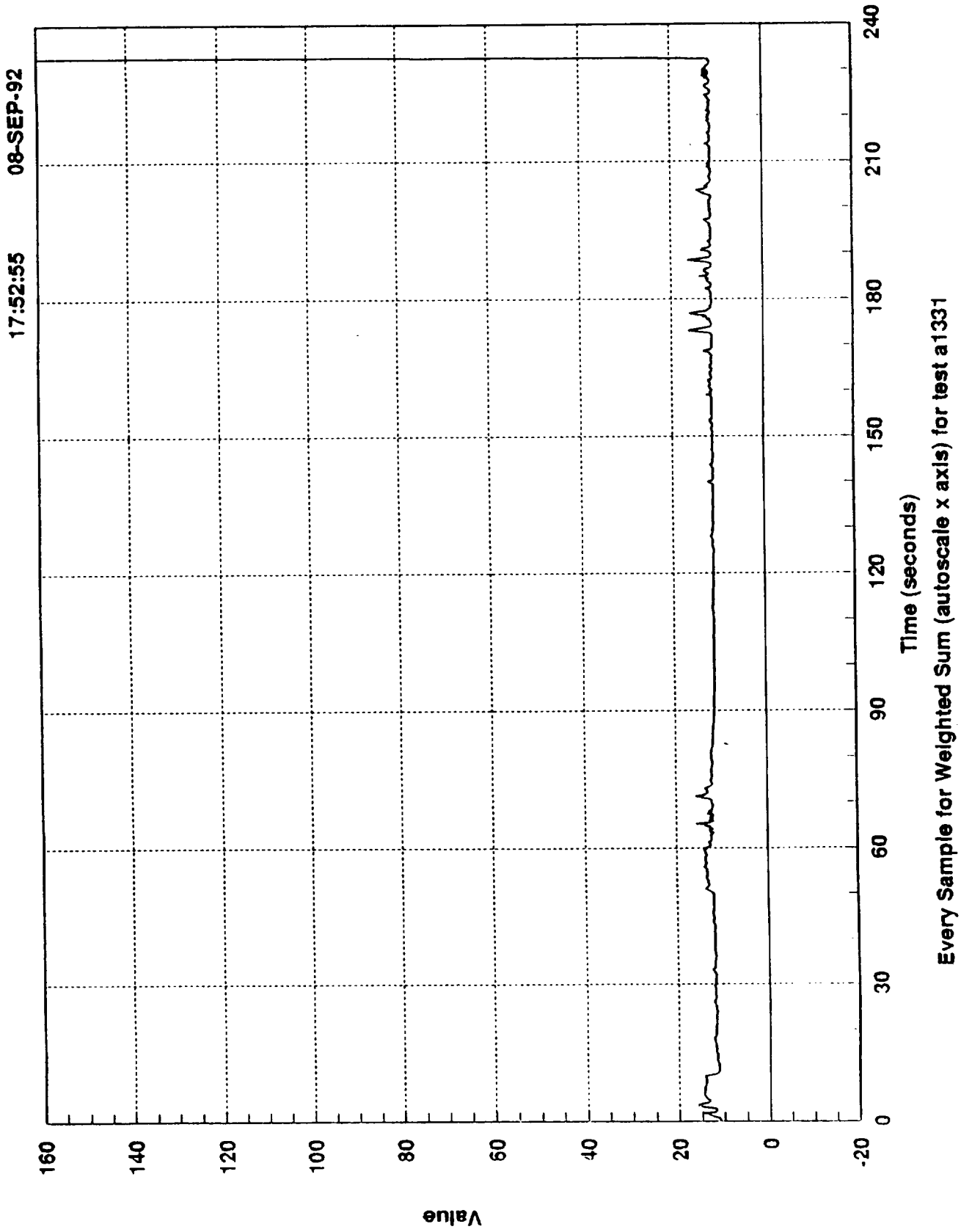
WSUM threshold 60 from 0-20 Seconds, 40 from 20 Seconds - C/O

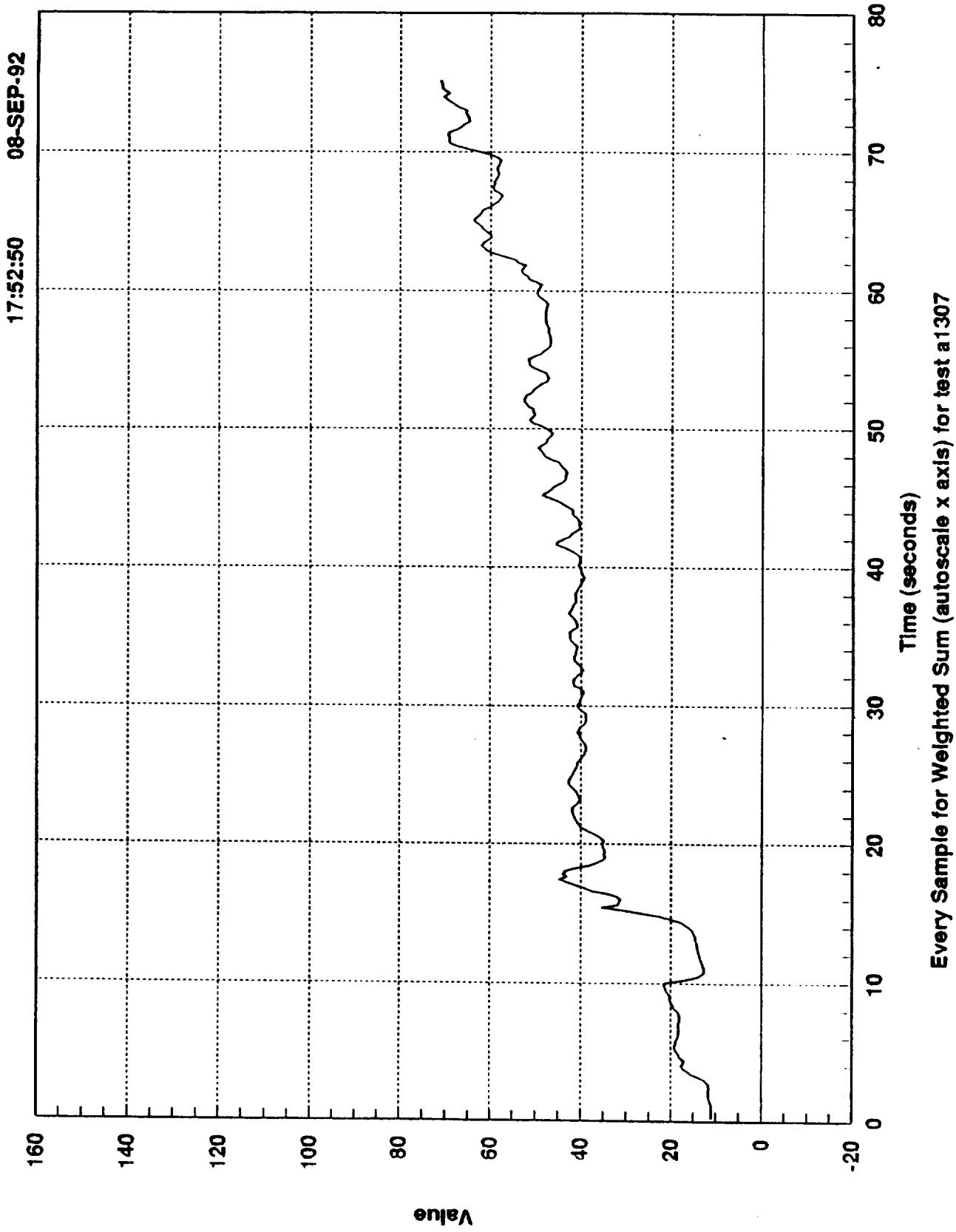
TEST #	CUT-OFF [sec]	RSA CUT-OFF	COMMENTS
A1331	233	NONE	Injector Lox post failure - very rapid - no time to respond prior to C/O
A1307	75	22	RSA C/O at 22 may be due to model differences causing level shift - if so RSA /C/O occurs between 50 and 64 seconds depending upon amount of shift
A3259	101	20	RSA C/O@ 20 to 22 seconds even if model difference shift has occurred.
A3175	115	NONE	Duct structural failure - no early warning for any system to detect using existing sensors
A1225	255.63	145	Lox ignition due to valve fretting resulting from high vibrations - RSA detected cause which eventually led to ignition
A1110	74	NONE	Fire occurred in HPOTP - C/O by controller - don't have data on speed of failure propagation or redline trigger
A1340	405	NONE	Indicators of failure occur at 290 seconds, but current algorithm does not detect due to sharpness of data spike - without filter, the failure is detected. Future work may provide improved filter technique which would detect this failure
A1363	250	135	HPFTP Turnaround duct cracked - RSA algorithm may have detected as early as 30 seconds, but certainly at 135 seconds
A1436	611	NONE	Coolant liner buckled at 610.36 - no time to react
A1364	392	40(25)	Kaiser hat nut failure early in test propogates to worse and worse failures as test continues until engine is destroyed. First RSA failure signal is about 40 seconds if model shift is assumed (25 seconds, if not) - Algorithm shows continued buildup of distress until cutoff occurs

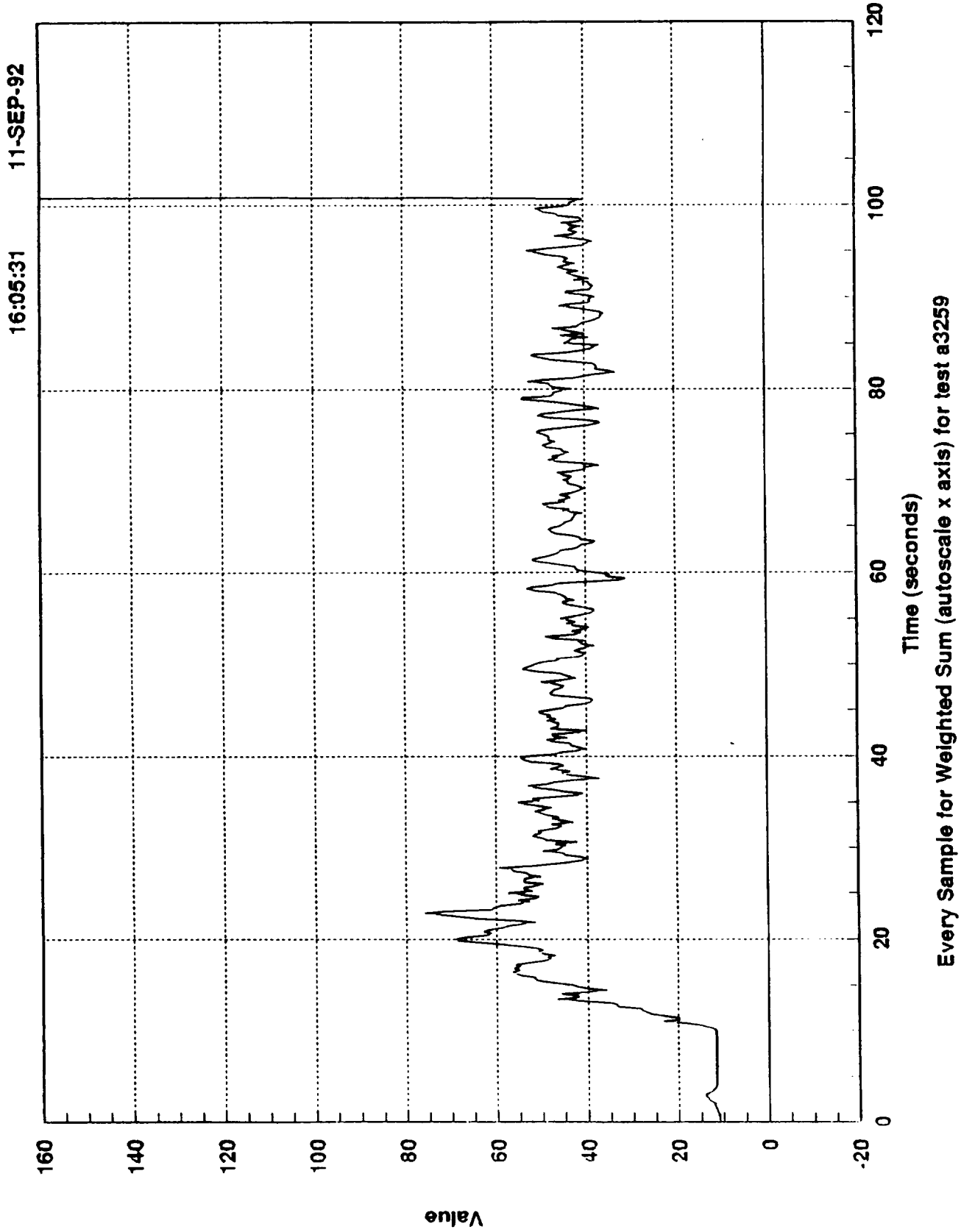
FAILURE TEST LIST

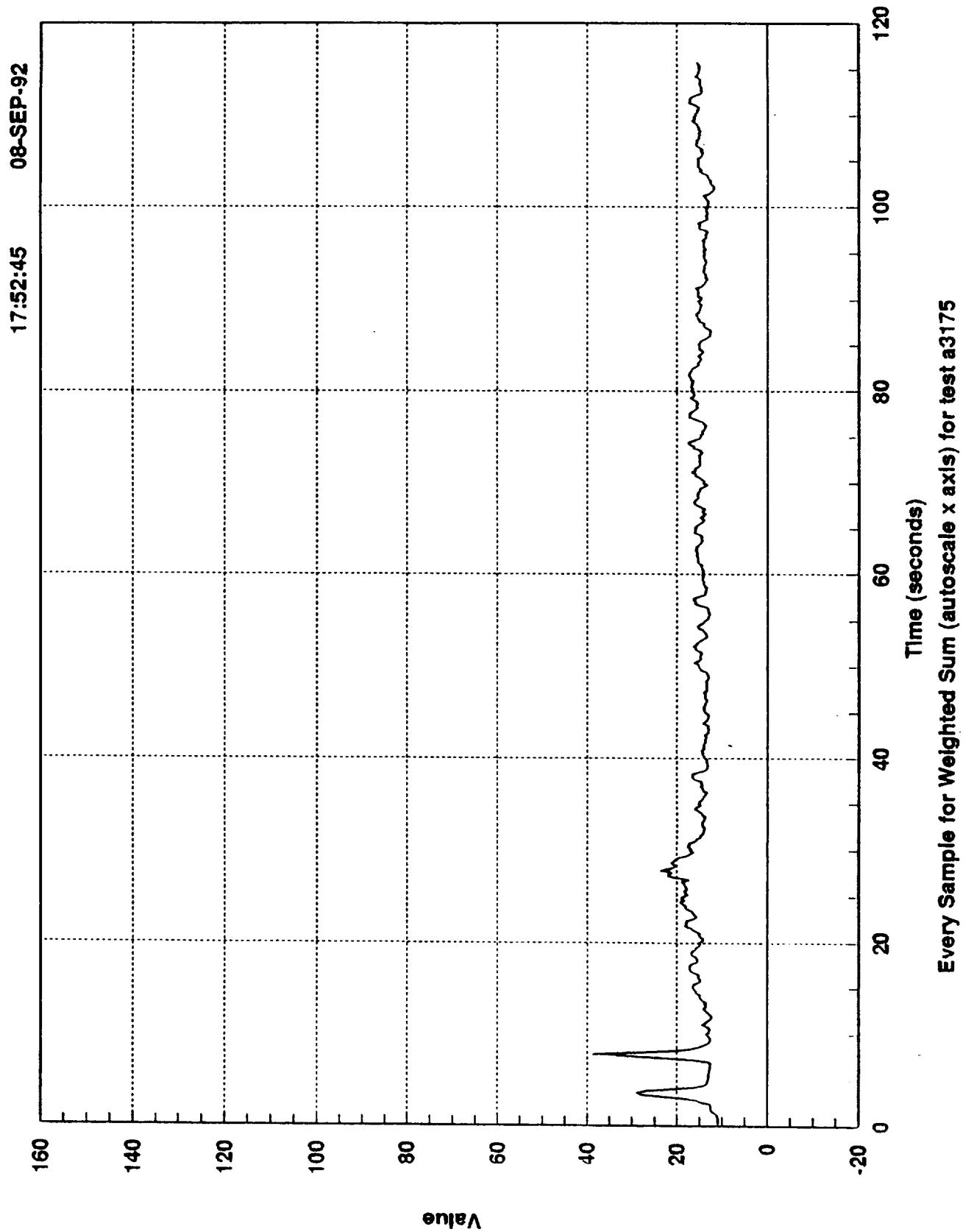
WSUM threshold 60 from 0-20 Seconds, 40 from 20 Seconds - C/O

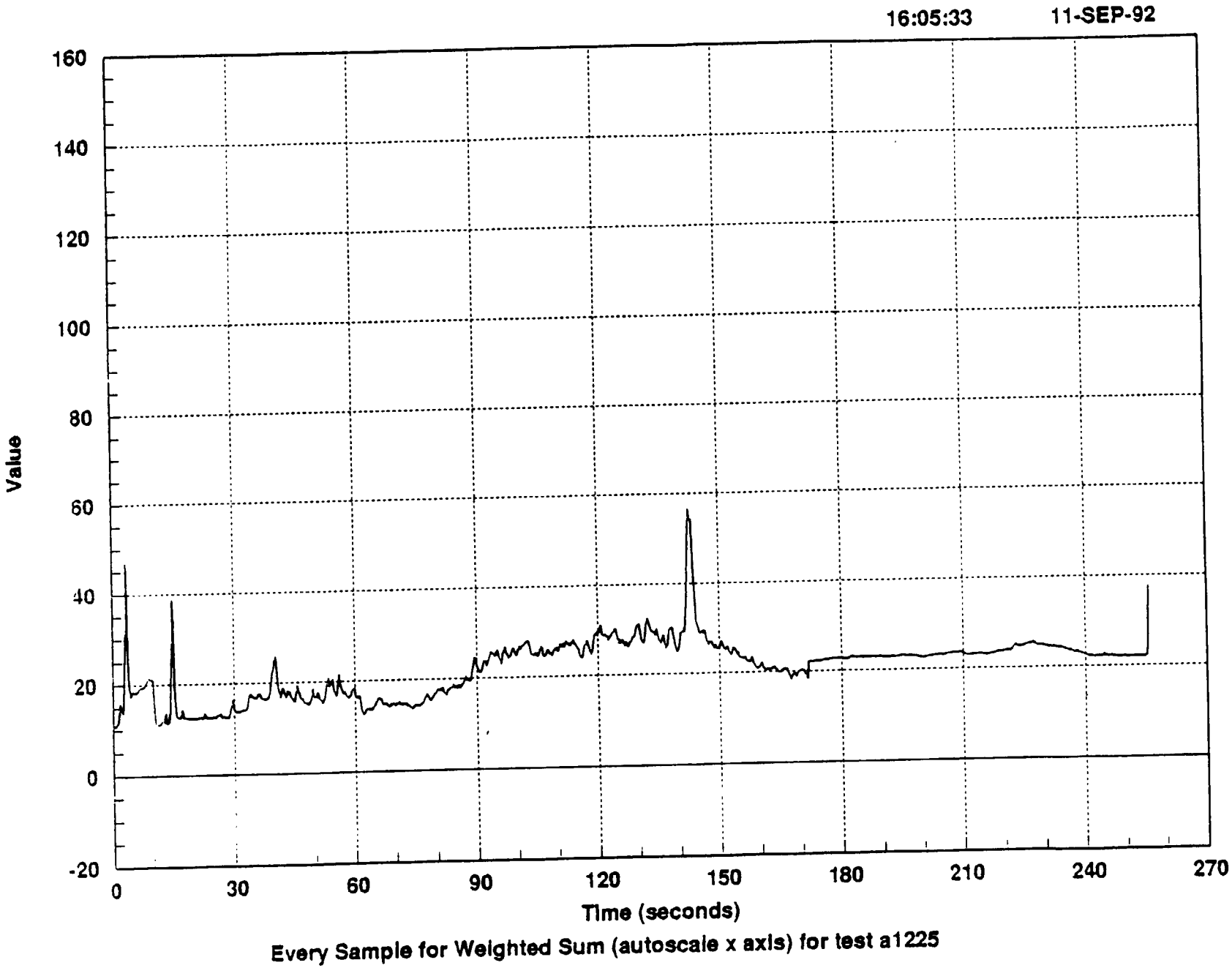
TEST #	CUT-OFF [sec]	RSA CUT-OFF	COMMENTS
A2209	823	595	HPFT end dome lock tab breaks loose and propagates damage - not catastrophic
A2249	450	150	Initial damage began early in test & slowly propogated. HPFP pump cavitation began at 108 seconds and slowly increased. KelF rub ring fails at 374 seconds, first stage turbine blade fails at 450 seconds. RSA detected failure during early cavitation period - algorithm shows increase in distress throughout test
A2095	51	8	Shutdown due to preburner pump radial accel redline - eroded turbine damage and 8 main injector Lox posts eroded - RSA detects early indication at 8 seconds, and shows buildup of distress beginning at 30 seconds
A1346	500	20	Damage to HPFT and MCC liner - Strong indication of problem by RSA at about 20 seconds
A1362	500	240(20)	HPOT and HPFT - damaged - RSA nominal value appears to be shifted up due to model difference - failure clearly indicated at 240 seconds even if shifted - if not shifted, failure noted at 20 seconds
A1183	50	2	Injector face burn through due to structural failure - self limiting - RSA indicates major problem between 2 and 10 seconds
A2428	204.1	130	Good example of recent failure (not SAFD database) - RSA shows clearly distress buildup as time progresses with cutoff at 130 seconds

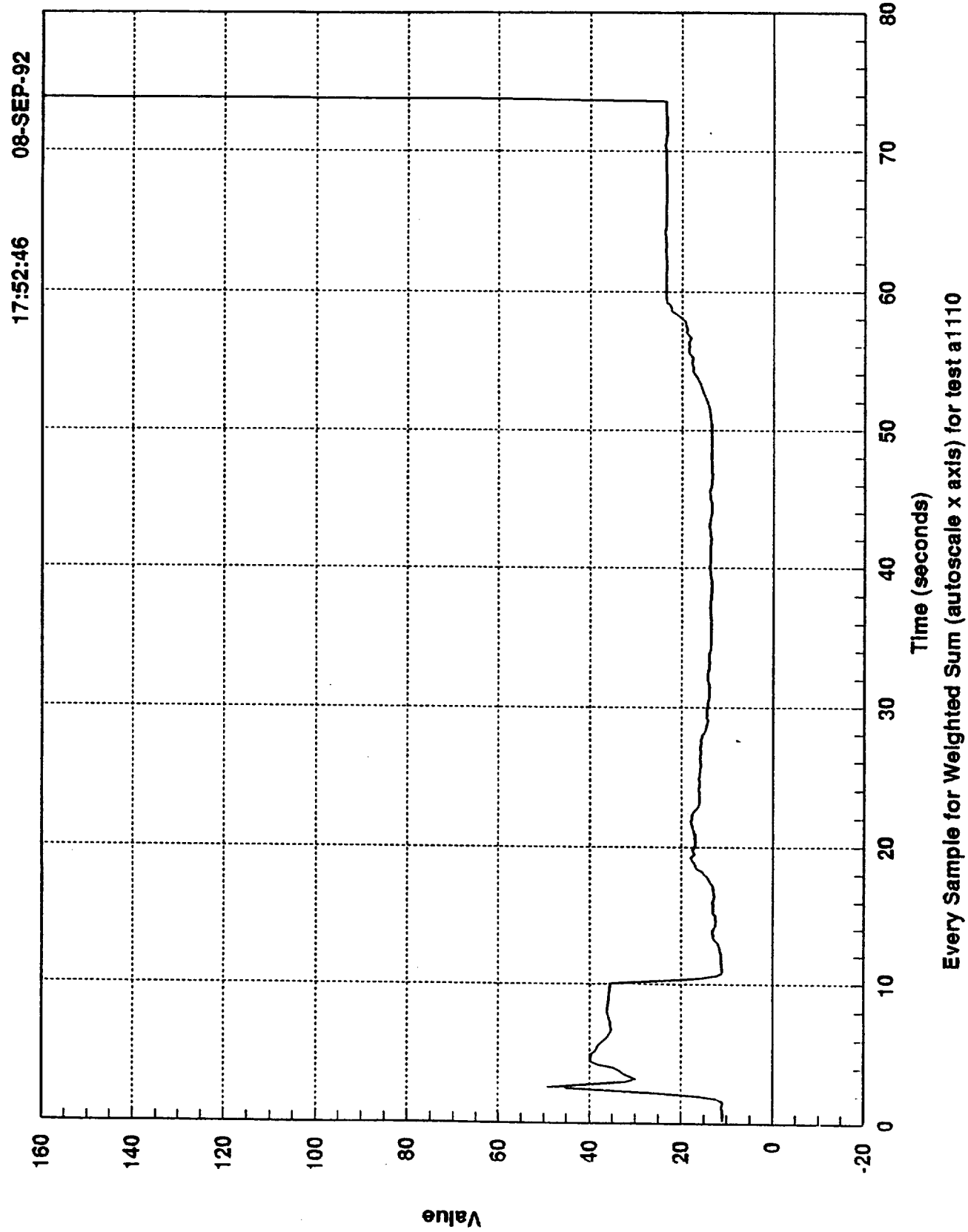

17:52:55
08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
30
60
90
120
150
180
210
240
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1331

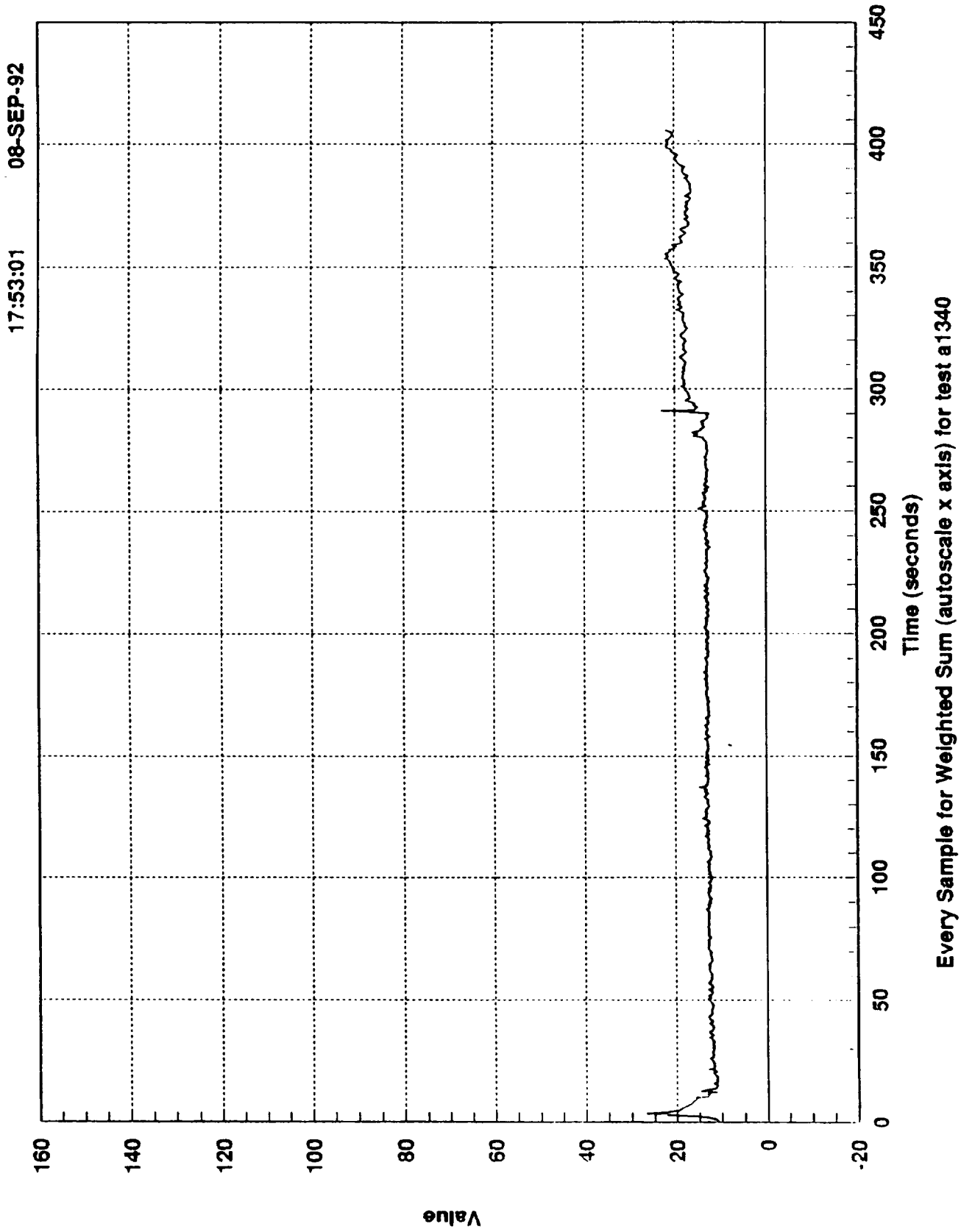

17:52:50 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
10
20
30
40
50
60
70
80
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1307

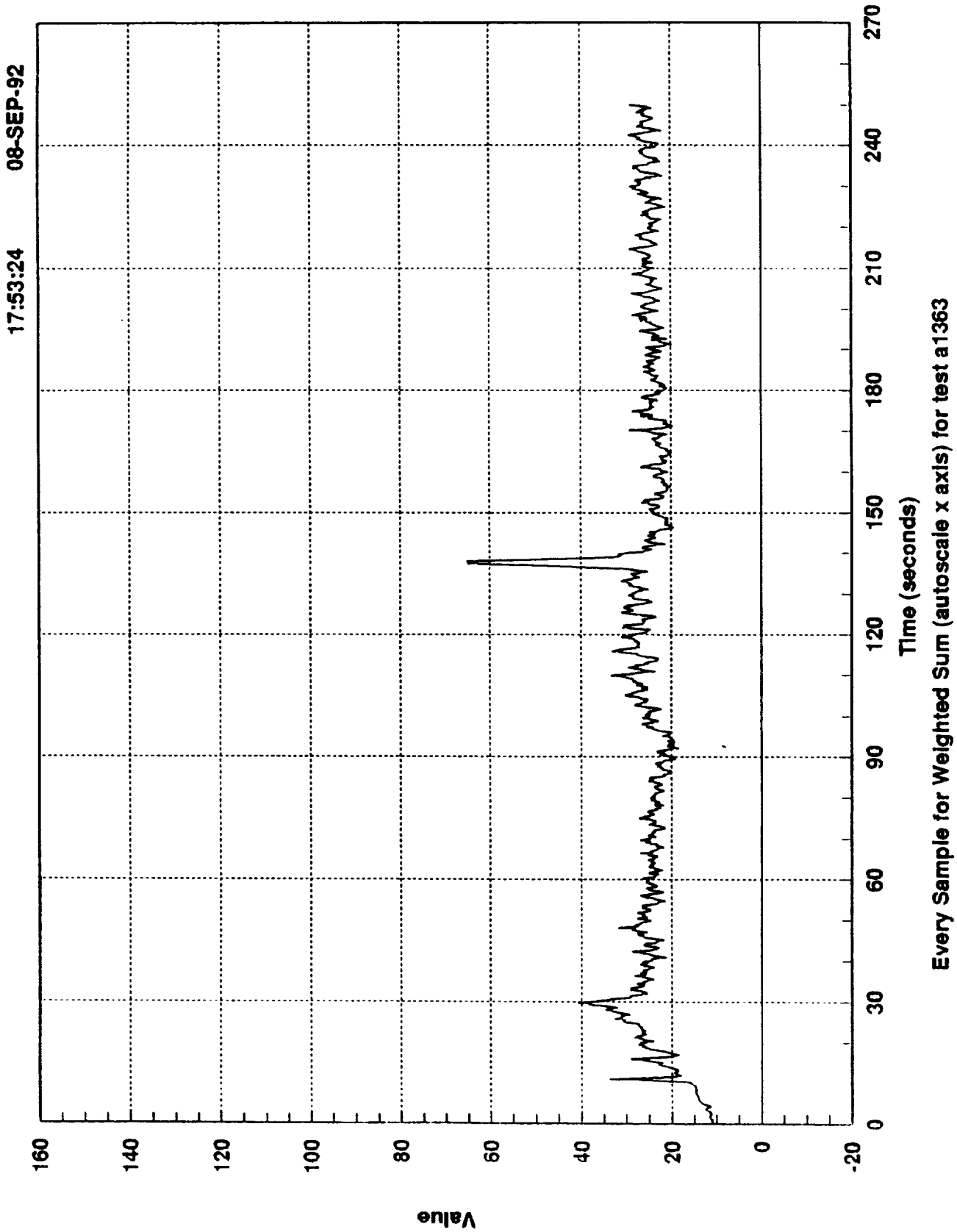

16:05:31
11-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
20
40
60
80
100
120
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a3259

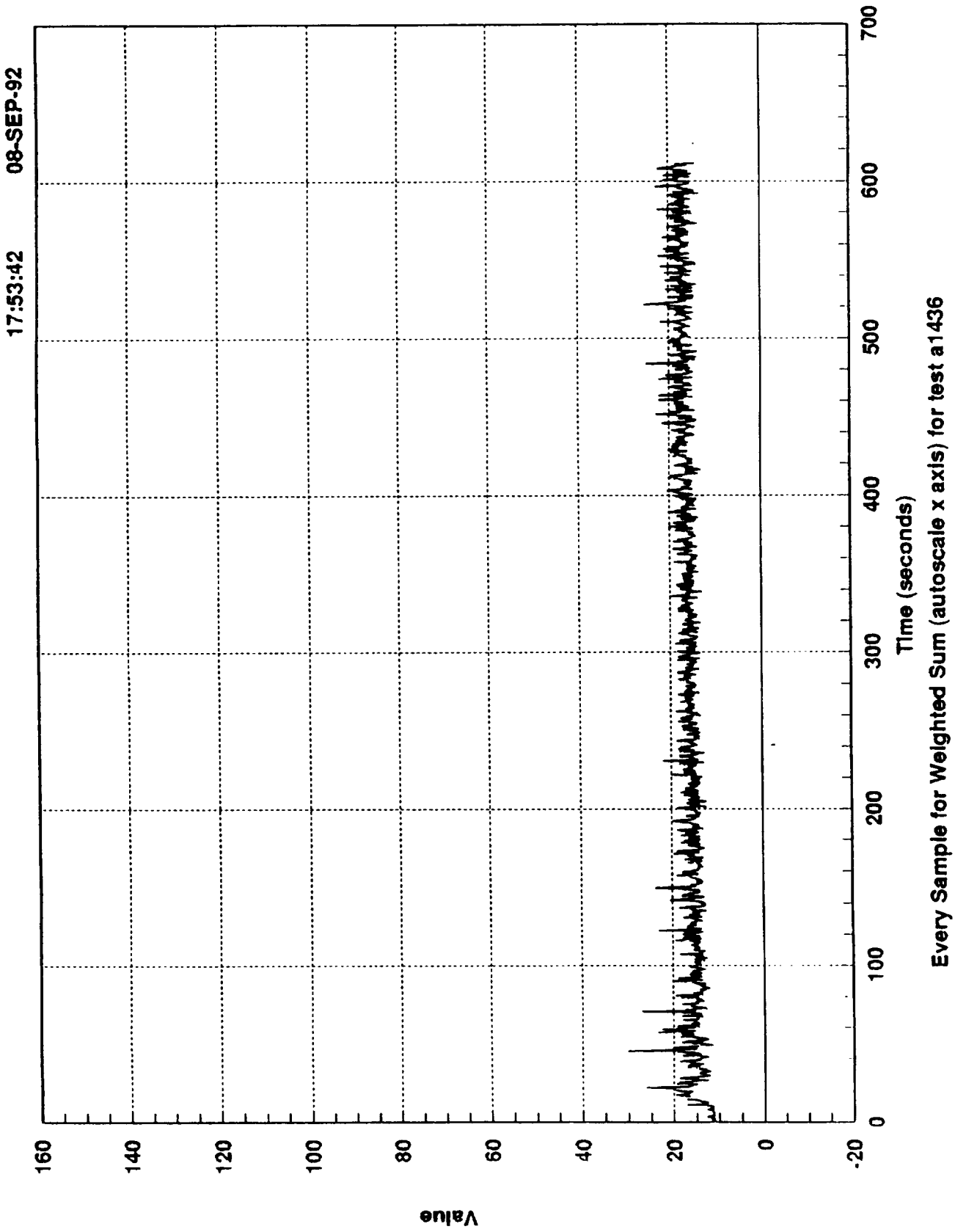

17:52:45
08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
20
40
60
80
100
120
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a3175

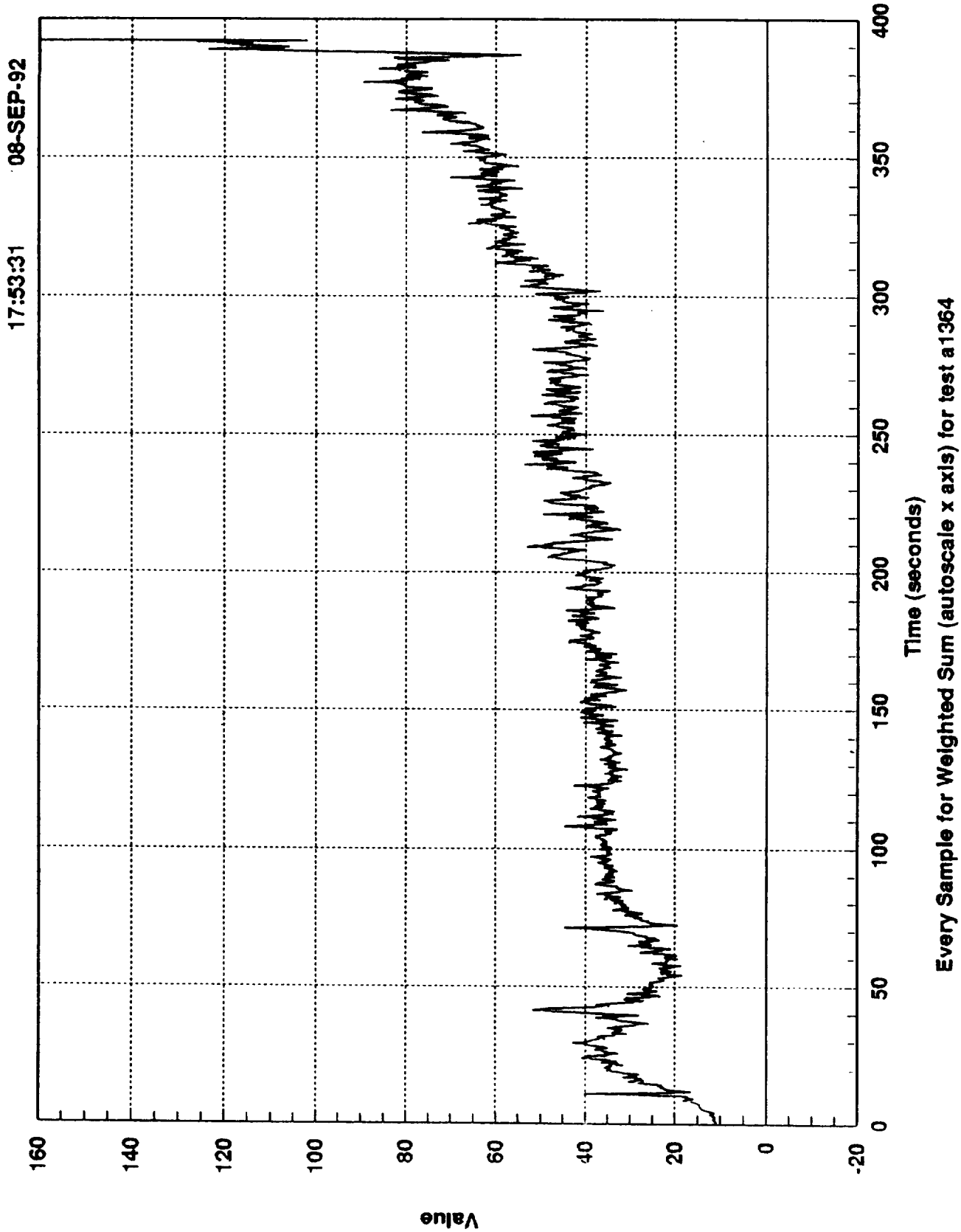

16:05:33
11-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
30
60
90
120
150
180
210
240
270
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1225

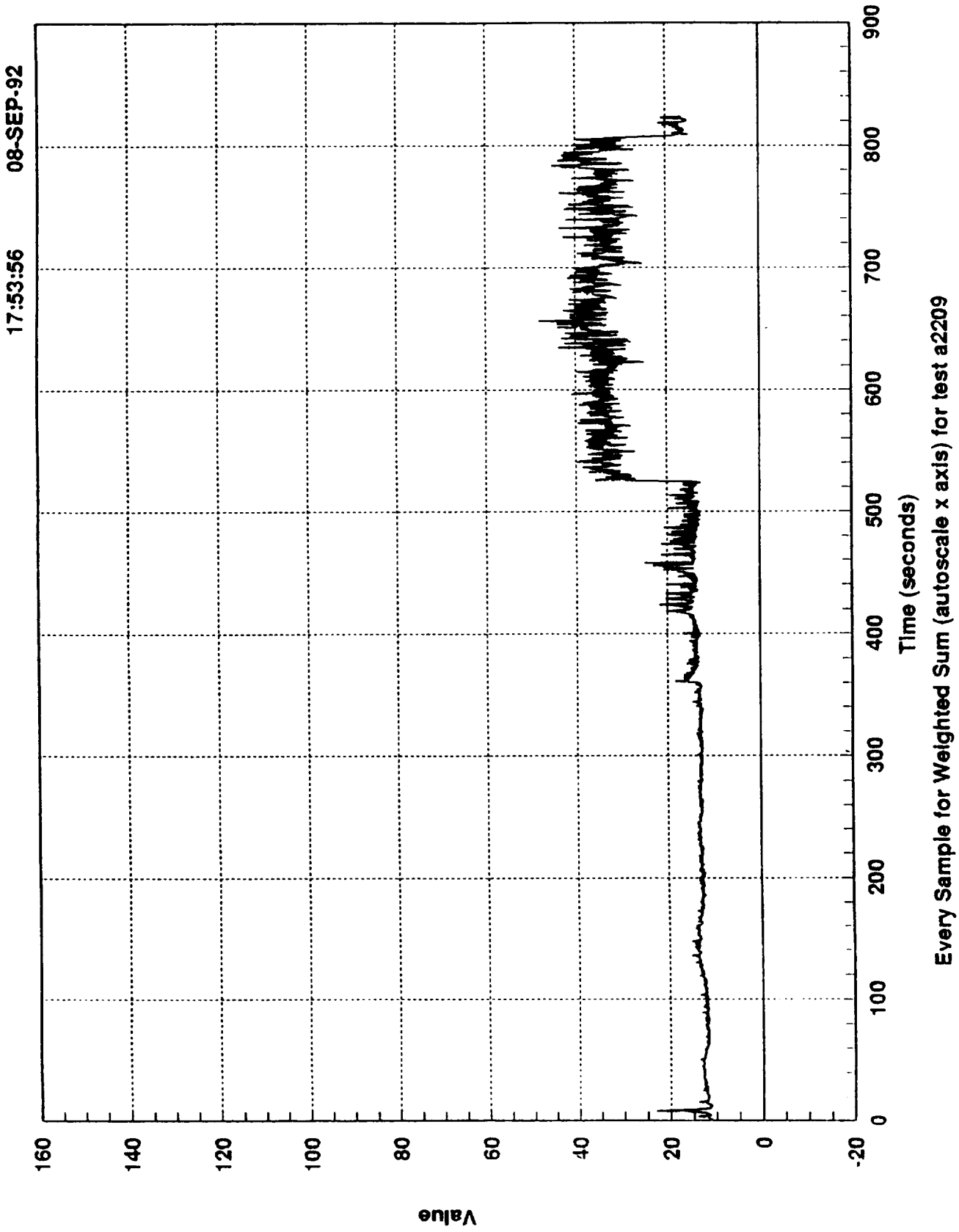

17:52:46 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
10
20
30
40
50
60
70
80
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1110

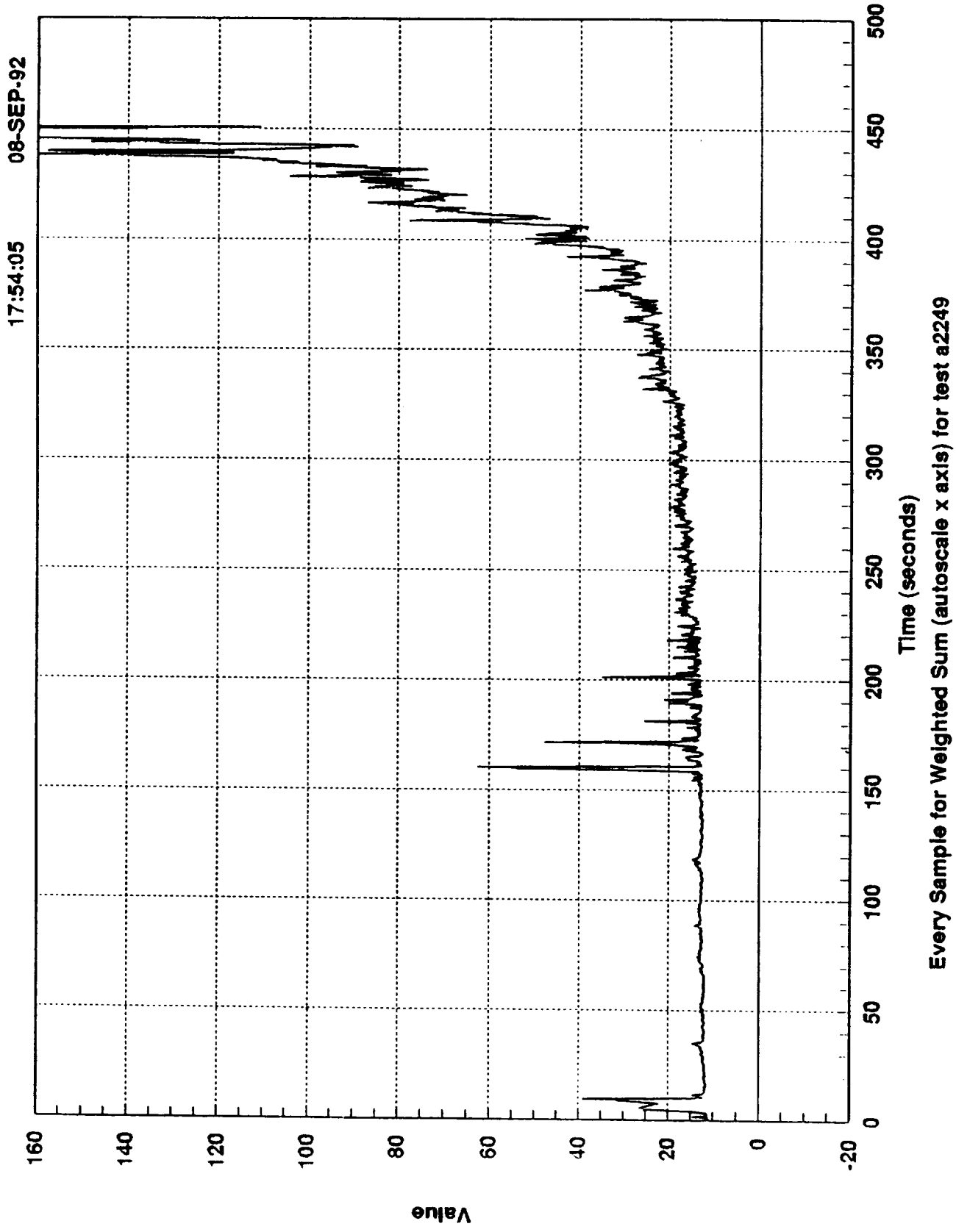

17:53:01 08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
50
100
150
200
250
300
350
400
450
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1340

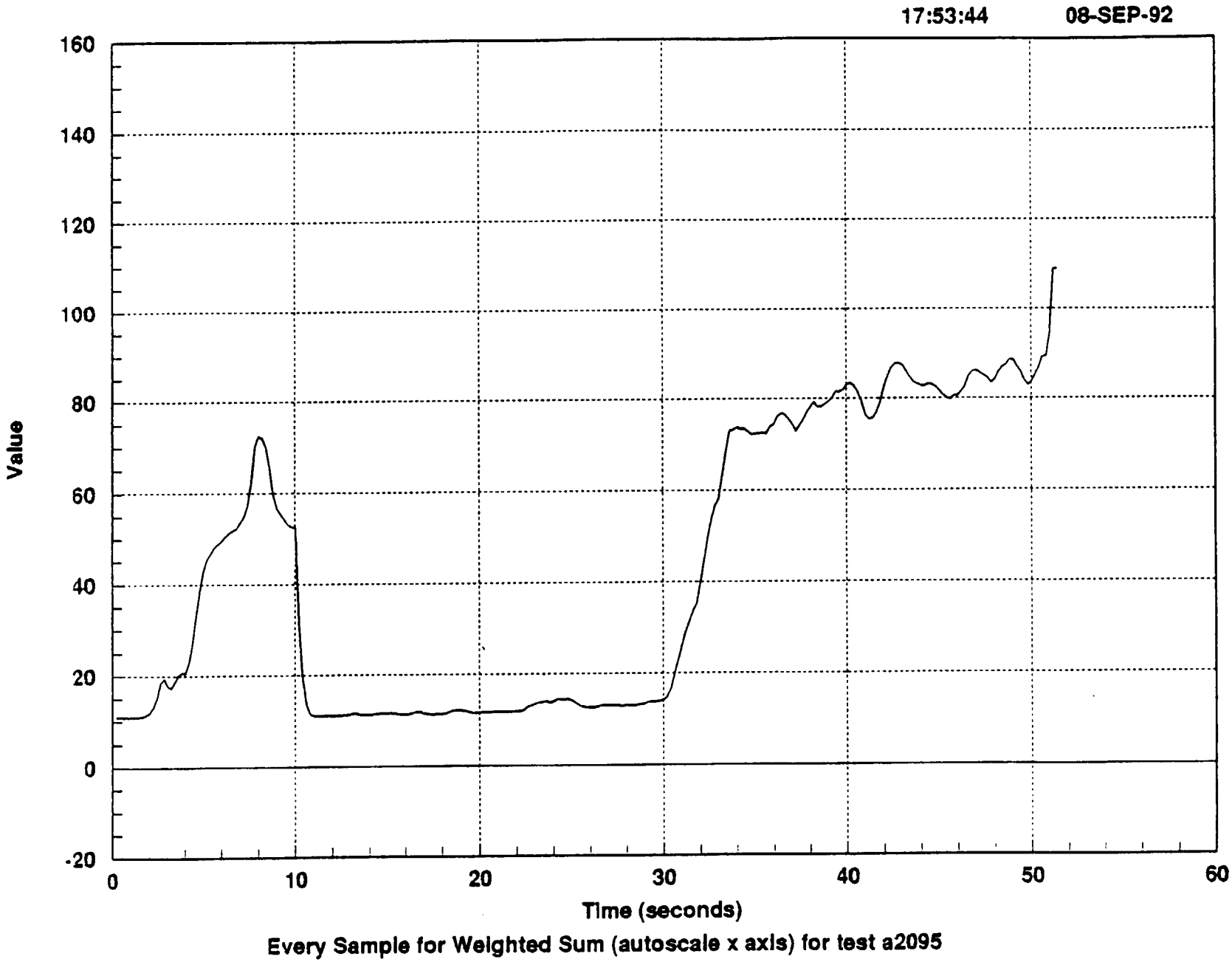

17:53:24 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
30
60
90
120
150
180
210
240
270
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1363


17:53:42
08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
100
200
300
400
500
600
700
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1436

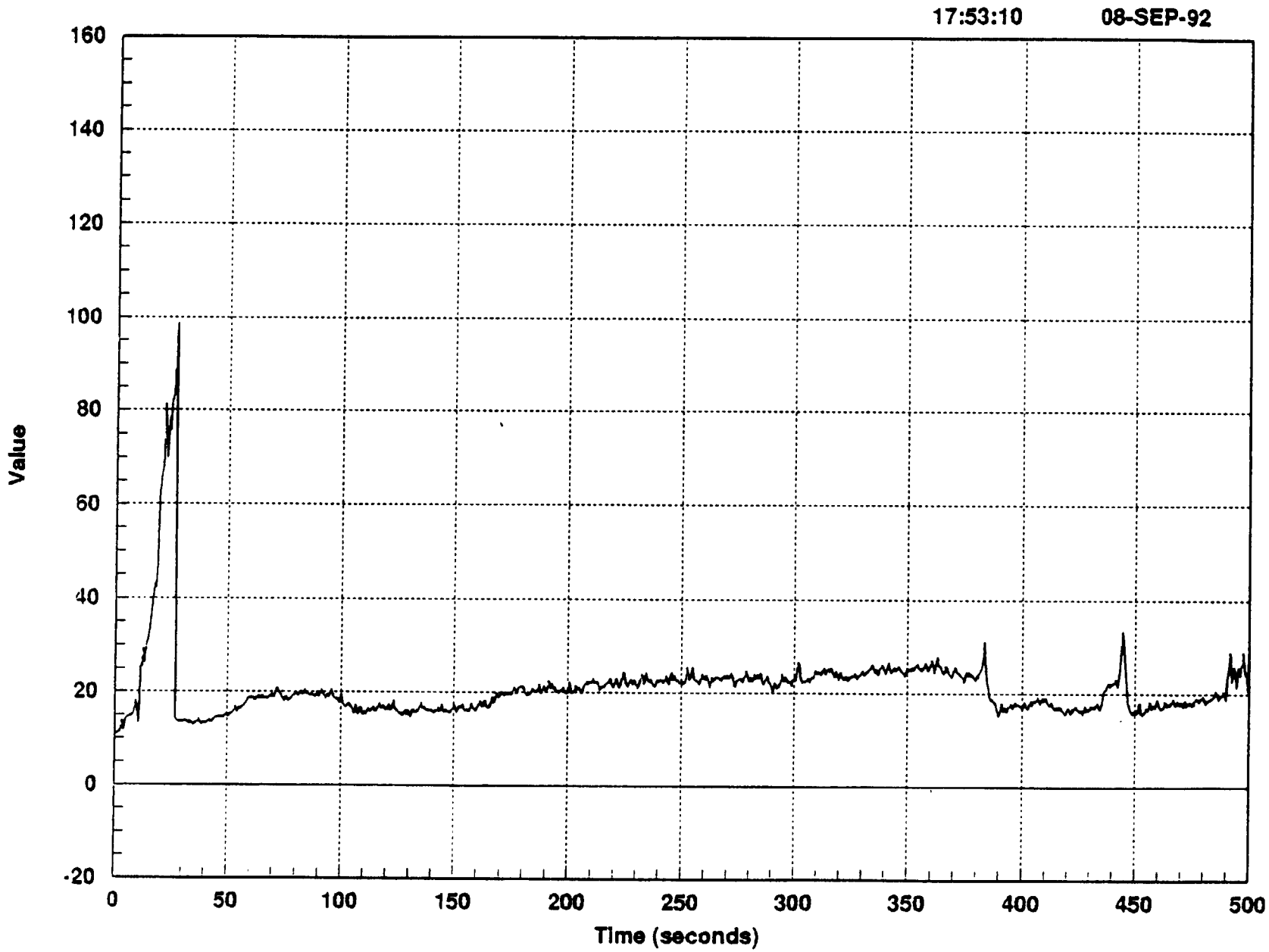

17:53:31
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
50
100
150
200
250
300
350
400
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1364

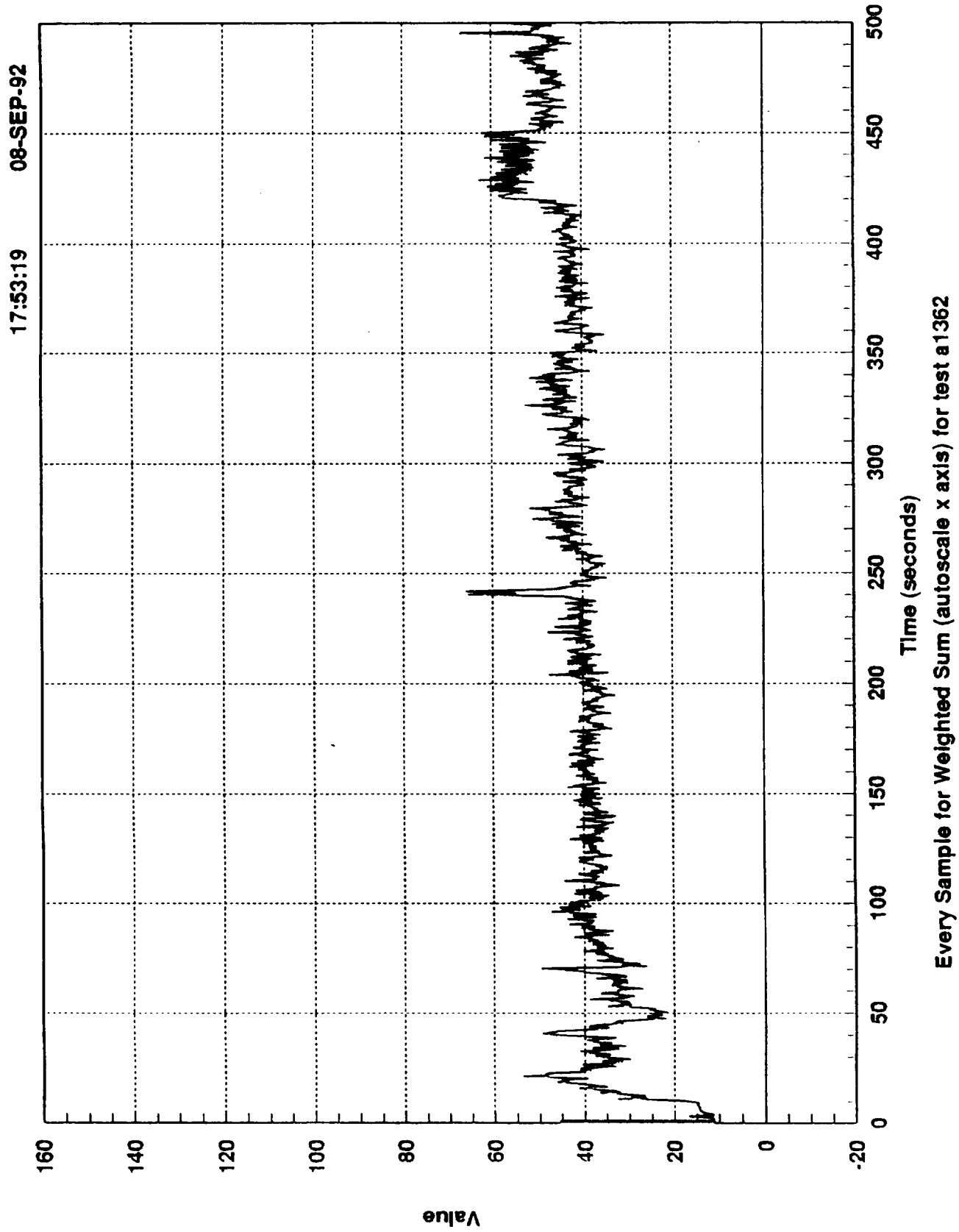

17:53:56 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
100
200
300
400
500
600
700
800
900
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2209

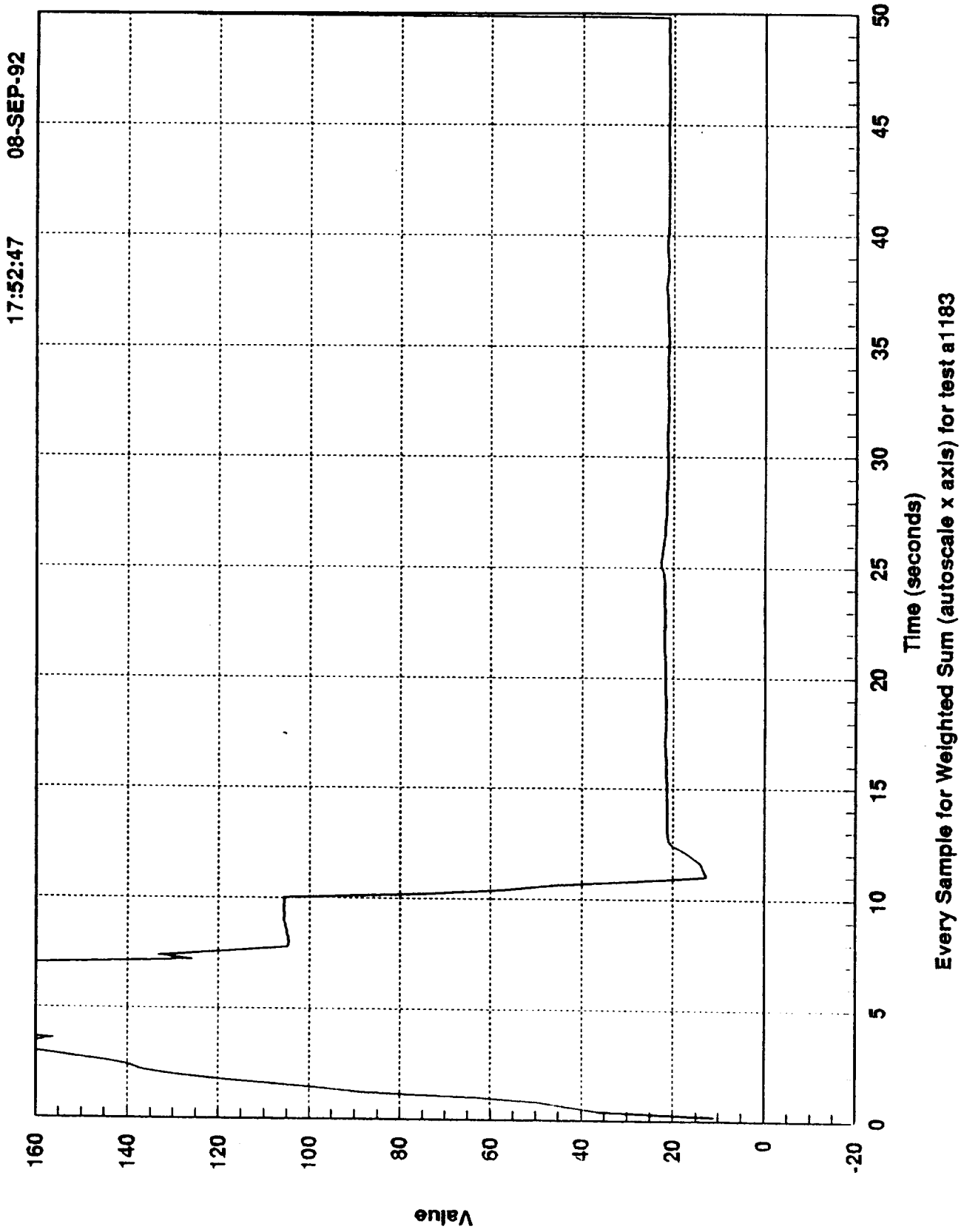

17:54:05 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
50
100
150
200
250
300
350
400
450
500
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2249

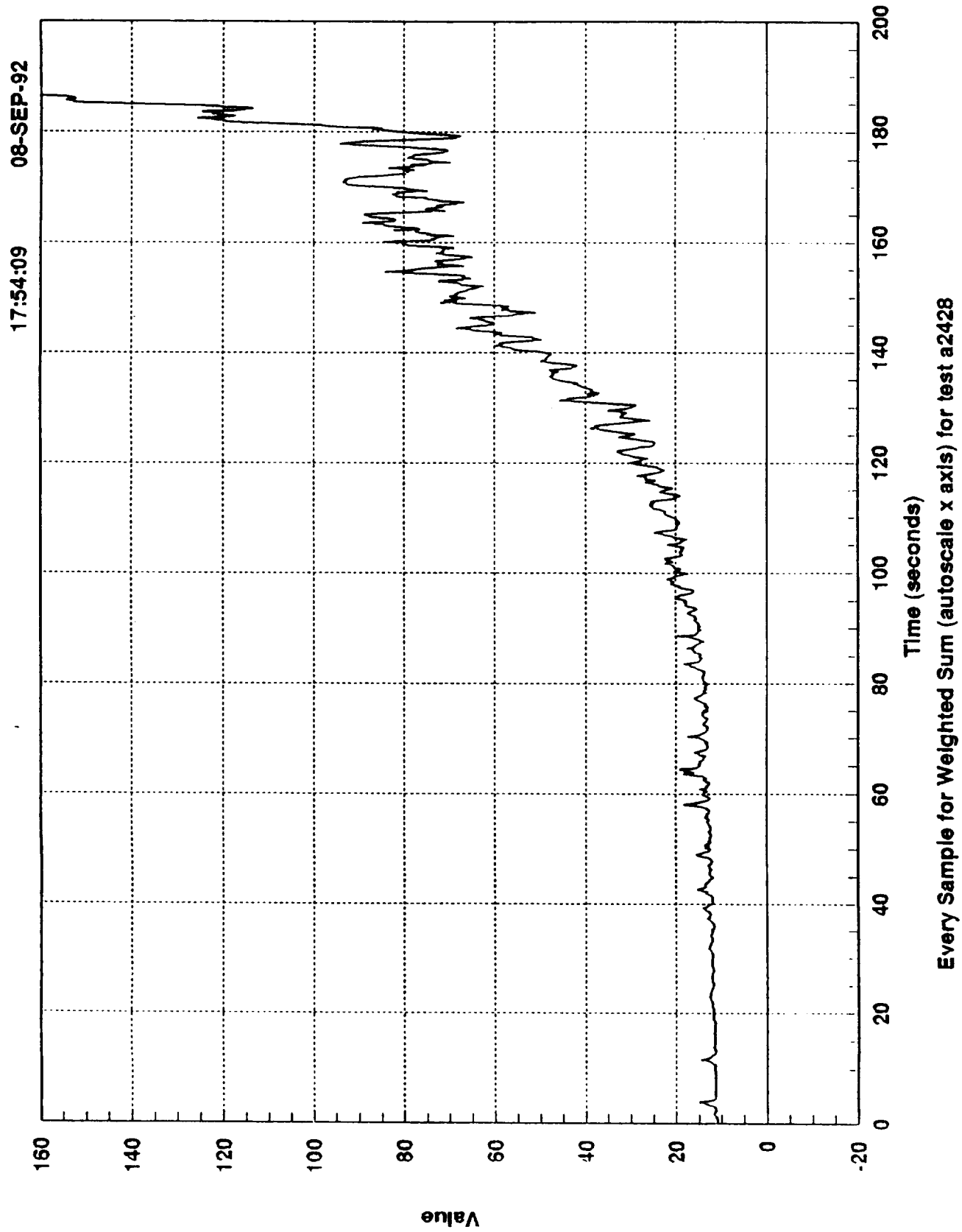


Every Sample for Weighted Sum (autoscale x axis) for test a2095

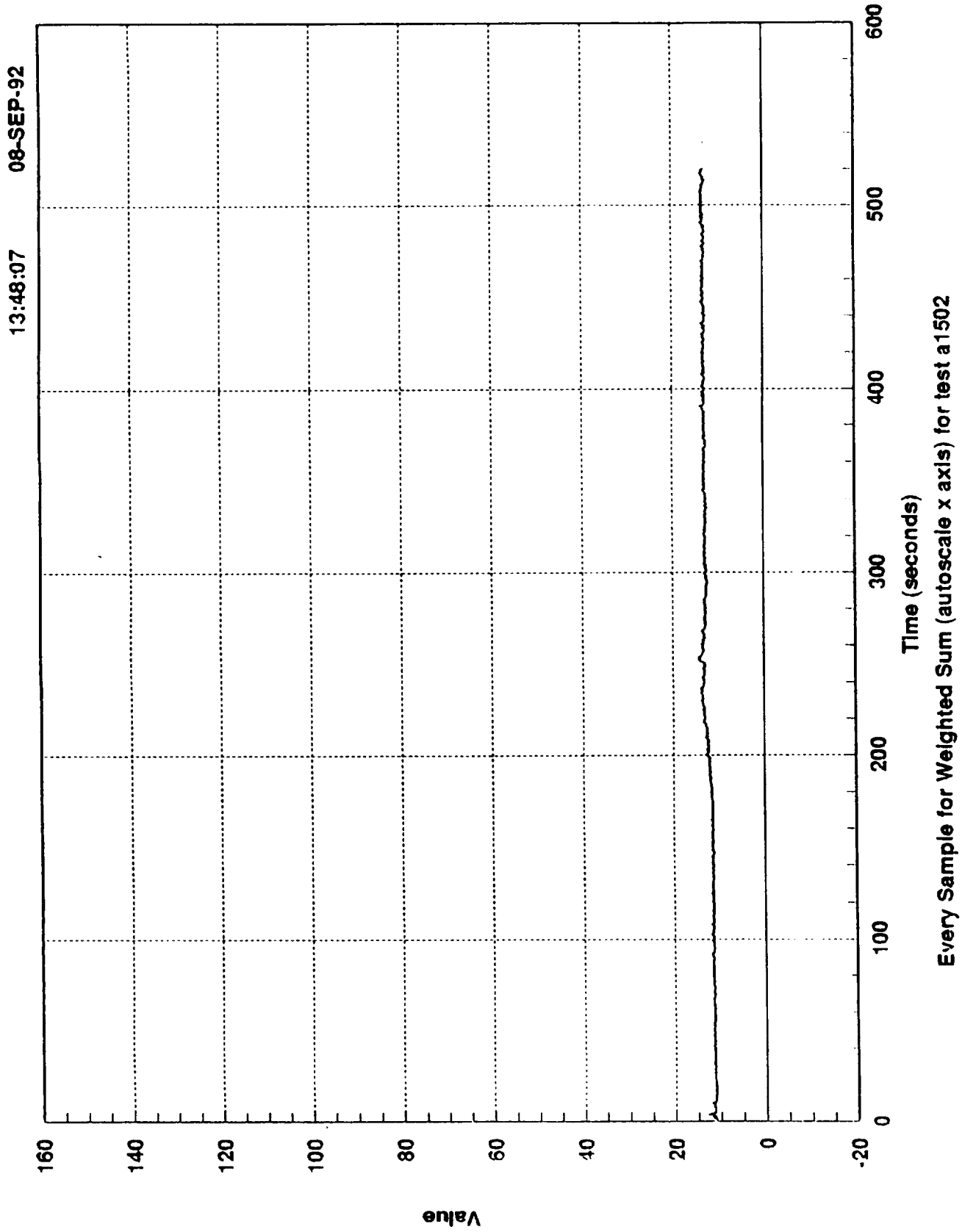

17:53:10
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
50
100
150
200
250
300
350
400
450
500
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1346

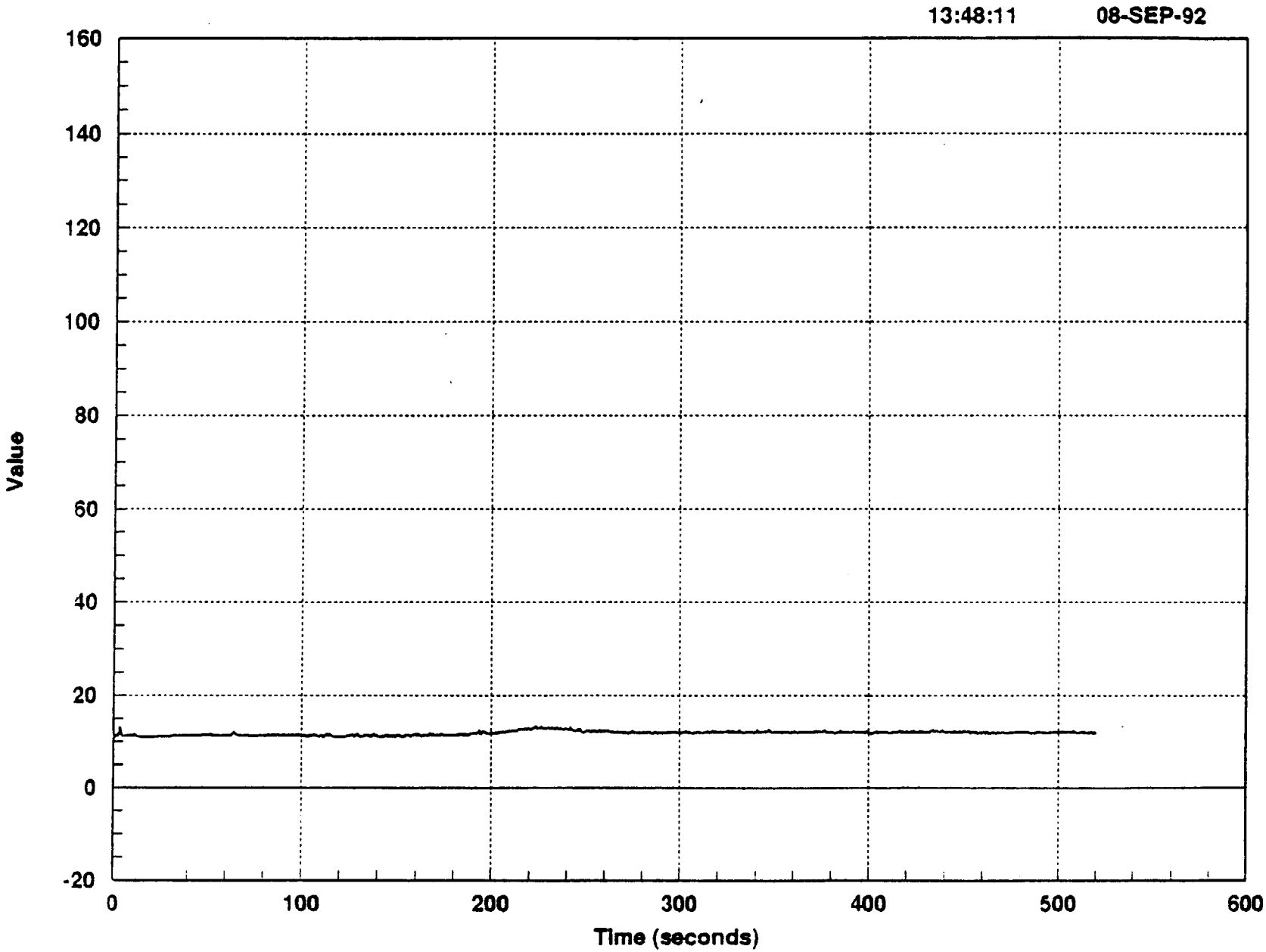

17:53:19
08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
50
100
150
200
250
300
350
400
450
500
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1362


17:52:47
08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
5
10
15
20
25
30
35
40
45
50
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1183

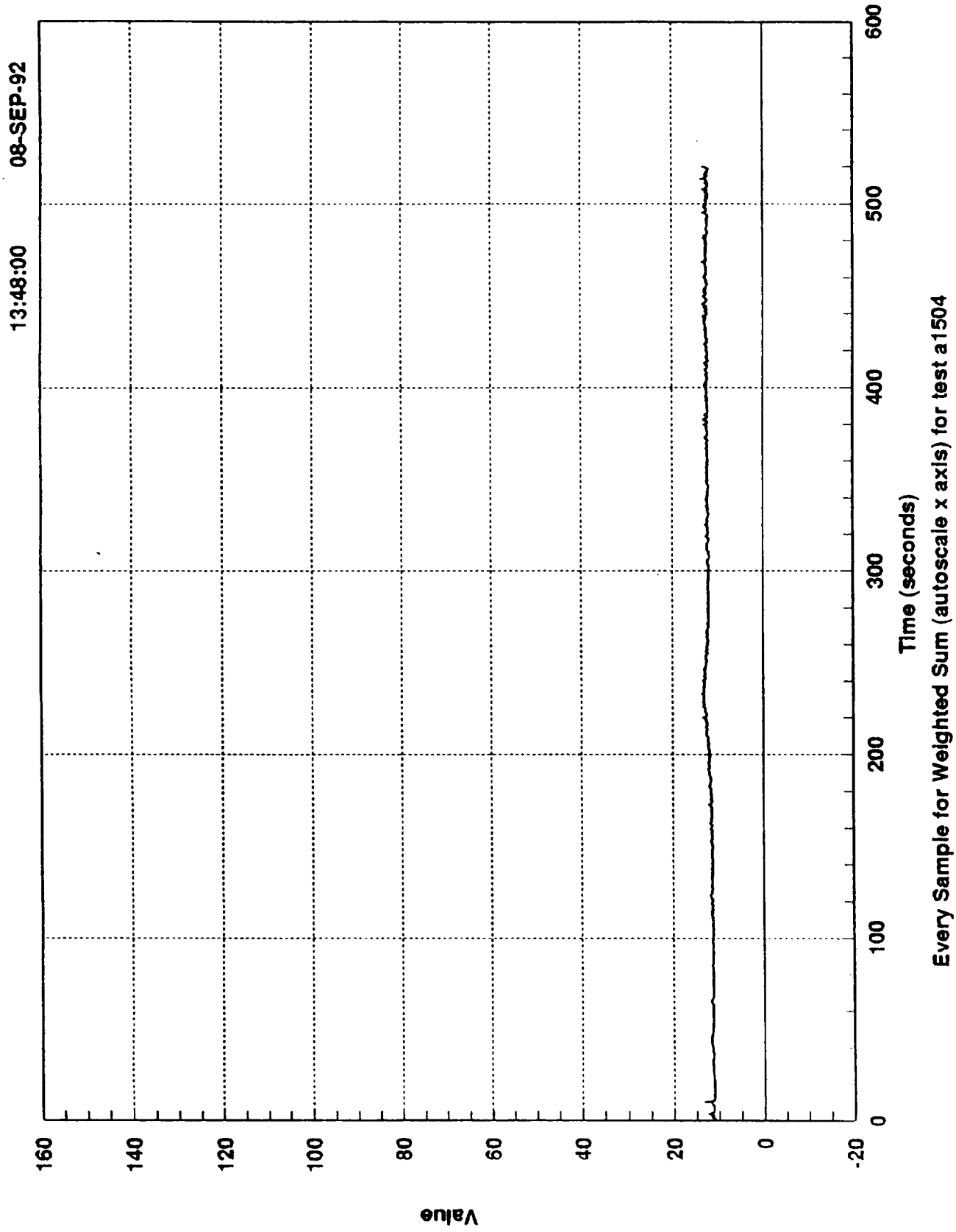

17:54:09 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
20
40
60
80
100
120
140
160
180
200
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2428

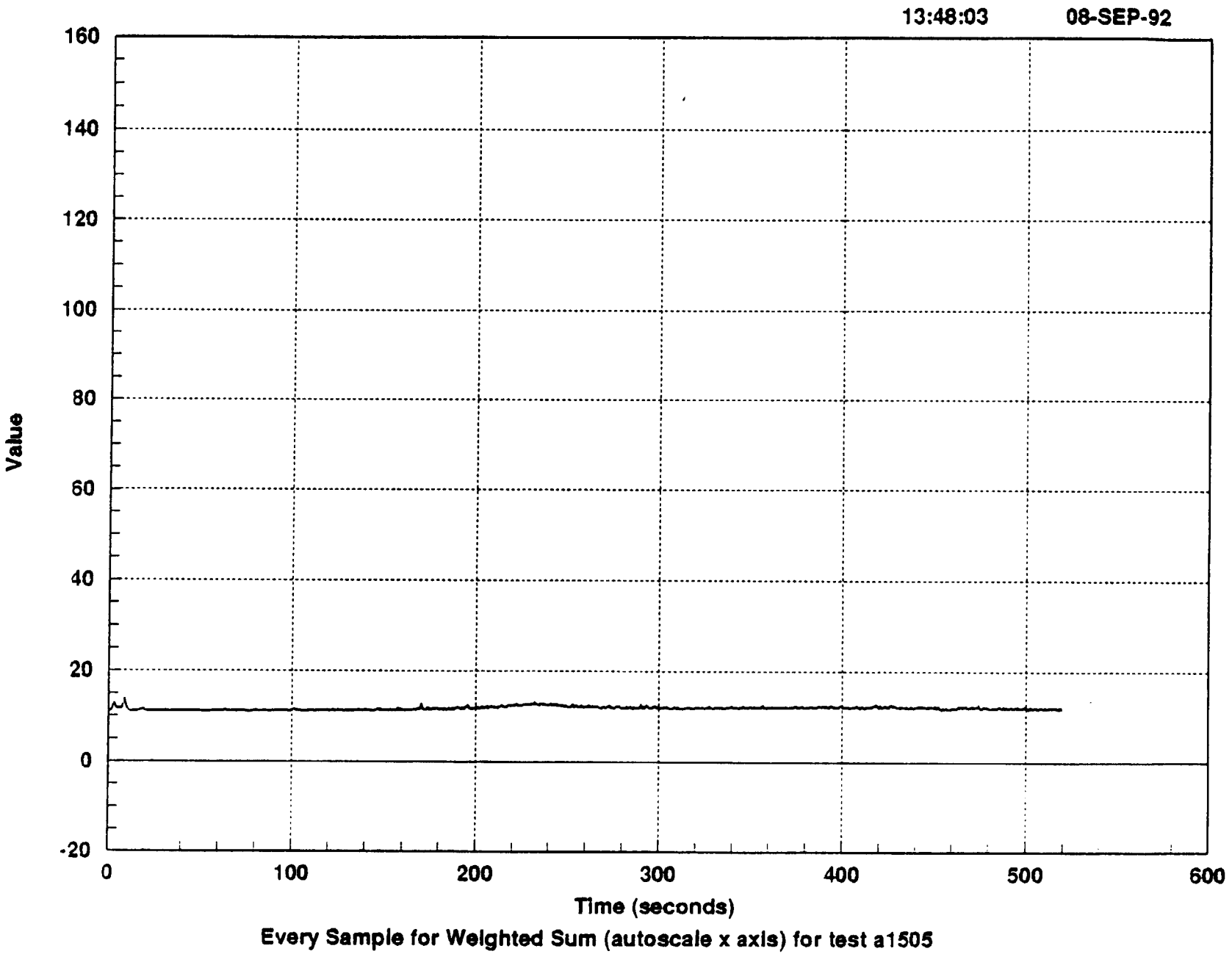
				Threshold Table					
				Limit = 60 from 0-20 Seconds					
				Limit = 40 from 20 Seconds to Cutoff					
Test Number	Engine S/N	Max Value 0-20 Sec	Max Value 20 Sec-C/O	% Threshold	Test Number	Engine S/N	Max Value 0-20 Sec	Max Value 20 Sec-C/O	% Threshold
A1502	2105	13	13	33	A2480	2206	13	15	38
A1503	2105	12	12	30	A2481	2206	13	24	60
A1504	2105	12	13	33	A2482	2206	15	15	38
A1505	2105	13	12	30	A2483	2206	12	17	43
A1514	2105	23	12	38	A2484	2206	13	15	38
A1515	2105	22	14	37	A2585	2206	13	12	30
A1520	2011	14	14	35	A2486	2206	13	12	30
A1521	2011	13	13	33	A2492	2012	14	19	48
A1522	2105	15	19	48	A2493	2012	14	29	73
A1539	O211	20	14	35	A2496	2026	13	25	63
A1544	O211	27	13	45	A2499	2026	13	15	38
A1550	2019	13	11	28	A2503	2206	15	12	30
A1551	2019	13	15	38	A2504	2206	13	13	33
A1558	O211	15	13	33	A2506	2030	15	21	53
A1559	O211	20	15	38	A2508	2206	13	15	38
A1560	O211	24	12	40	A2510	2032	15	28	70
A1561	O211	17	13	33	A2511	2032	17	24	60
A1562	O211	23	15	38	A2516	2206	19	21	53
A1592	O211	17	22	55	A2521	2033	15	12	30
A1667	2011	13	13	33	A2522	2033	13	36	90
A2410	2106	12	22	55	A2523	2206	16	15	38
A2411	2106	12	23	58	A2524	2206	14	14	35
A2413	2106	15	23	58	A2528	2206	12	17	43
A2417	2106	17	23	58	A2529	2206	17	19	48
A2430	2027	16	13	33	A2530	2206	18	15	38
A2436	2028	13	14	37	A2531	2035	15	17	43
A2437	2028	13	21	53	A2533	2206	13	15	38
A2439	2029	13	16	40	A2535	2206	13	17	43
A2441	2029	13	18	45	A2537	2035	18	15	38
A2443	2030	15	18	45	B1008	2206	13	19	48
A2446	2027	13	17	43	B1010	2206	13	14	35
A2447	2027	13	30	75	B1012	2206	13	20	50
A2449	2029	14	30	75	B1013	2206	12	23	58
A2452	2107	18	15	38	B1027	O212	23	17	43
A2454	2015	16	17	43	B1028	O212	40	22	66
A2455	2015	15	18	45	B1030	O212	52	17	87
A2457	2031	17	18	45	B1031	O212	20	20	50
A2458	2031	17	18	45	B1045	2031	14	15	38
A2460	2019	16	18	45	B1046	2031	16	21	53
A2461	2019	15	29	73	B1047	2031	15	21	53
A2462	2019	15	20	50	B1049	O213	23	28	70
A2463	2019	15	16	40	B1050	O213	18	11	28
A2466	2107	23	24	60	B1051	O213	42	31	78
A2468	2206	15	22	55	B1059	O213	28	27	68
A2470	2206	52	21	87	B1060	O213	31	23	58
A2474	2011	15	12	30					
A2477	2024	14	17	43					


13:48:07
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1502

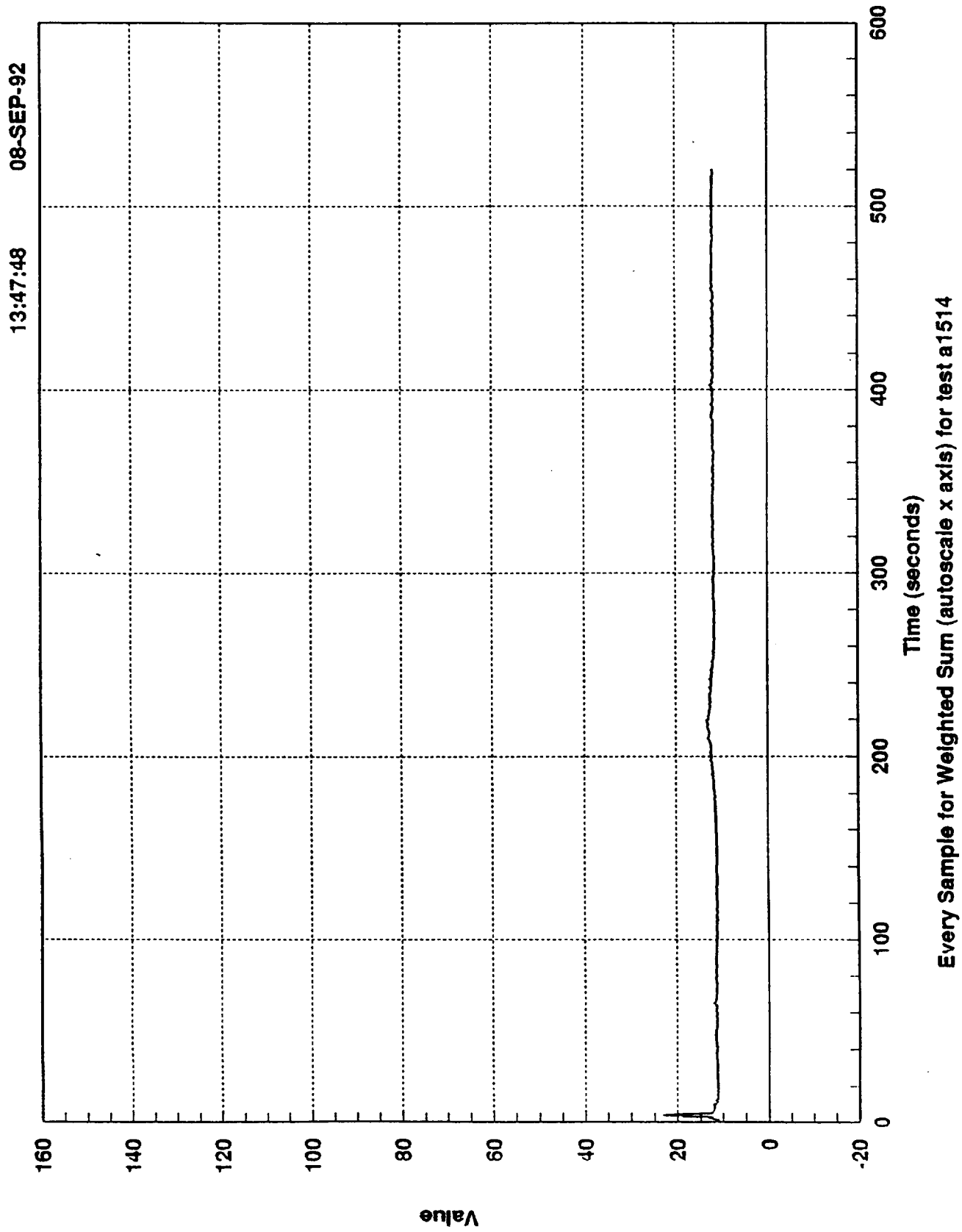


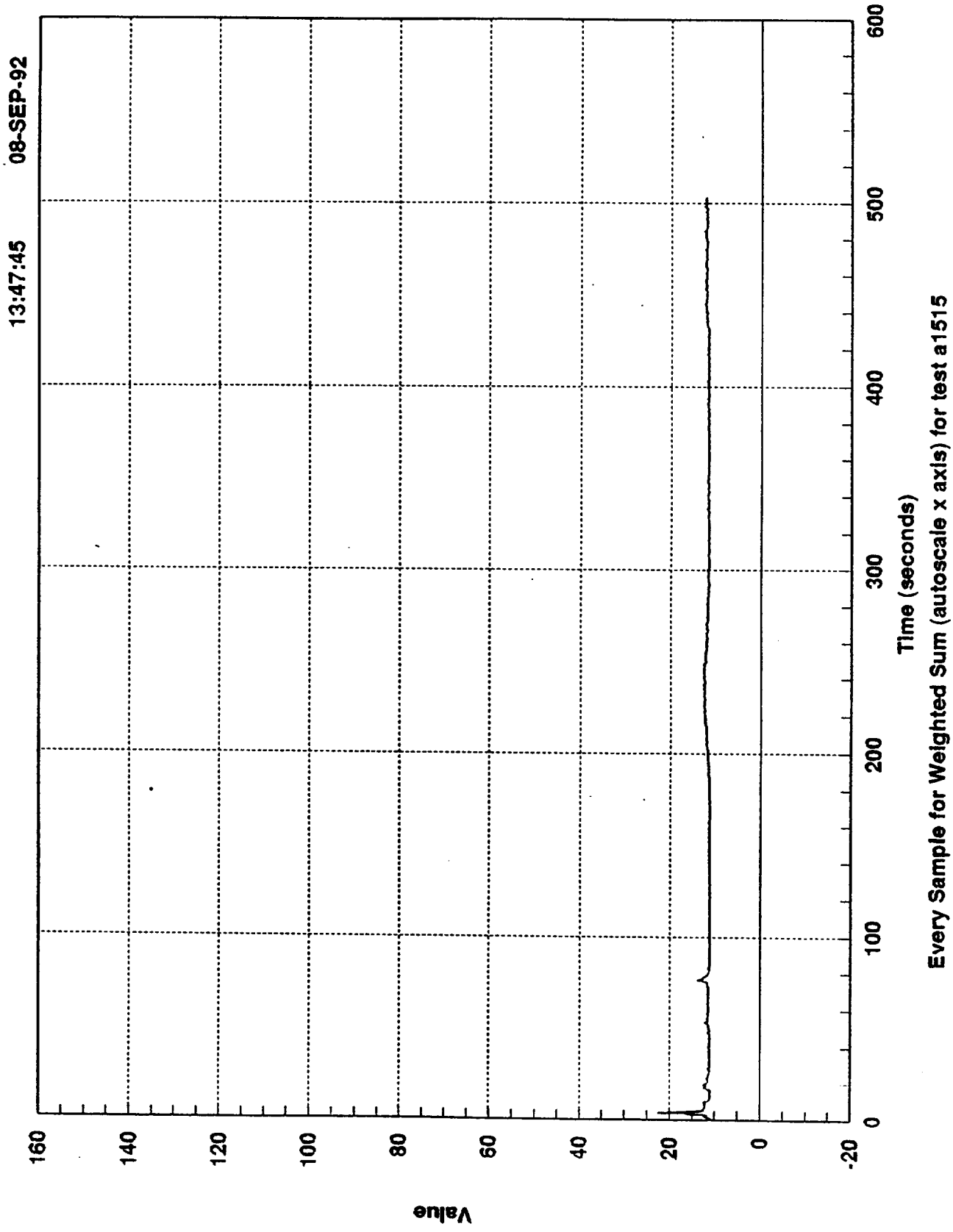
Every Sample for Weighted Sum (autoscale x axis) for test a1503

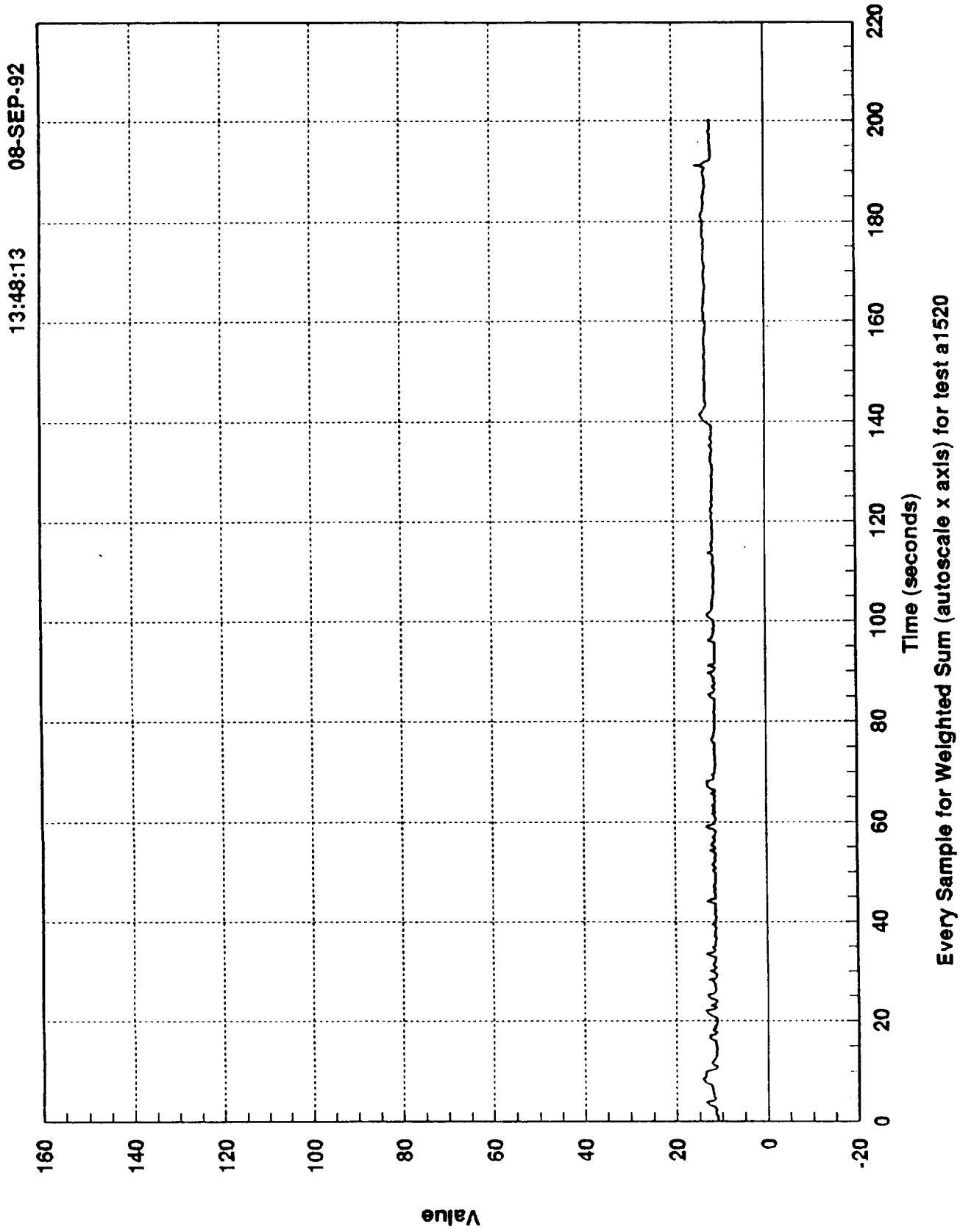

13:48:00
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1504

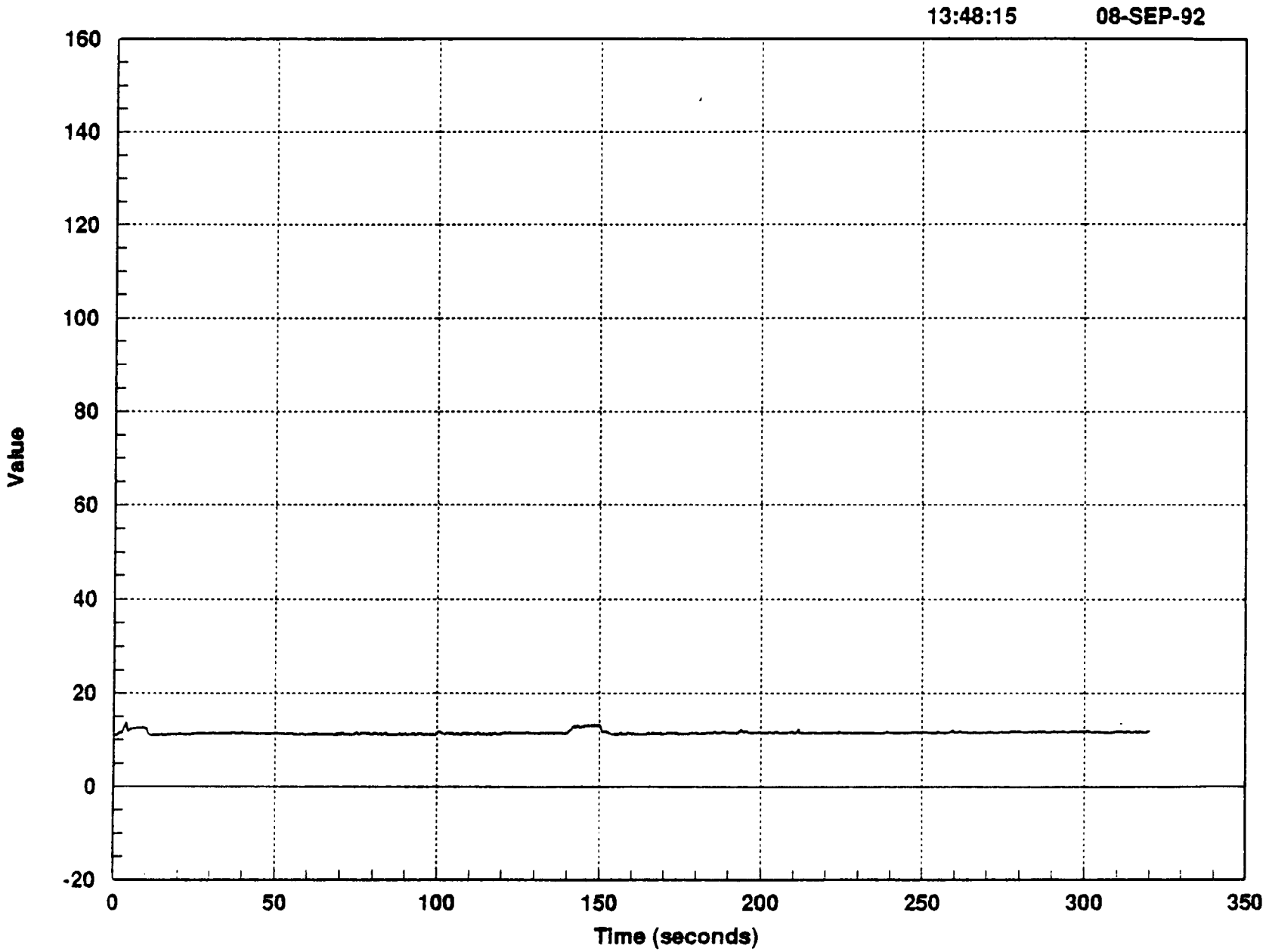


Every Sample for Weighted Sum (autoscale x axis) for test a1505

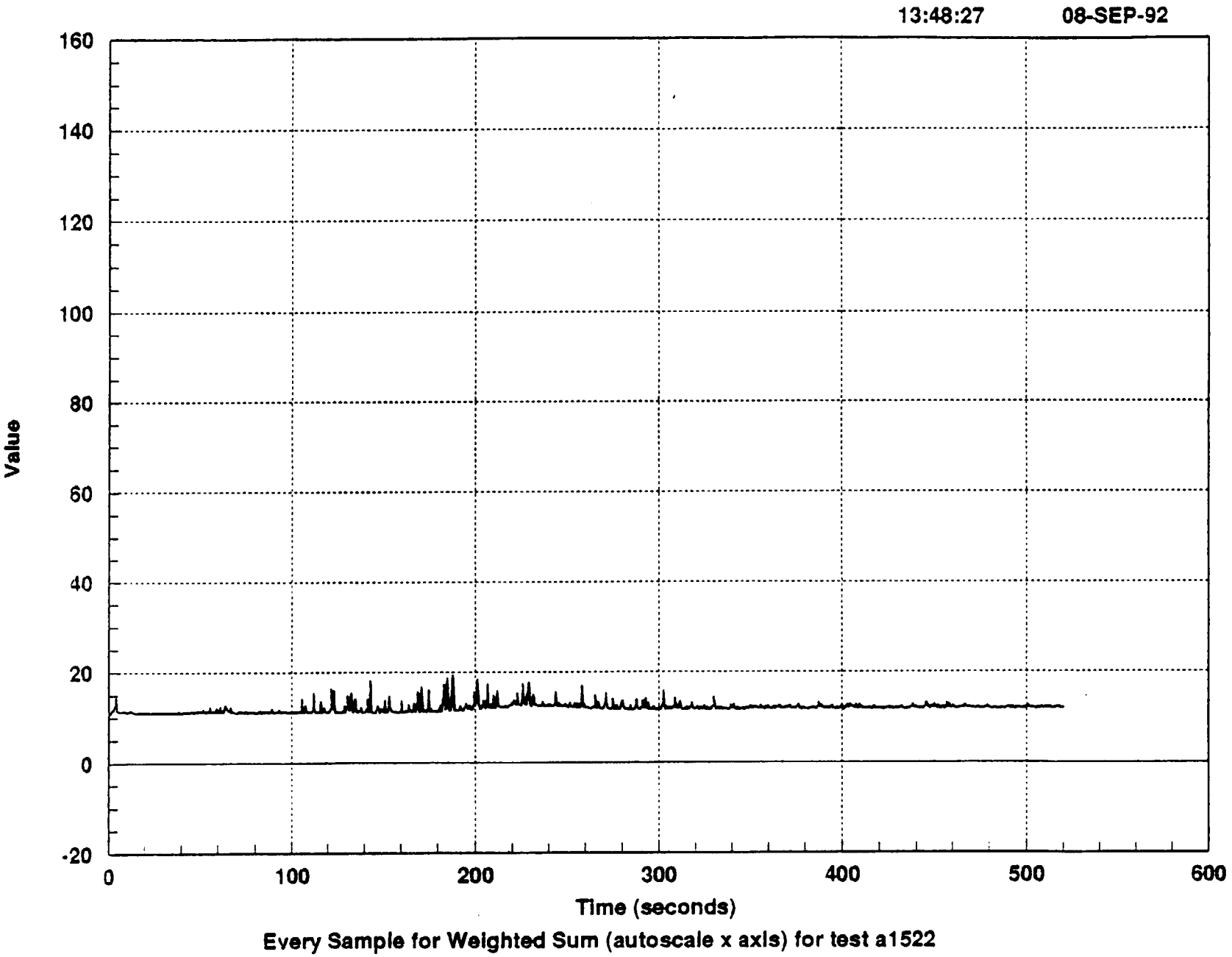

13:47:48
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1514


13:47:45 08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1515

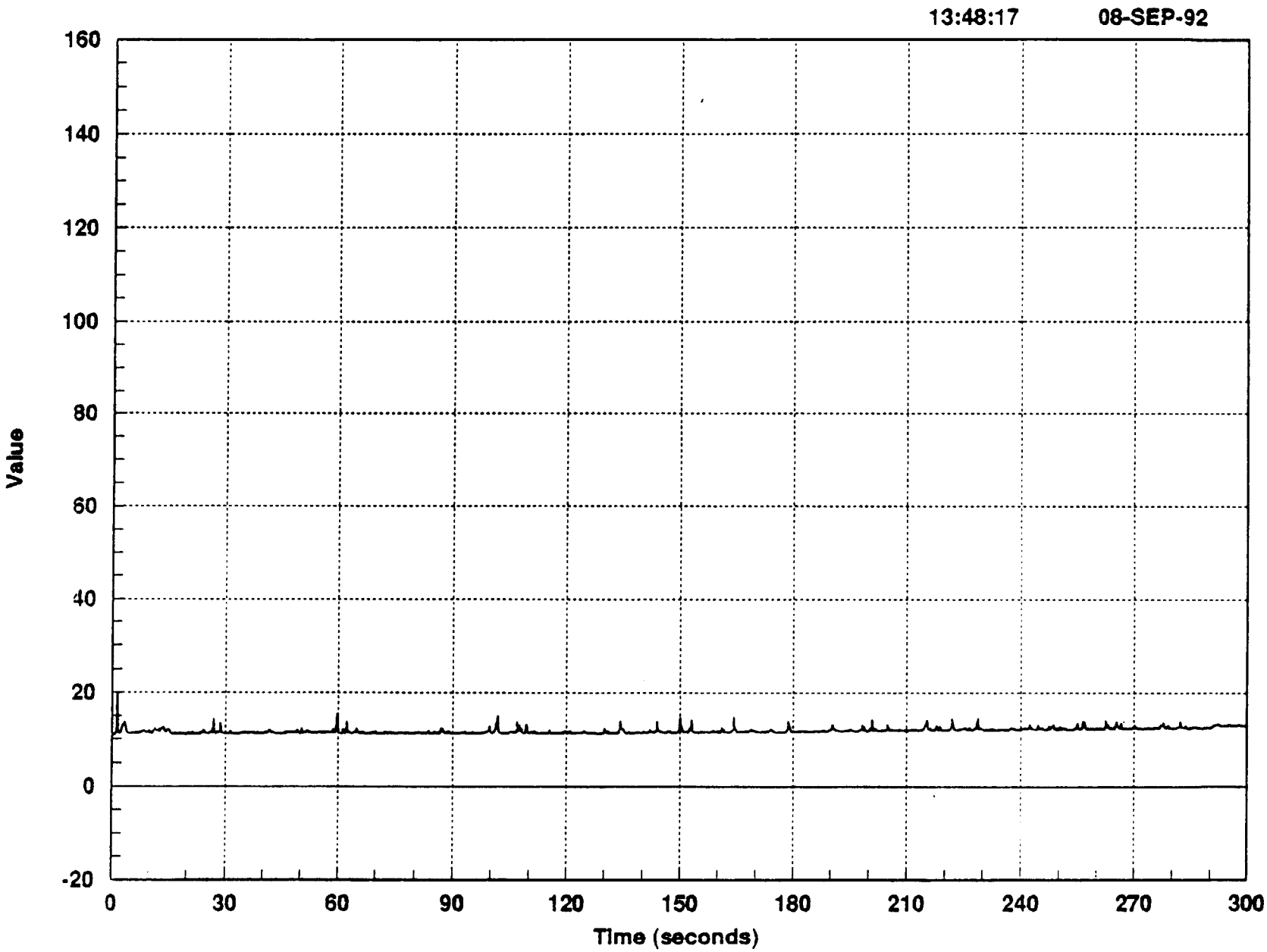

13:48:13
08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
20
40
60
80
100
120
140
160
180
200
220
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1520



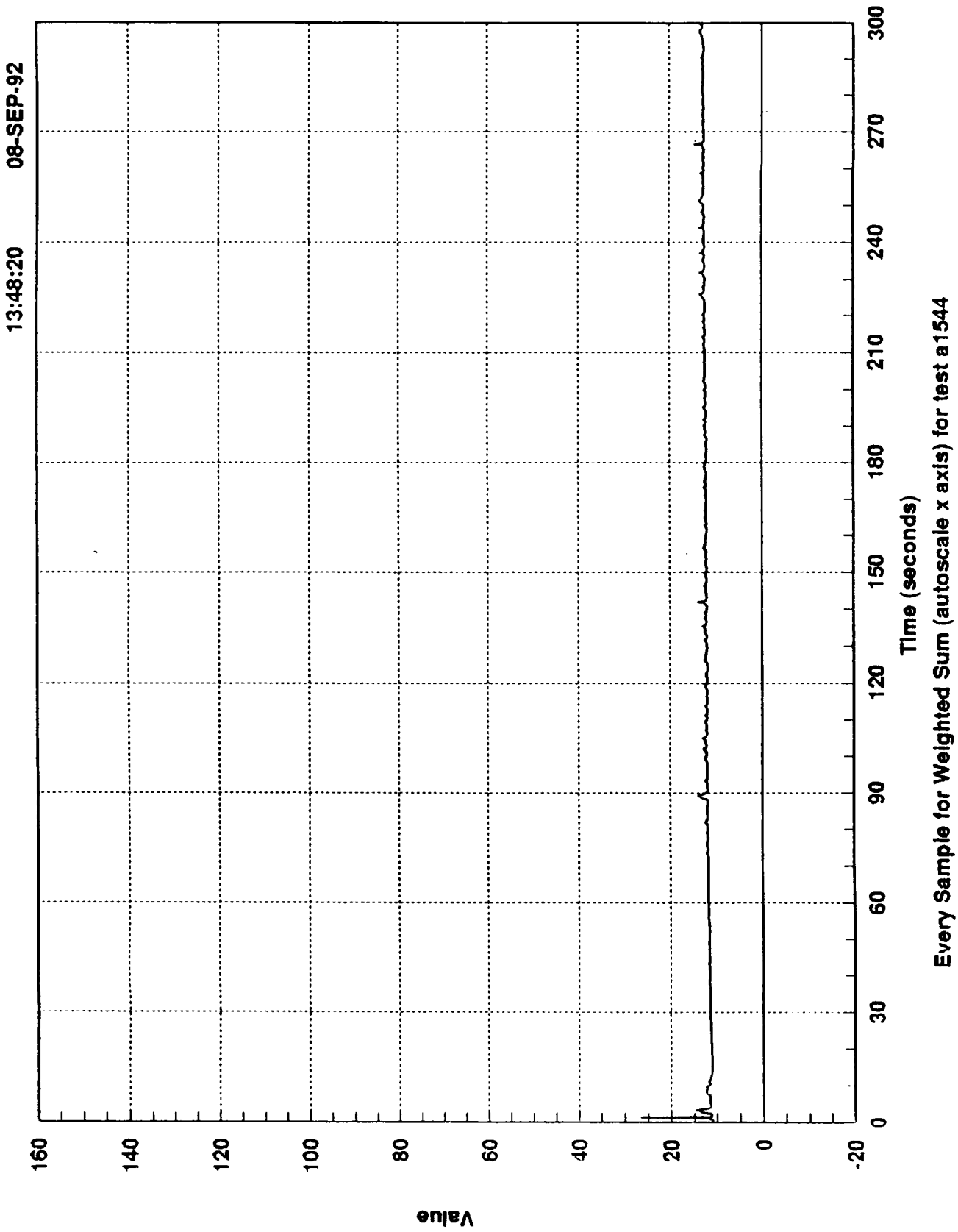
Every Sample for Weighted Sum (autoscale x axis) for test a1521

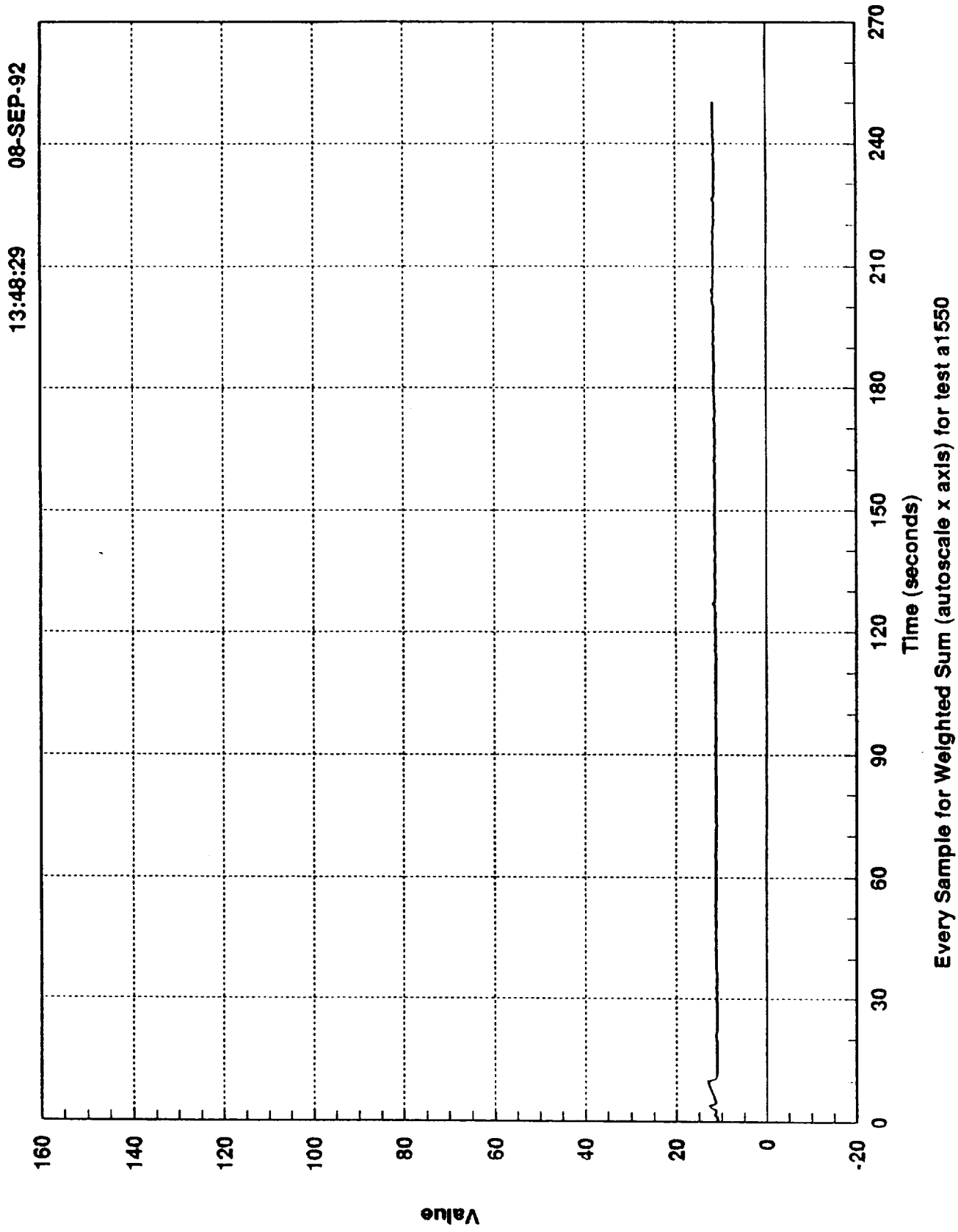


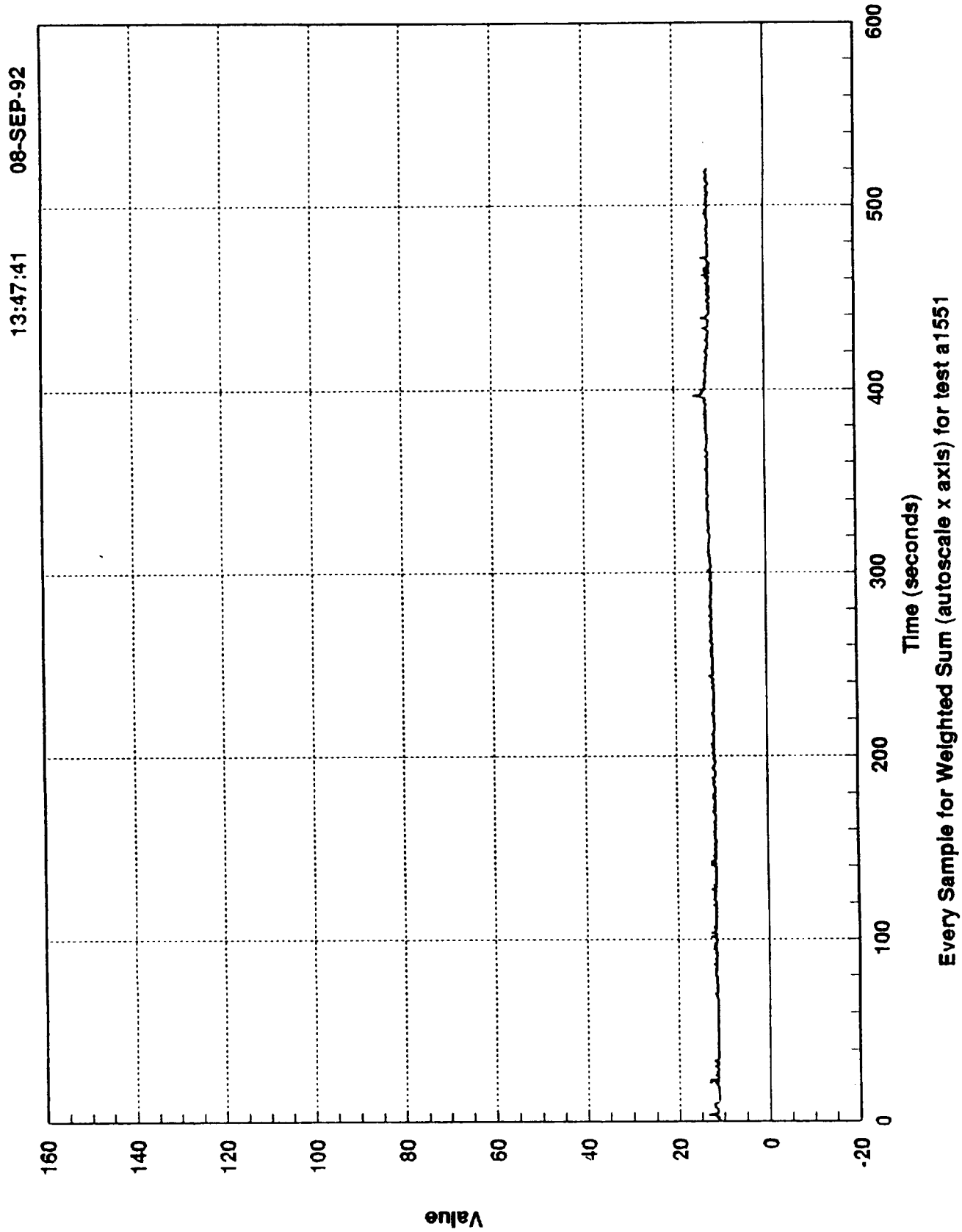
Every Sample for Weighted Sum (autoscale x axis) for test a1522

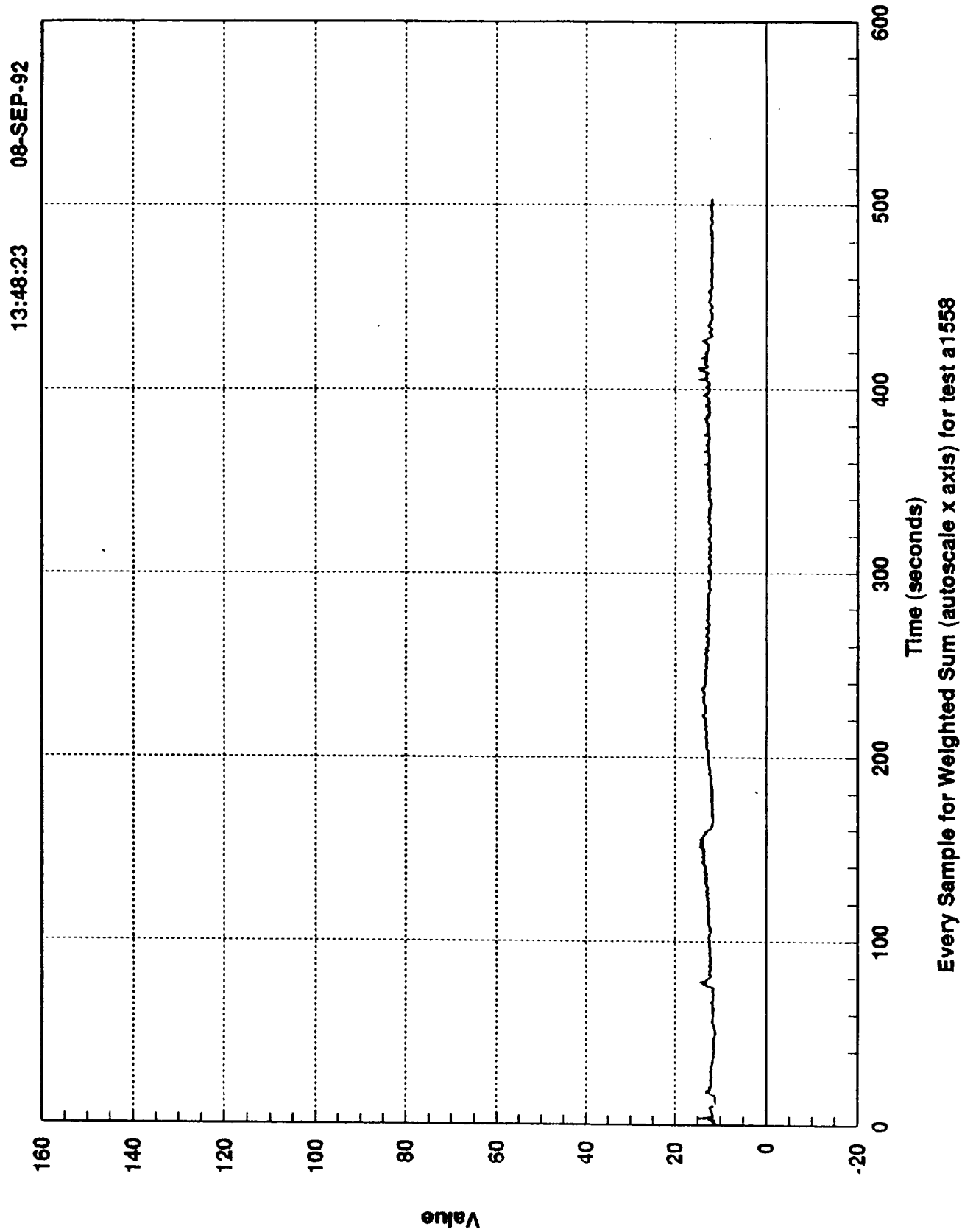


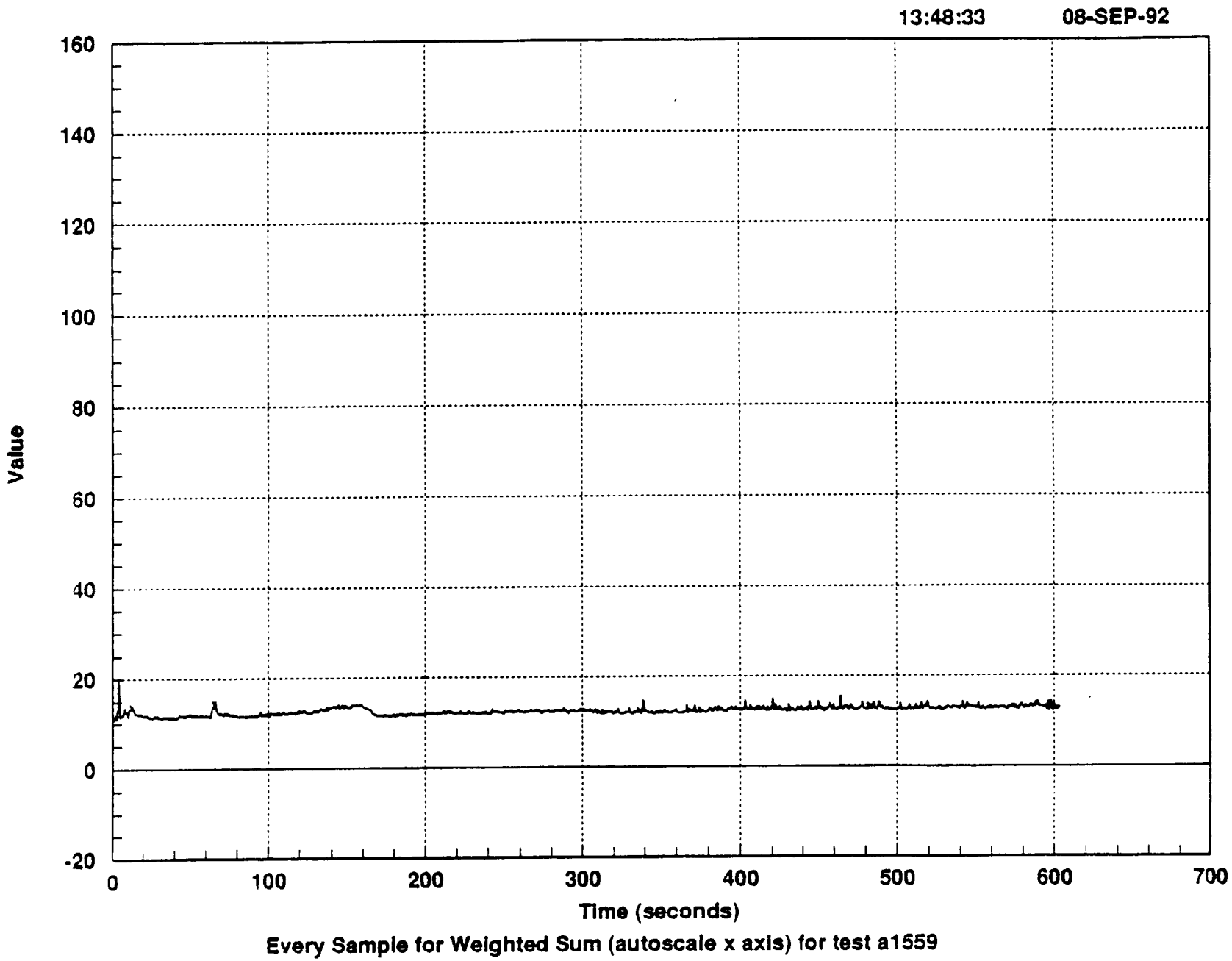
Every Sample for Weighted Sum (autoscale x axis) for test a1539


13:48:20
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
30
60
90
120
150
180
210
240
270
300
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1544

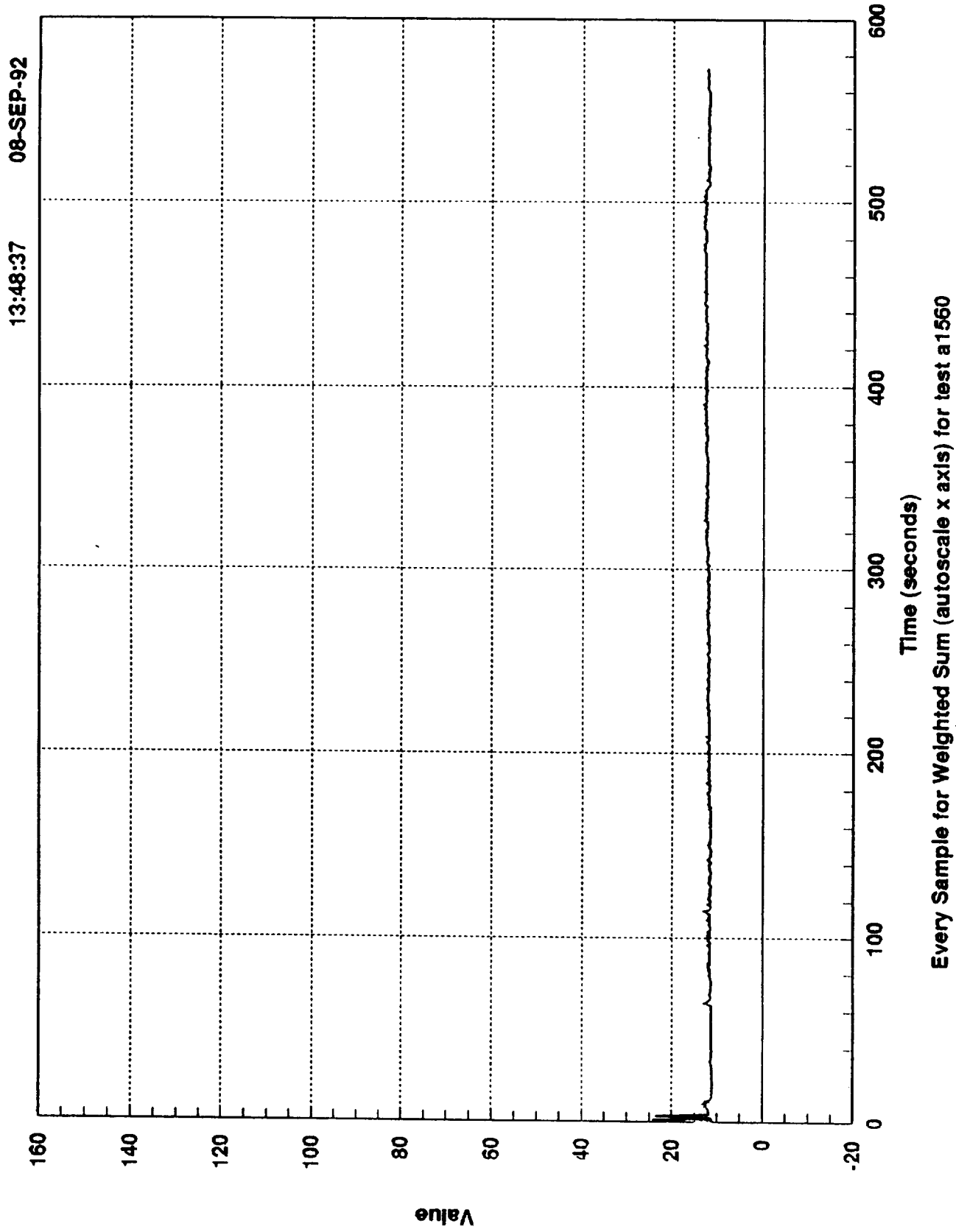

13:48:29 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
30
60
90
120
150
180
210
240
270
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1550

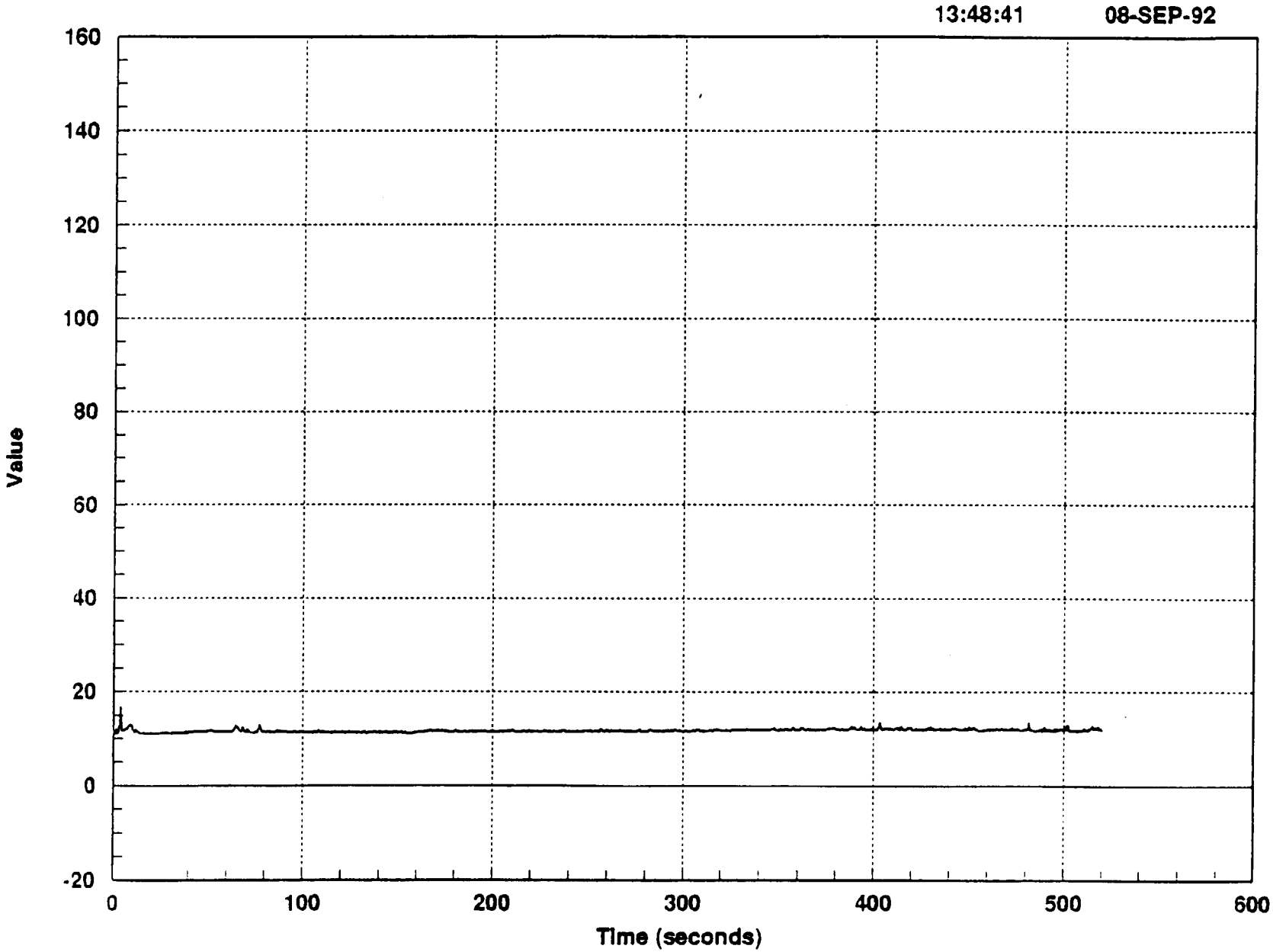

13:47:41
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1551


13:48:23
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1558

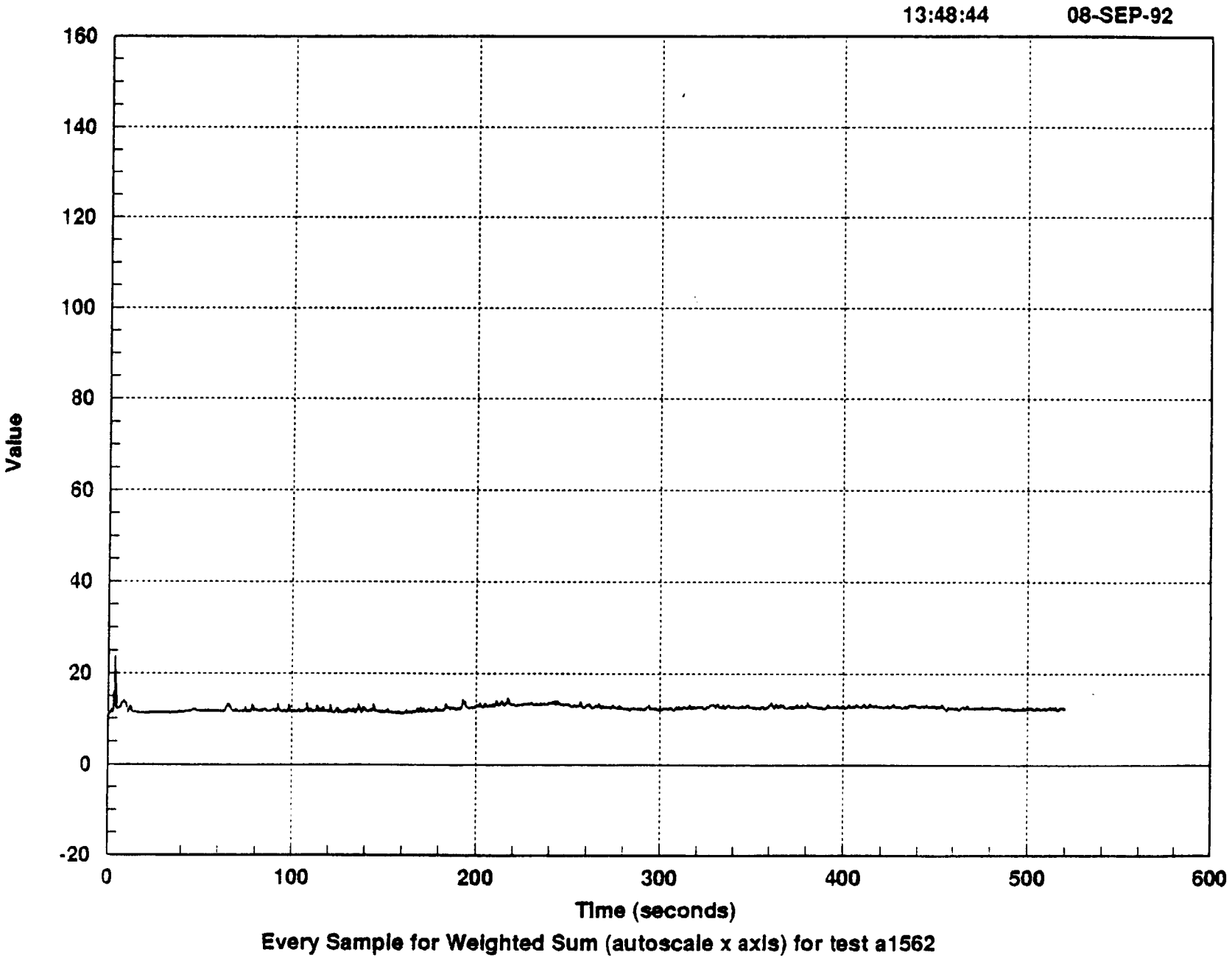


Every Sample for Weighted Sum (autoscale x axis) for test a1559

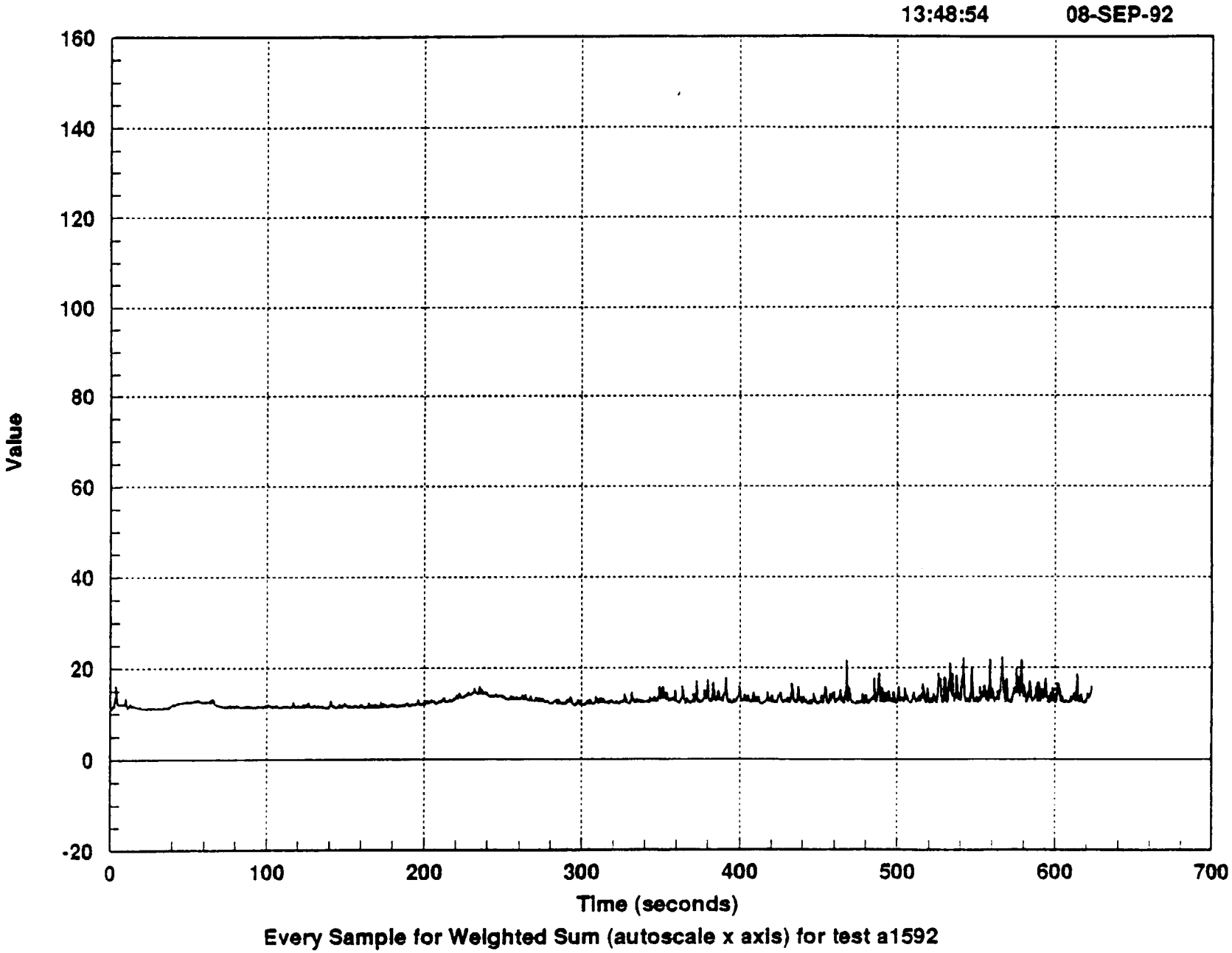

13:48:37 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1560



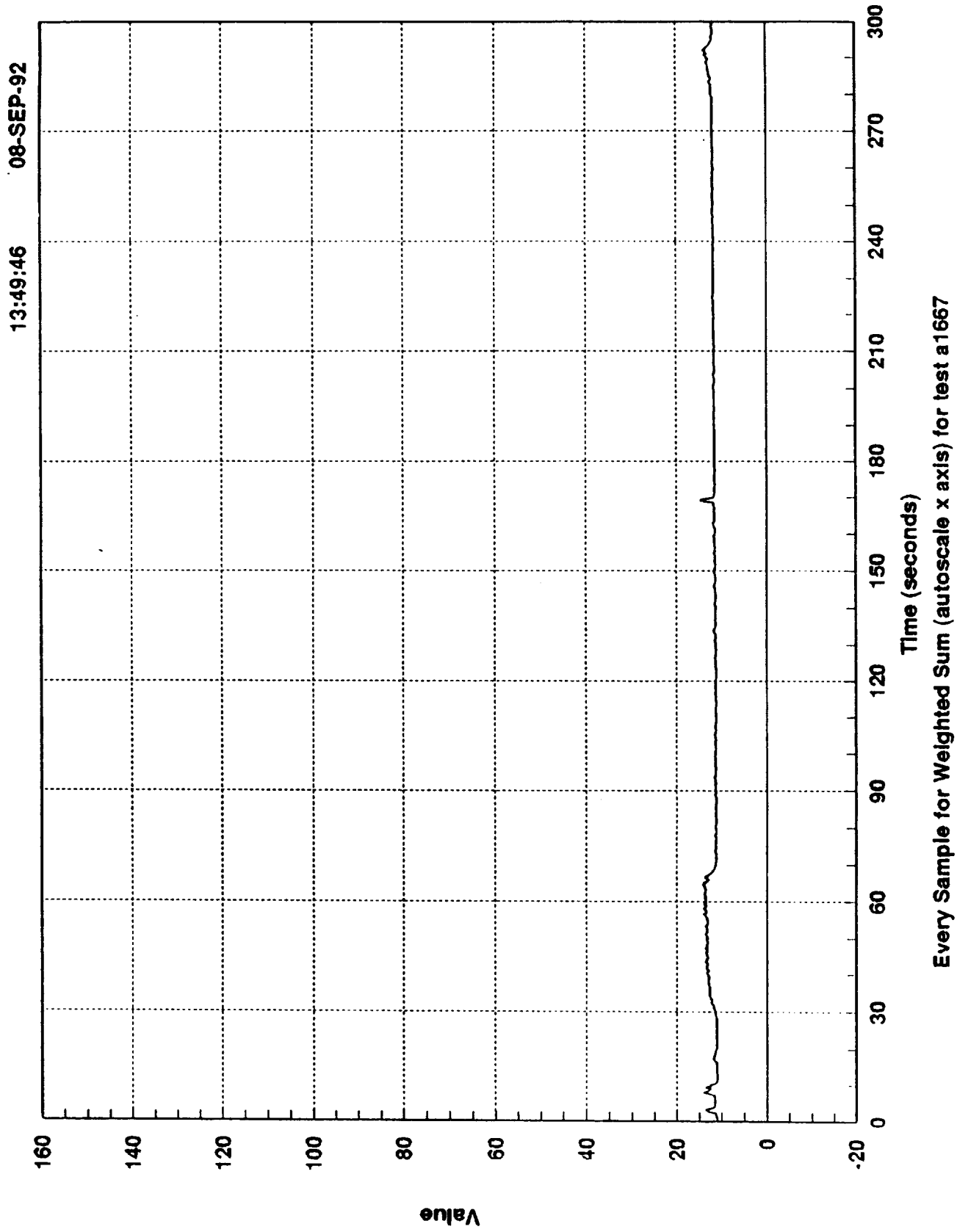
Every Sample for Weighted Sum (autoscale x axis) for test a1561

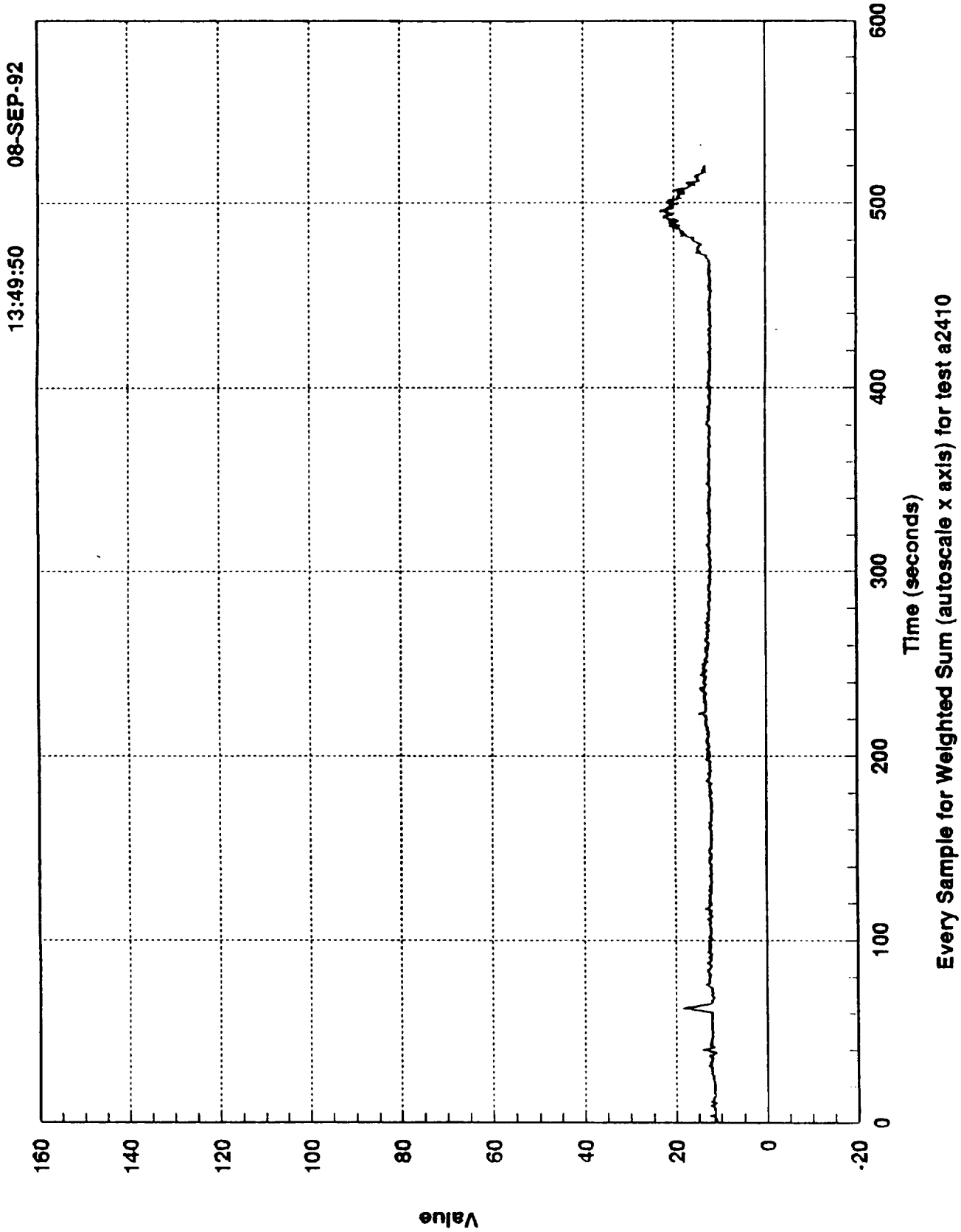


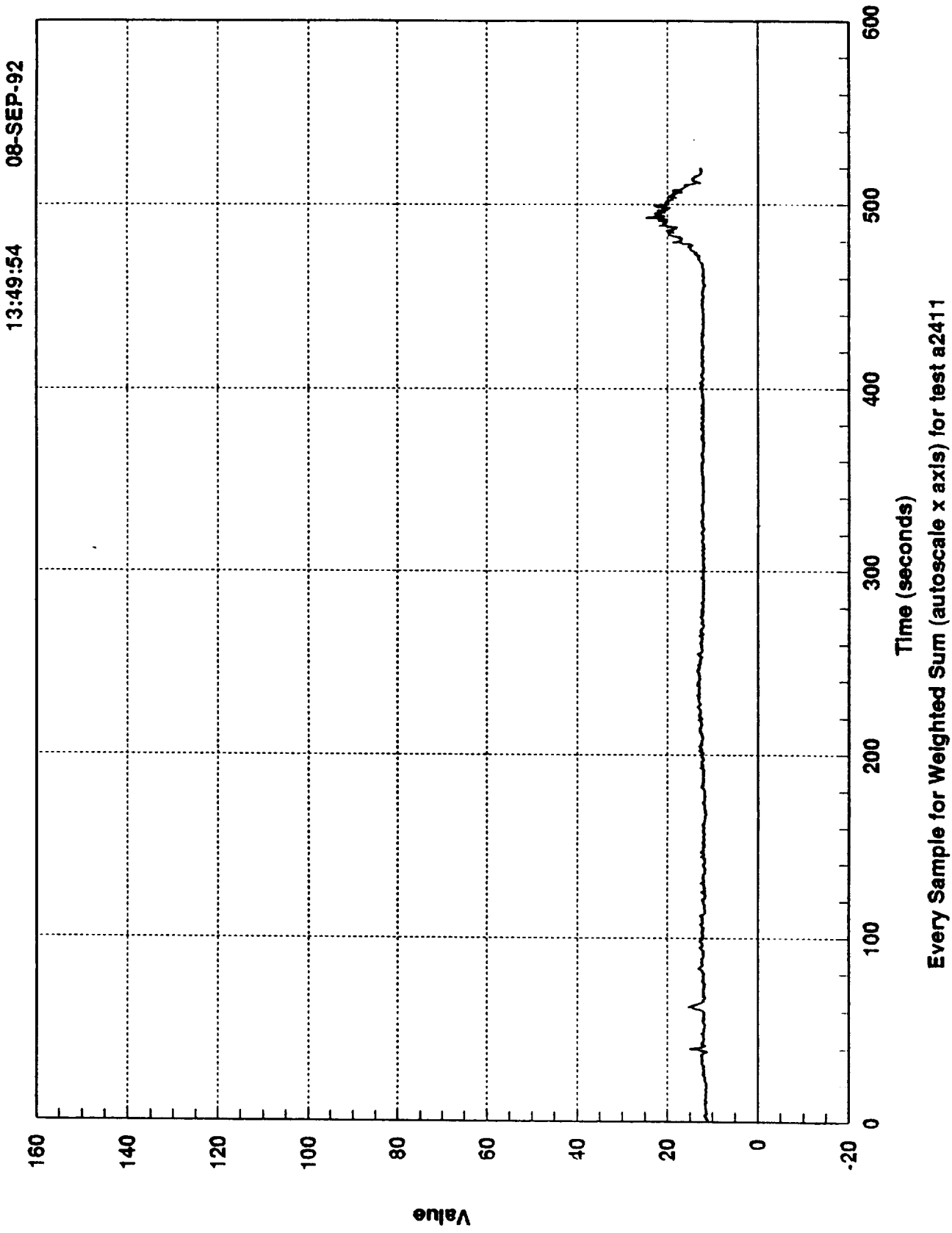
Every Sample for Weighted Sum (autoscale x axis) for test a1562

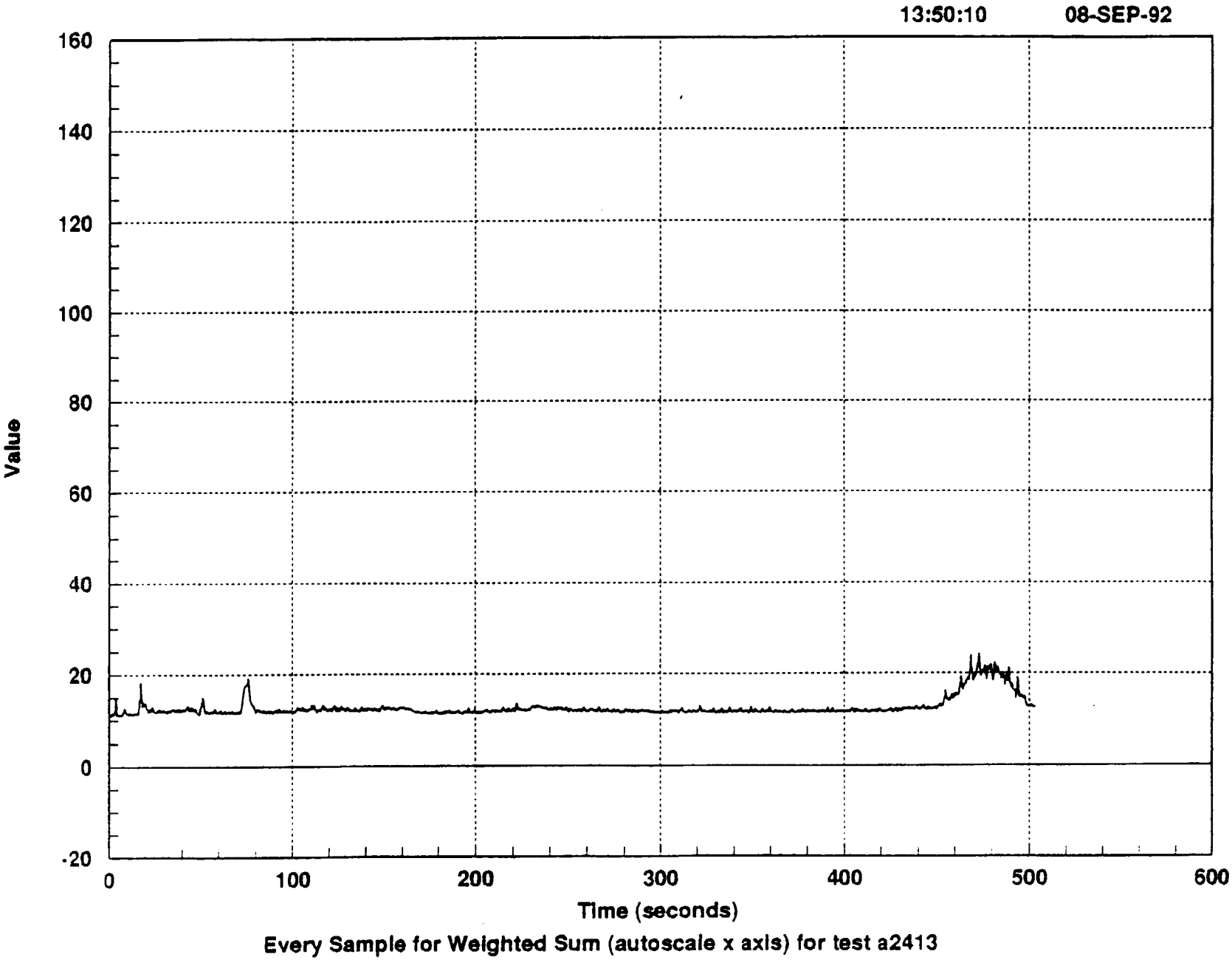


Every Sample for Weighted Sum (autoscale x axis) for test a1592

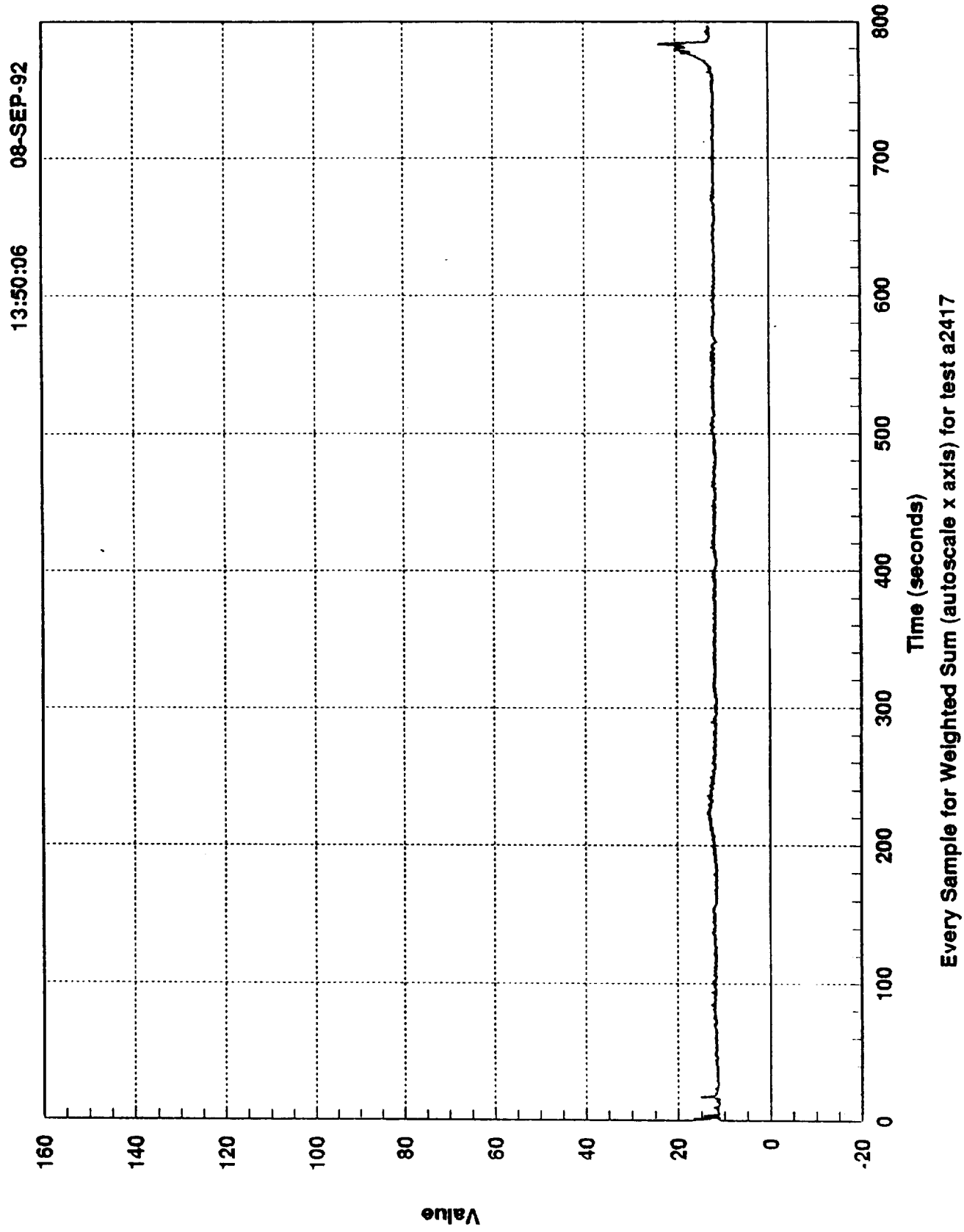

13:49:46 08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
30
60
90
120
150
180
210
240
270
300
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a1667

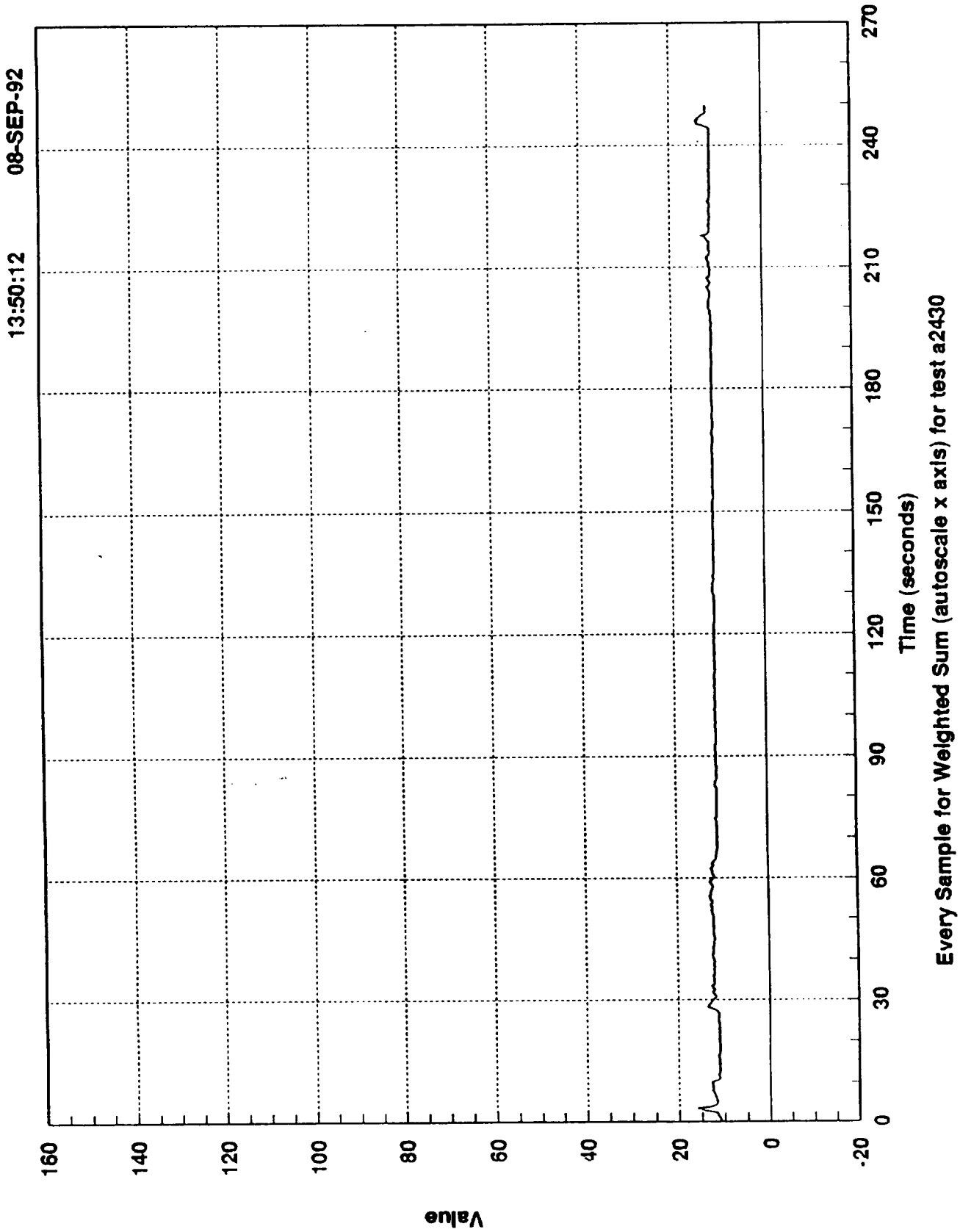

13:49:50 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2410

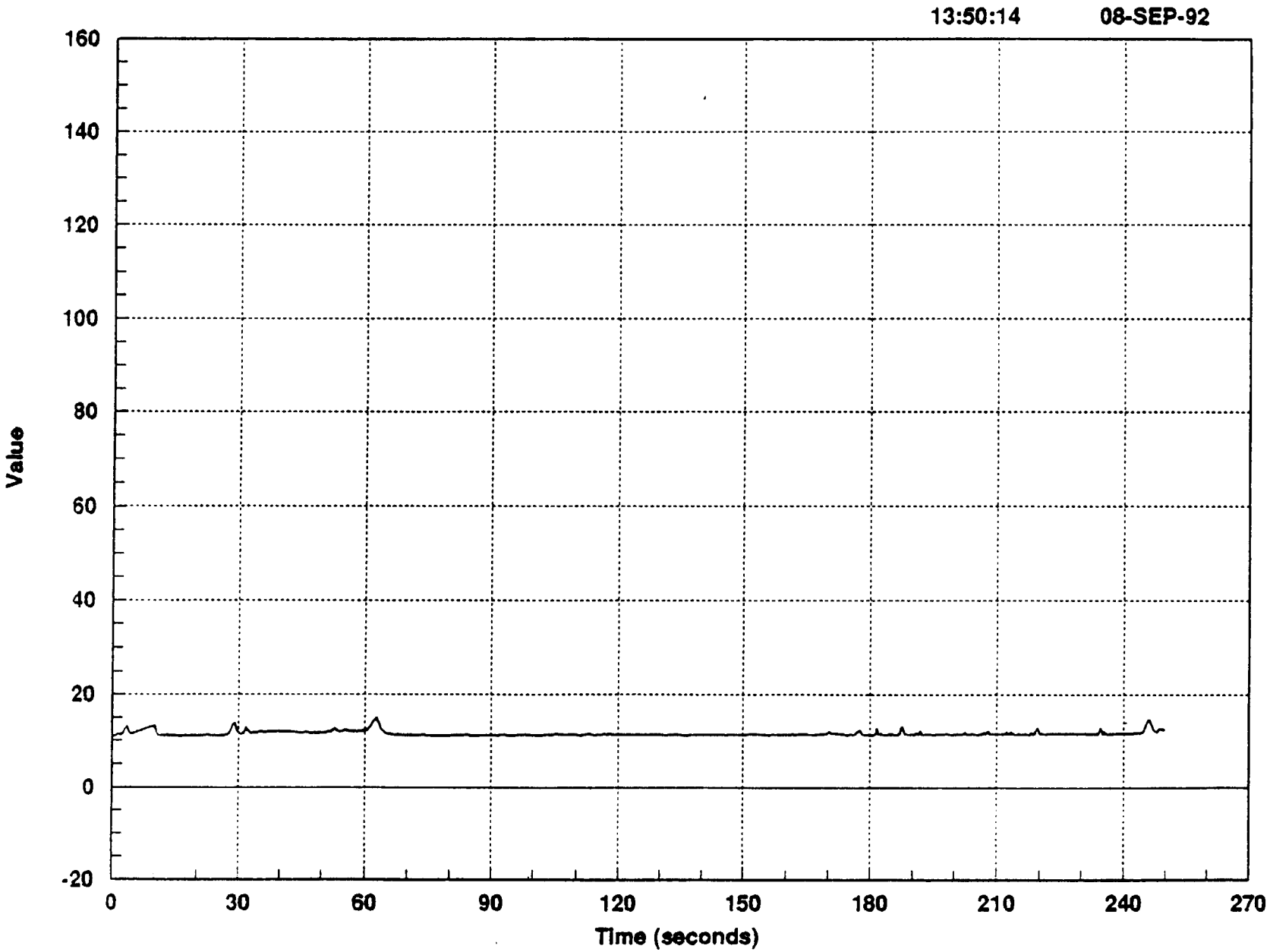

13:49:54 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2411



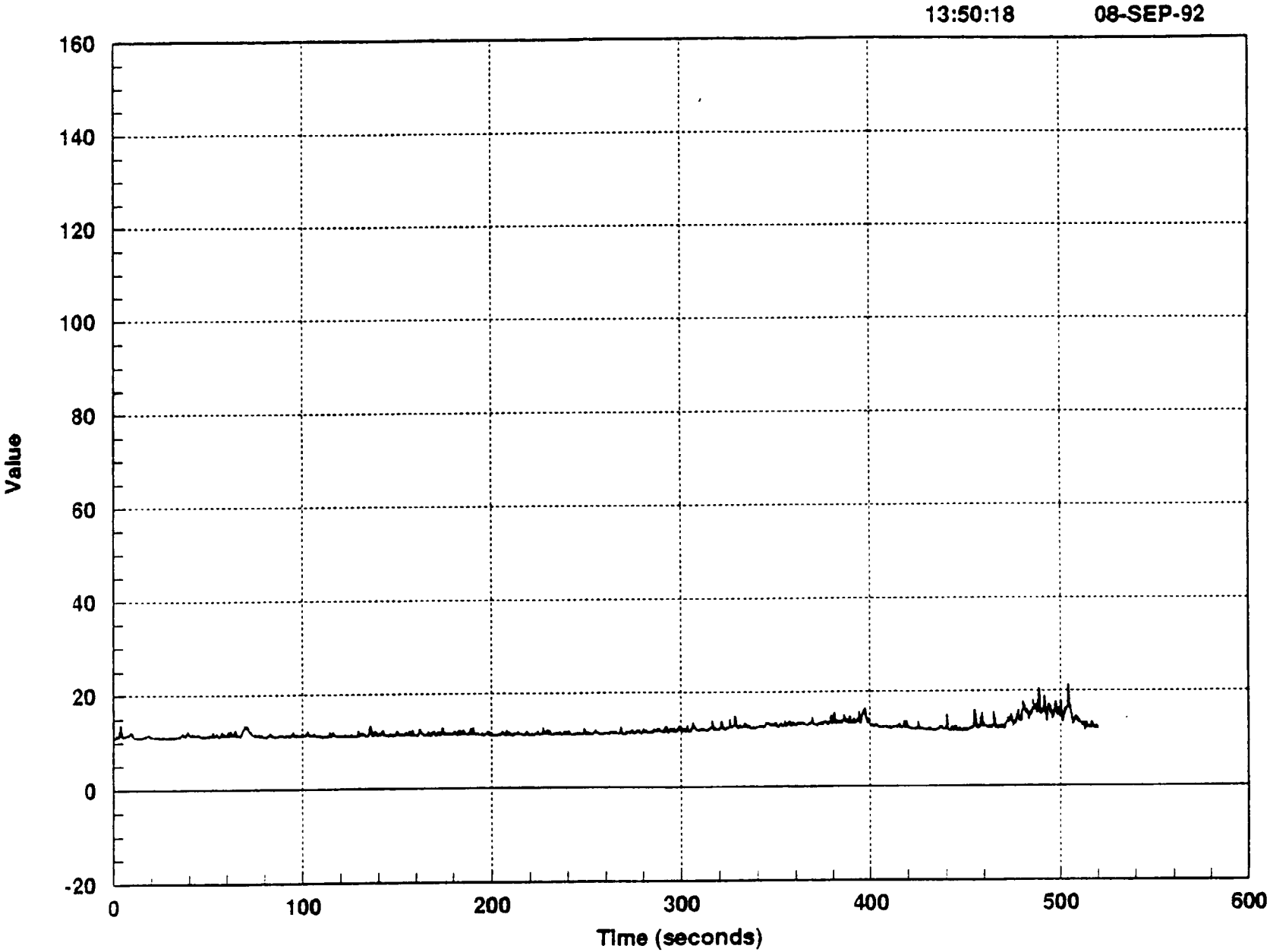
Every Sample for Weighted Sum (autoscale x axis) for test a2413


13:50:06 08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
700
800
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2417

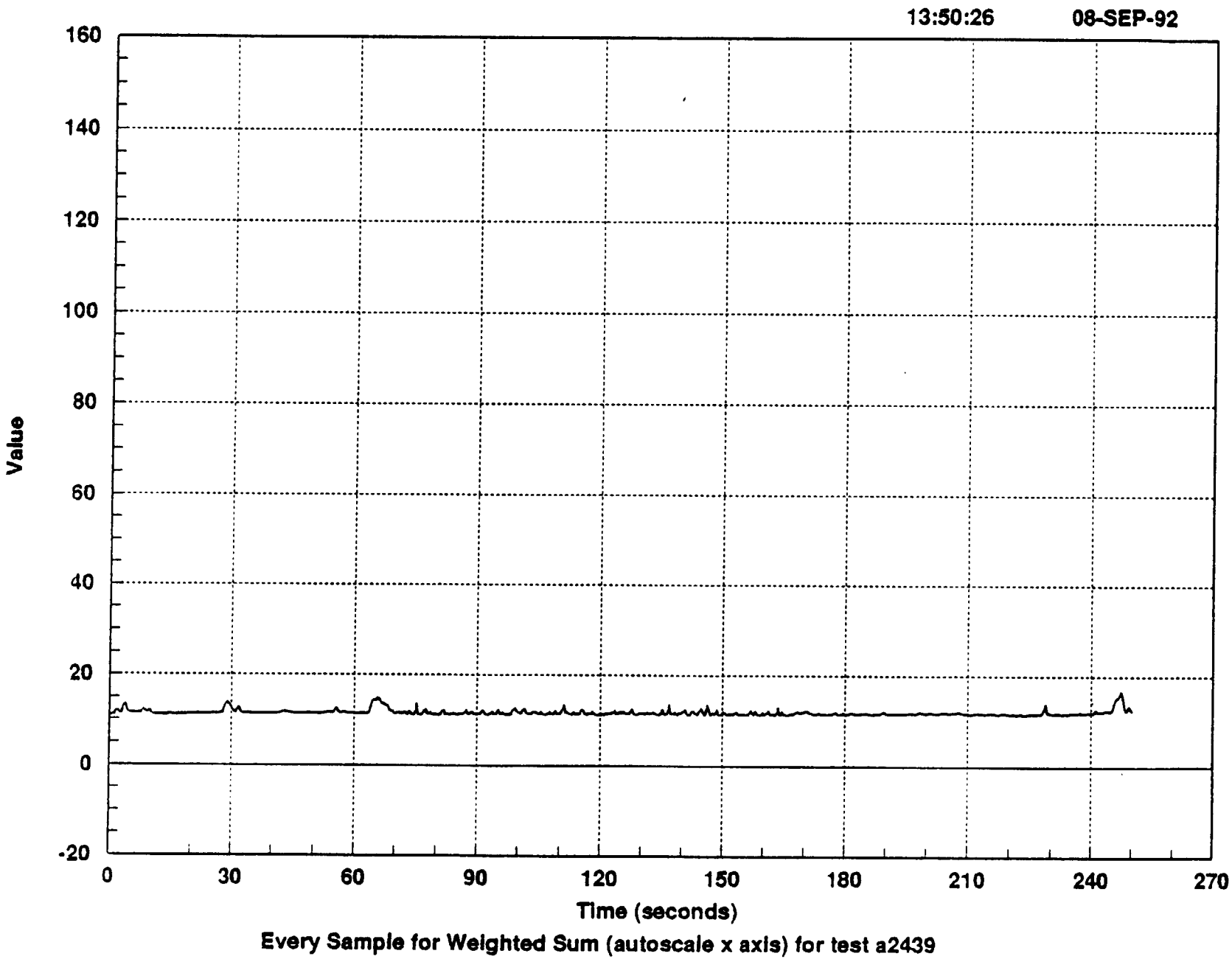

13:50:12
08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
30
60
90
120
150
180
210
240
270
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2430



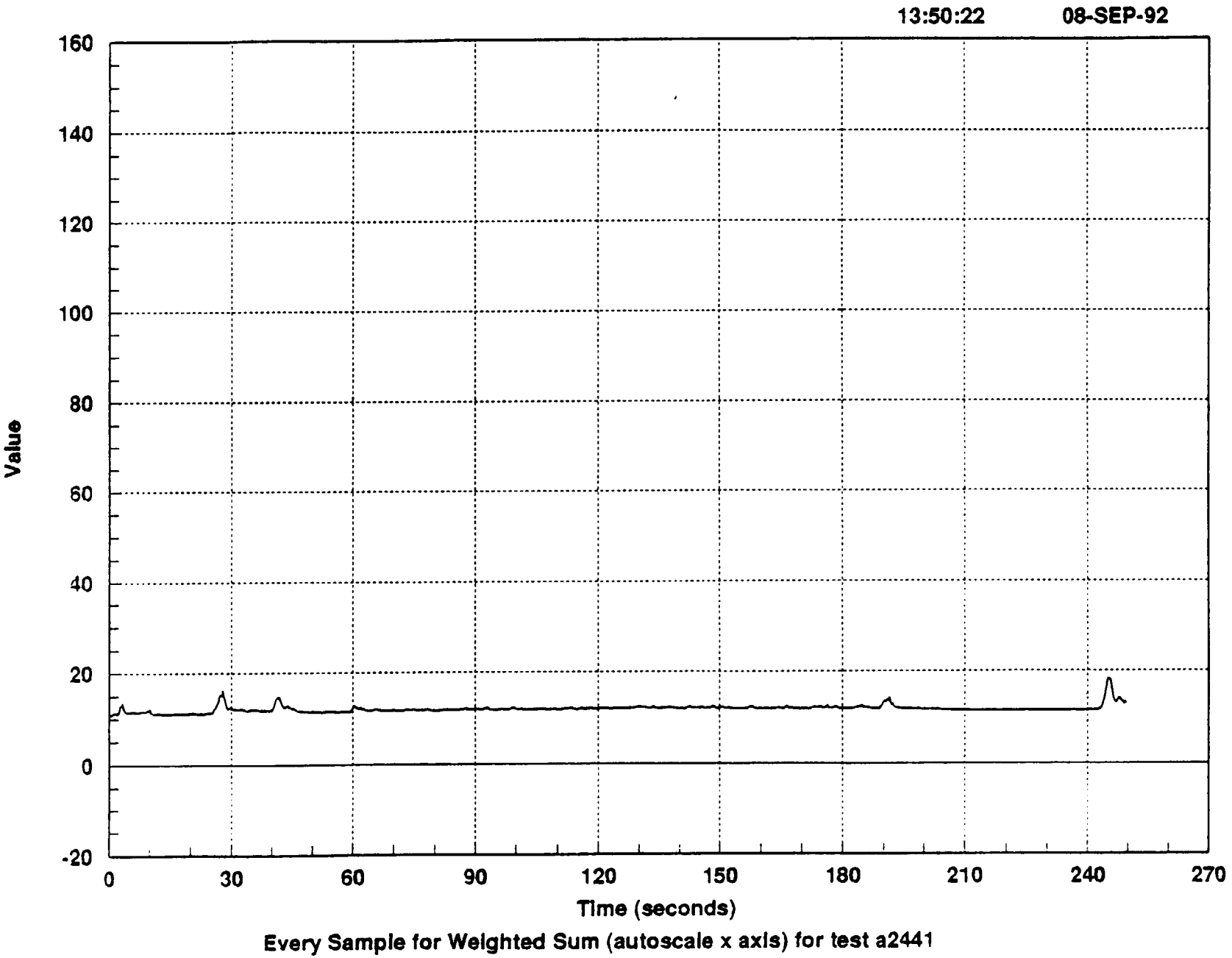
Every Sample for Weighted Sum (autoscale x axis) for test a2436



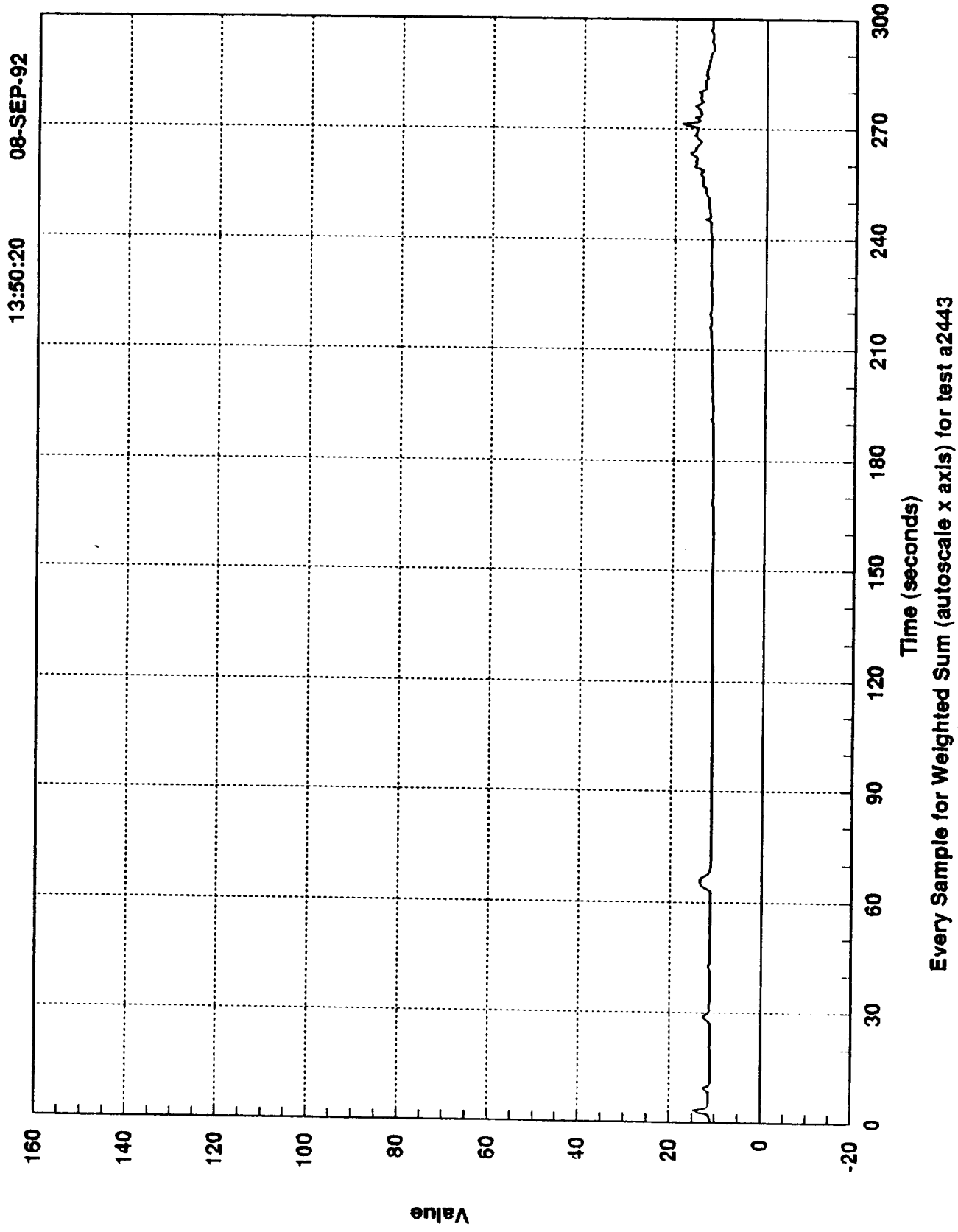
Every Sample for Weighted Sum (autoscale x axis) for test a2437

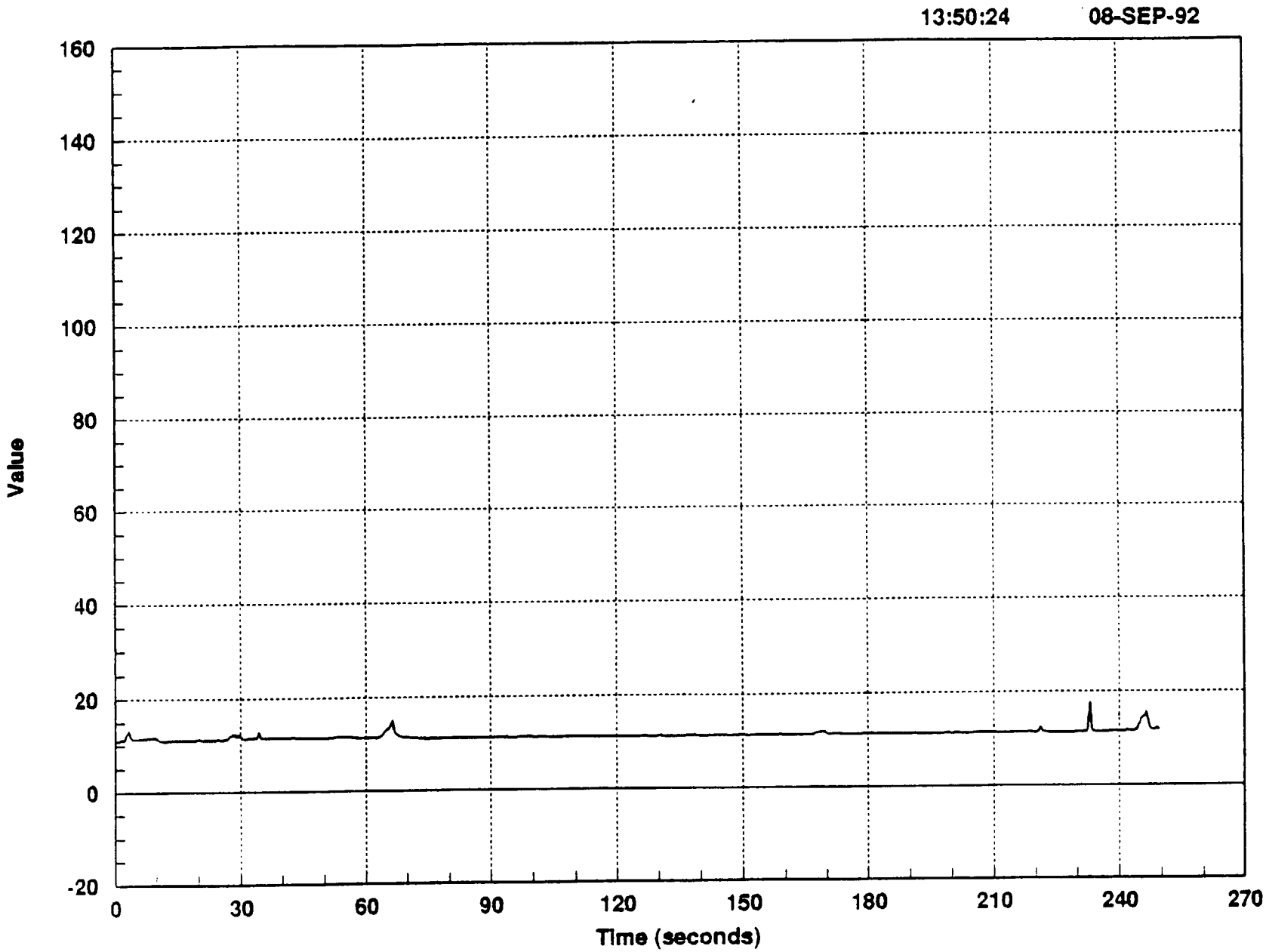


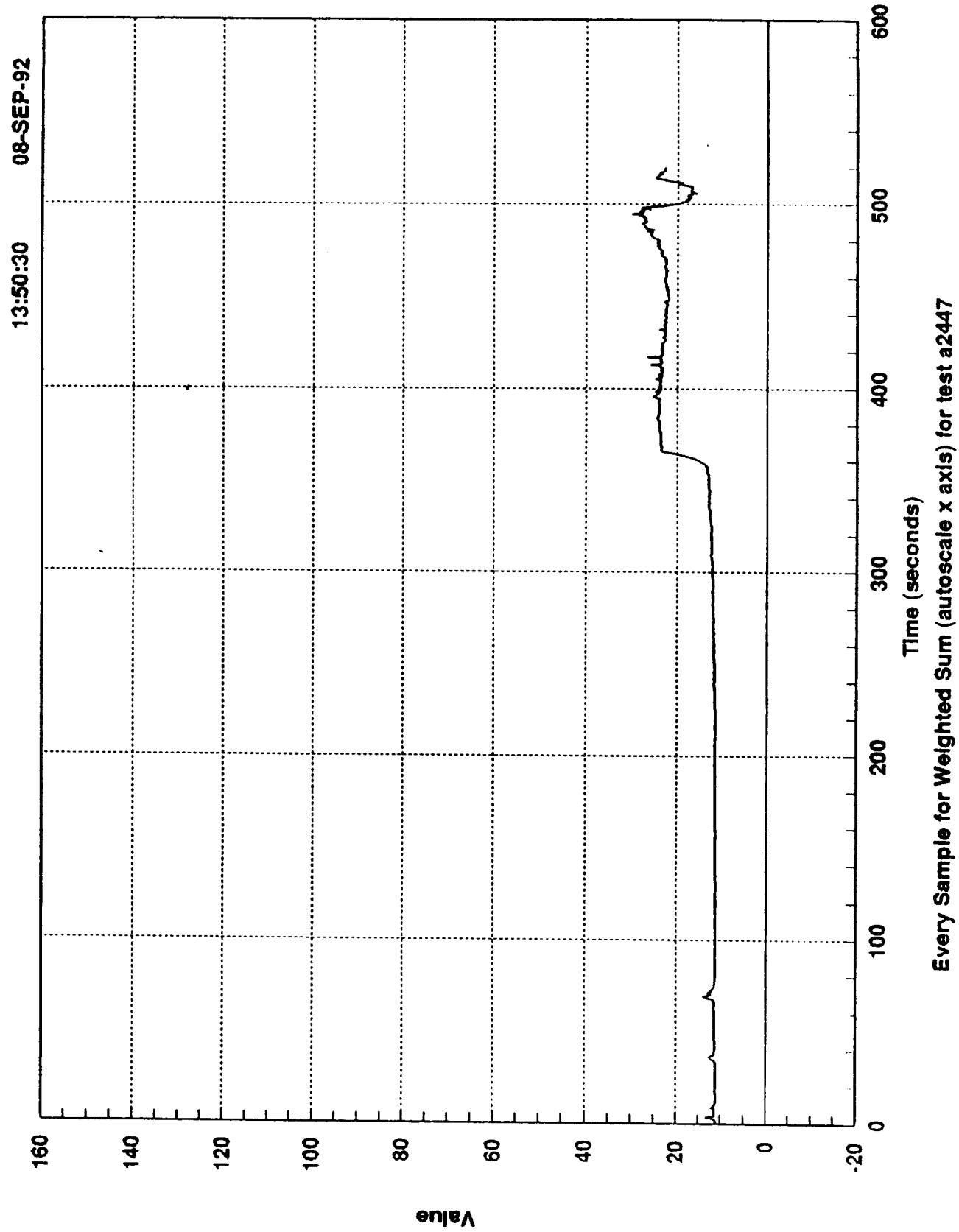
Every Sample for Weighted Sum (autoscale x axis) for test a2439

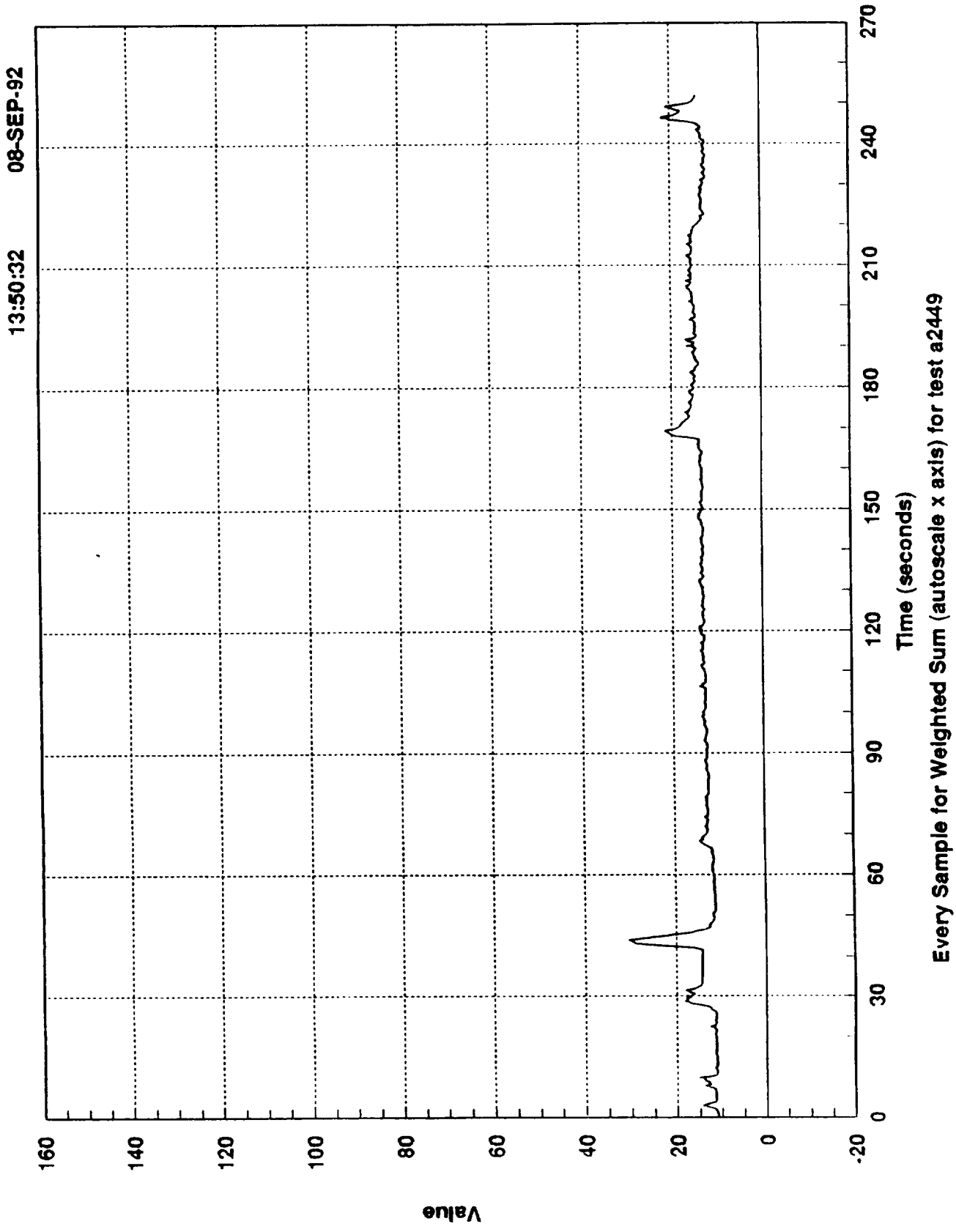


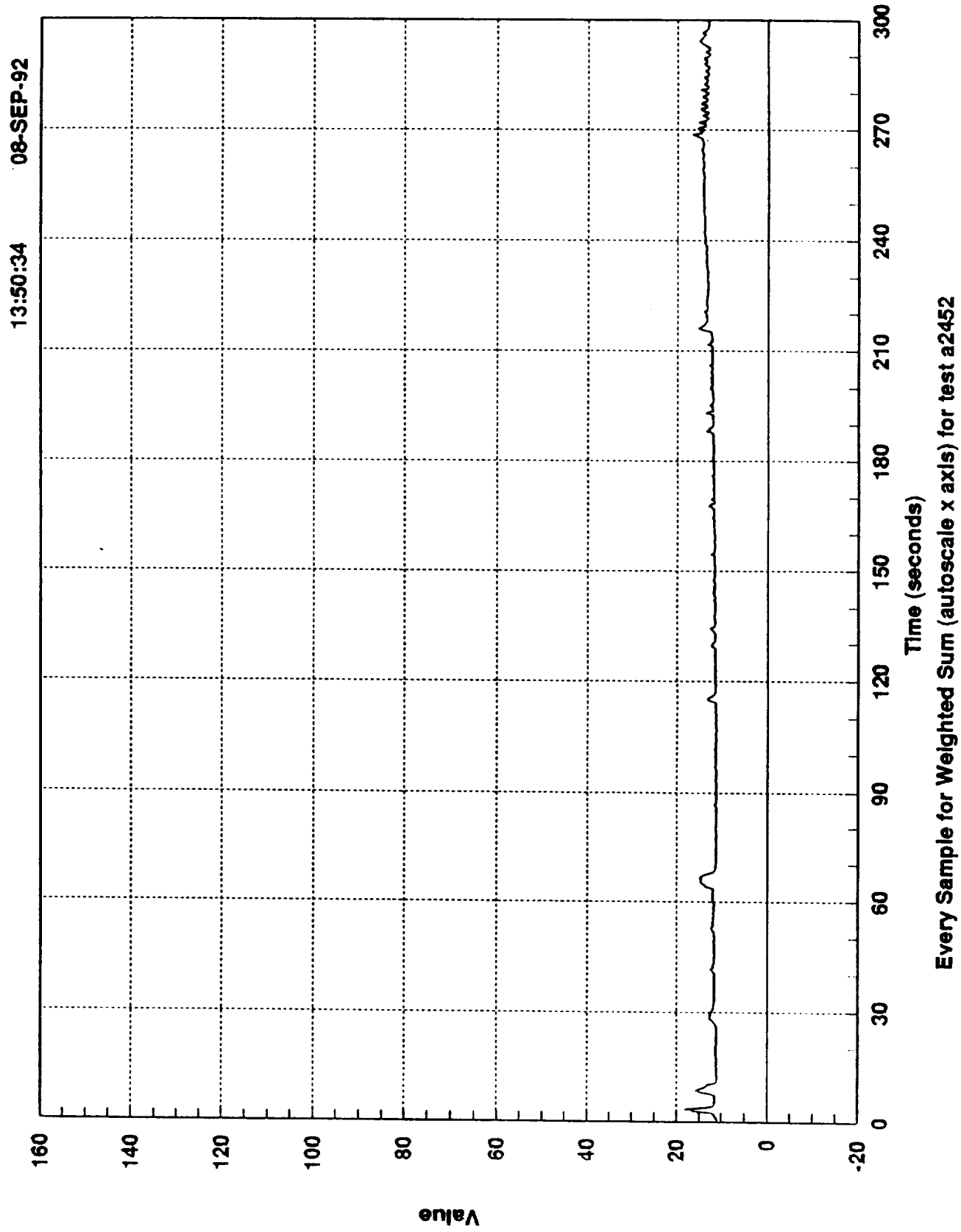
Every Sample for Weighted Sum (autoscale x axis) for test a2441


13:50:20 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
30
60
90
120
150
180
210
240
270
300
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2443

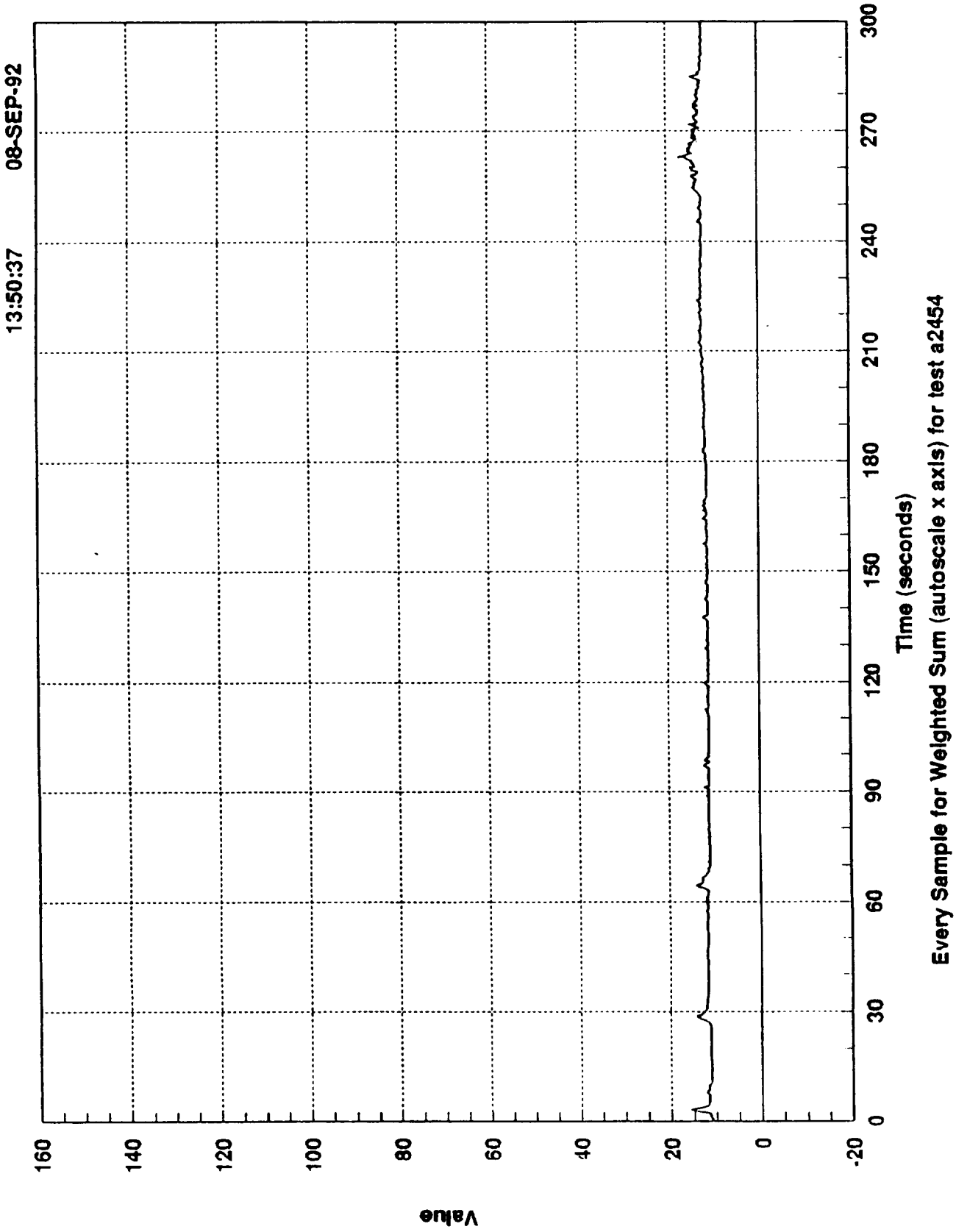

13:50:24
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
30
60
90
120
150
180
210
240
270
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2446

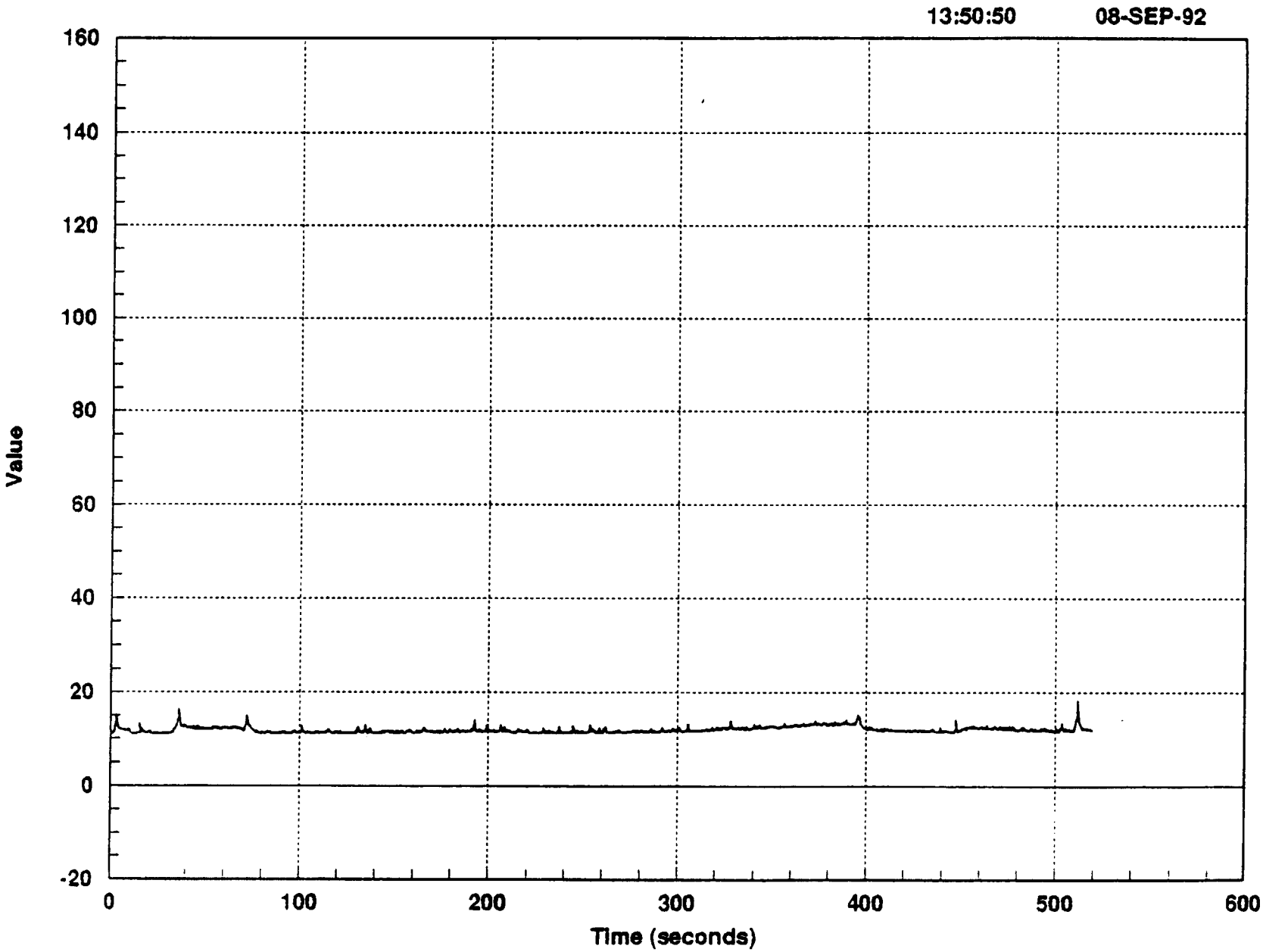

13:50:30 08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2447


13:50:32 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
30
60
90
120
150
180
210
240
270
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2449

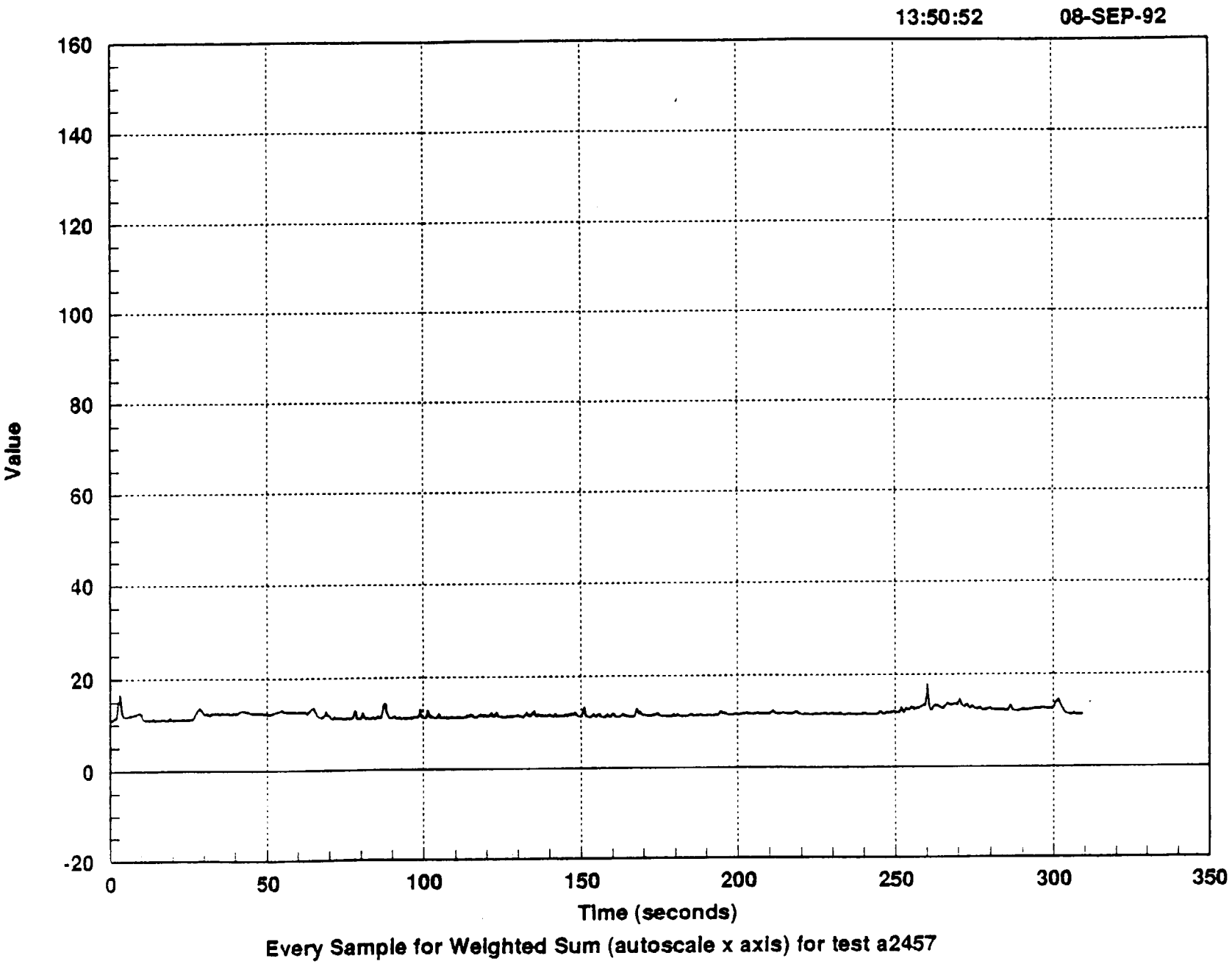

13:50:34
08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
30
60
90
120
150
180
210
240
270
300
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2452

C-2.

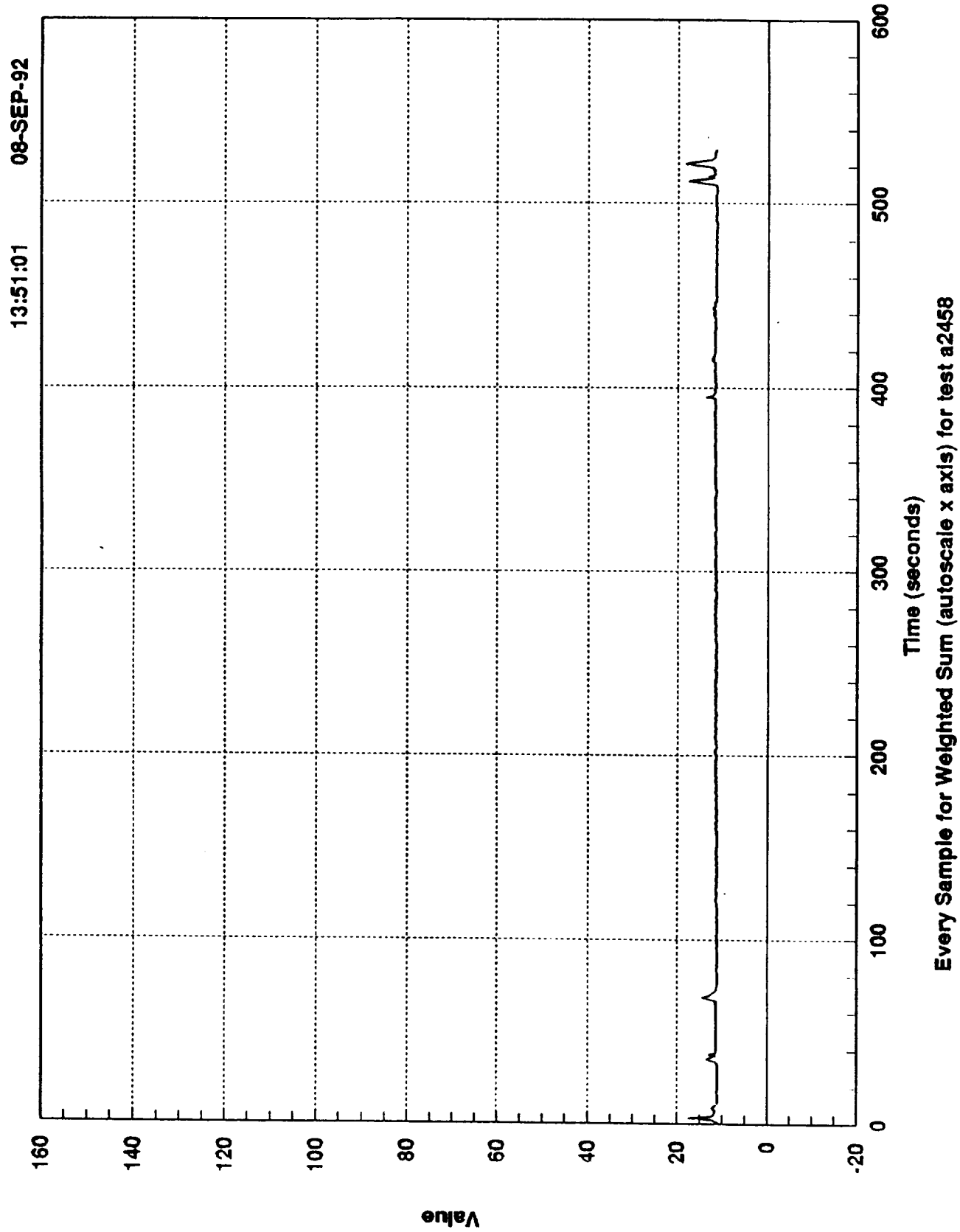

13:50:37
08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
30
60
90
120
150
180
210
240
270
300
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2454

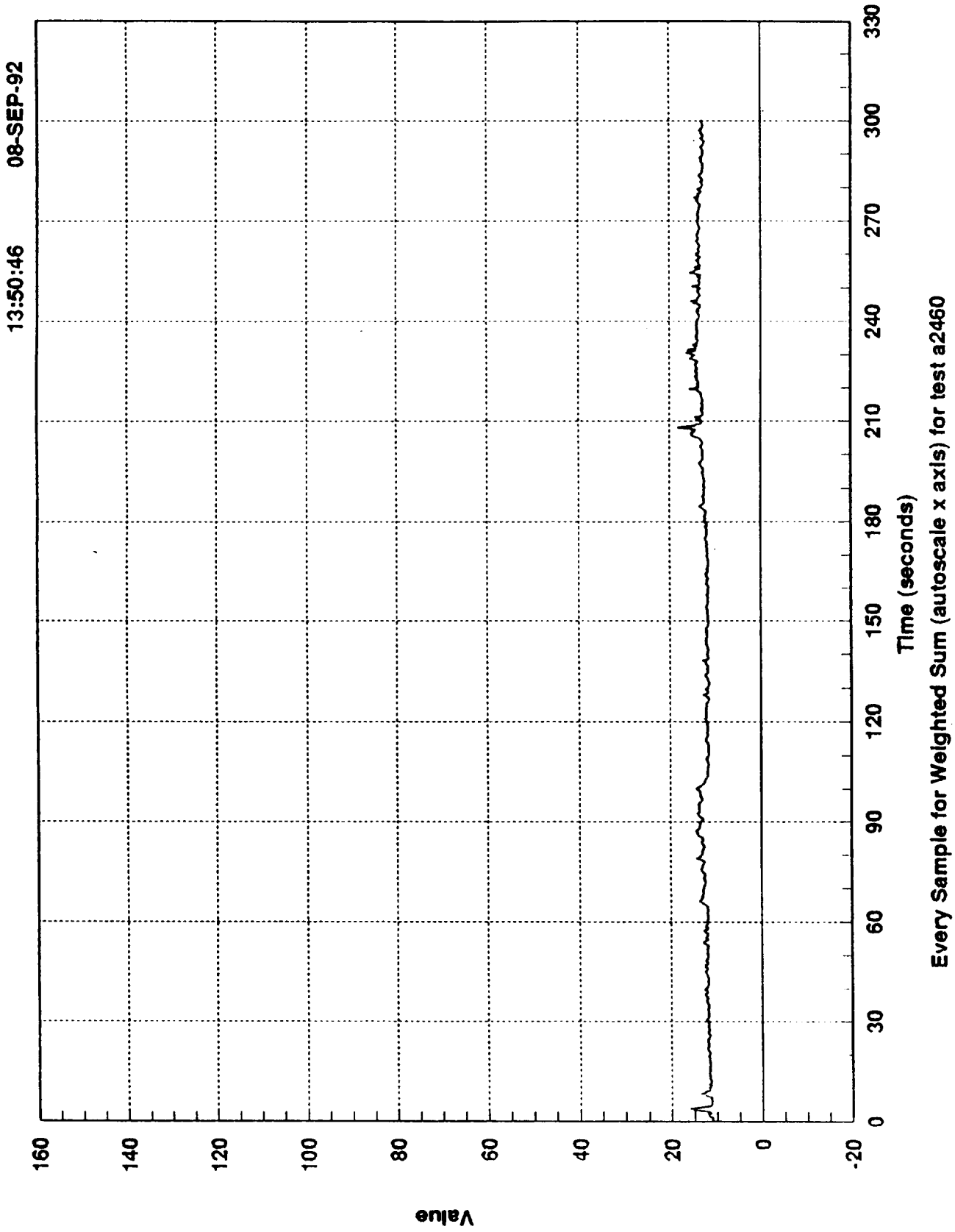


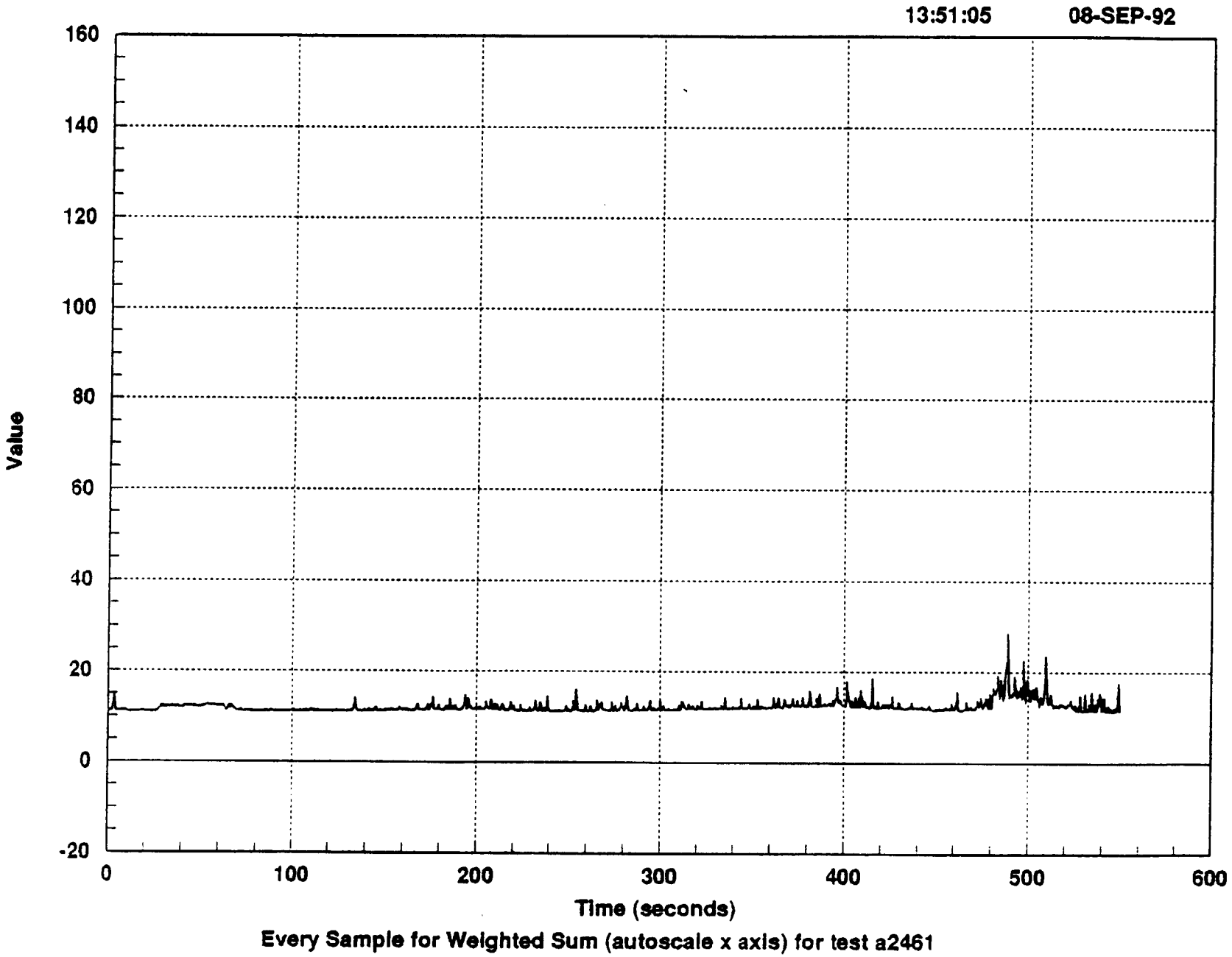
Every Sample for Weighted Sum (autoscale x axis) for test a2455



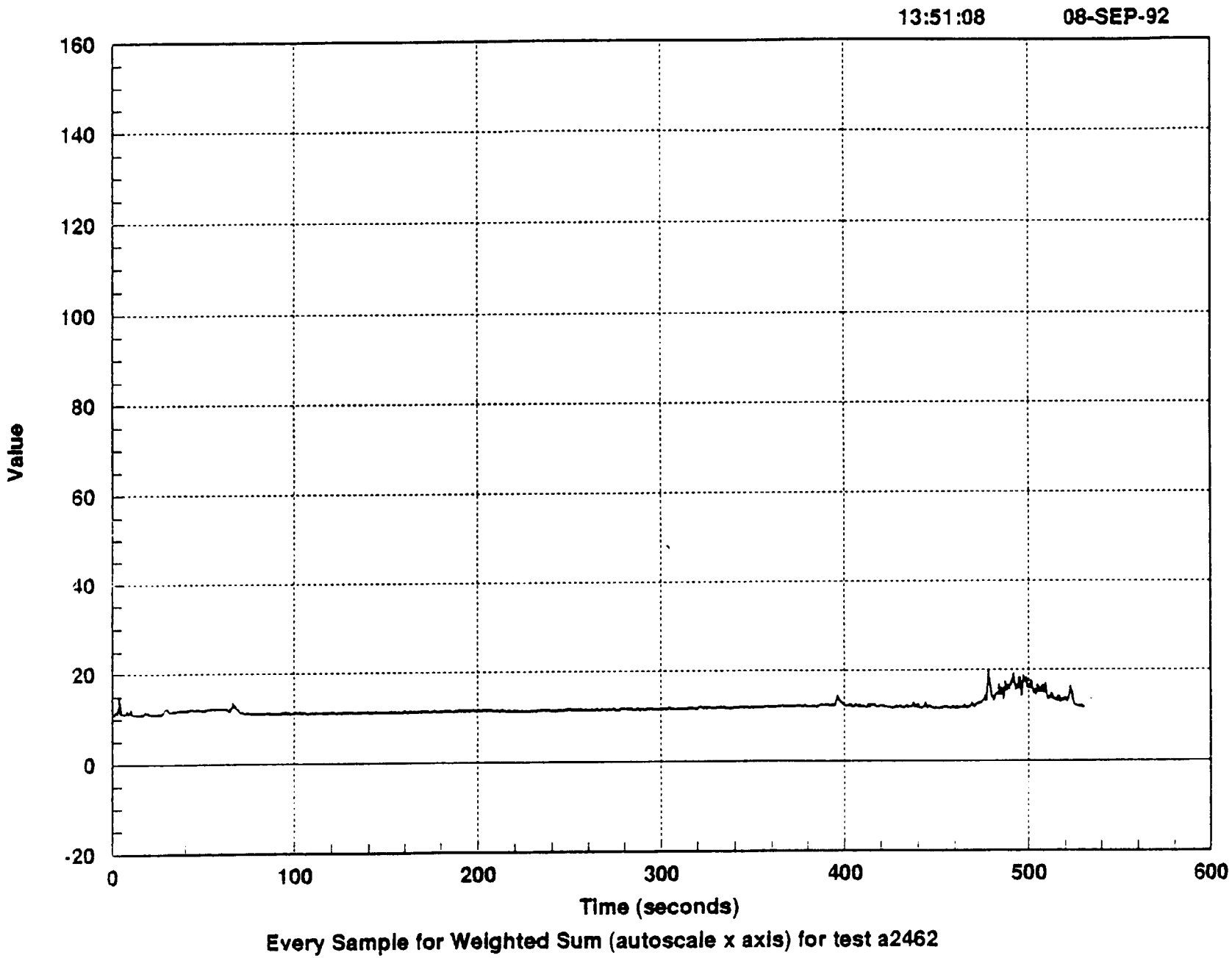
Every Sample for Weighted Sum (autoscale x axis) for test a2457


13:51:01
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2458

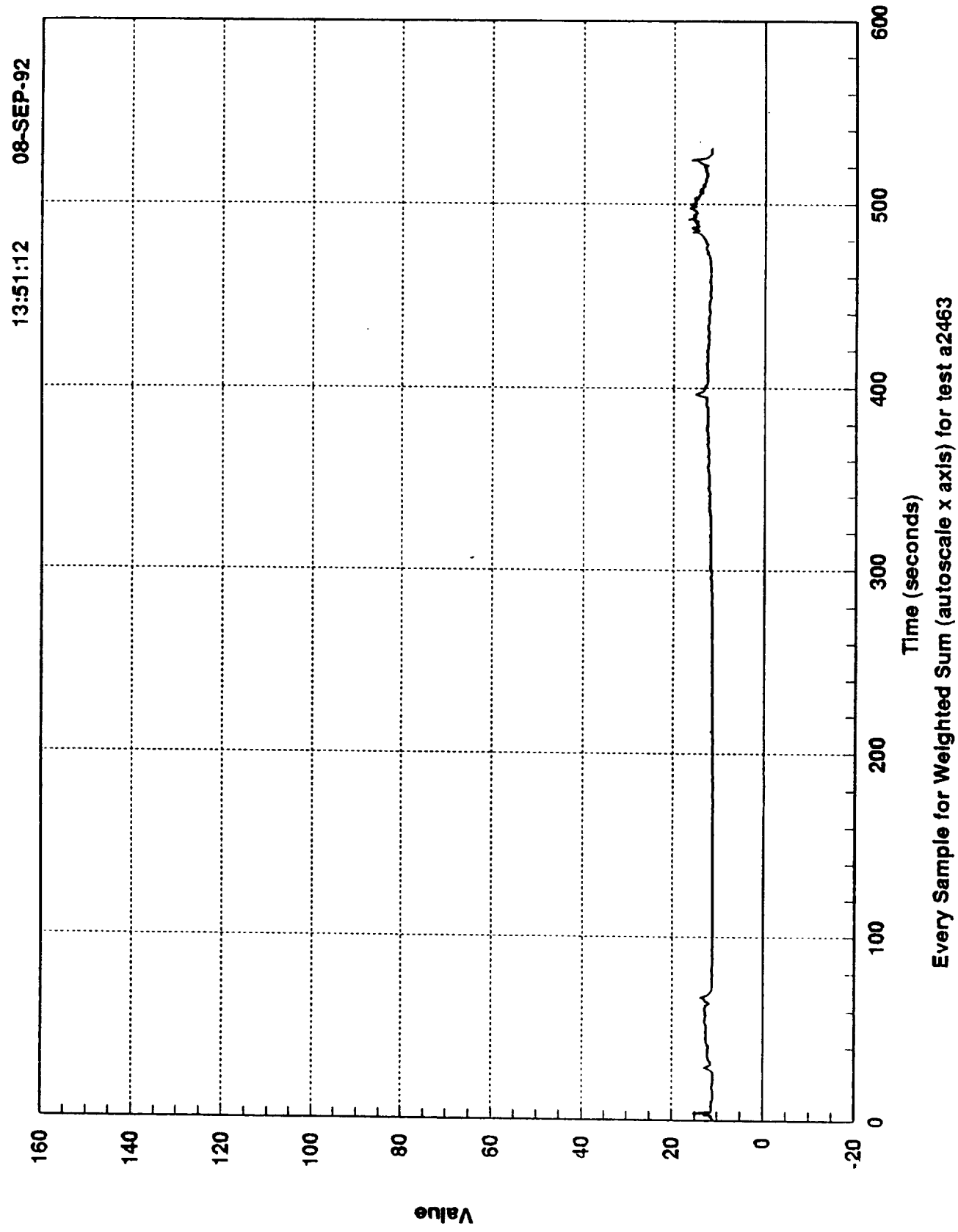

13:50:46 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
30
60
90
120
150
180
210
240
270
300
330
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2460

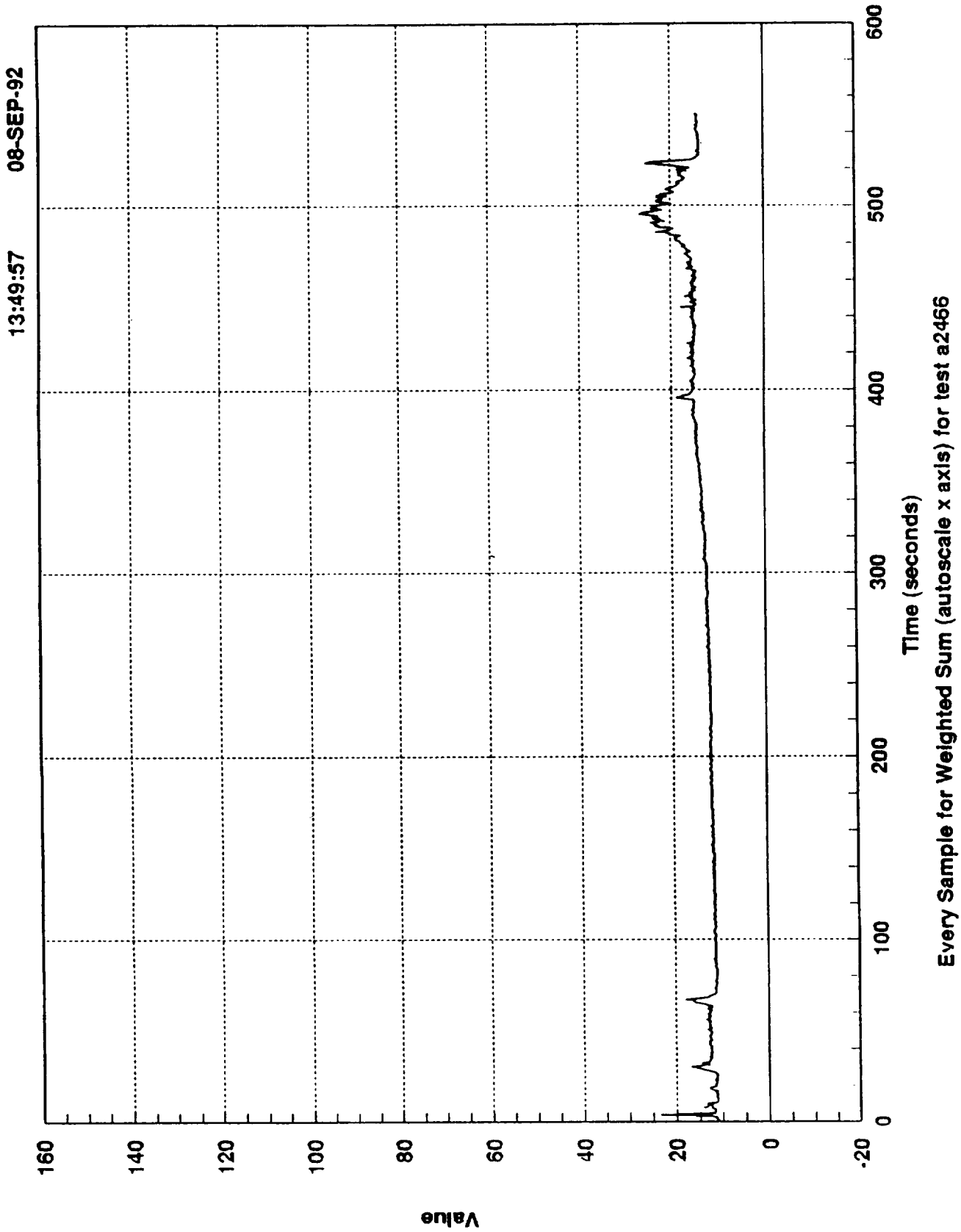


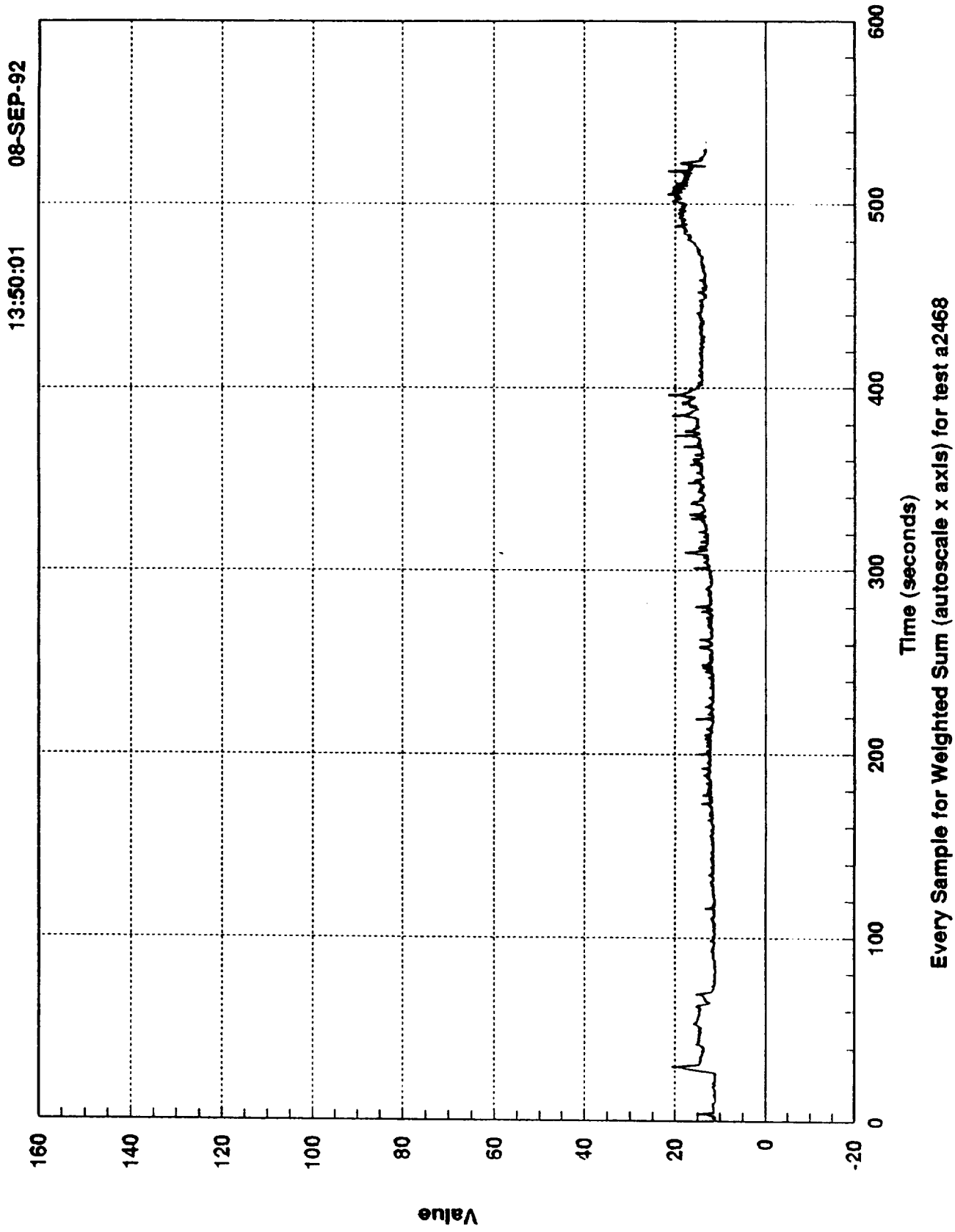
Every Sample for Weighted Sum (autoscale x axis) for test a2461

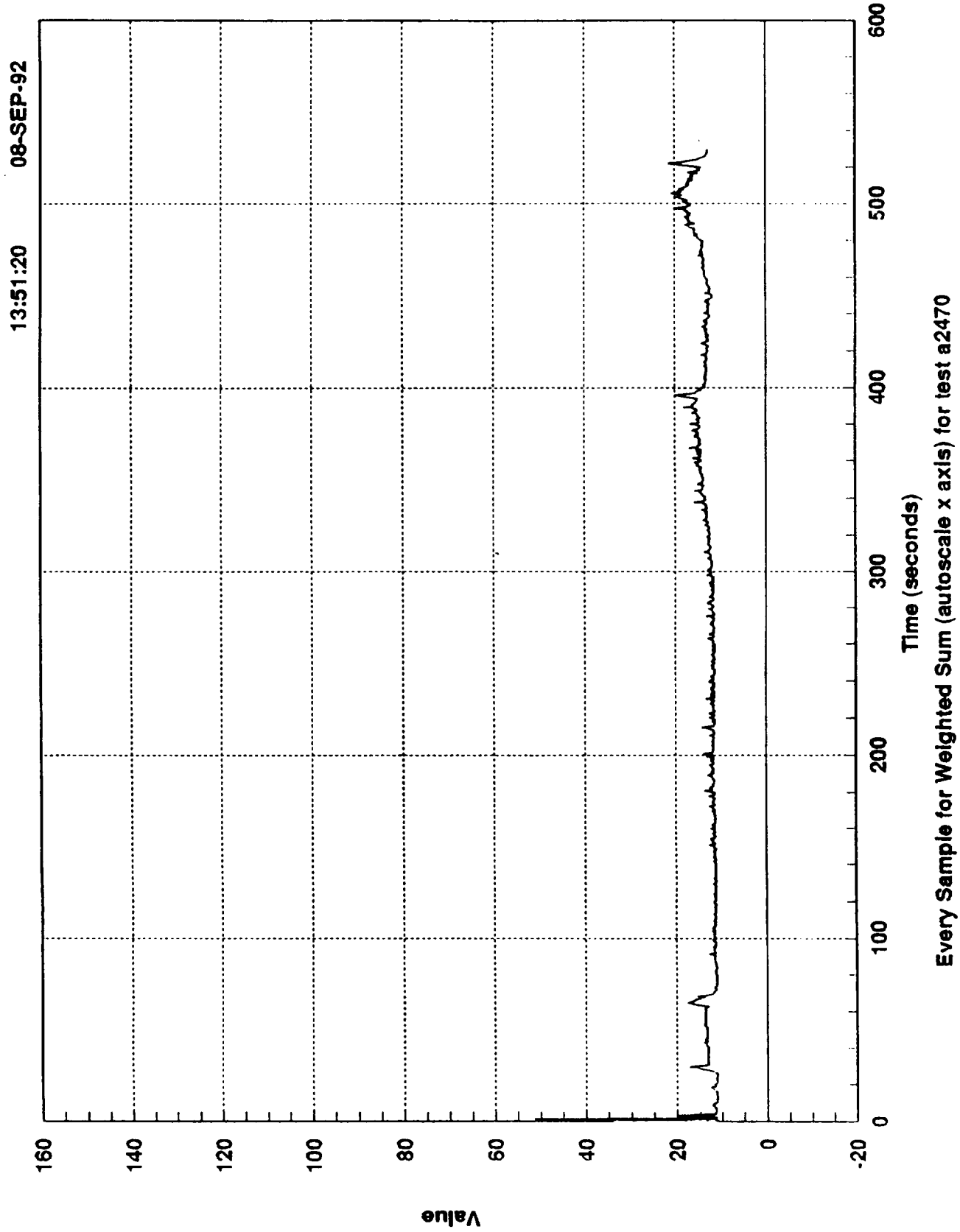


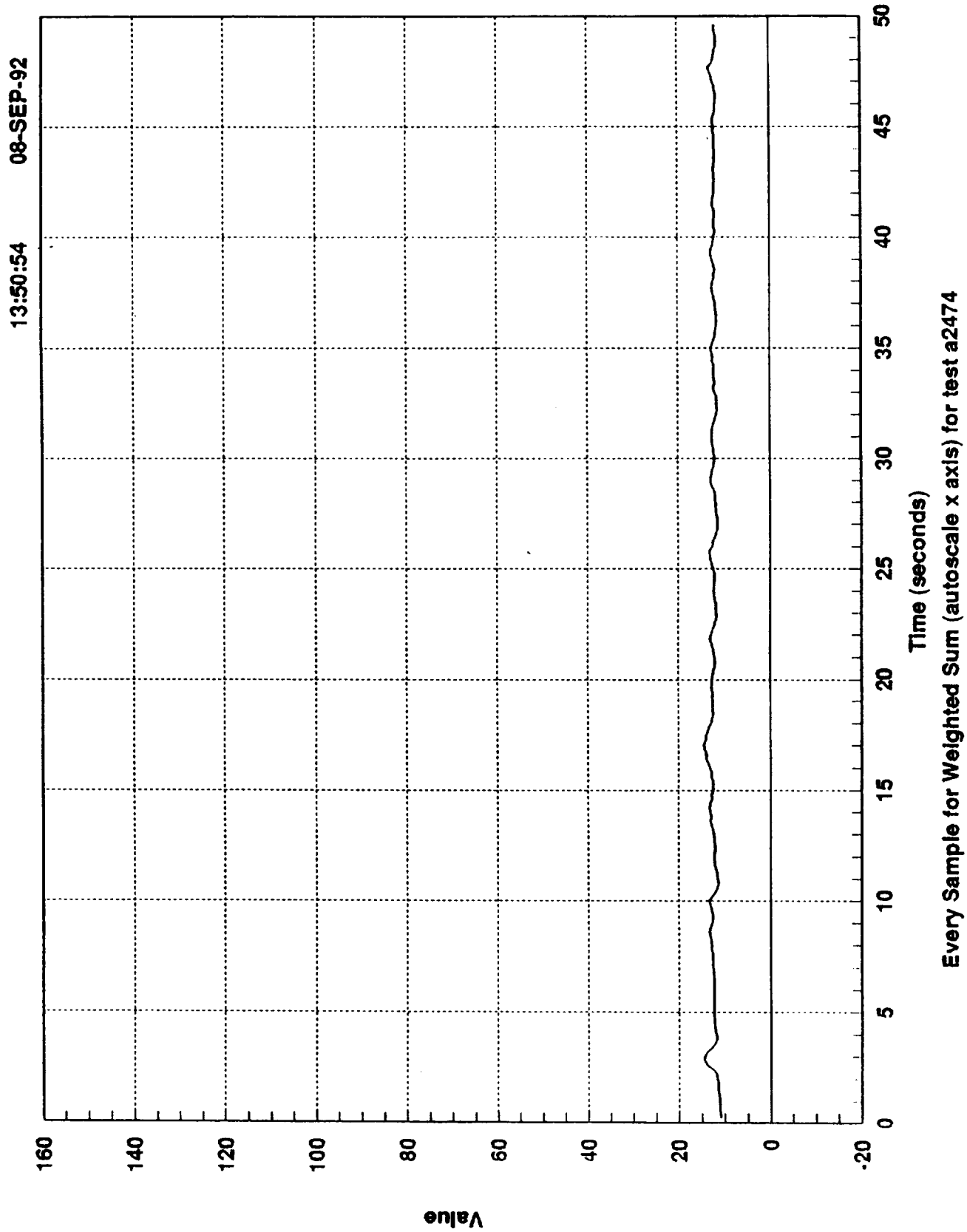
Every Sample for Weighted Sum (autoscale x axis) for test a2462

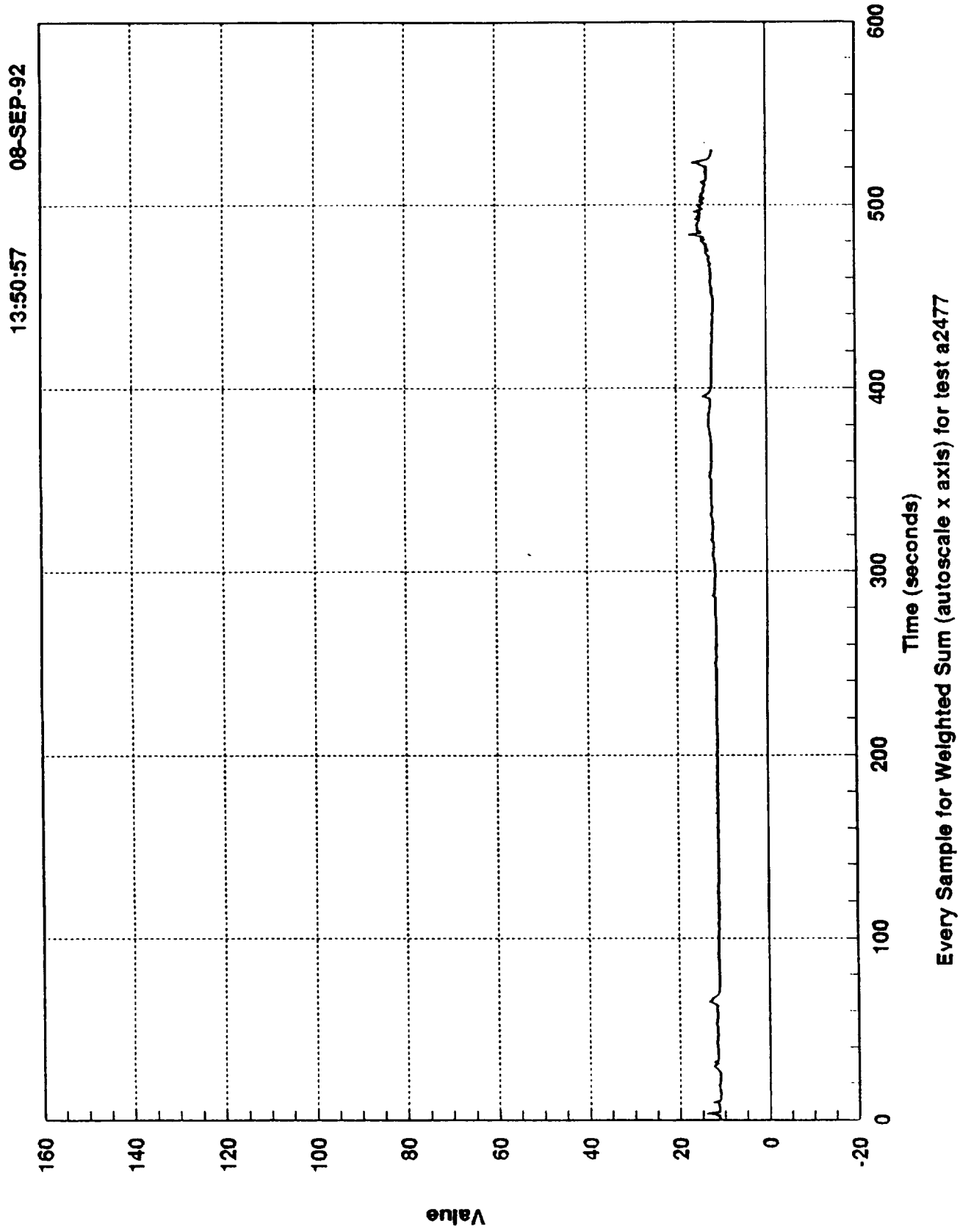

13:51:12
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2463

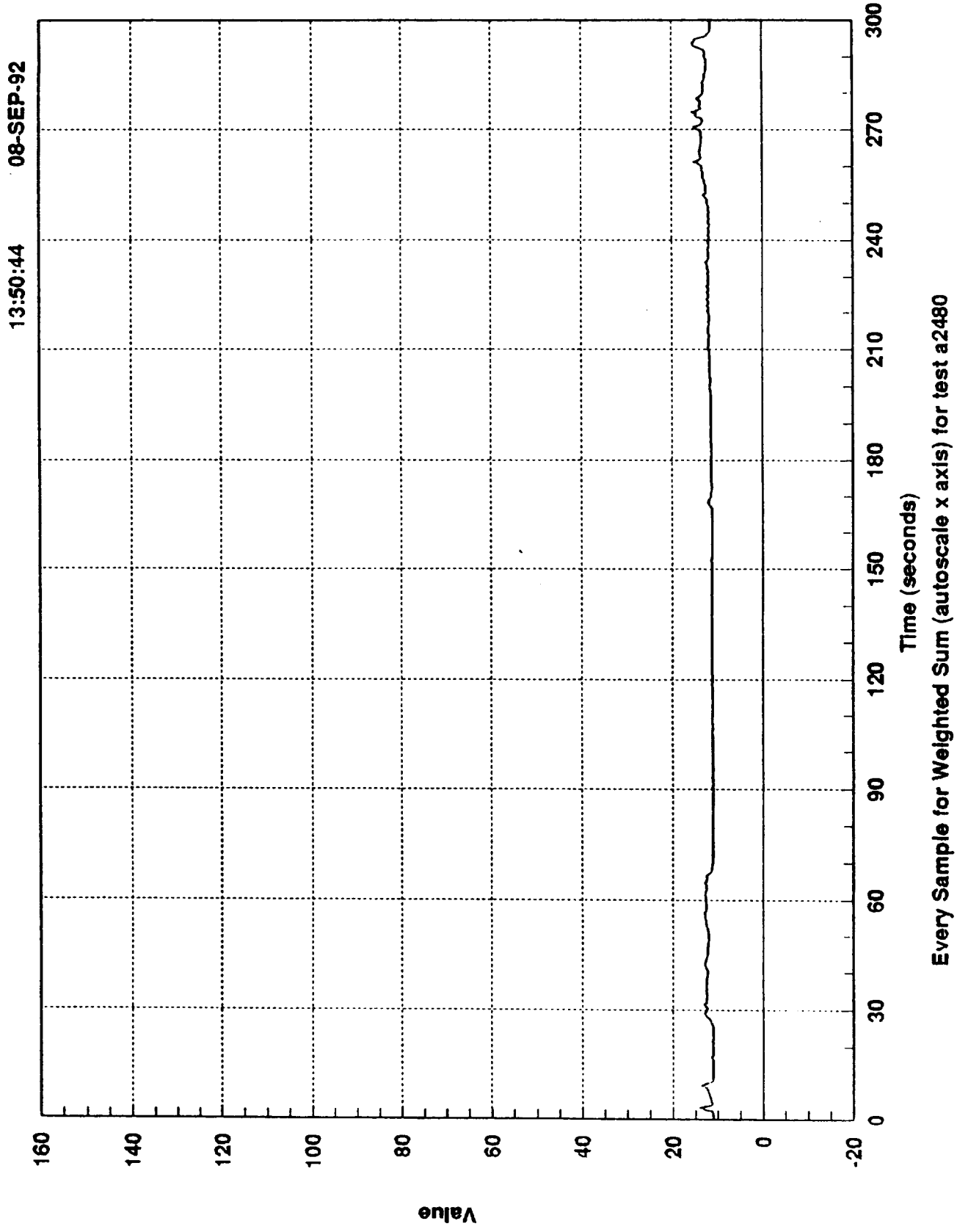

13:49:57
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2466

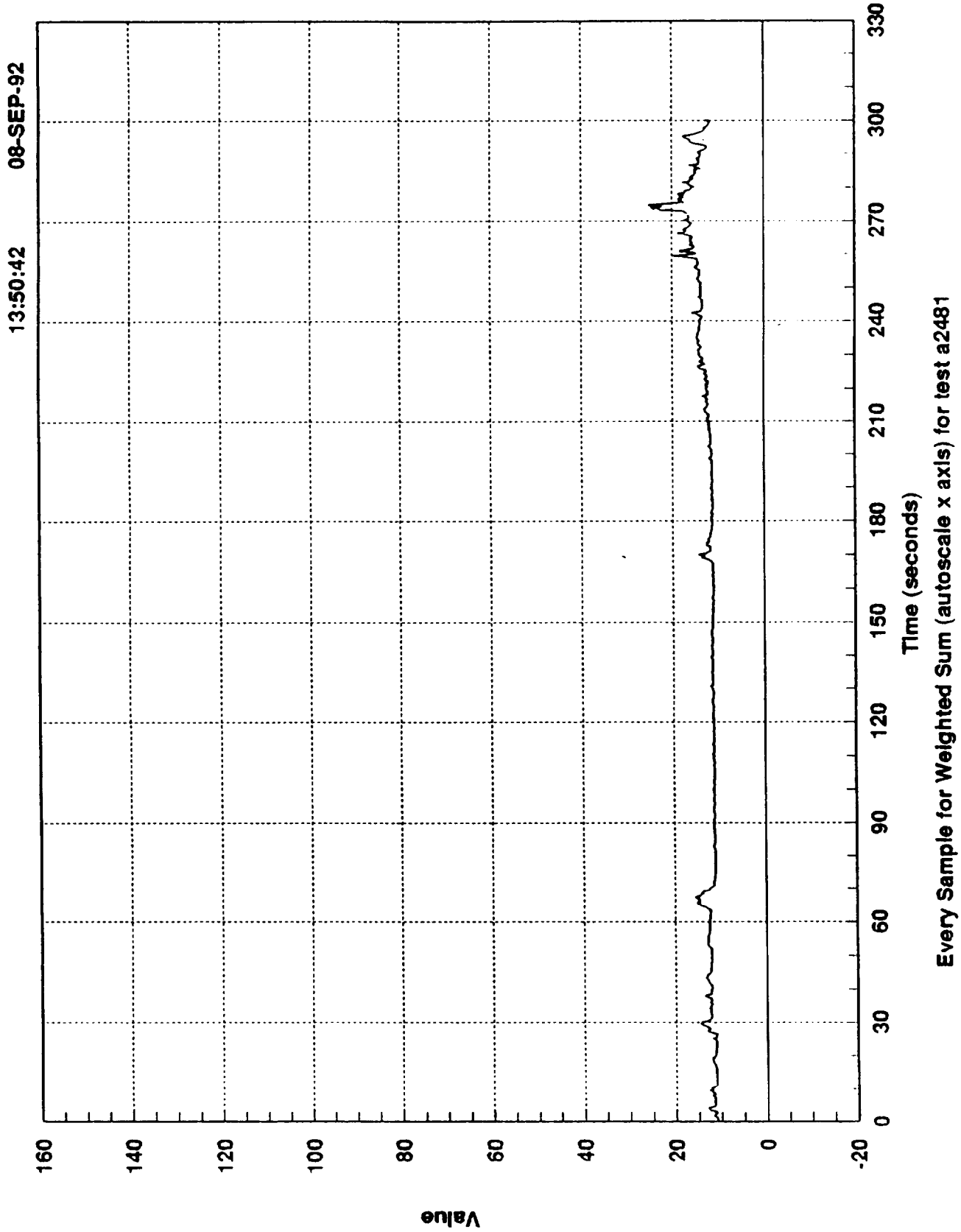

13:50:01
08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2468

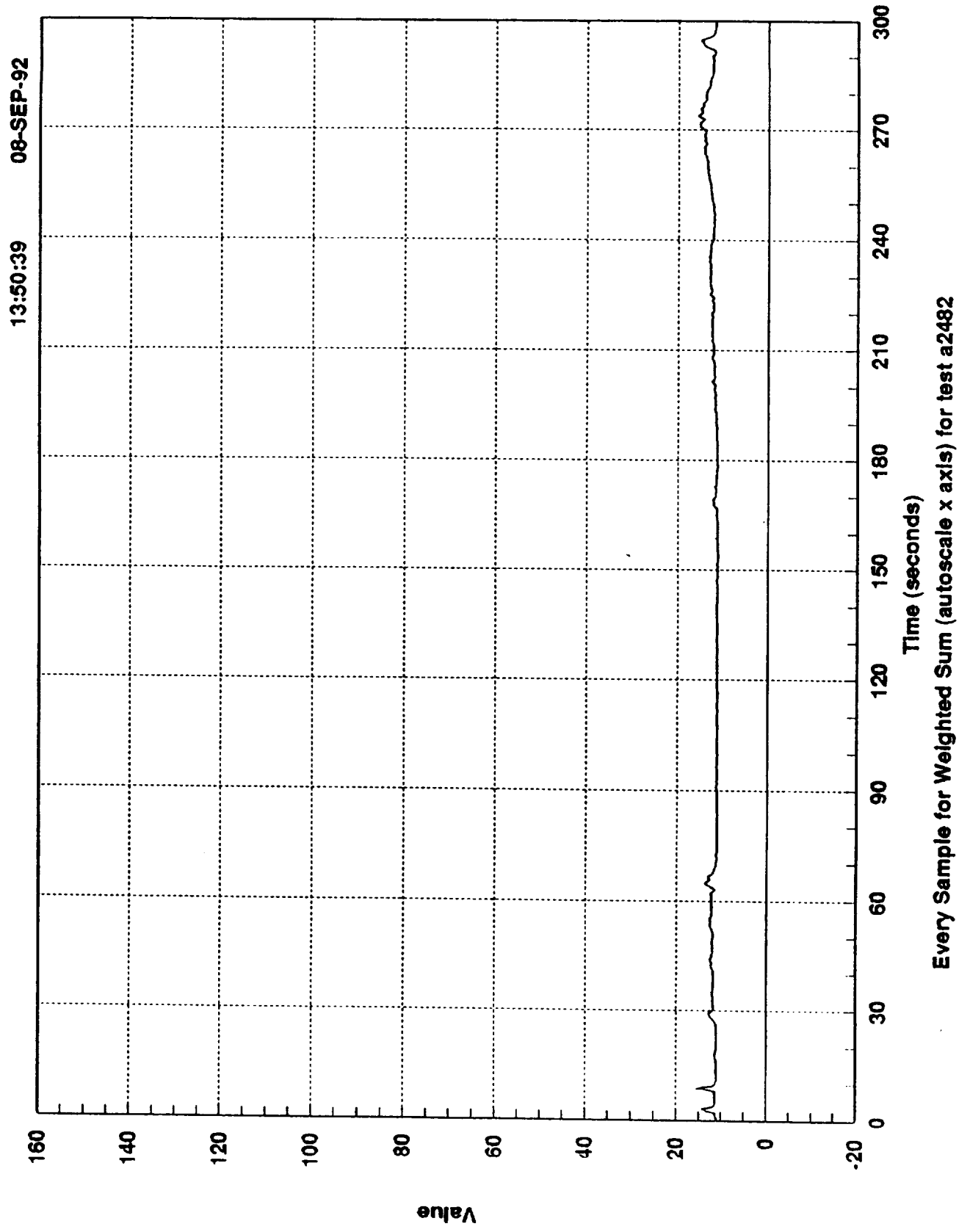

13:51:20
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2470

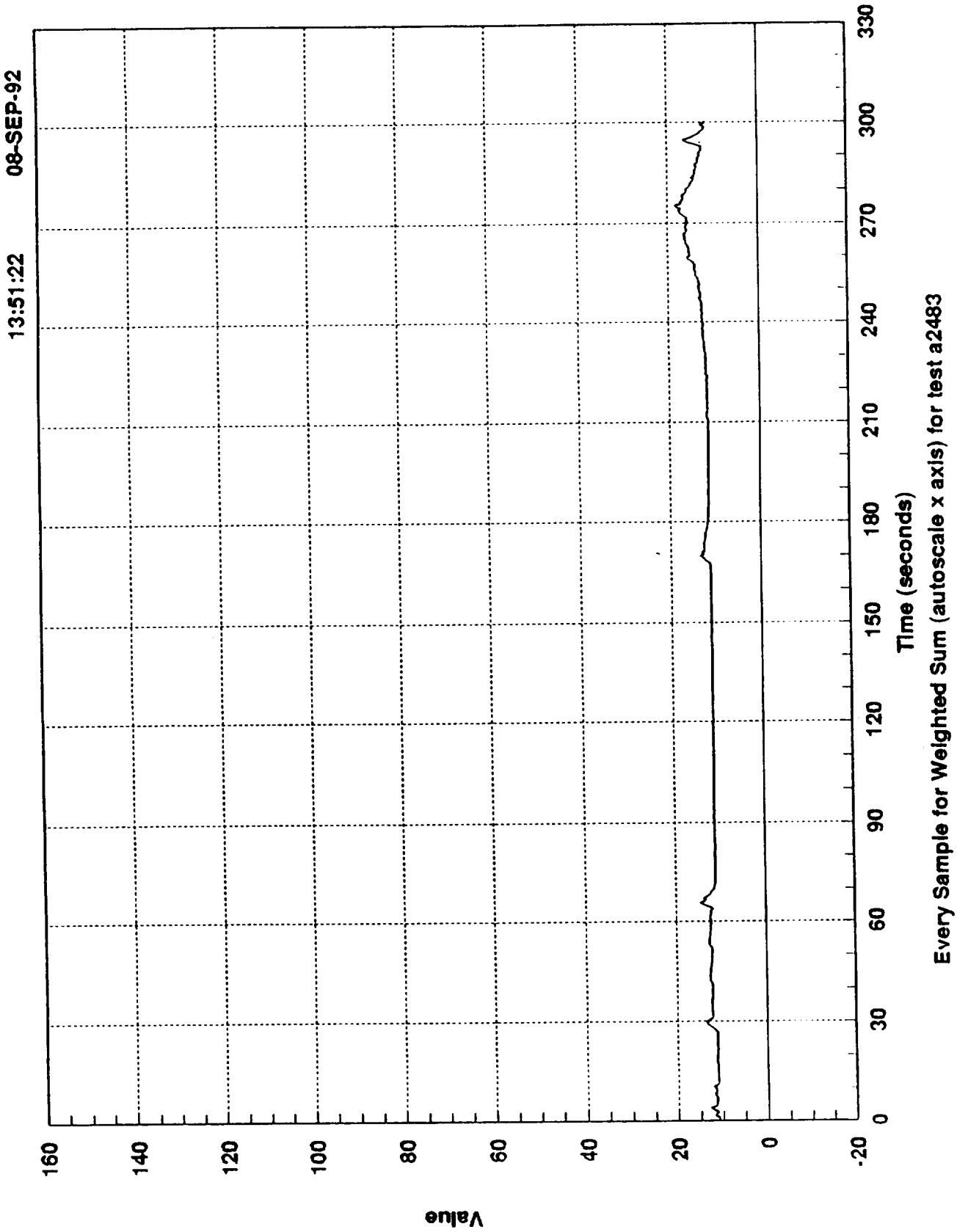

13:50:54 08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
5
10
15
20
25
30
35
40
45
50
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2474

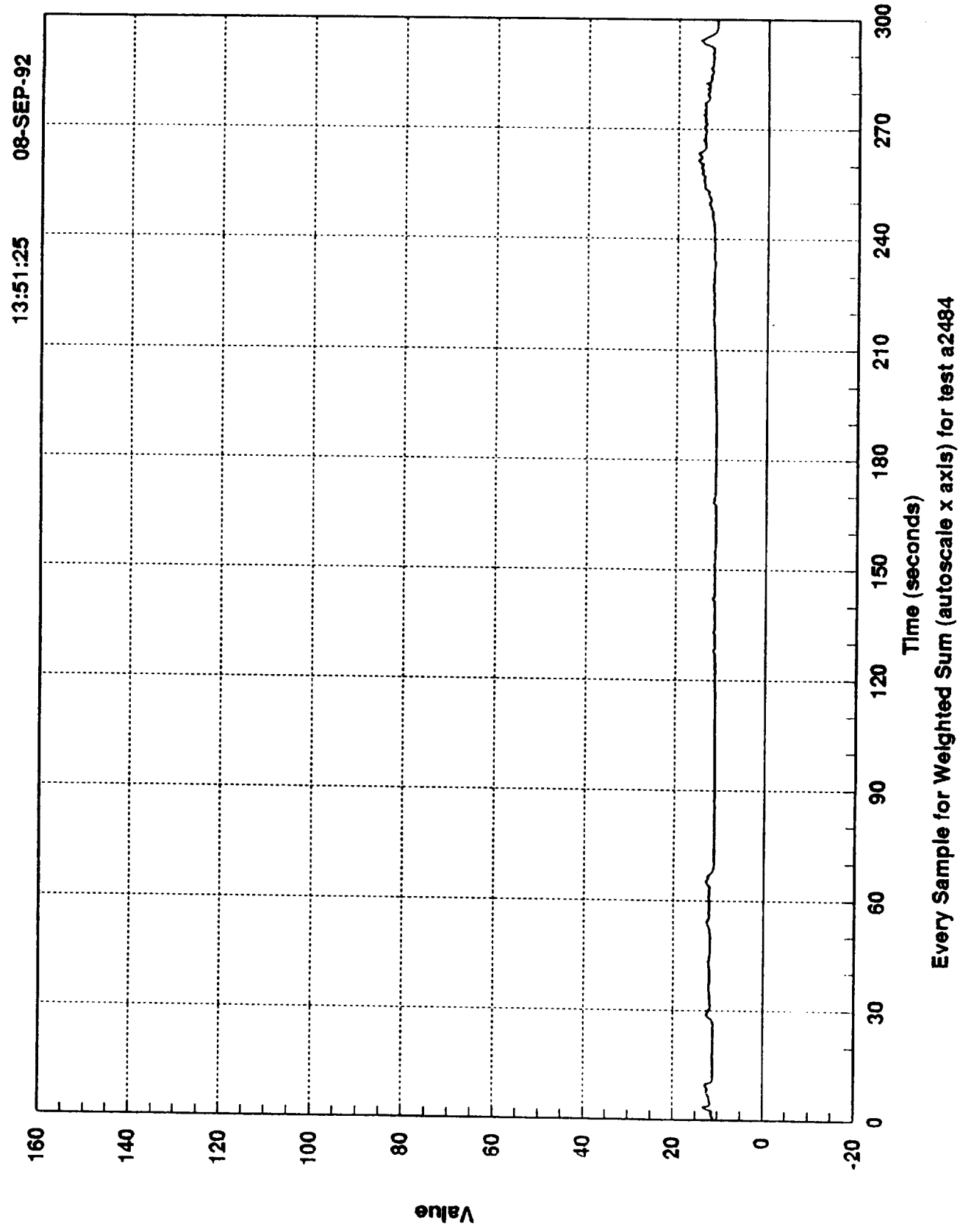

13:50:57 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2477

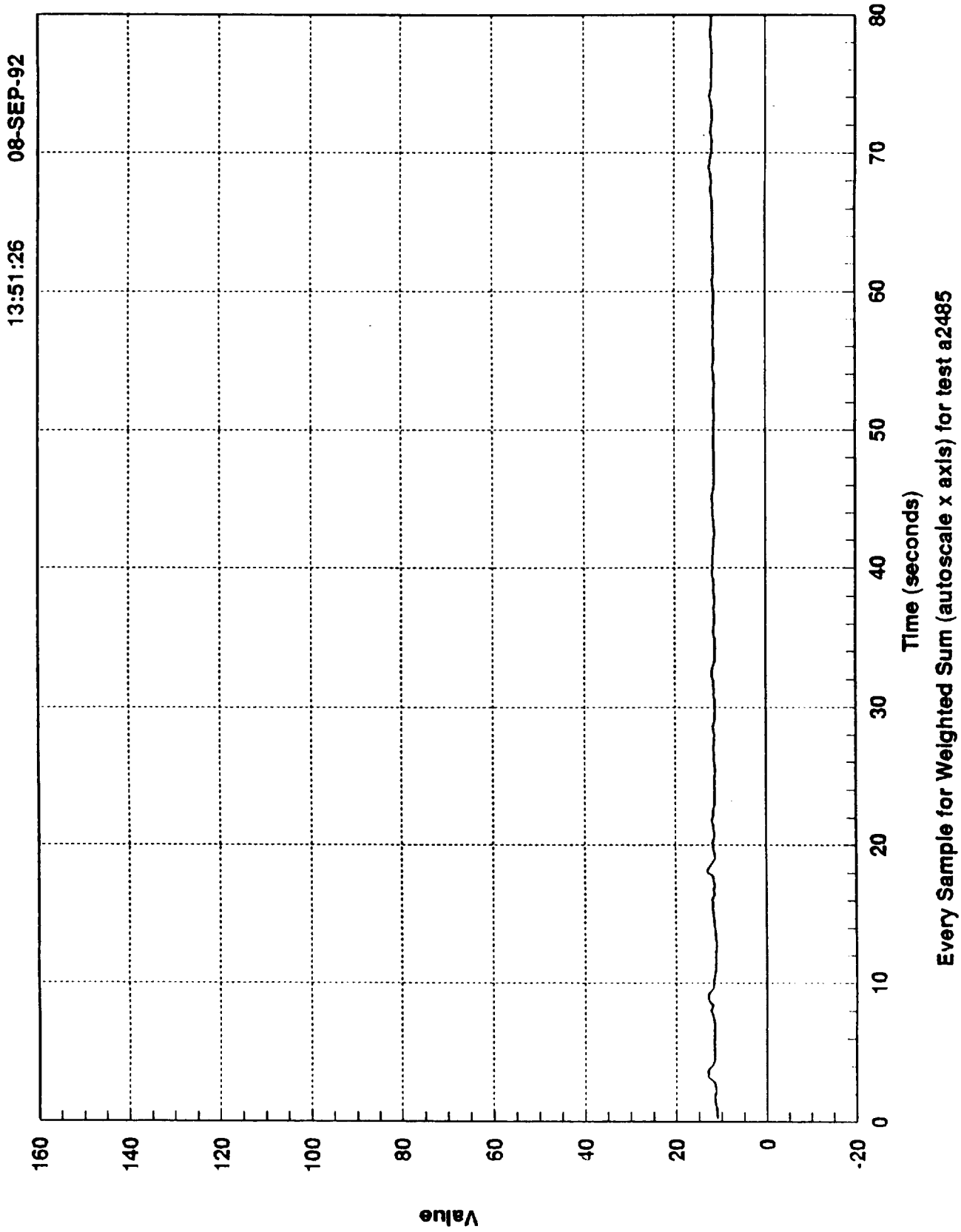

13:50:44 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
30
60
90
120
150
180
210
240
270
300
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2480

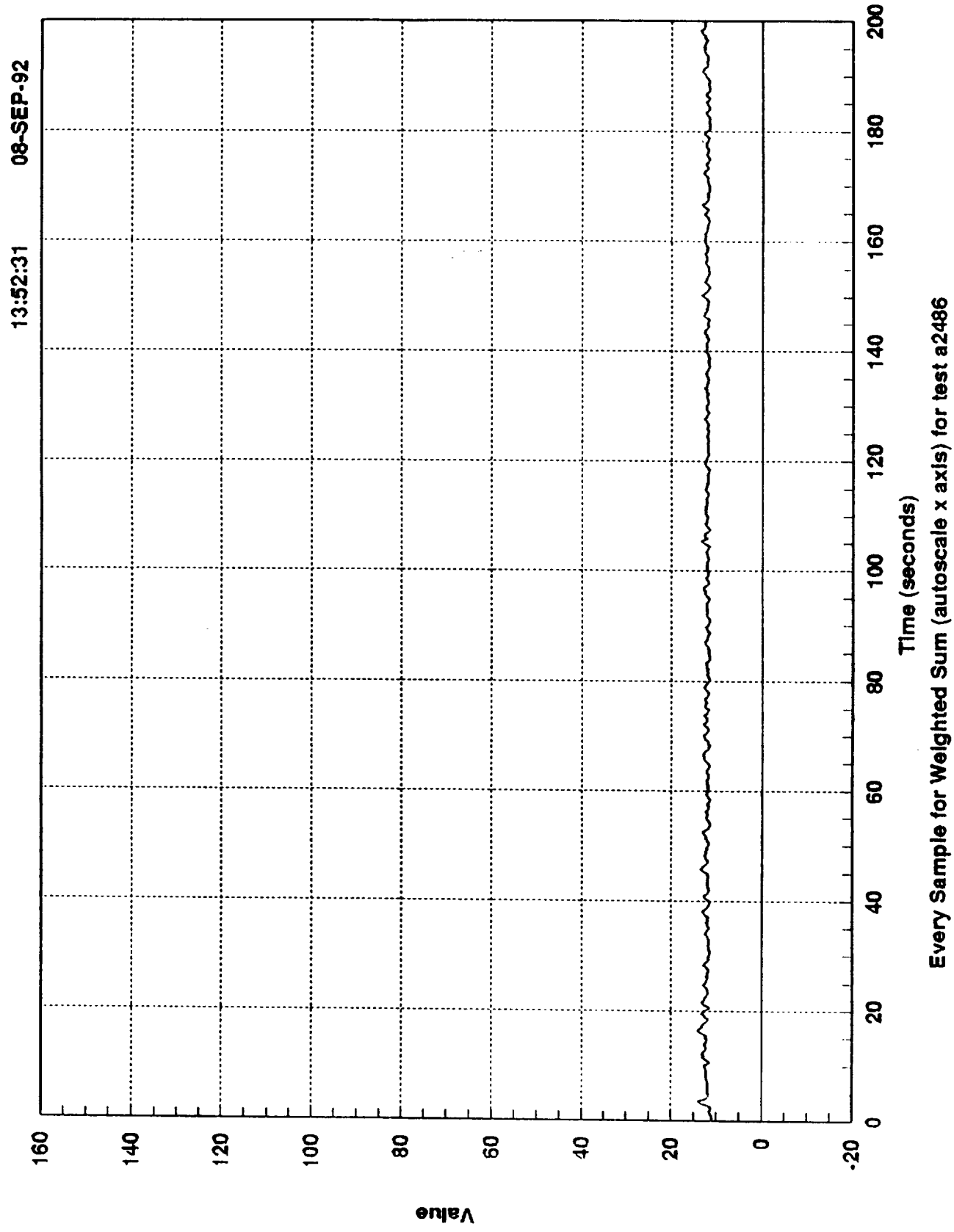

13:50:42 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
30
60
90
120
150
180
210
240
270
300
330
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2481

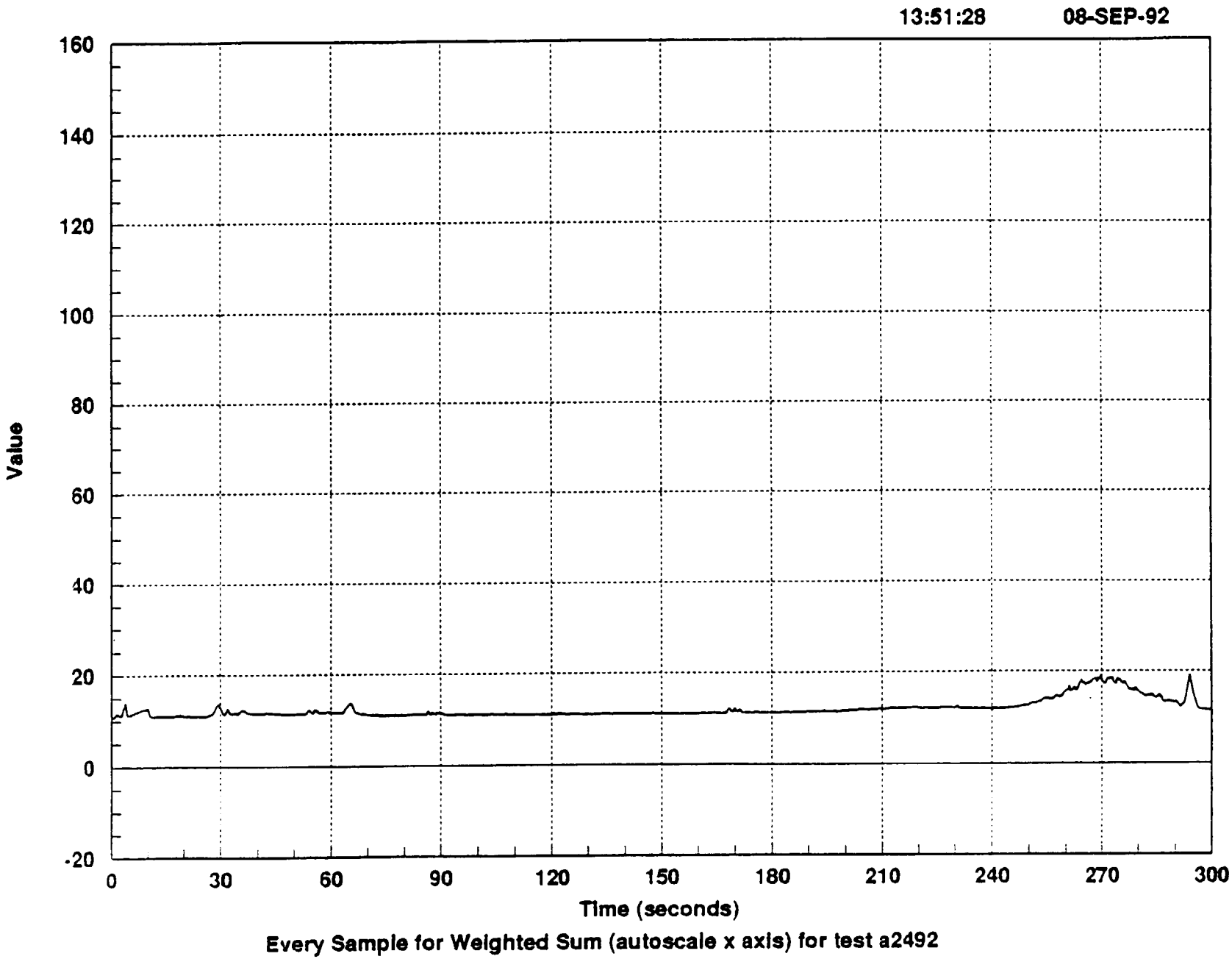

13:50:39 08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
30
60
90
120
150
180
210
240
270
300
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2482

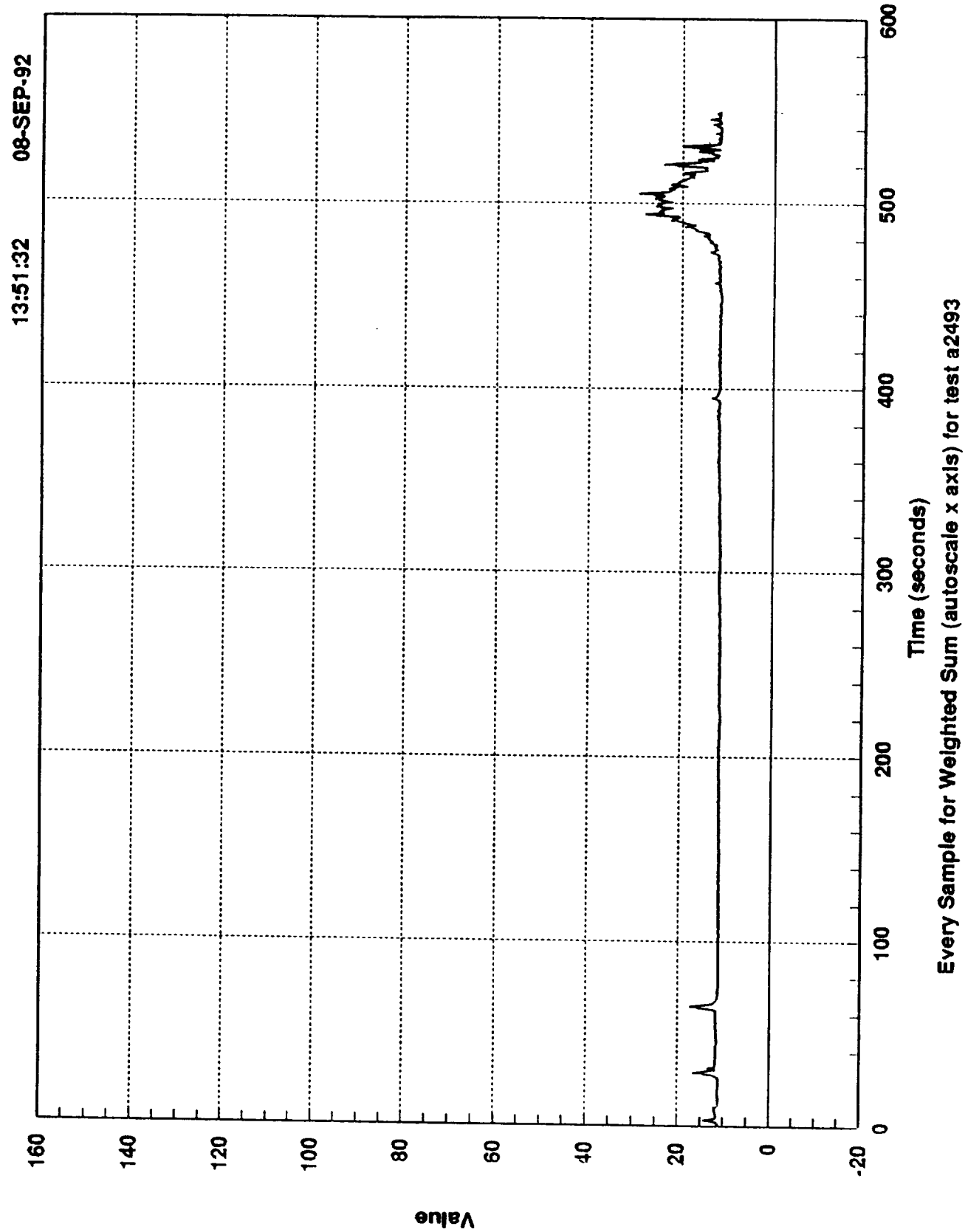

13:51:22 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
30
60
90
120
150
180
210
240
270
300
330
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2483

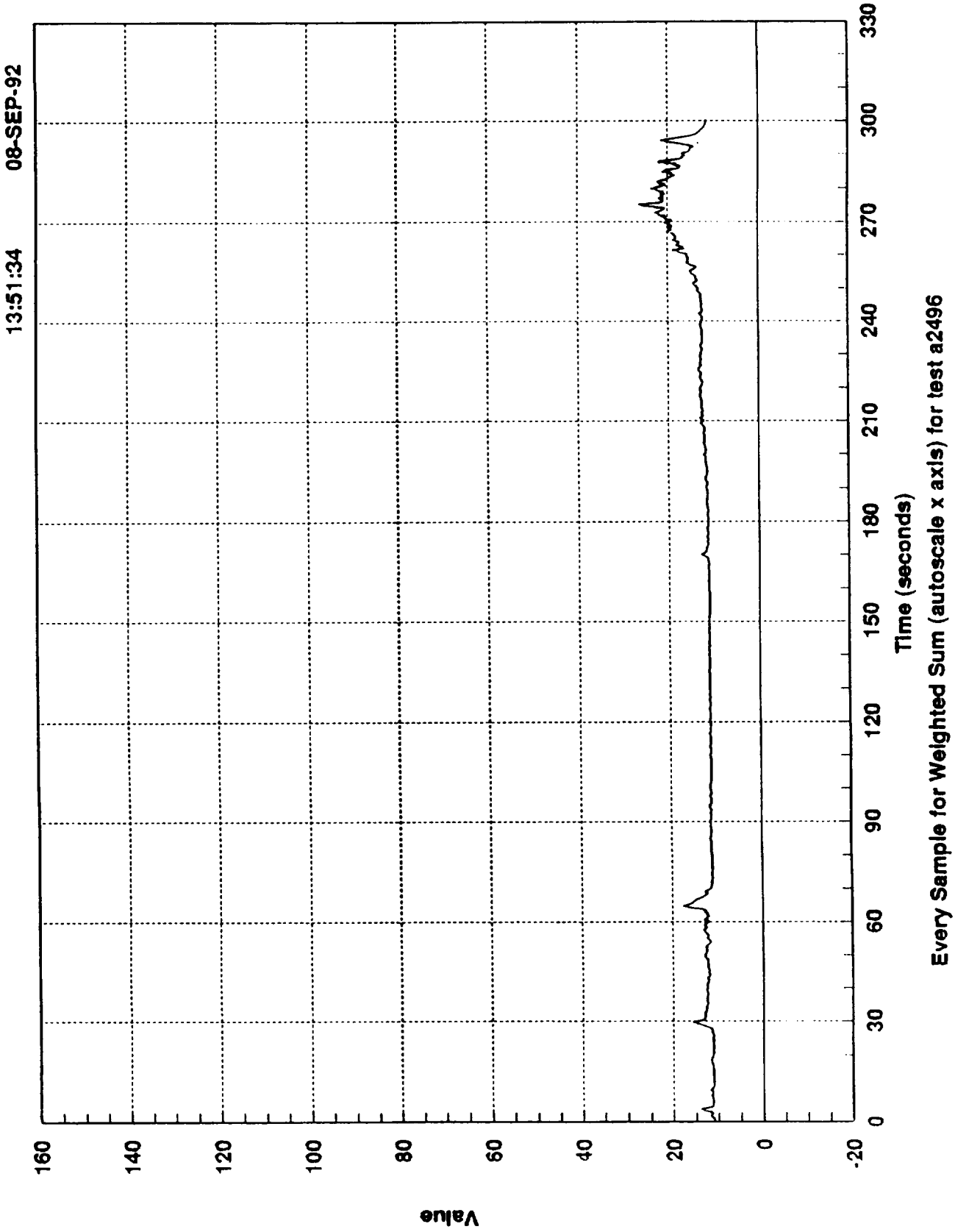

13:51:25 08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
30
60
90
120
150
180
210
240
270
300
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2484

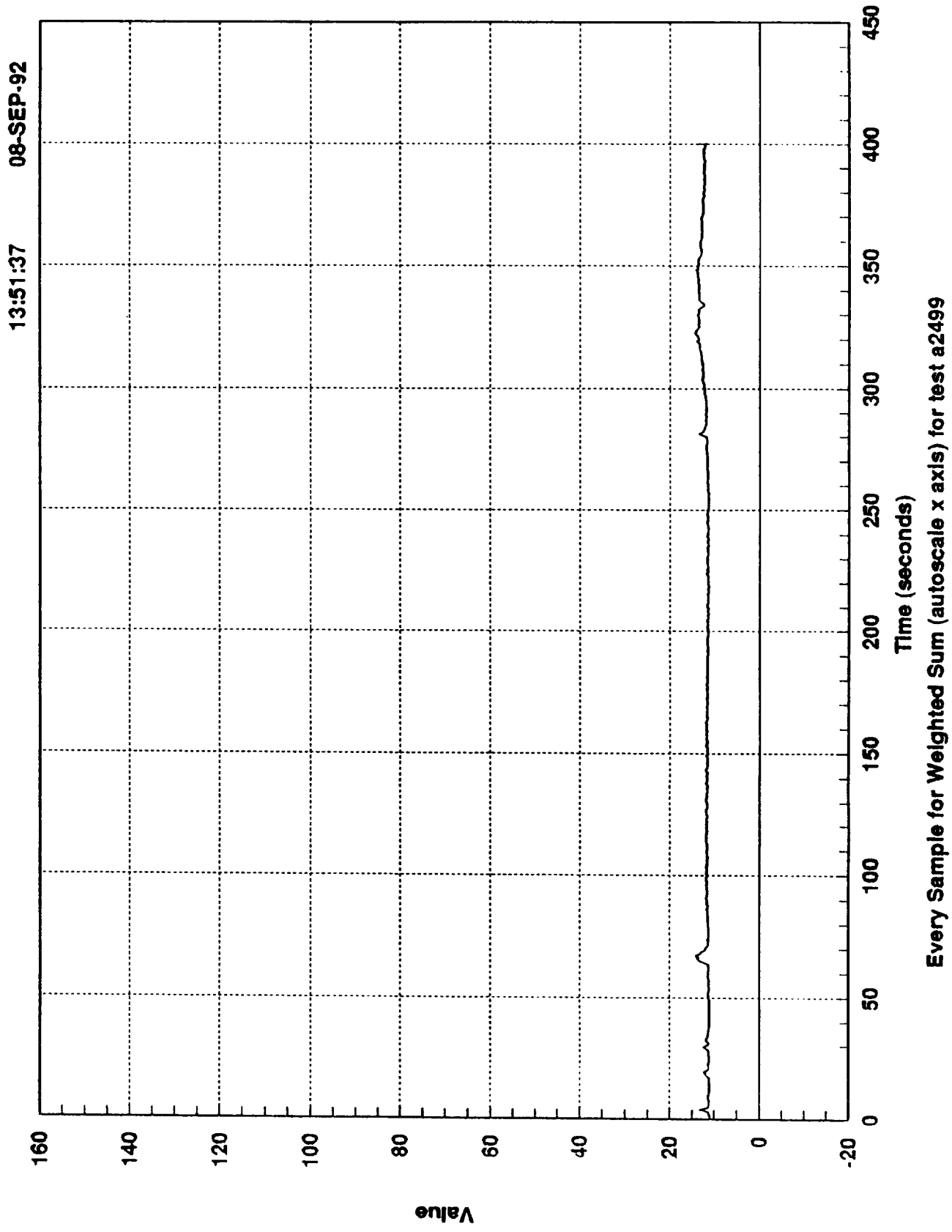

13:51:26 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
10
20
30
40
50
60
70
80
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2485

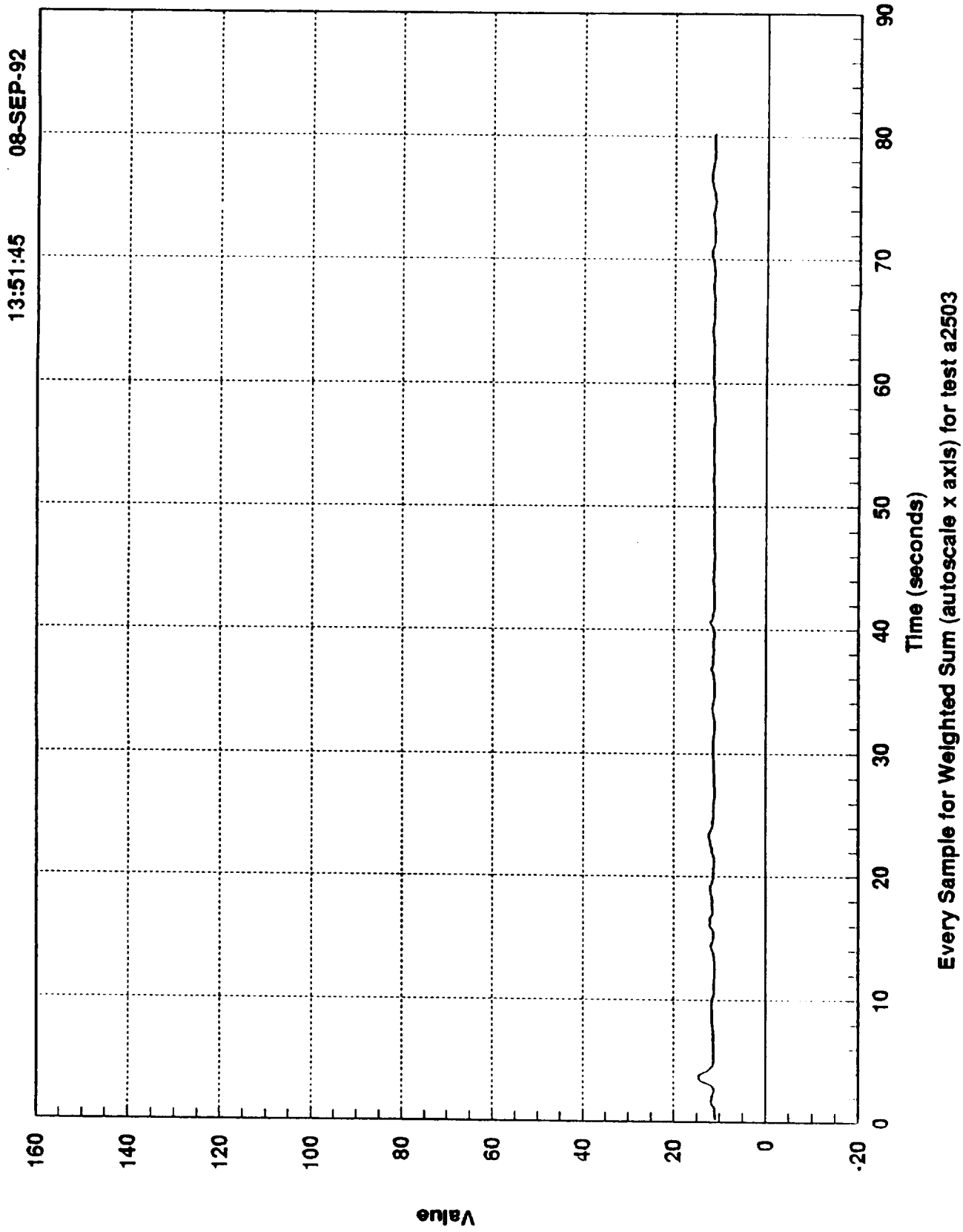

13:52:31 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
20
40
60
80
100
120
140
160
180
200
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2486

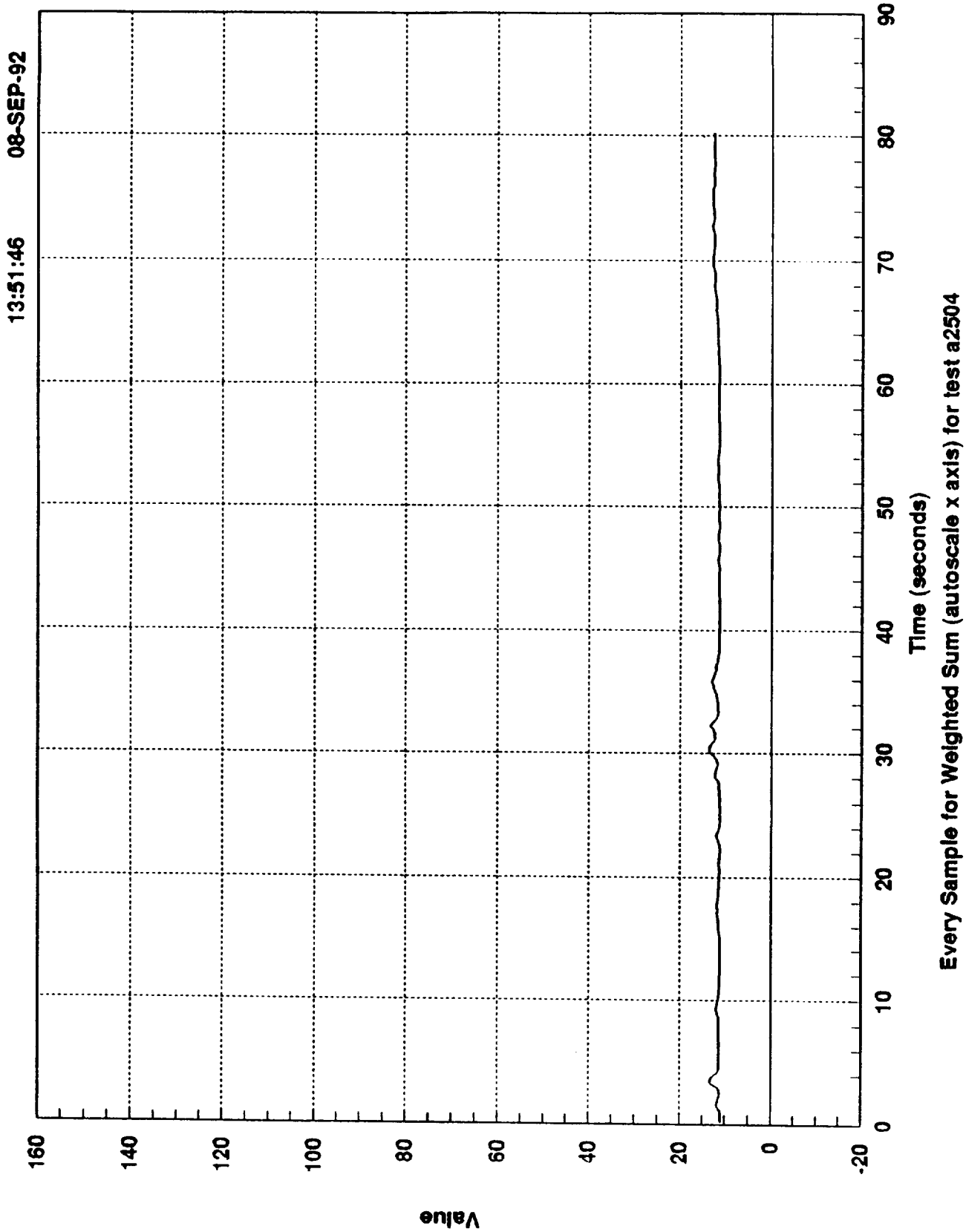

13:51:28
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
30
60
90
120
150
180
210
240
270
300
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2492

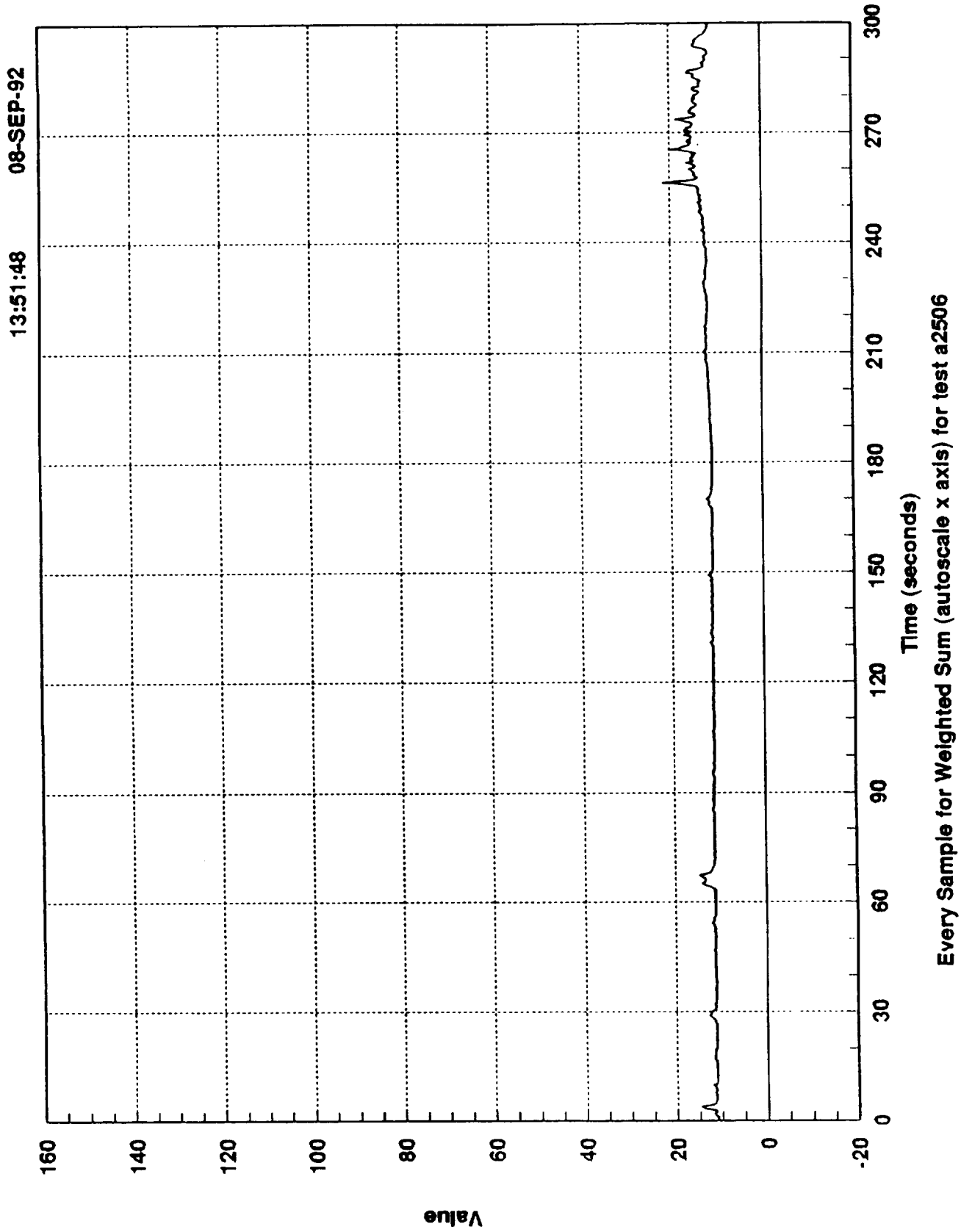

13:51:32
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2493

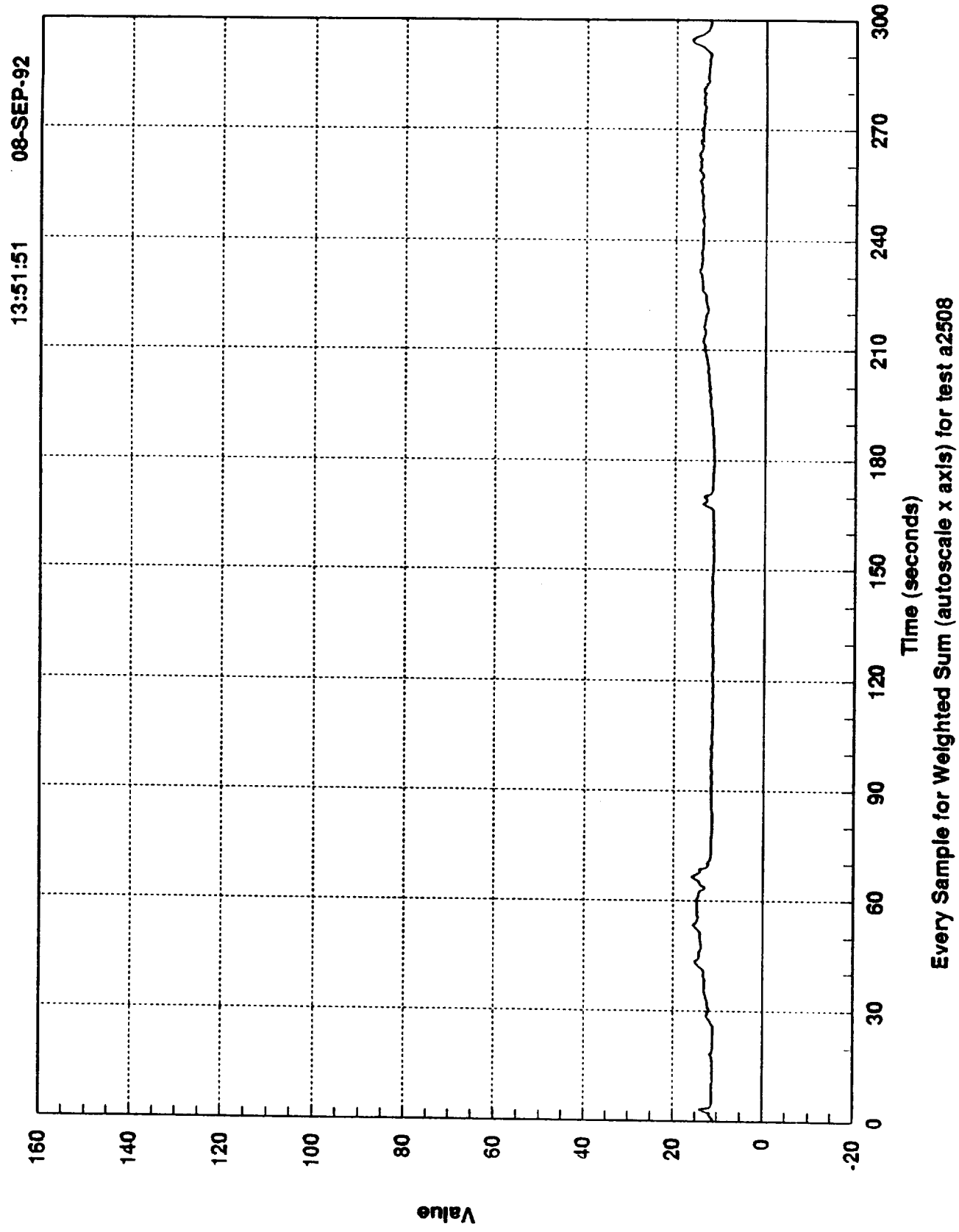

13:51:34
08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
30
60
90
120
150
180
210
240
270
300
330
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2496

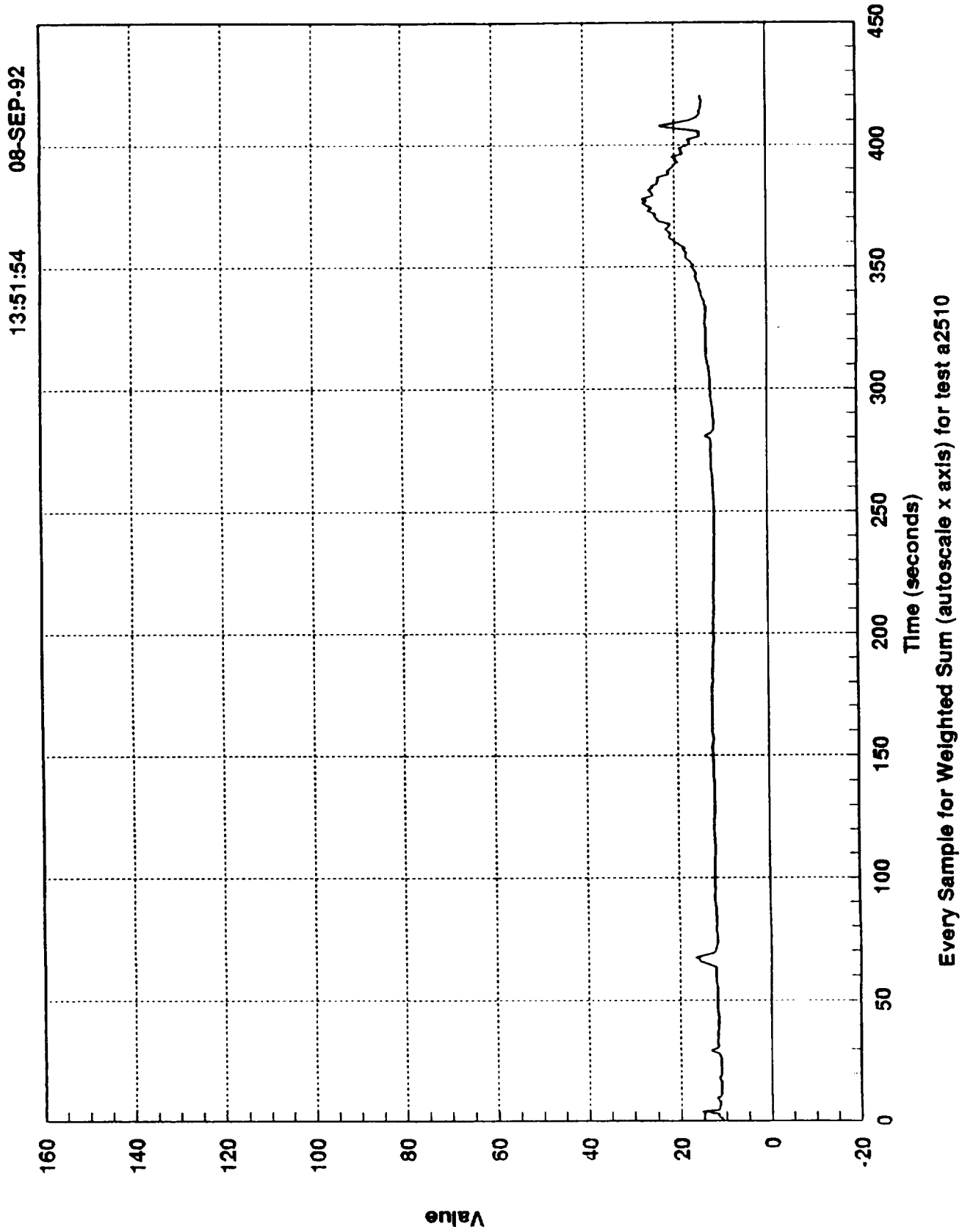

13:51:37 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
50
100
150
200
250
300
350
400
450
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2499

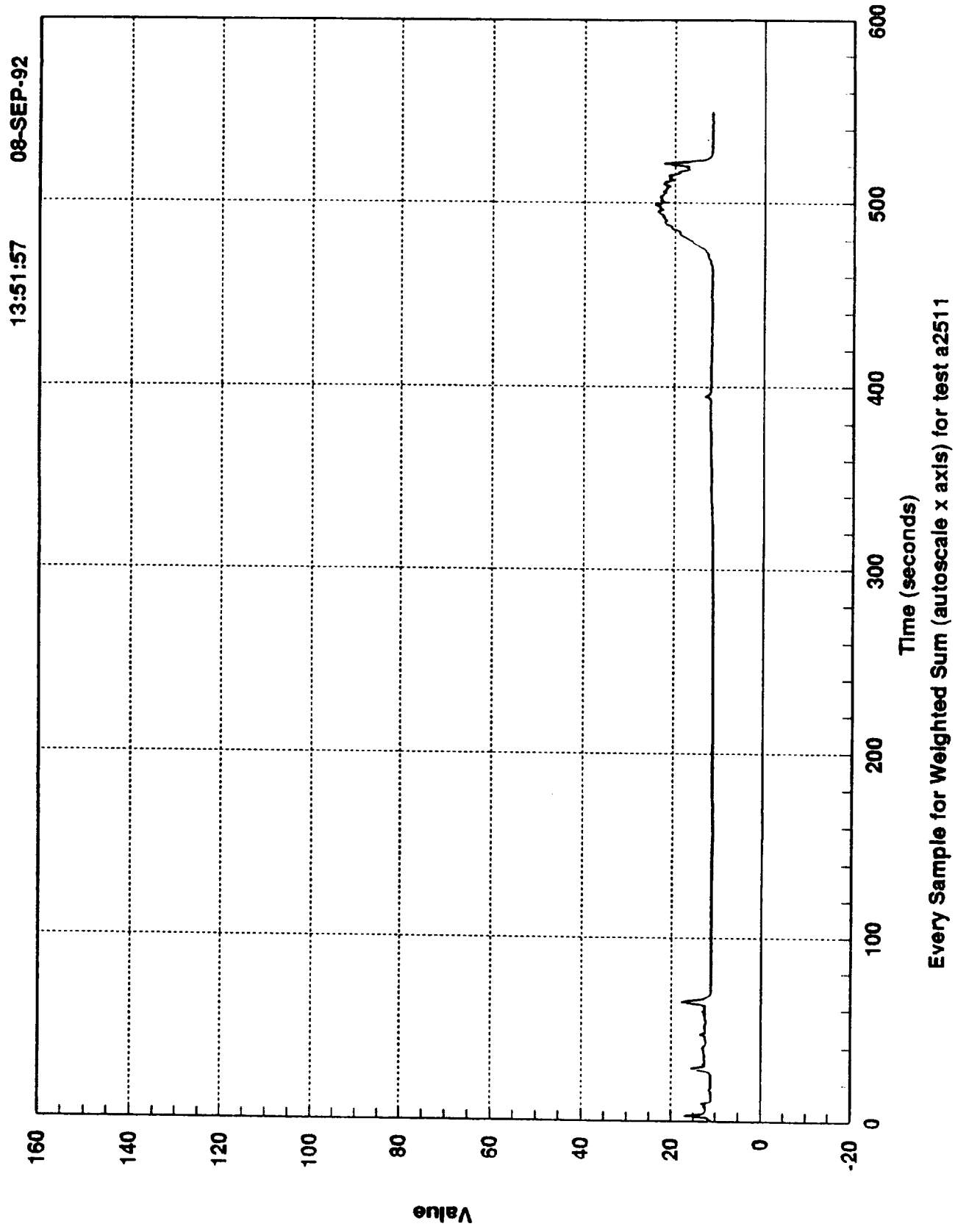

13:51:45
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
10
20
30
40
50
60
70
80
90
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2503

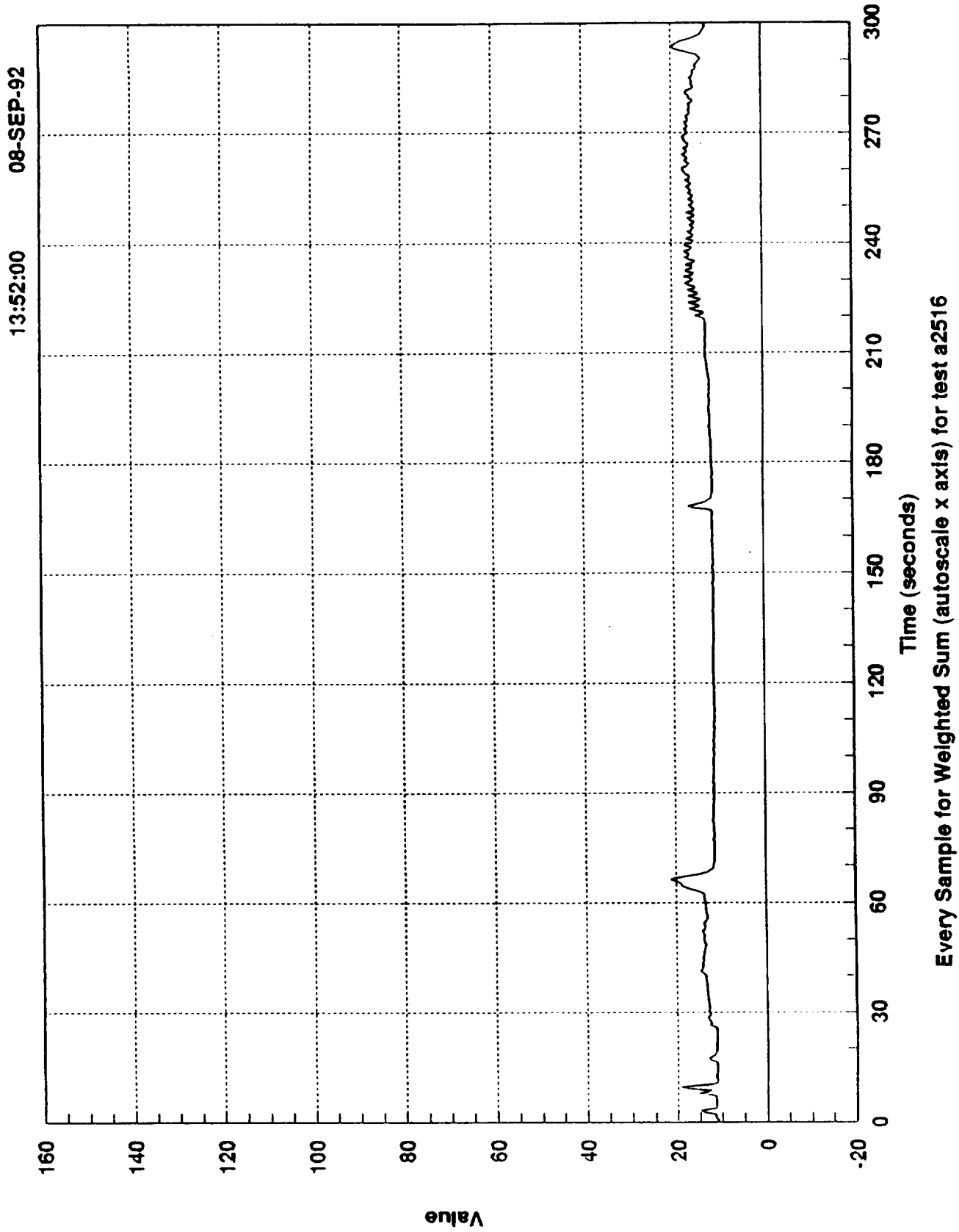

13:51:46
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
10
20
30
40
50
60
70
80
90
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2504

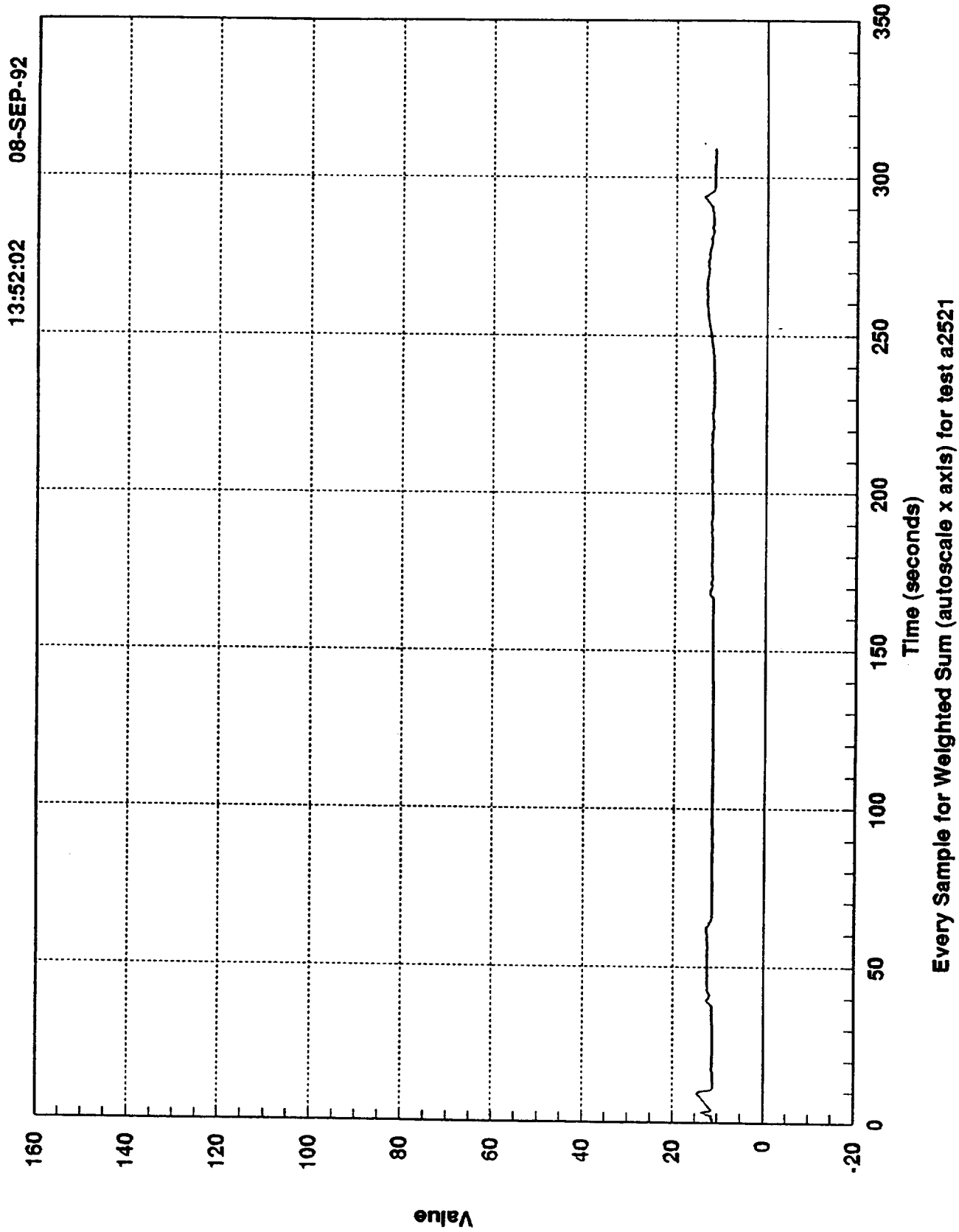

13:51:48
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
30
60
90
120
150
180
210
240
270
300
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2506

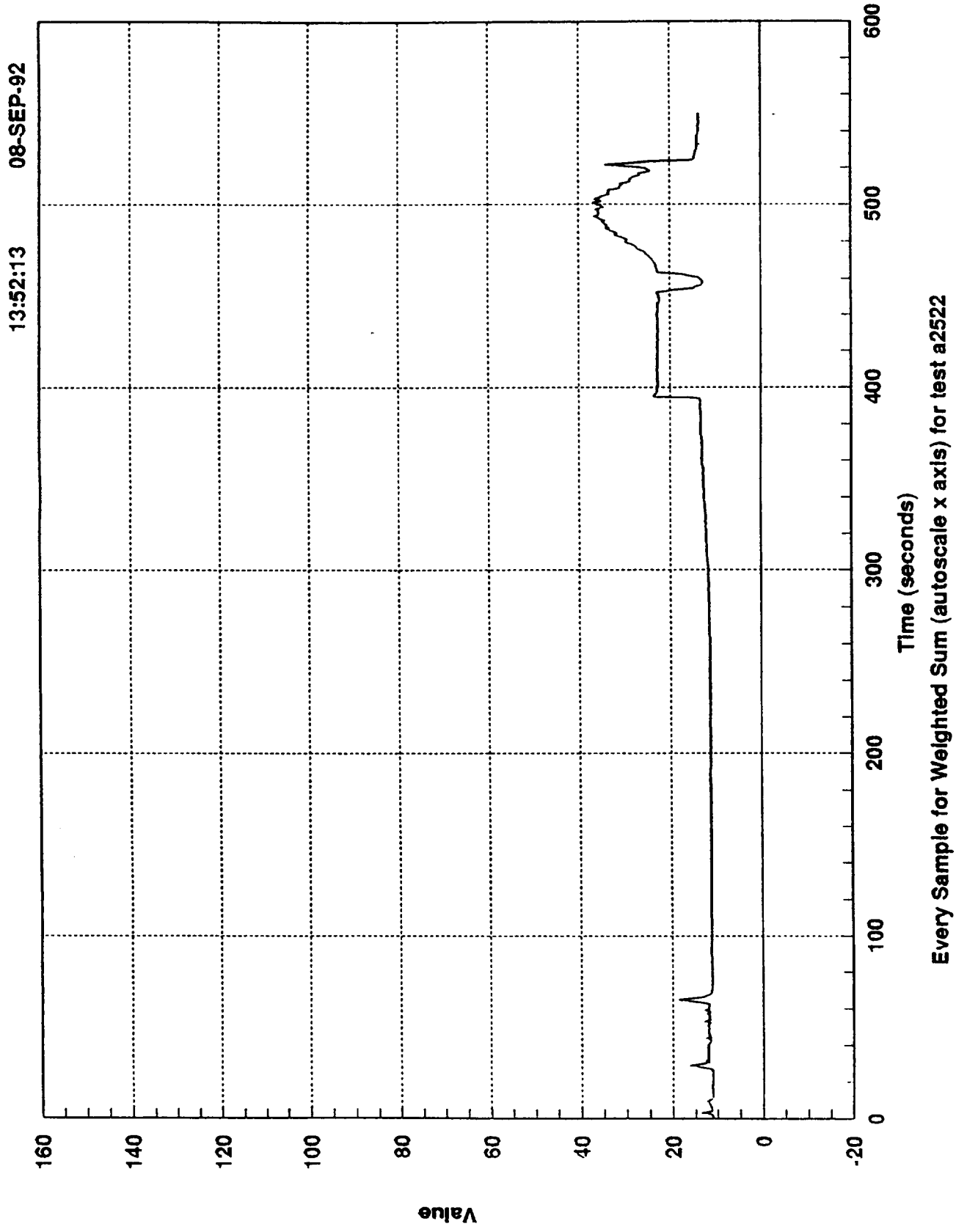

13:51:51 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
30
60
90
120
150
180
210
240
270
300
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2508

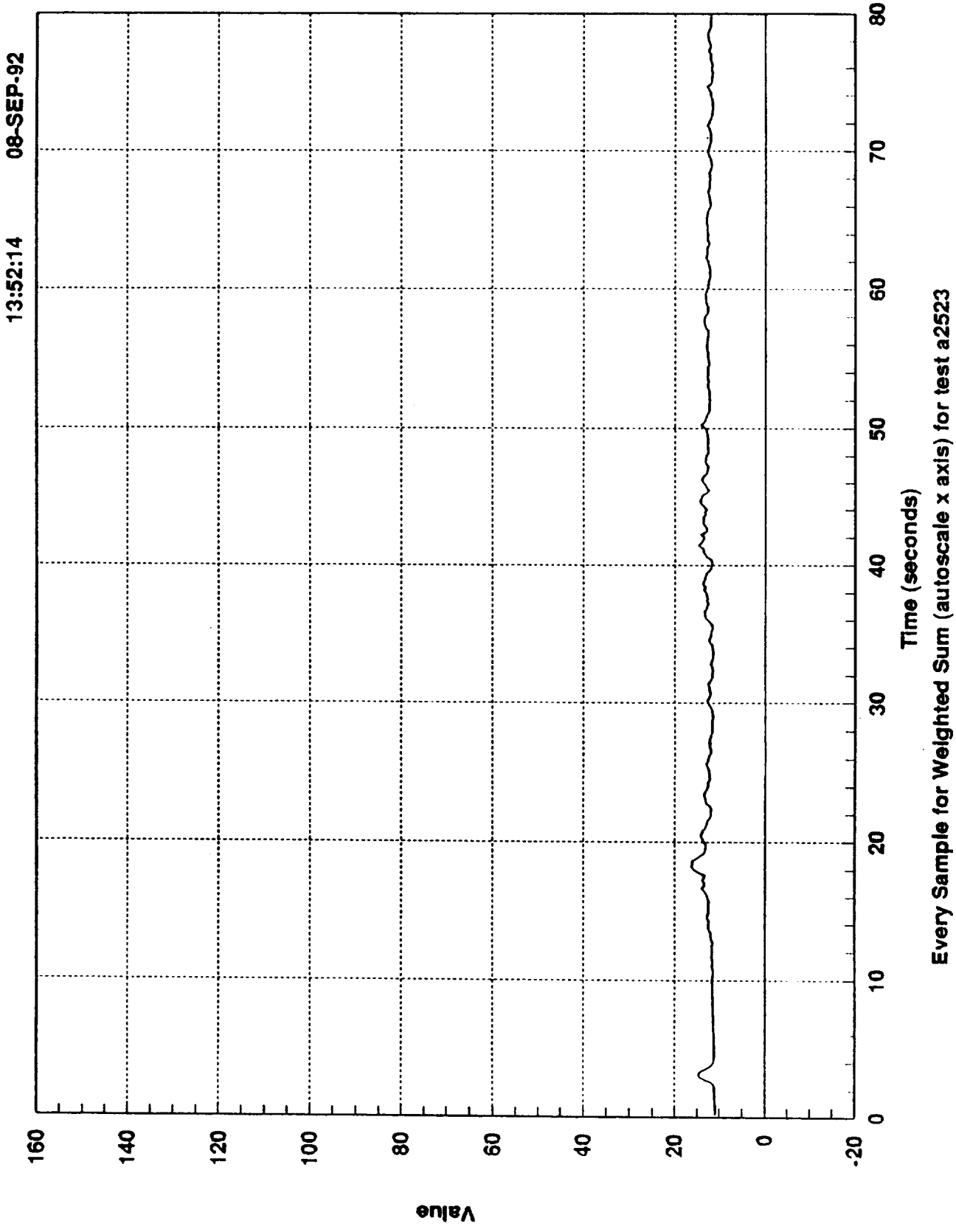

13:51:54 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
50
100
150
200
250
300
350
400
450
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2510

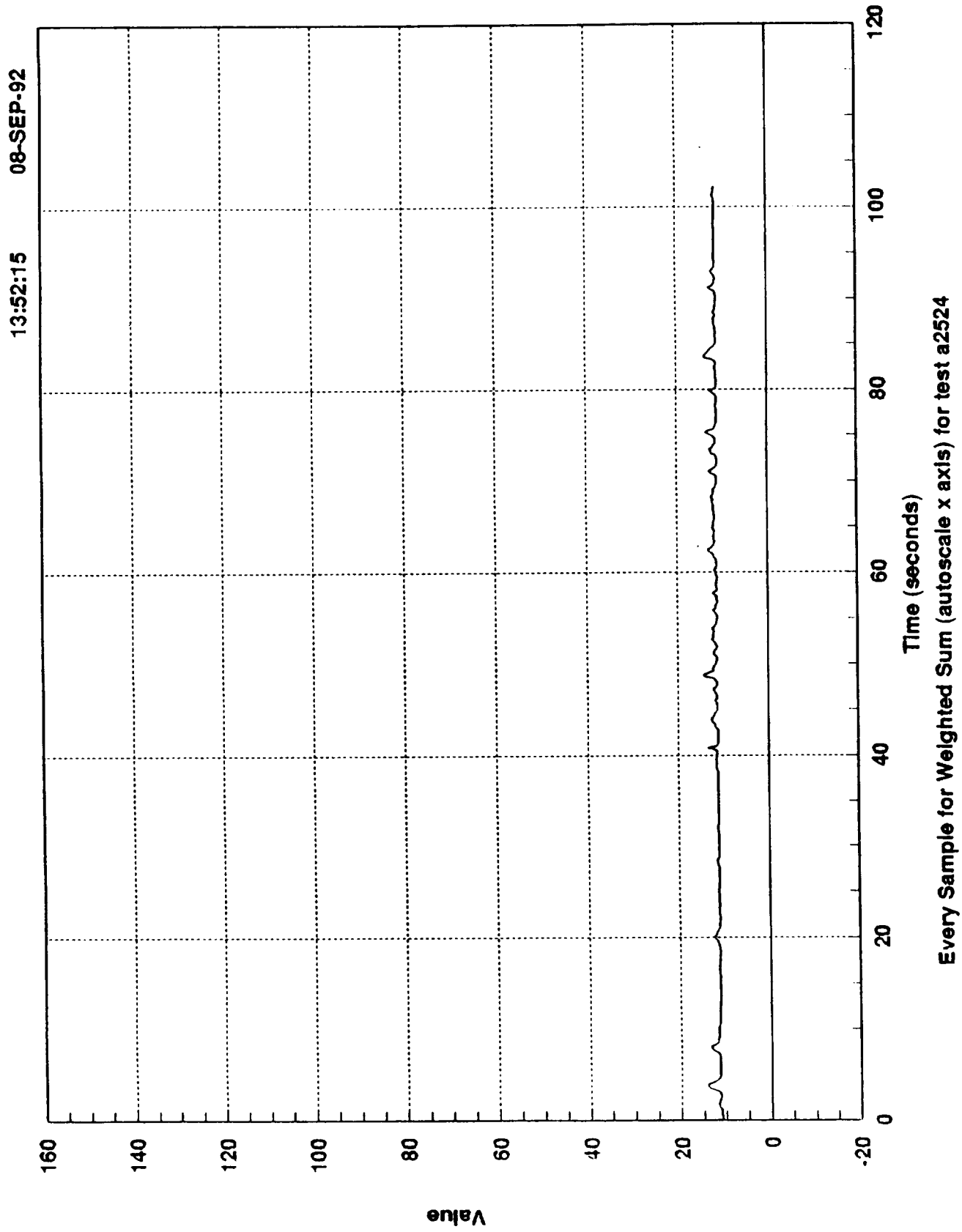

13:51:57
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2511

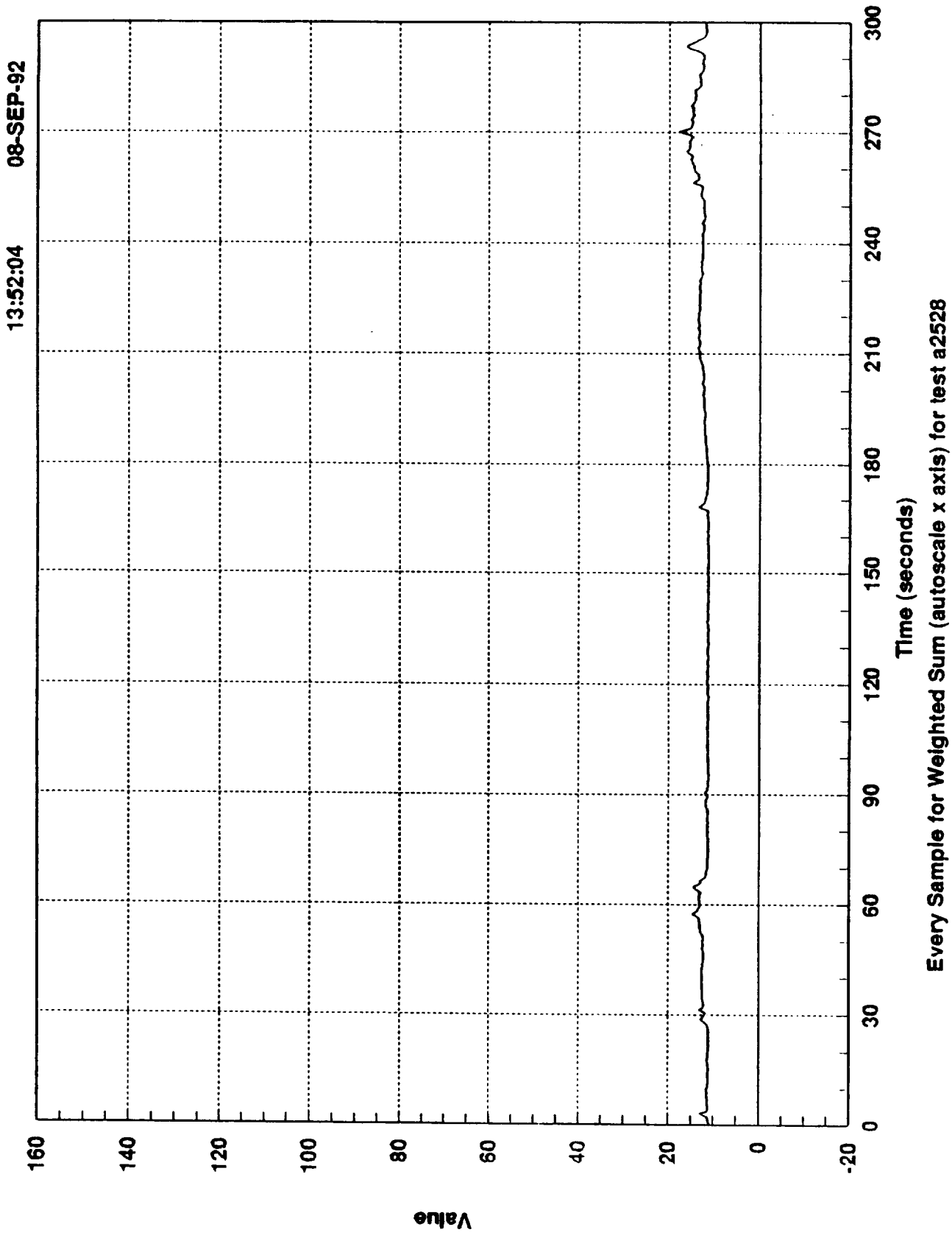

13:52:00 08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
30
60
90
120
150
180
210
240
270
300
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2516

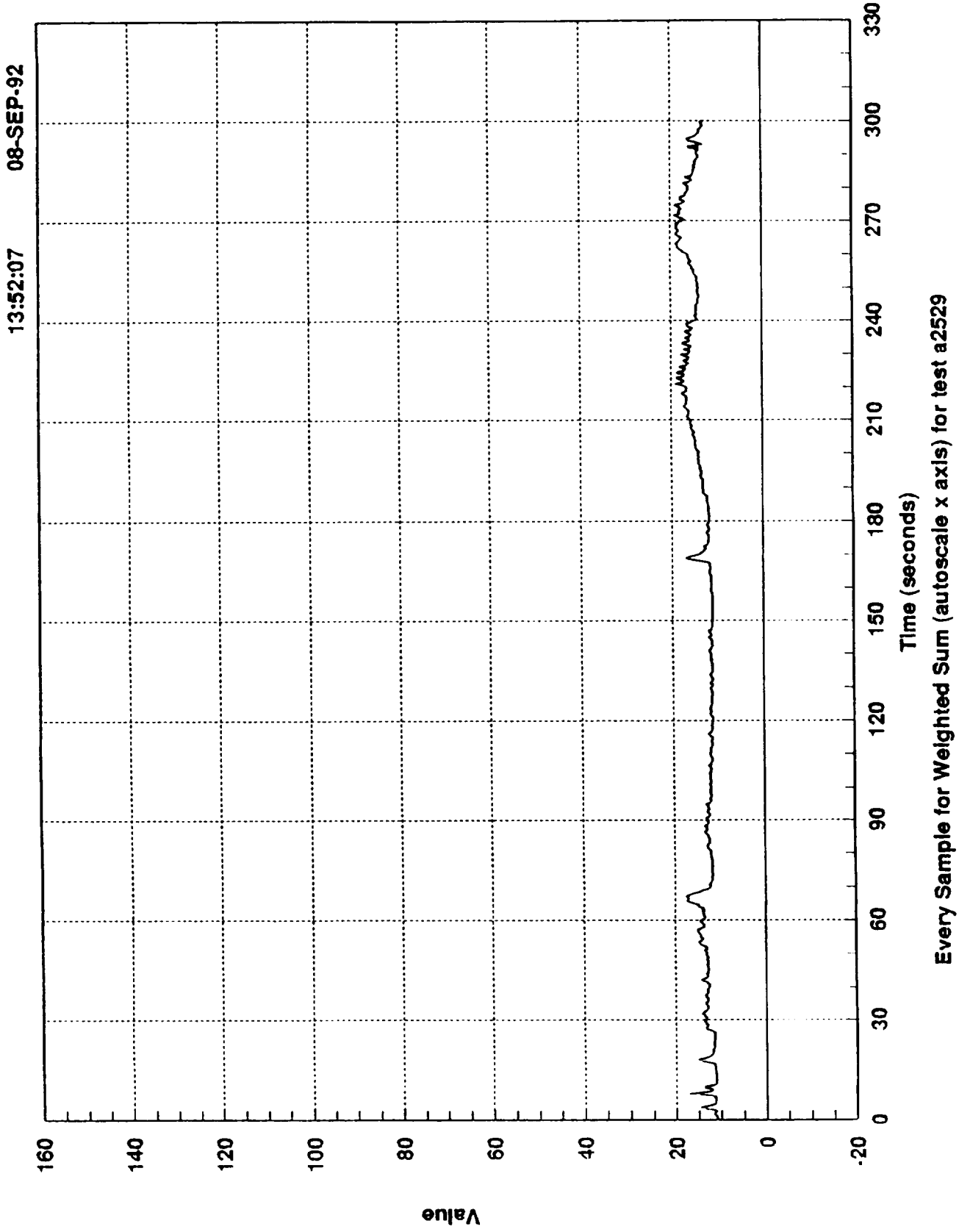

13:52:02 08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
50
100
150
200
250
300
350
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2521

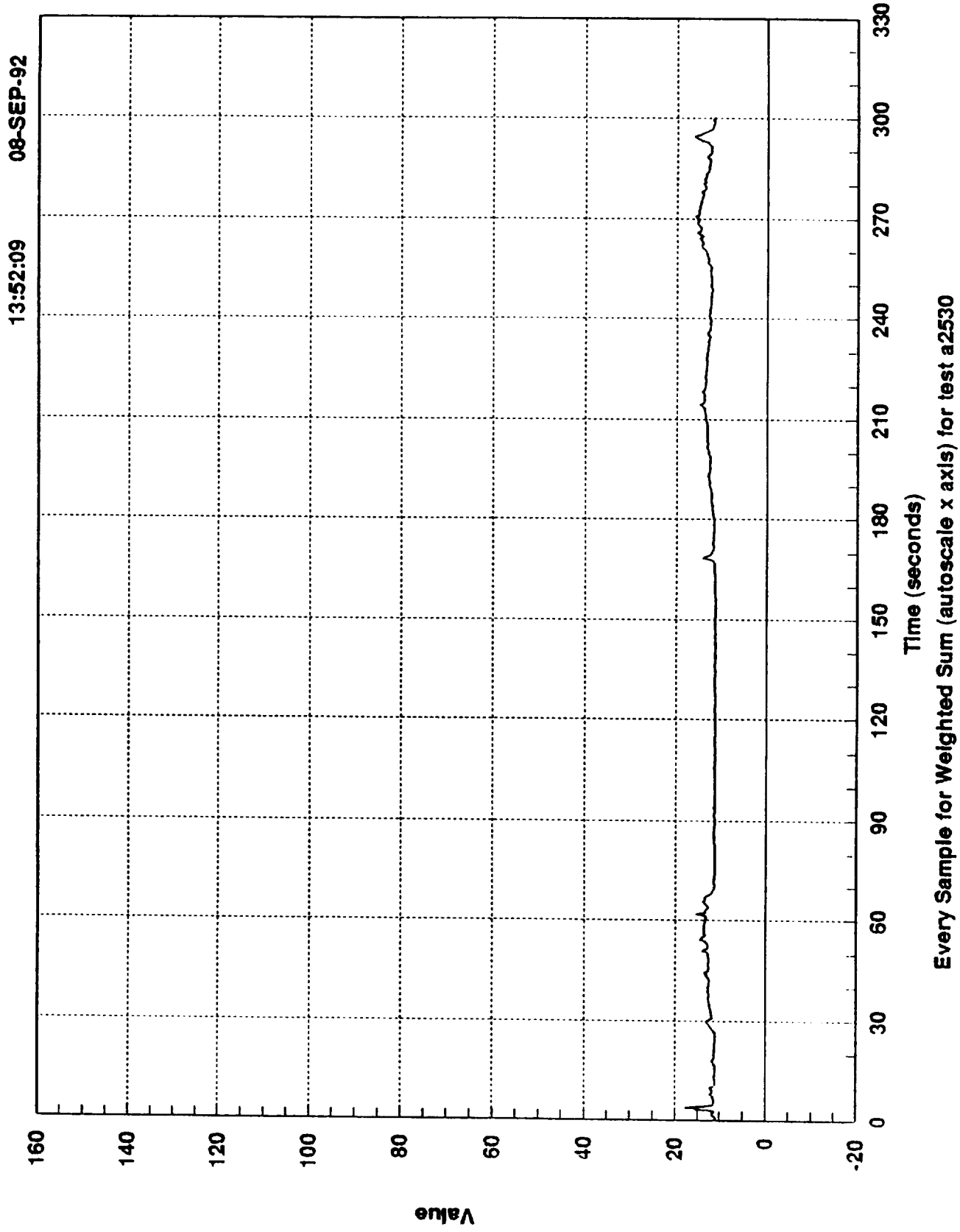

13:52:13 08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2522

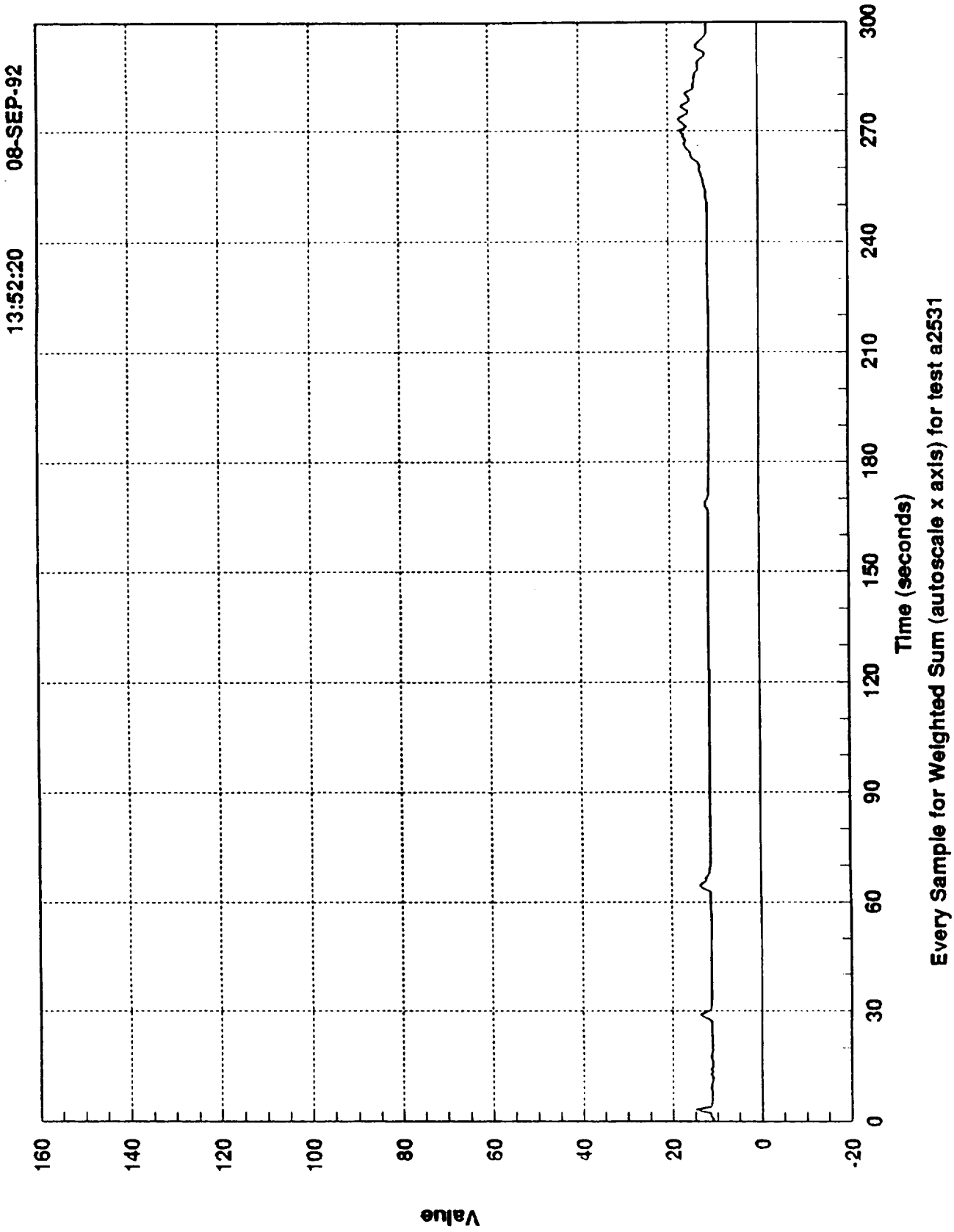

13:52:14 08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
10
20
30
40
50
60
70
80
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2523

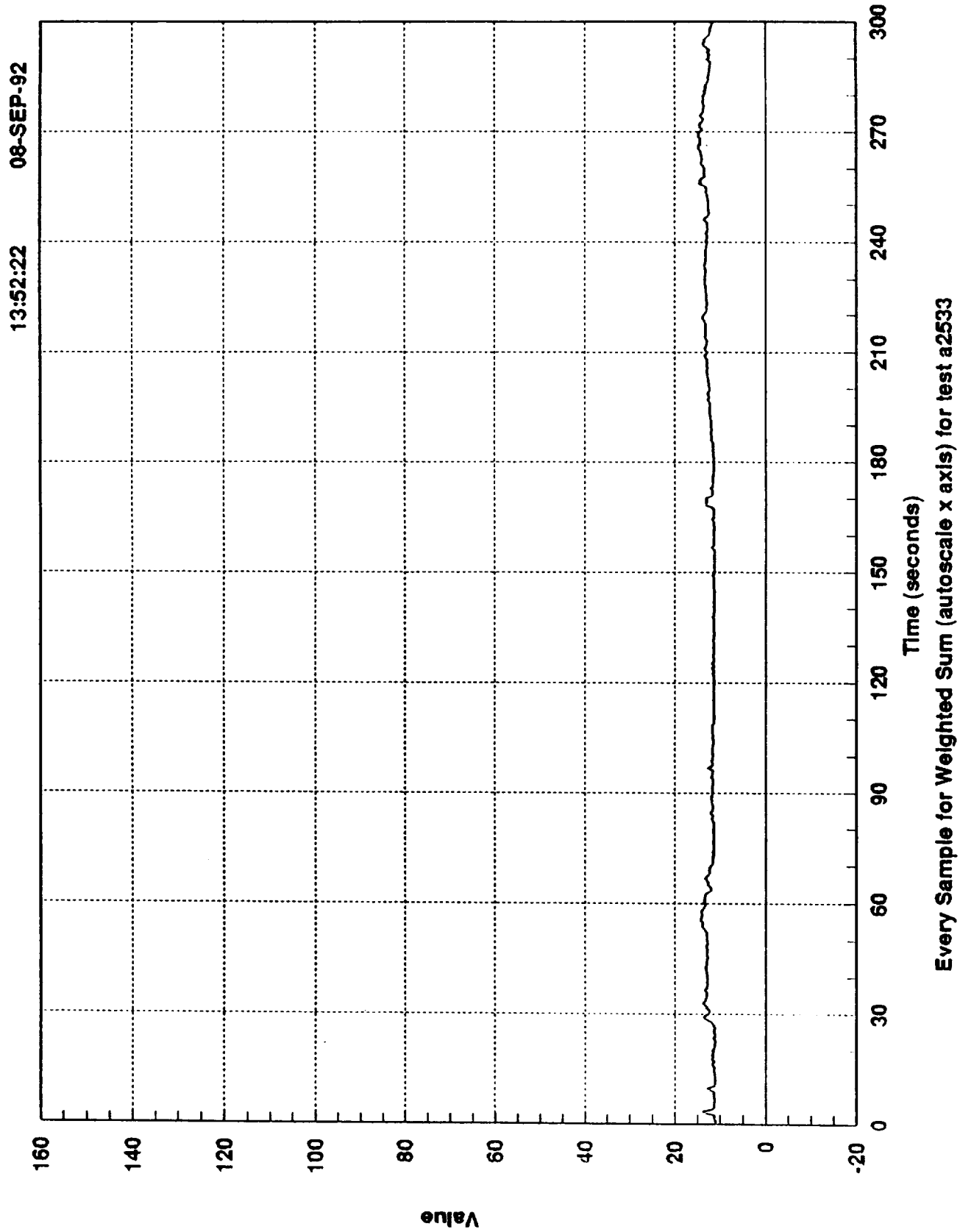

13:52:15 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
20
40
60
80
100
120
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2524

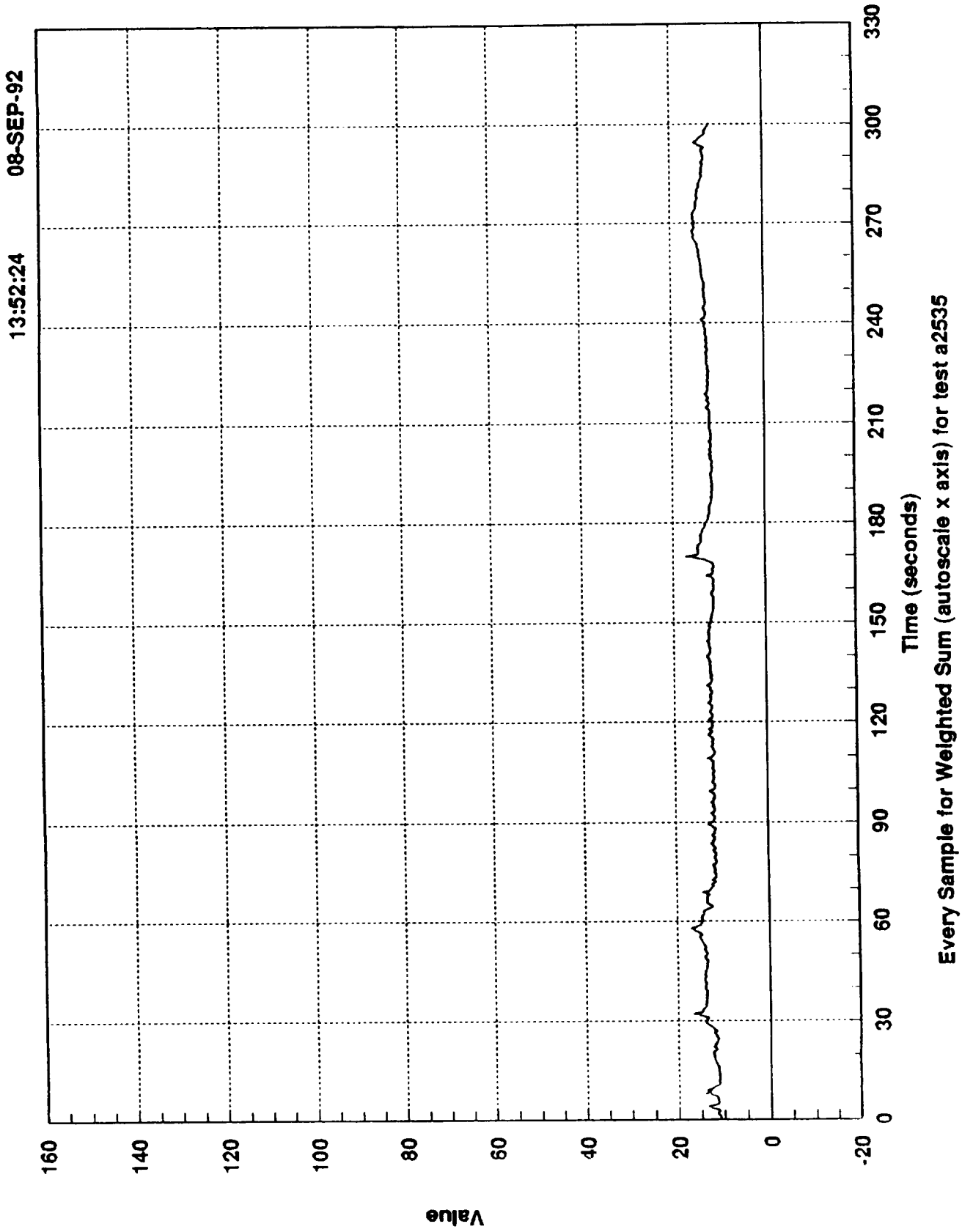

13:52:04 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
30
60
90
120
150
180
210
240
270
300
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2528

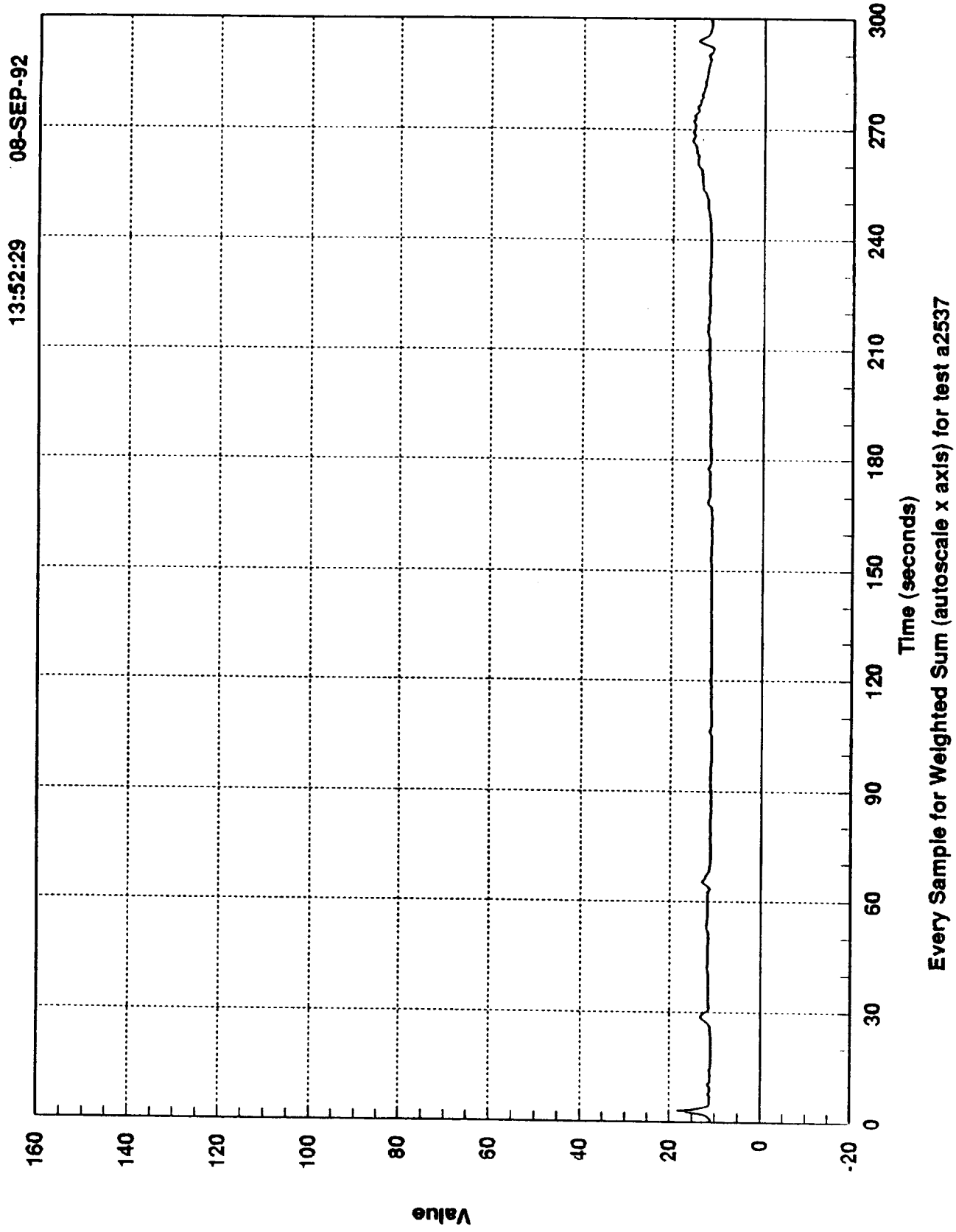

13:52:07
08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
30
60
90
120
150
180
210
240
270
300
330
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2529

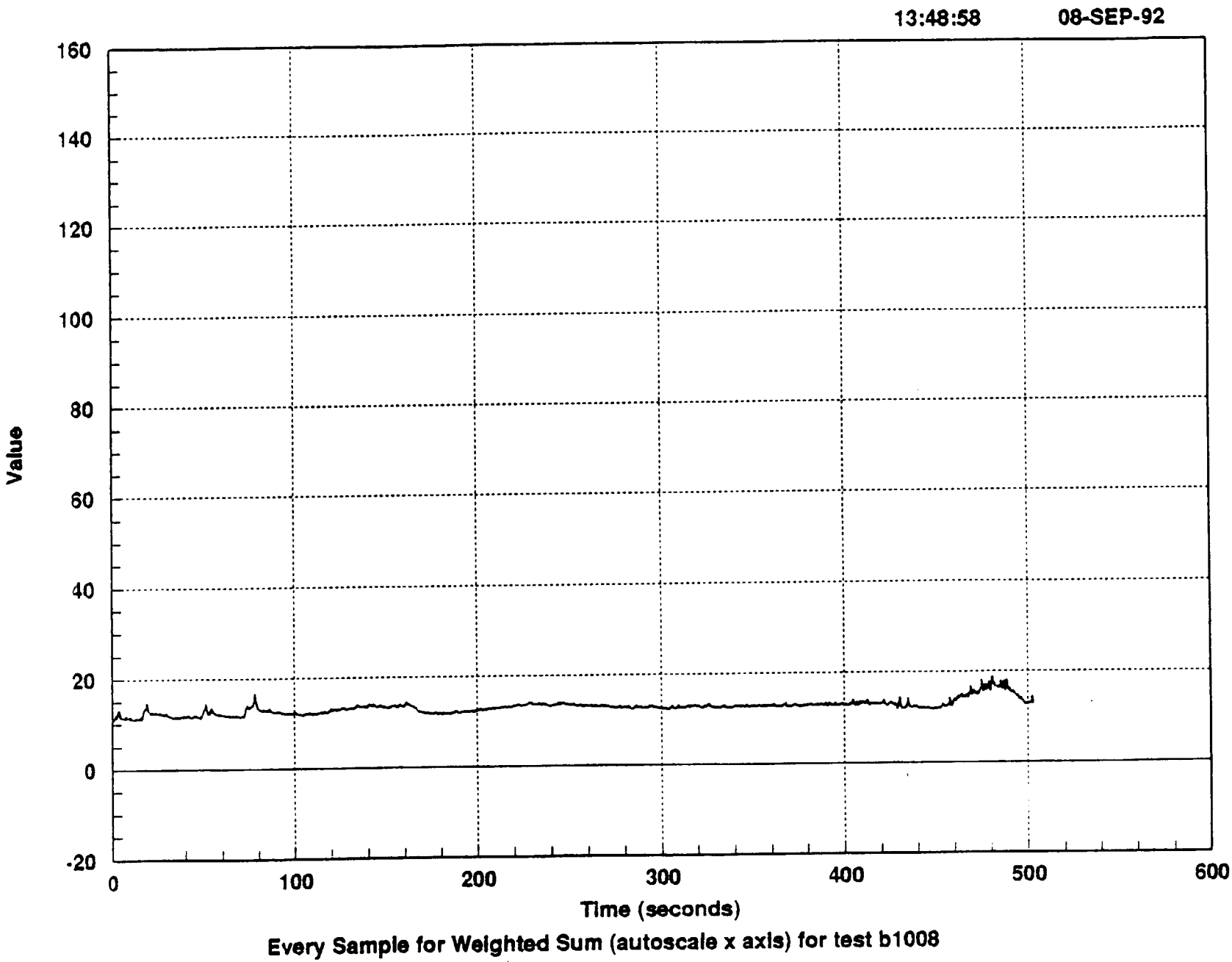

13:52:09
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
30
60
90
120
150
180
210
240
270
300
330
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2530


13:52:20 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
30
60
90
120
150
180
210
240
270
300
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2531

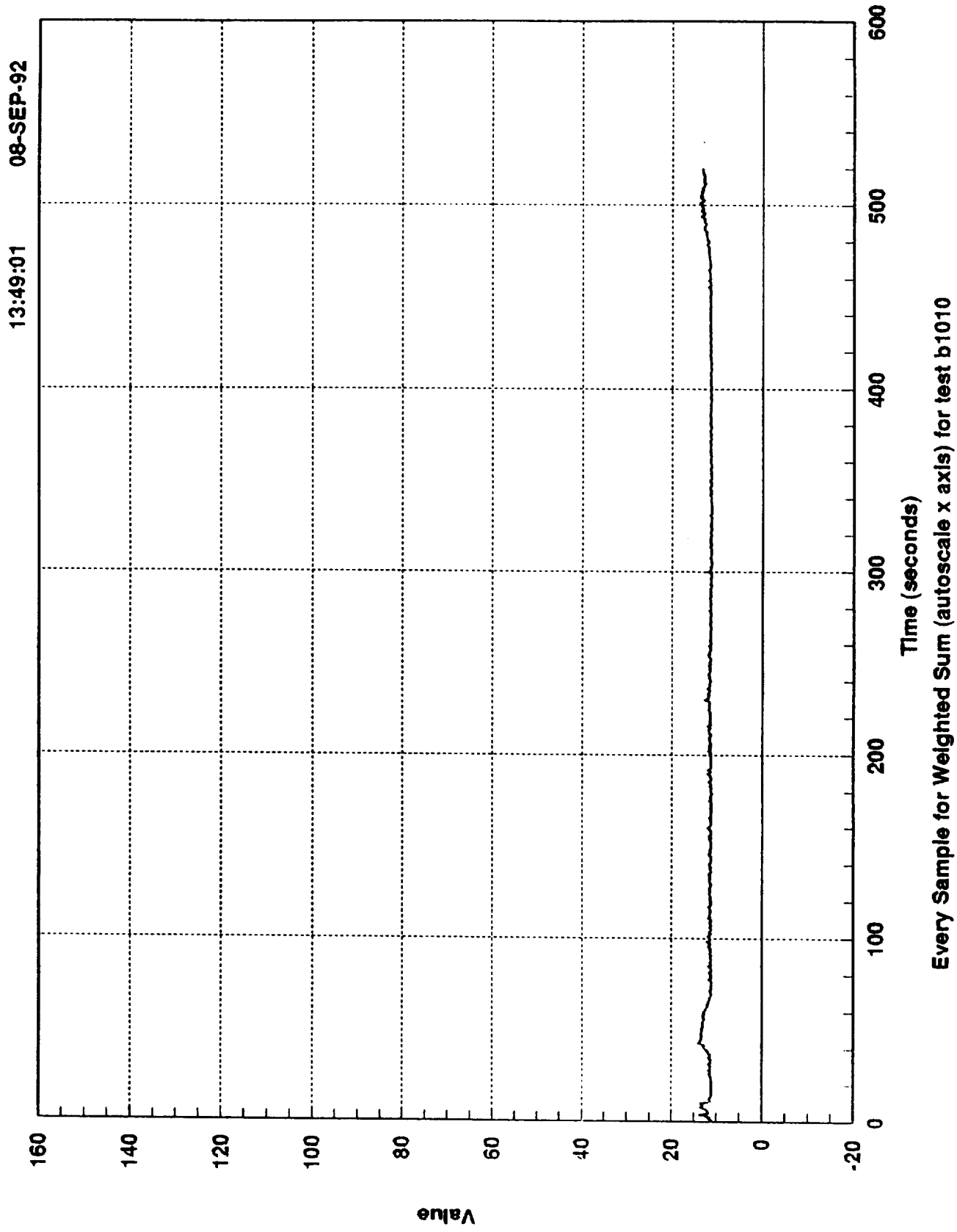

13:52:22
08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
30
60
90
120
150
180
210
240
270
300
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2533

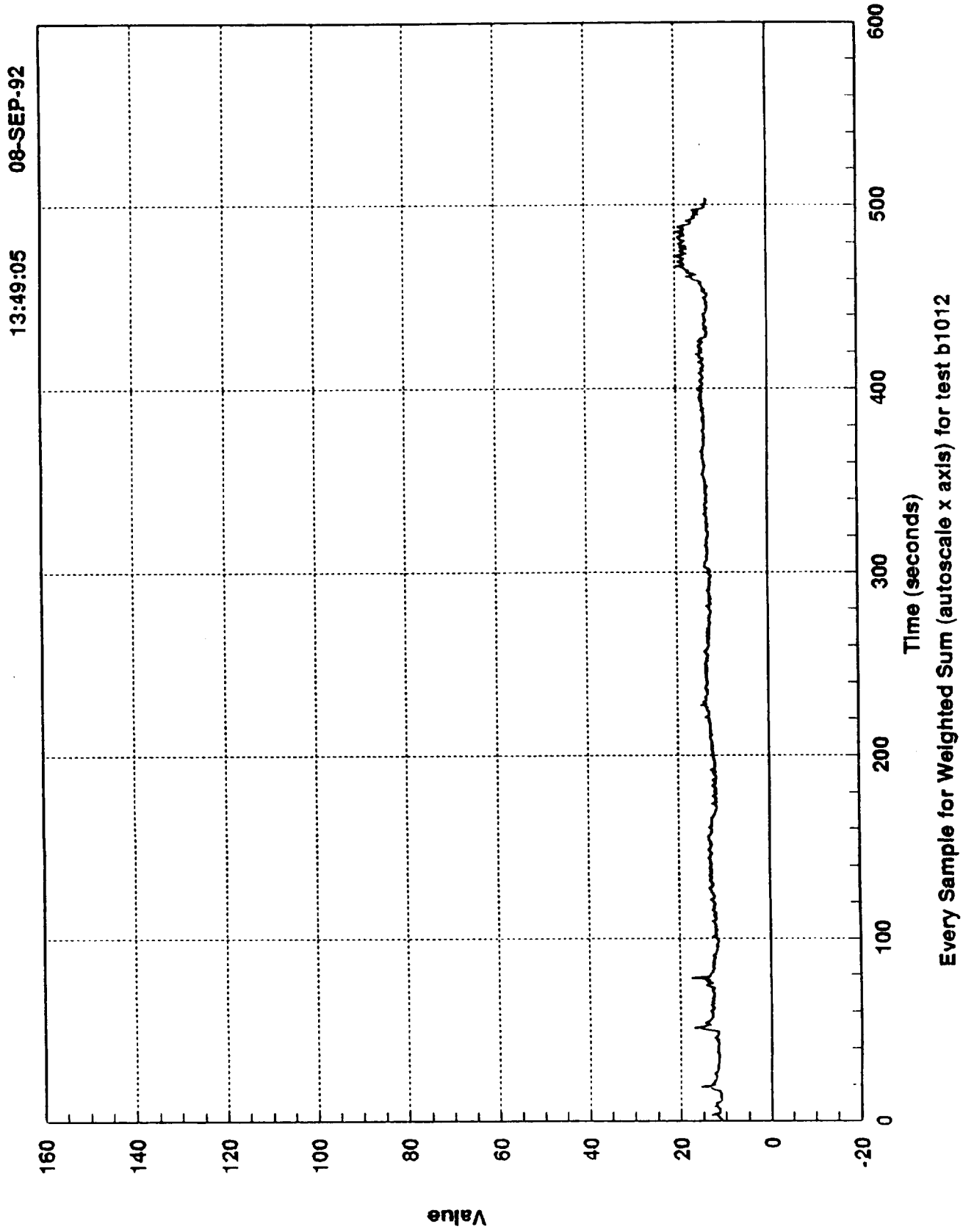

13:52:24 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
30
60
90
120
150
180
210
240
270
300
330
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2535

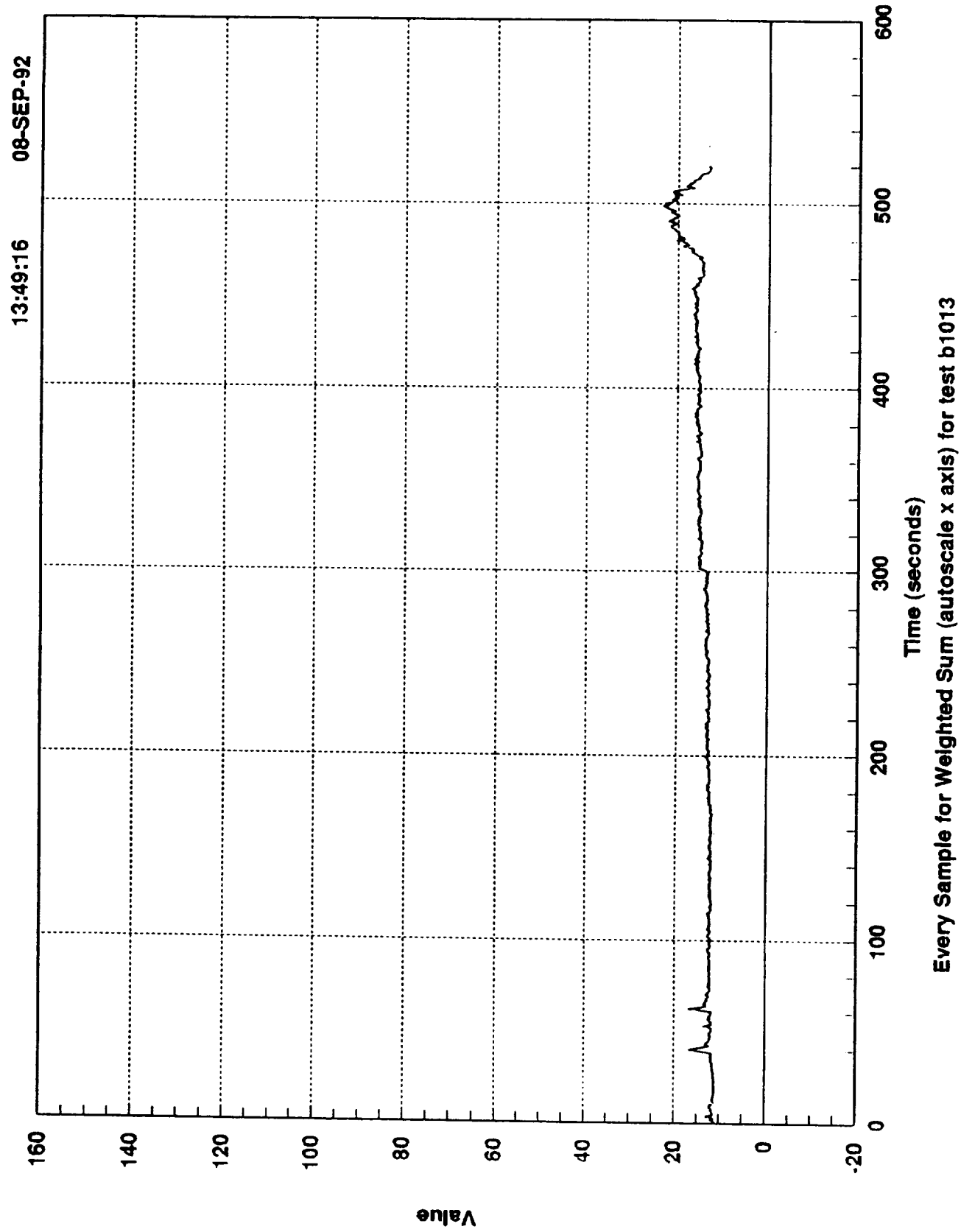

13:52:29
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
30
60
90
120
150
180
210
240
270
300
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test a2537

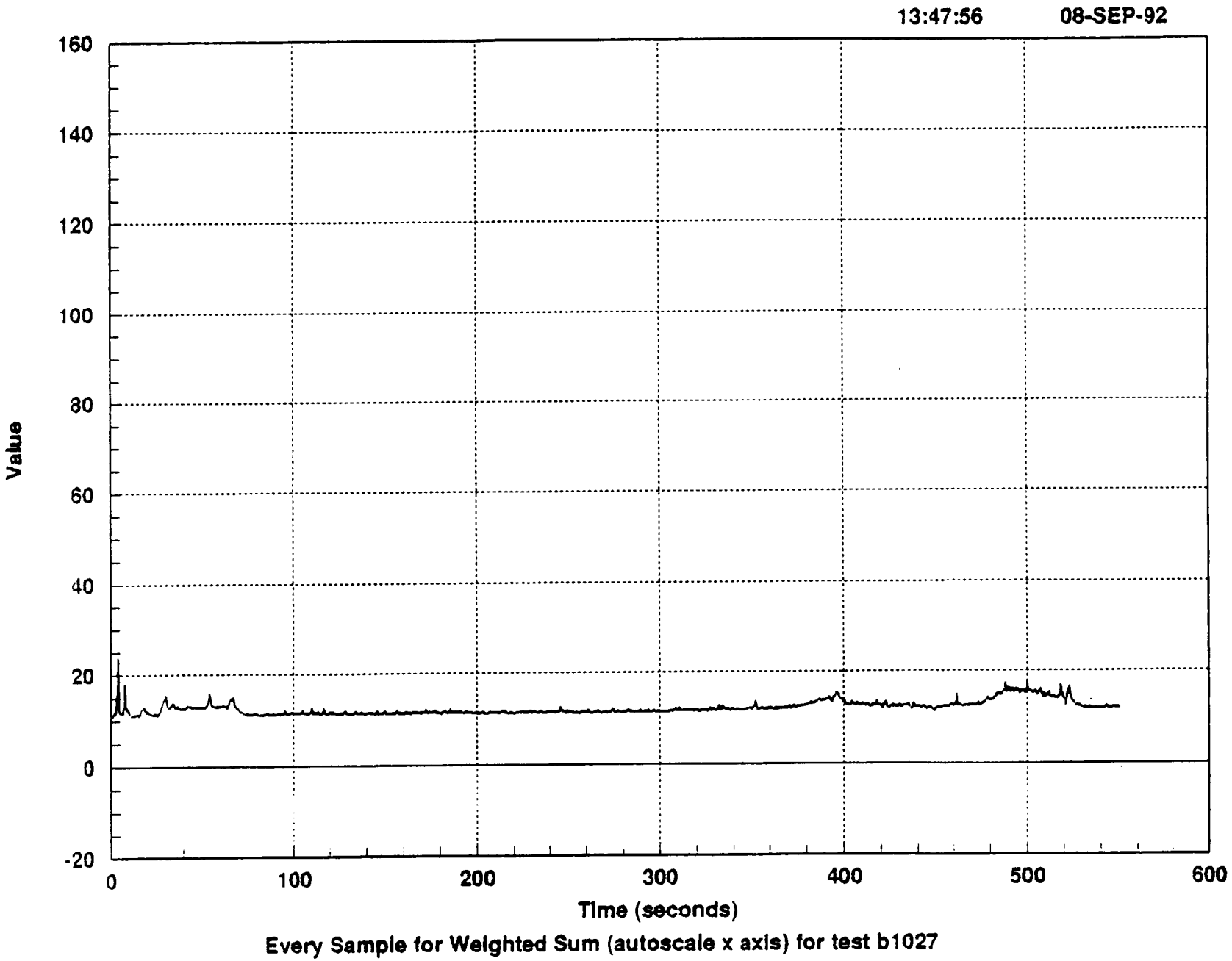


Every Sample for Weighted Sum (autoscale x axis) for test b1008

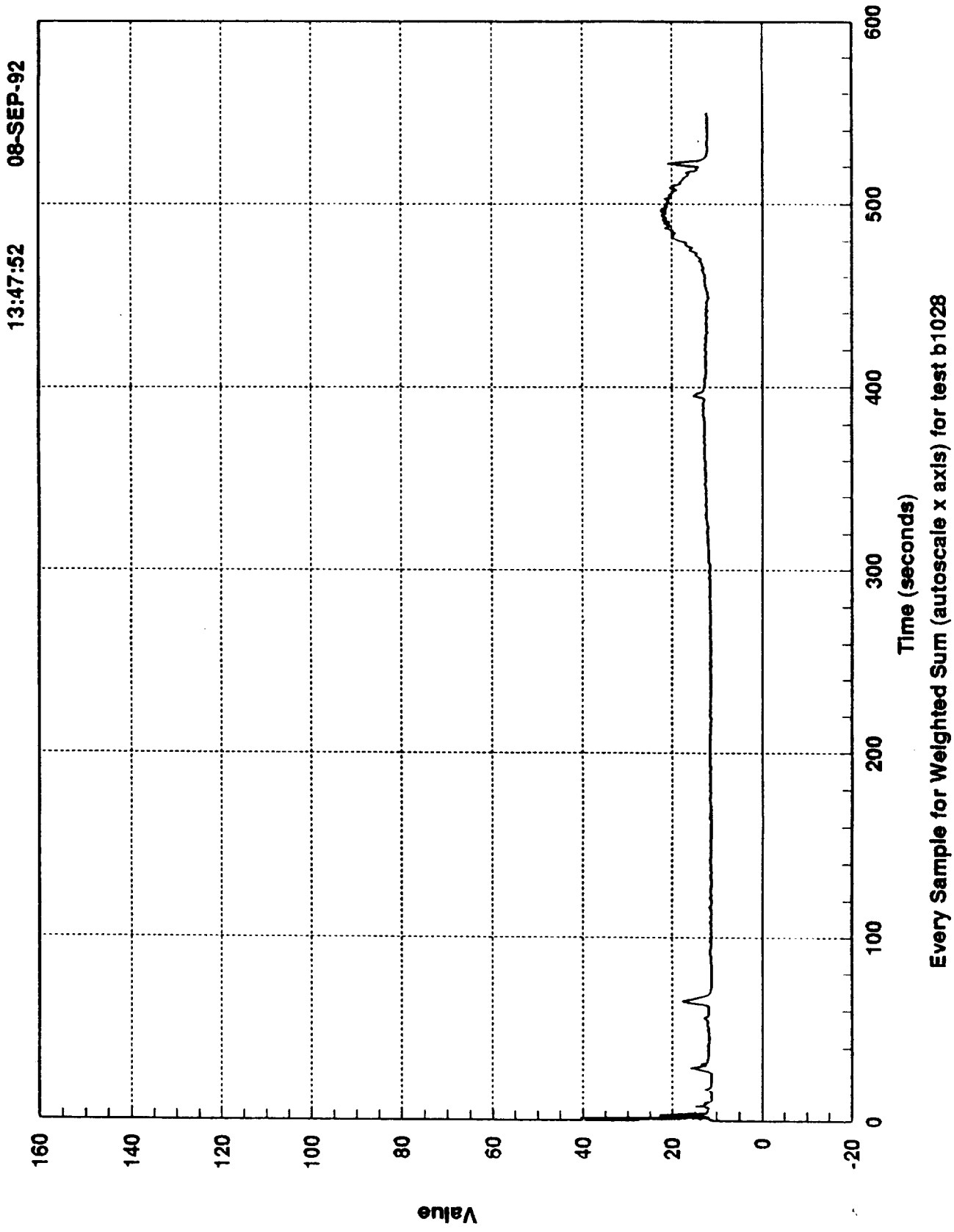

13:49:01 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test b1010

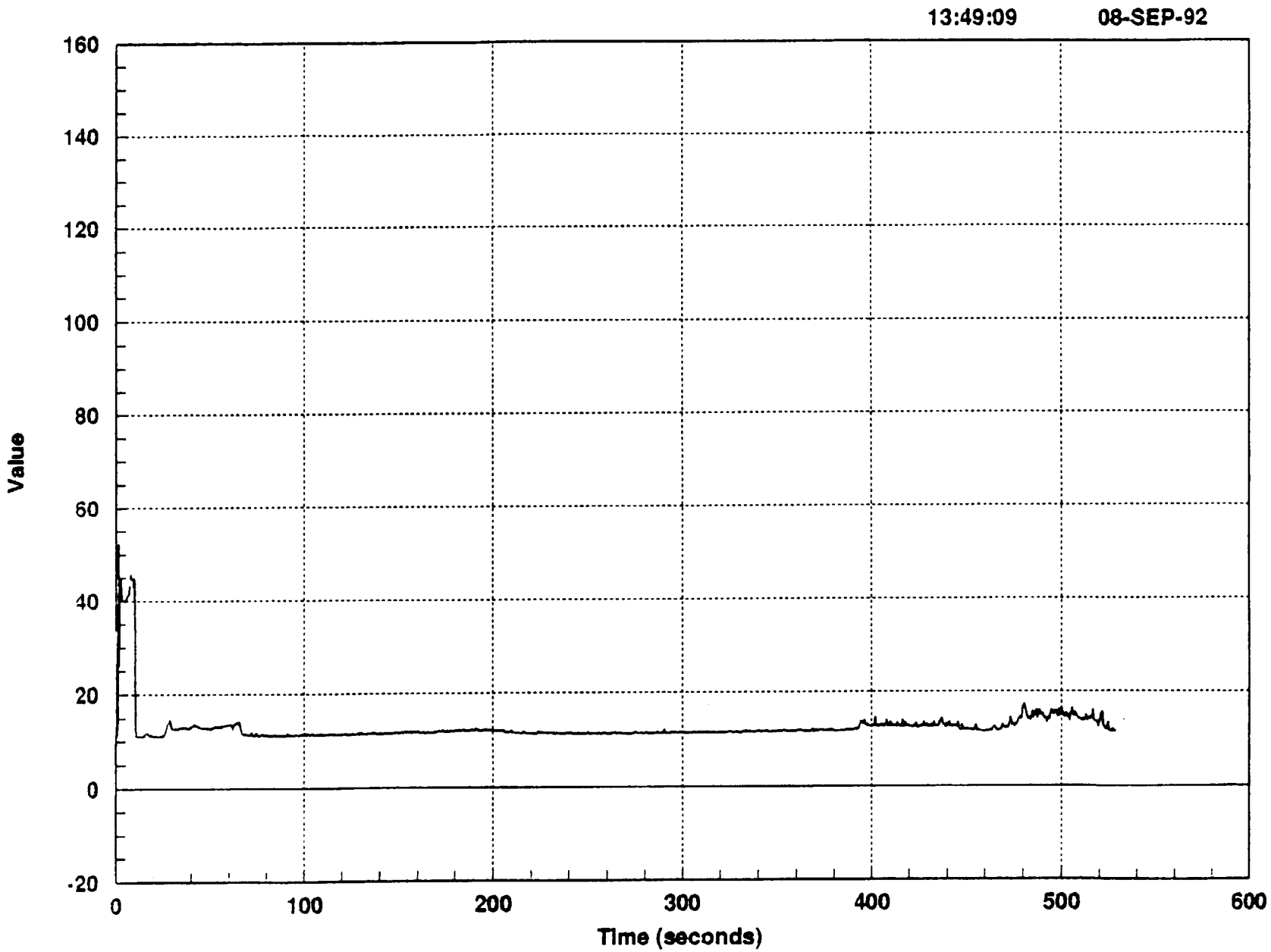

13:49:05 08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test b1012


13:49:16
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test b1013

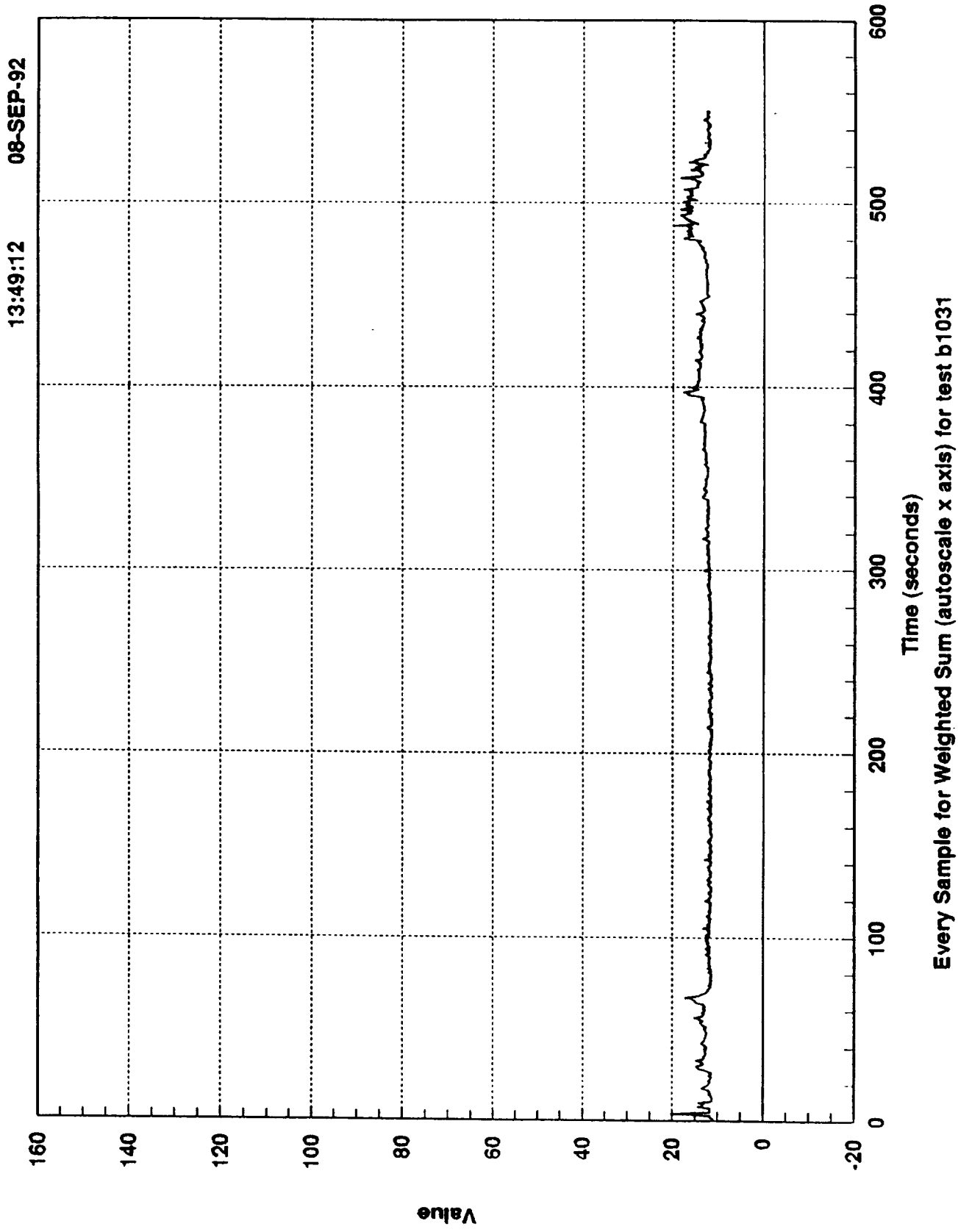


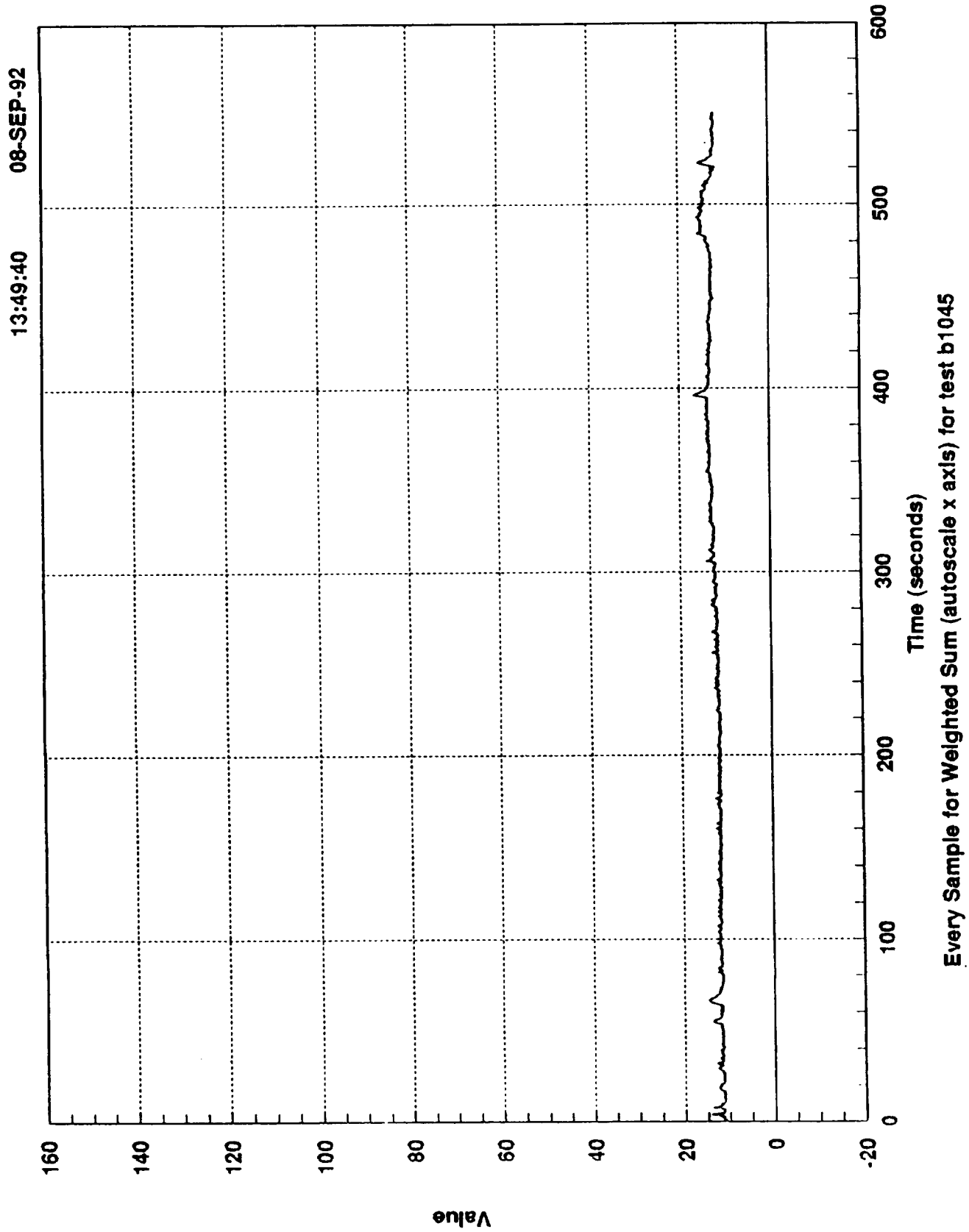
Every Sample for Weighted Sum (autoscale x axis) for test b1027

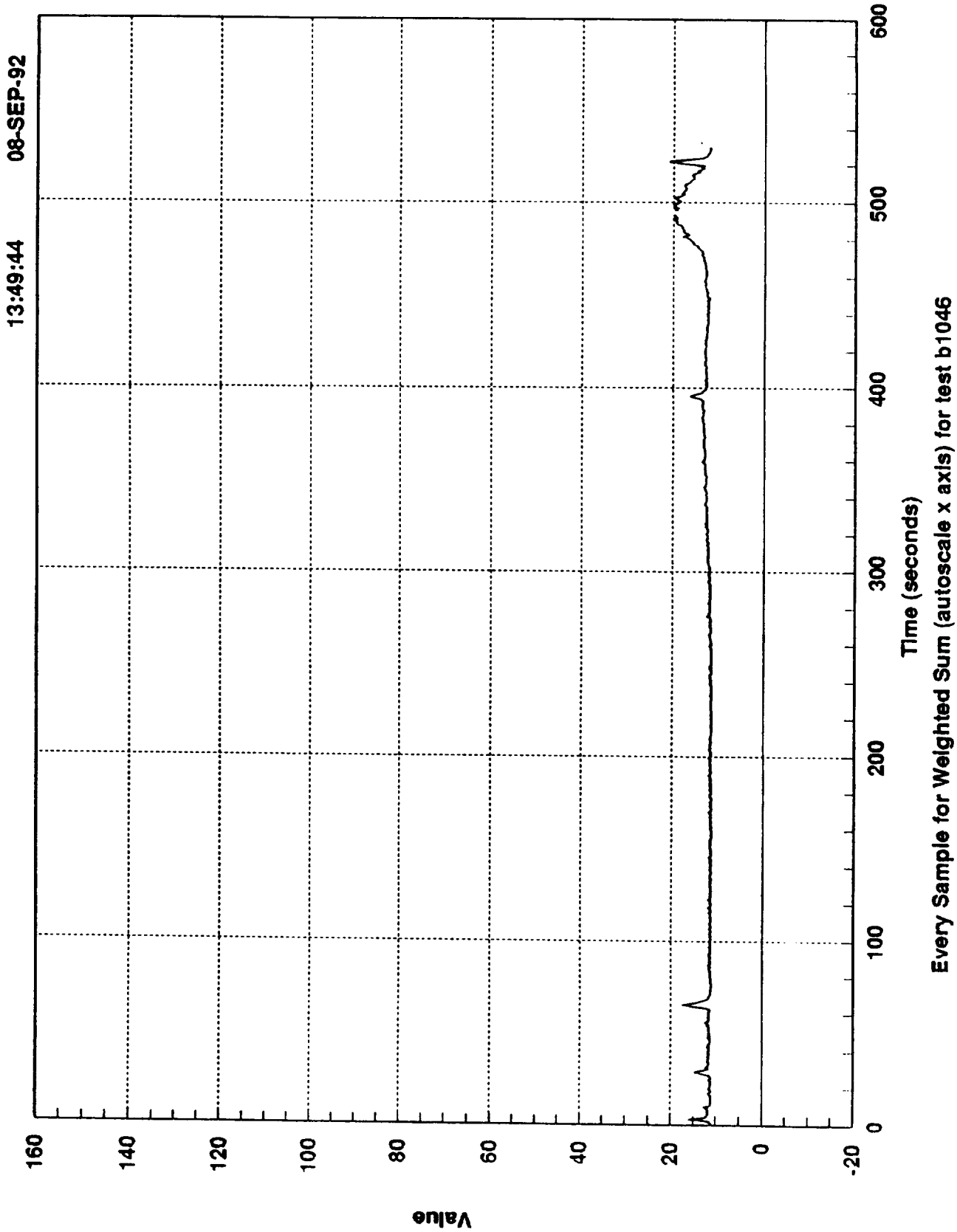

13:47:52
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test b1028

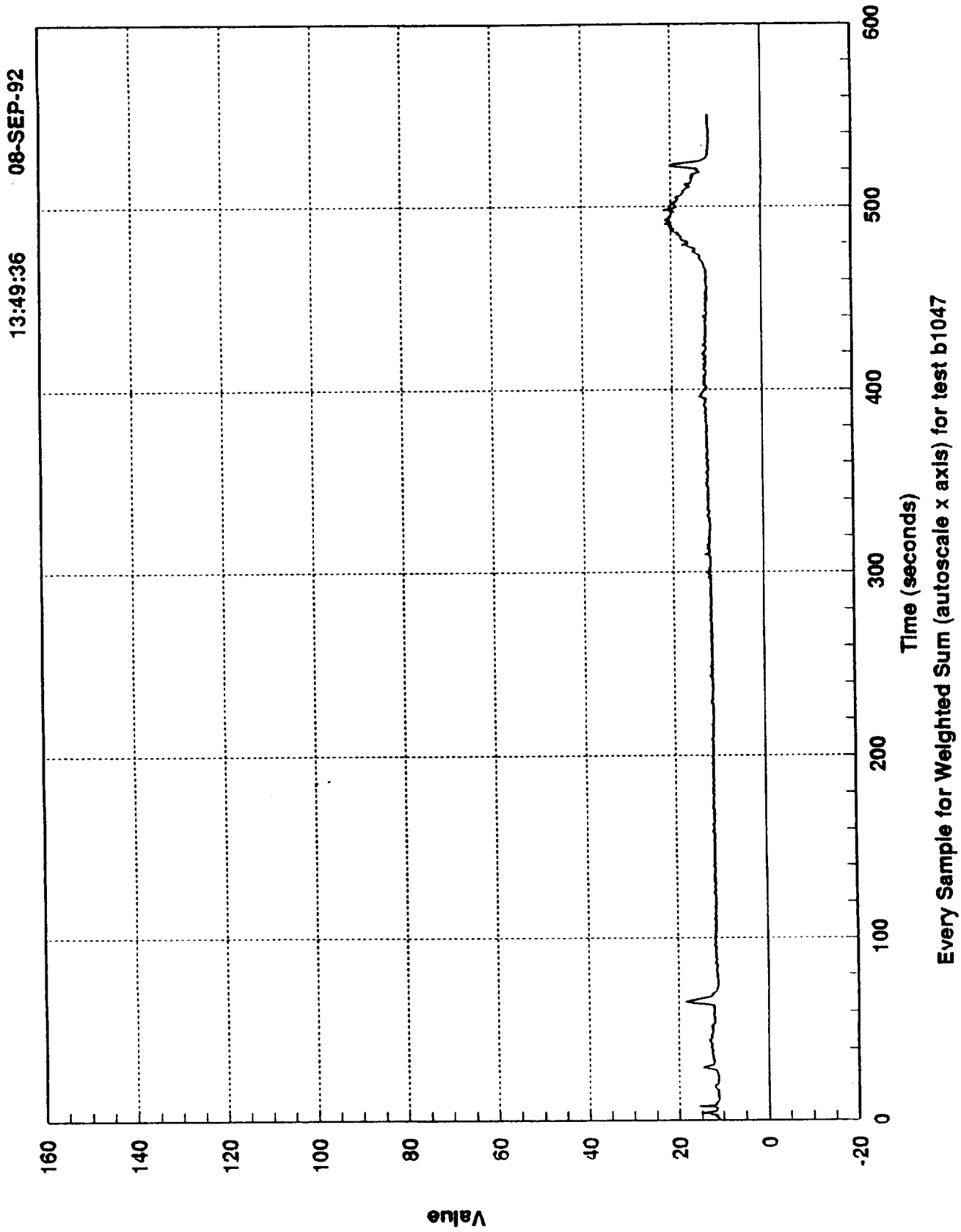


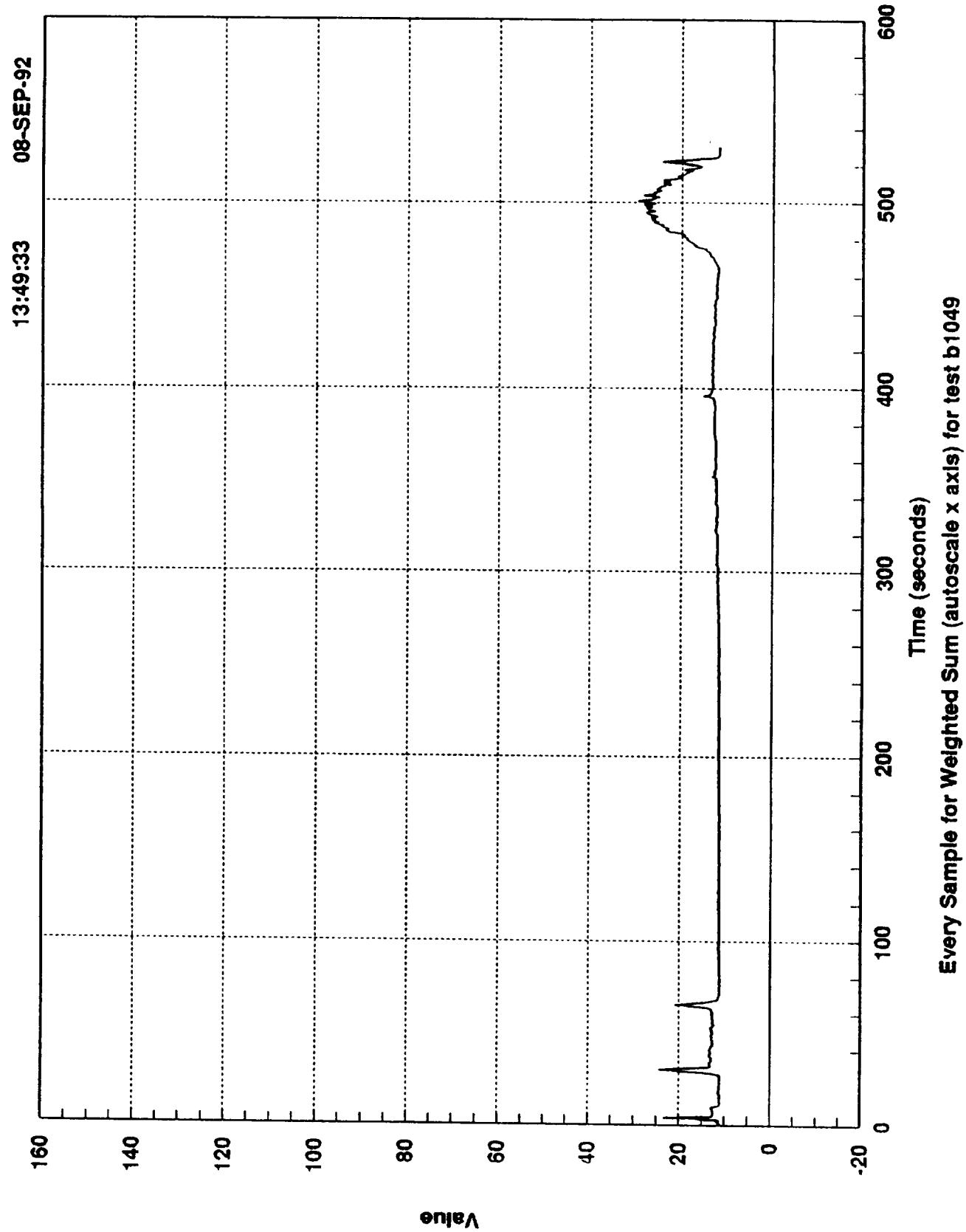
Every Sample for Weighted Sum (autoscale x axis) for test b1030

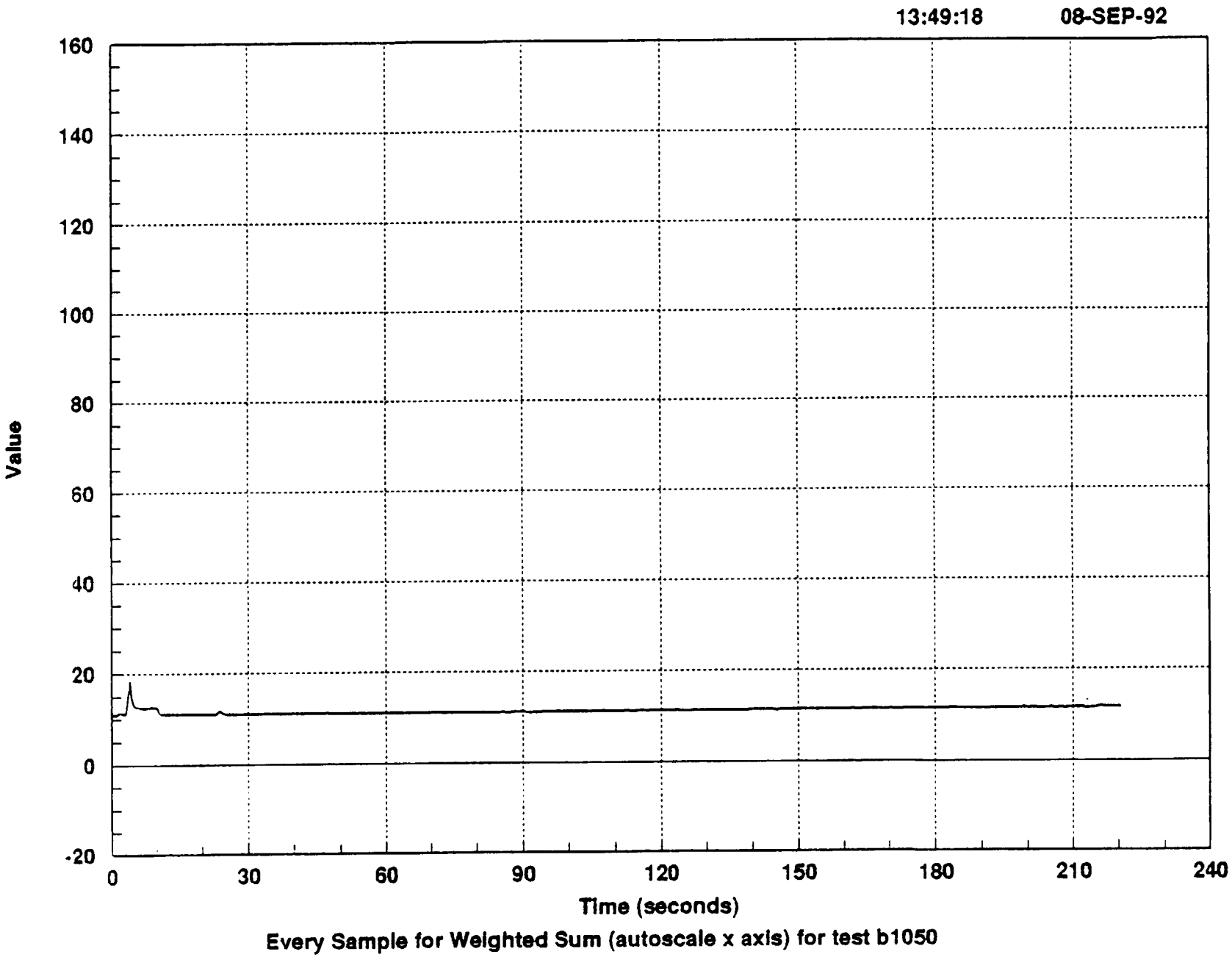

13:49:12
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test b1031

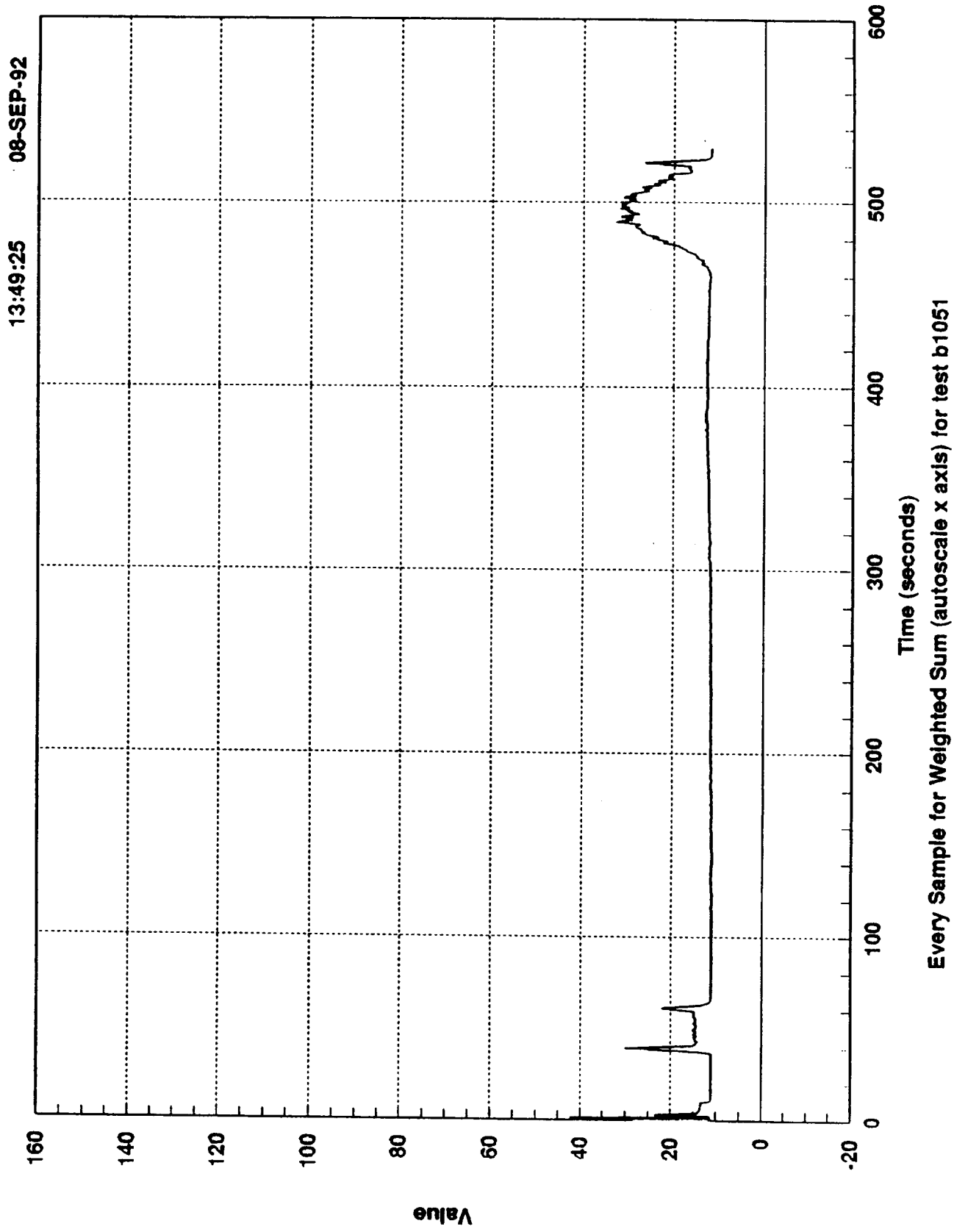

13:49:40
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test b1045

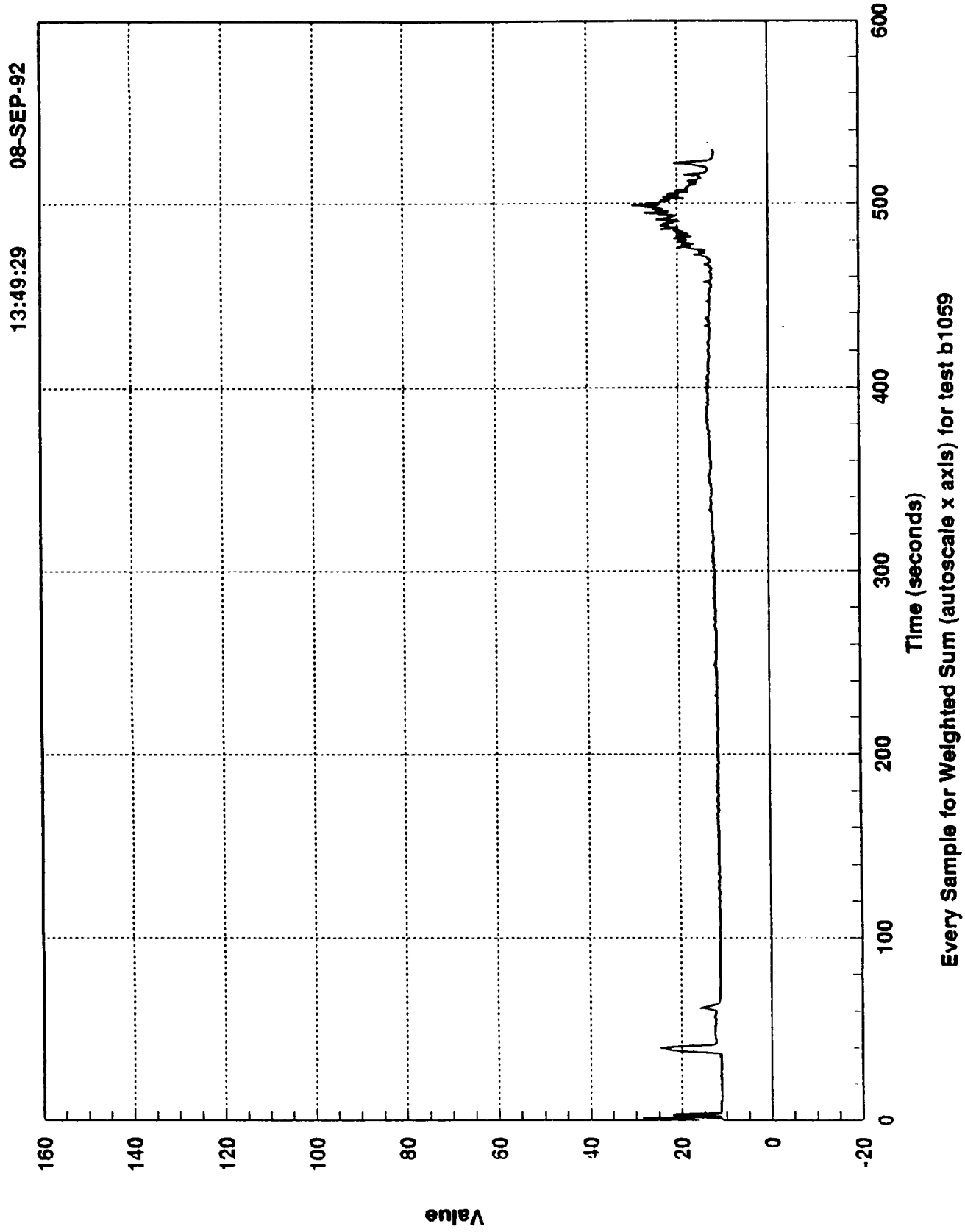

13:49:44
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test b1046

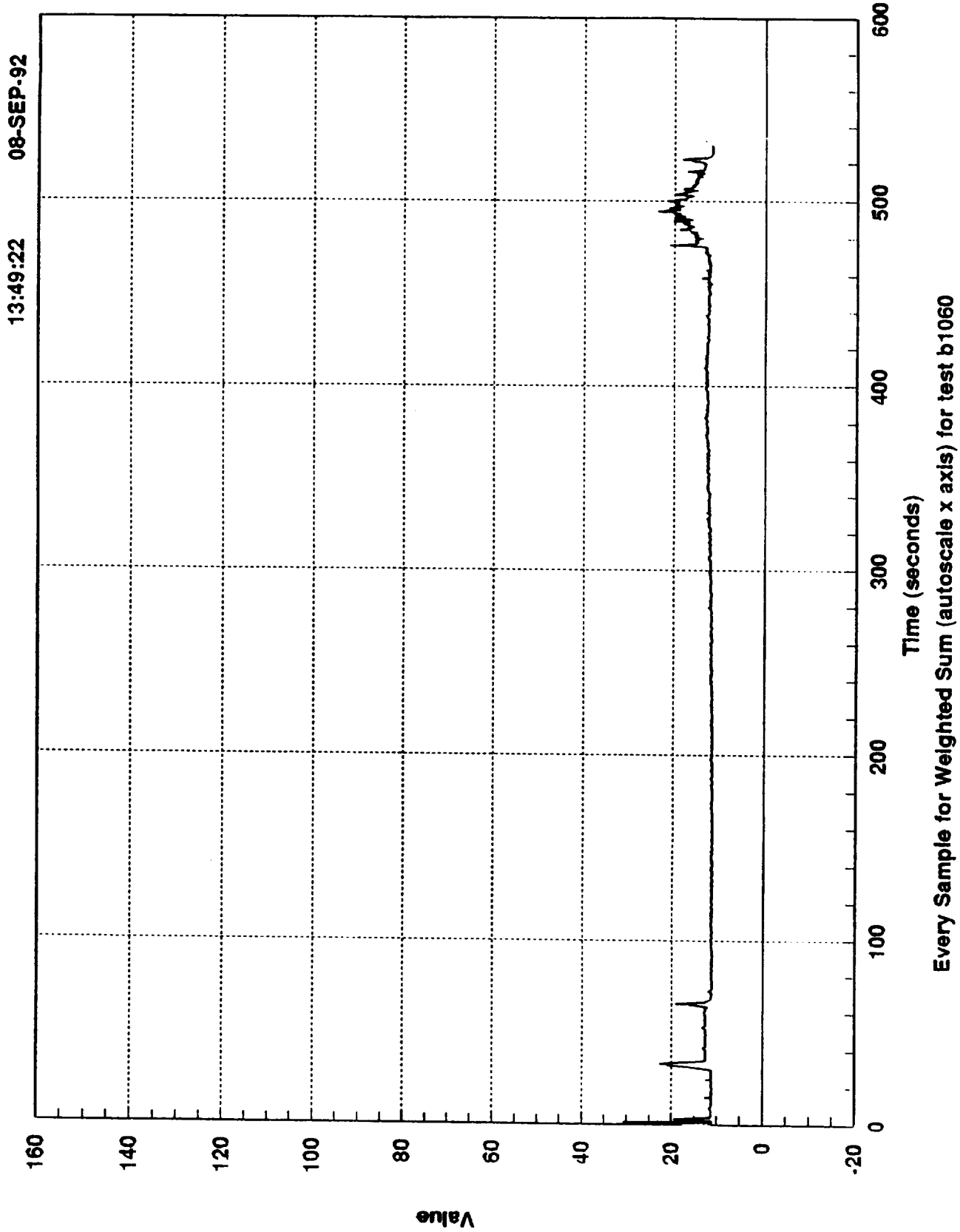

13:49:36
08-SEP-92
Value
160
140
120
100
80
60
40
20
0
-20
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test b1047


13:49:33 08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test b1049


13:49:18
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
30
60
90
120
150
180
210
240
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test b1050


13:49:25 08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test b1051


13:49:29
08-SEP-92
160
140
120
100
80
60
40
20
0
-20
Value
0
100
200
300
400
500
600
Time (seconds)
Every Sample for Weighted Sum (autoscale x axis) for test b1059


13:49:22 08-SEP-92
Every Sample for Weighted Sum (autoscale x axis) for test b1060
Time (seconds)
Value
160
140
120
100
80
60
40
20
0
-20
0
100
200
300
400
500
600

APPENDIX B
RSA CODE

```
#include <math.h>

/*****************************************************************************
*                                                                            *
*                                                                            *
*                       ROCKETDYNE SAFETY ALGORITHM                          *
*                                                                            *
*                         ALGORITHM MODULE INIT()                            *
*                                                                            *
*   PURPOSE:                                                                 *
*                                                                            *
*   This module initializes the local variables of the RSA module. It is     *
*   called prior to processing a test with RSA. The SAFD platform requires   *
*   a separate init() routine in addition to the RSA main module, thus       *
*   this code.                                                               *
*                                                                            *
*   This module does nothing except tell RSA to initialize itself.           *
*                                                                            *
*****************************************************************************/



/*****************************************************************************
******************************************************************************
*                                                                            *
*                                                                            *
*                       ROCKETDYNE SAFETY ALGORITHM                          *
*                                                                            *
*                         ALGORITHM MODULE RSA()                             *
*                                                                            *
*   PURPOSE:                                                                 *
*                                                                            *
*   This module contains the code for the RSA. The interface to this module *
*   from the outside world is the parameter list ONLY.                       *
*                                                                            *
*****************************************************************************/

/*****************************************************************************/
/**                                                                         **/
/**                       RSA FLOW-CONTROL VARIABLES                        **/
/**            (not directly used for calculation of redlining sum)         **/
/**            (Used to track power levels, keep track of time, etc.)       **/
/**                                                                         **/
/*****************************************************************************/

 /* Numeric constants needed for data structure sizes: */

    #define MAX 80                  /* constants for data structure sizes */
    #define ARRAY_MAX 2500          /* Mostly used in WSUM() routine.     */
    #define max_adt 100             /* and support subs (engine model)    */
    #define max_est 22
    #define max_depnom 22

 /* Sampling is done to average data for the power level signature. */
 /* This signature is otherwise known as the "initialization point" */
 /* These variables define sampling times in seconds:               */
```

```
#include <math.h>

/*****************************************************************************
*                                                                            *
*                                                                            *
*                       ROCKETDYNE SAFETY ALGORITHM                          *
*                                                                            *
*                         ALGORITHM MODULE INIT()                            *
*                                                                            *
*   PURPOSE:                                                                 *
*                                                                            *
*   This module initializes the local variables of the RSA module. It is     *
*   called prior to processing a test with RSA. The SAFD platform requires   *
*   a separate init() routine in addition to the RSA main module, thus       *
*   this code.                                                               *
*                                                                            *
*   This module does nothing except tell RSA to initialize itself.           *
*                                                                            *
*****************************************************************************/

/*****************************************************************************
******************************************************************************
*                                                                            *
*                                                                            *
*                       ROCKETDYNE SAFETY ALGORITHM                          *
*                                                                            *
*                         ALGORITHM MODULE RSA()                             *
*                                                                            *
*   PURPOSE:                                                                 *
*                                                                            *
*   This module contains the code for the RSA. The interface to this module *
*   from the outside world is the parameter list ONLY.                       *
*                                                                            *
*****************************************************************************/

/*****************************************************************************/
/**                                                                         **/
/**                        RSA FLOW-CONTROL VARIABLES                       **/
/**             (not directly used for calculation of redlining sum)        **/
/**             (Used to track power levels, keep track of time, etc.)      **/
/**                                                                         **/
/*****************************************************************************/

 /* Numeric constants needed for data structure sizes: */

    #define MAX 80                  /* constants for data structure sizes */
    #define ARRAY_MAX 2500          /* Mostly used in WSUM() routine.     */
    #define max_adt 100             /* and support subs (engine model)    */
    #define max_est 22
    #define max_depnom 22

 /* Sampling is done to average data for the power level signature. */
 /* This signature is otherwise known as the "initialization point" */
 /* These variables define sampling times in seconds:               */
```

```
   static float time_that_we_begin_sampling = 5.9;
   static float time_main_stage_is_reached = 5.9;
   static float time_to_wait = 4.0;
   static float end_of_sampling_period = -1;   /* set at run-time */

/* Variables to make us wait after power level transients so that we */
/* don't model incorrectly. 01/16/92 -- D'Valentine/RVdH             */

   static float transient_count = 0;
   static float transient_flag = 1;

/* Arrays used for sampling, averaging of data to produce engine */
/* signatures: */

   static float VH[22], woax_tot, wfax_tot, deltay_L[22];
   static float average_H[max_adt];   /* averages of data for power level  */
   static float average_L[max_adt];   /* modelling. */

/* Counters and flags to control sampling process: */

   static int sample_count = 0;         /* number of samples taken so far   */
   static float Pc_old = -999999.0;     /* previous history memory MccPc    */

   static int done_sampling = 0;        /* Are we done sampling?            */
   static int very_first_data_point = 1; /* for initial processing...       */


/*****************************************************************************/
/**                                                                         **/
/**                          REDLINING VARIABLES                            **/
/**              (directly used to calculate redlining sum in WSUM())       **/
/**                                                                         **/
/*****************************************************************************/


 /* INPUT/OUTPUT PARAMETER LIST TYPES for calling the RSA module, */
 /* done the stupidest (but most transportable) way possible,     */
 /* as the specifications require us to do....                    */
 /* This code is identical in both RSA.C and SAFD.C, and really   */
 /* should be extracted to a separate include file for generality */


    struct inp          { float adt0, adt1, adt2, adt3, adt4, adt5,
                                adt6, adt7, adt8, adt9, adt10, adt11,
                                adt12, adt13, adt14, adt15, adt16, adt17,
                                adt18, adt19, adt20, adt21, adt22, adt34,
                                adt38, adt39, adt40, adt41, adt42, adt43,
                                adt47, adt48, adt49, adt50, adt51, adt52,
                                adt53, adt54, adt55, adt56, adt57, adt90;

                          long int engine_status; };


    struct inp_tags     { long int adt0, adt1, adt2, adt3, adt4, adt5,
                                adt6, adt7, adt8, adt9, adt10, adt11,
                                adt12, adt13, adt14, adt15, adt16, adt17,
                                adt18, adt19, adt20, adt21, adt22, adt34,
                                adt38, adt39, adt40, adt41, adt42, adt43,
                                adt47, adt48, adt49, adt50, adt51, adt52,
```

```
                        adt53, adt54, adt55, adt56, adt57, adt90,
                        engine_status;
                    };


    struct outp     { long int shutdown;
                      float adt0, adt1, adt2, adt3, adt4, adt5,
                        adt6, adt7, adt8, adt9, adt10, adt11,
                        adt12, adt13, adt14, adt15, adt16, adt17,
                        adt18, adt19, adt20, adt21, adt22, adt34,
                        adt38, adt39, adt40, adt41, adt42, adt43,
                        adt47, adt48, adt49, adt50, adt51, adt52,
                        adt53, adt54, adt55, adt56, adt57, adt90,
                        est0, est1, est2, est3, est4, est5, est6,
                        est7, est8, est9, est10, est11, est12,
                        est13, est14, est15, est16, est17, est18,
                        est19, est20, est21, wsum;
                     };


    struct outp_tags    {  long int shutdown, adt0, adt1, adt2, adt3, adt4, adt5,
                        adt6, adt7, adt8, adt9, adt10, adt11,
                        adt12, adt13, adt14, adt15, adt16, adt17,
                        adt18, adt19, adt20, adt21, adt22, adt34,
                        adt38, adt39, adt40, adt41, adt42, adt43,
                        adt47, adt48, adt49, adt50, adt51, adt52,
                        adt53, adt54, adt55, adt56, adt57, adt90,
                        est0, est1, est2, est3, est4, est5, est6,
                        est7, est8, est9, est10, est11, est12,
                        est13, est14, est15, est16, est17, est18,
                        est19, est20, est21, wsum;
                    };


/* ACTUAL POINTERS TO INPUT/OUTPUT ARRAYS FOR GLOBAL USAGE: */

  struct inp  *indata;
  struct outp *outdata;
  struct outp_tags *out_tags;

/* Local Data Table to which data is copied from outside sources: */

static float ADT[100];


static float val[8];            /* misc. variables used for numeric calc. */
static float nval;              /* Mostly used in either WSUM() or wfax() */
static double indnom[7];        /* or woax() routines. */
static float w[12];
static double depnom[22];
static double dep_star[22];
static double influ [22][7];
static float indval[7];
static float vnd[22];
static float v[22];
static float npd[12];
static float nd[22];
static float ed_nd[22];
static float filt_nd[22][10];
static float ad;
```

```
static float deltay[22];
static float deltax[7];
static float PCHK[8] = {180.0, 10.0, 25.0, 6.0, 18.0, 85.0, 1.0, 1.0};


/* The matrix representing our models of engine parameters (derived later): */

static float est[22] = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                0.0, 0.0, 0.0, 0.0 };

/* matrixes added to enhance ability to handle power level changes */
/* Added 9/19/91 by Mark D'Valentine. */

static float d_st[22] = { 141.48, 134.71, 167.83, 160.74, -1.000, -1.000,
                          -1.000, -1.000, 5252.1, 5251.9, -1.000, -1.000,
                          688.65, 688.65, 176.85, 161.71, 80.222, 48.221,
                          5.7192, 5.9477, 25.349, 26.213 };
static float d_ms[22] = { 63.614, 66.340, 112.44, 104.73, 4.0549, 4.0648,
                          2.6113, 2.5301, 302.95, 306.82, 424.91, 424.91,
                          95.727, 95.727, 194.03, 191.24, 30.237, 22.150,
                          10.200, 10.790, 11.009, 12.525 };
static float vmax_st[11] = { 0.2144, 0.1708, 0.0000, 0.0000, 0.0150, 0.0000,
                             0.0000, 0.9965, 2.5076, 2.0956, 0.5132 };
static float vmax_ms[11] = { 1.7390, 0.7460, 0.1050, 0.2639, 0.5132, 0.5000,
                             0.5000, 0.3350, 2.4651, 0.7917, 0.8479 };


/* Cross-reference arrays used for automating a clumsy numbering scheme */
/* that we use to identify ADT/EST parameters, thus the repetitions of  */
/* 7,7,8,8,9,9... etc. RVdH 2/3/92 */

static short vmaxcross [11] = {0,1,2,3,4,5,6,7,8,9,9};
static short vndcross [11] = {0,2,4,6,8,10,12,14,16,18,20};
static short ndcross [22] = {0,1,2,3,4,4,5,5,6,6,7,7,8,8,9,9,10,10,11,11,11,11};
static short crossref [6][3] = {35, 36, 37,
                                38, 39, 40,
                                47, 48, 49,
                                41, 42, 43,
                                50, 51, 52,
                                44, 45, 46  };

/* Matrixes of coefficients to use in polynomial engine model. Used in */
/* WSUM() and support routines which set up model. */

static float XNOM [7][4] = {   6.0, 0.0, 0.0, 0.0,
                               30.0, 0.0, 0.0, 0.0,
                               100.0, 0.0, 0.0, 0.0,
                               37.0, 0.0, 0.0, 0.0,
                               164.0, 0.0, 0.0, 0.0,
                               5.97545E-02, 0.5719, 6.801E-02, 0.0,
                               0.202924, 1.679, -0.2849, 0.0};

static float ED_NOM[7];

static double ynomt[22];                /* need not be initialized; derived. */
static float YINI [22][4];
static float YNOM [22][4] = {  1684.00,   -746.336,  721.239,  0.0,
                       1684.00,   -746.336,  721.239,  0.0,
```

```
                    478.000,   714.638,   110.014,  0.0,
                    478.000,   714.638,   110.014,  0.0,
                    76.83,     -25.4172,  29.1003,  0.0,
                    76.83,     -25.4172,  29.1003,  0.0,
                    72.59,     -63.8407,  57.6847,  0.0,
                    72.59,     -63.8407,  57.6847,  0.0,
                    17204.0,   13012.3,   4720.79,  0.0,
                    17204.0,   13012.3,   4720.79,  0.0,
                    4884.0,    23191.3,   -836.435, 0.0,
                    4884.0,    23191.3,   -836.435, 0.0,
                    -160.00,   4604.87,   187.075,  554.564,
                    -160.00,   4604.87,   187.075,  554.564,
                    -493.00,   8396.85,   -4697.71, 2445.17,
                    -493.00,   8396.85,   -4697.71, 2445.17,
                    132.00,    3003.94,   196.742,  0.0,
                    132.00,    3003.94,   196.742,  0.0,
                    0.0,       3006.0,    0.0,      0.0,
                    0.0,       3006.0,    0.0,      0.0,
                    0.0,       3006.0,    0.0,      0.0,
                    0.0,       3006.0,    0.0,      0.0};


/* Special matrix of coefficients for time = 0 to 10 seconds */
/* added for start transients 6/26/91 by Mark D'Valentine:   */


static float YNOM_START [22][4] = { 0.0,  520.0420,  1777.250, -1176.211,
                            0.0,  520.0420,  1777.250, -1176.211,
                            0.0, -537.0478,   2449.364,  -1004.361,
                            0.0, -537.0478,   2449.364 , -1004.361,
                            0.0,  0.0,       0.0,       0.0,
                            0.0,  0.0,       0.0,       0.0,
                            0.0,  0.0,       0.0,       0.0,
                            0.0,  0.0,       0.0,       0.0,
                            0.0 , 93179.35, -124648.3,  63325.11,
                            0.0 , 93179.35, -124648.3,  63325.11,
                            0.0,  0.0,       0.0,       0.0,
                            0.0,  0.0,       0.0,       0.0,
                            0.0,  4604.870,  187.0750,  554.5640,
                            0.0,  4604.870,  187.0750,  554.5640,
                            0.0,  4952.407, -1274.236,  1205.108,
                            0.0,  4952.407, -1274.236,  1205.108,
                            0.0,  3003.940,  196.7420,  0.0,
                            0.0,  3003.940,  196.7420,  0.0,
                            0.0,  3006.000,  0.0,       0.0,
                            0.0,  3006.000,  0.0,       0.0,
                            0.0,  3006.000,  0.0,       0.0,
                            0.0,  3006.000,  0.0,       0.0};


/* Matrix of polynomial coefficients for model simulation; */
/* Mark's altered (improved) coefficients, circa 2/20/91:  */


static float COEF [22][7][4] = {
             6.75581E-02,  3.13551E-01, -6.33715E-01,  0.00000E+00,
            -1.70377E-02,  1.42833E-02, -4.50418E-03,  0.00000E+00,
             1.33900E-02, -3.66618E-03, -3.06278E-03,  0.00000E+00,
             4.45270E-01, -5.47878E-01,  3.84827E-01,  0.00000E+00,
             7.81299E-02, -3.78262E-01,  1.77979E-01,  0.00000E+00,
```

```
 2.48217E-03, -6.37037E-03,  2.89615E-03,  0.00000E+00,
-6.22494E-04, -3.36372E-04,  3.61408E-04,  0.00000E+00,

 6.75581E-02,  3.13551E-01, -6.33715E-01,  0.00000E+00,
-1.70377E-02,  1.42833E-02, -4.50418E-03,  0.00000E+00,
 1.33900E-02, -3.66618E-03, -3.06278E-03,  0.00000E+00,
 4.45270E-01, -5.47878E-01,  3.84827E-01,  0.00000E+00,
 7.81299E-02, -3.78262E-01,  1.77979E-01,  0.00000E+00,
 2.48217E-03, -6.37037E-03,  2.89615E-03,  0.00000E+00,
-6.22494E-04, -3.36372E-04,  3.61408E-04,  0.00000E+00,

-1.79474E-01,  5.30960E+00, -1.78281E+00,  0.00000E+00,
-9.45672E-04,  1.75486E-02, -1.05341E-02,  0.00000E+00,
-3.15037E-02, -4.35284E-02,  4.25821E-02,  0.00000E+00,
 3.09397E-02, -2.52552E-01,  0.00000E+00,  0.00000E+00,
-4.63672E-01,  1.84179E+00, -7.88768E-01,  0.00000E+00,
-5.18391E-03,  1.99238E-02, -8.76068E-03,  0.00000E+00,
 1.87290E-03,  3.37052E-03, -2.36554E-03,  0.00000E+00,

-1.79474E-01,  5.30960E+00, -1.78281E+00,  0.00000E+00,
-9.45672E-04,  1.75486E-02, -1.05341E-02,  0.00000E+00,
-3.15037E-02, -4.35284E-02,  4.25821E-02,  0.00000E+00,
 3.09397E-02, -2.52552E-01,  0.00000E+00,  0.00000E+00,
-4.63672E-01,  1.84179E+00, -7.88768E-01,  0.00000E+00,
-5.18391E-03,  1.99238E-02, -8.76068E-03,  0.00000E+00,
 1.87290E-03,  3.37052E-03, -2.36554E-03,  0.00000E+00,

-3.27388E+00,  8.50257E+00, -6.35848E+00,  0.00000E+00,
-2.72697E-02,  6.61322E-02, -4.57526E-02,  0.00000E+00,
 4.14552E-02, -8.71833E-02,  5.34435E-02,  0.00000E+00,
 1.65703E+00, -4.27165E+00,  2.88507E+00,  0.00000E+00,
-7.63844E-02,  2.17812E-01, -1.79975E-01,  0.00000E+00,
-1.04713E-02,  2.78641E-02, -2.02631E-02,  0.00000E+00,
-2.40354E-03,  6.10207E-03, -4.52982E-03,  0.00000E+00,

-3.27388E+00,  8.50257E+00, -6.35848E+00,  0.00000E+00,
-2.72697E-02,  6.61322E-02, -4.57526E-02,  0.00000E+00,
 4.14552E-02, -8.71833E-02,  5.34435E-02,  0.00000E+00,
 1.65703E+00, -4.27165E+00,  2.88507E+00,  0.00000E+00,
-7.63844E-02,  2.17812E-01, -1.79975E-01,  0.00000E+00,
-1.04713E-02,  2.78641E-02, -2.02631E-02,  0.00000E+00,
-2.40354E-03,  6.10207E-03, -4.52982E-03,  0.00000E+00,

 1.33173E+00, -3.68307E+00,  3.04542E+00,  0.00000E+00,
 6.04503E-04,  0.00000E+00,  0.00000E+00,  0.00000E+00,
 1.30019E-02, -2.74375E-02,  0.00000E+00,  0.00000E+00,
 2.20889E-02, -6.10283E-02,  0.00000E+00,  0.00000E+00,
 8.40340E-01, -2.36836E+00,  1.88330E+00,  0.00000E+00,
-1.19188E-03,  2.29577E-03,  0.00000E+00,  0.00000E+00,
-9.63561E-04,  2.13802E-03,  0.00000E+00,  0.00000E+00,

 1.33173E+00, -3.68307E+00,  3.04542E+00,  0.00000E+00,
 6.04503E-04,  0.00000E+00,  0.00000E+00,  0.00000E+00,
 1.30019E-02, -2.74375E-02,  0.00000E+00,  0.00000E+00,
 2.20889E-02, -6.10283E-02,  0.00000E+00,  0.00000E+00,
 8.40340E-01, -2.36836E+00,  1.88330E+00,  0.00000E+00,
-1.19188E-03,  2.29577E-03,  0.00000E+00,  0.00000E+00,
-9.63561E-04,  2.13802E-03,  0.00000E+00,  0.00000E+00,

-6.31739E-01,  1.12647E+00, -8.22106E-01,  0.00000E+00,
```

```
-1.17272E-02,  1.26742E-02, -5.26788E-03,  0.00000E+00,
 1.19841E-03, -4.42708E-03,  2.67544E-03,  0.00000E+00,
 3.06835E-01, -3.73406E-01,  2.76127E-01,  0.00000E+00,
-2.97835E-02,  8.02084E-02, -4.00085E-02,  0.00000E+00,
-1.32908E-04,  0.00000E+00,  0.00000E+00,  0.00000E+00,
-1.11418E-04,  3.62329E-04, -2.01927E-04,  0.00000E+00,

-6.31739E-01,  1.12647E+00, -8.22106E-01,  0.00000E+00,
-1.17272E-02,  1.26742E-02, -5.26788E-03,  0.00000E+00,
 1.19841E-03, -4.42708E-03,  2.67544E-03,  0.00000E+00,
 3.06835E-01, -3.73406E-01,  2.76127E-01,  0.00000E+00,
-2.97835E-02,  8.02084E-02, -4.00085E-02,  0.00000E+00,
-1.32908E-04,  0.00000E+00,  0.00000E+00,  0.00000E+00,
-1.11418E-04,  3.62329E-04, -2.01927E-04,  0.00000E+00,

-1.53861E-01,  7.19883E-01, -3.08034E-01,  0.00000E+00,
 7.03175E-05,  0.00000E+00,  0.00000E+00,  0.00000E+00,
-5.88139E-02,  9.10559E-02, -4.02095E-02,  0.00000E+00,
-4.07650E-03,  7.66015E-03, -5.73094E-03,  0.00000E+00,
 1.13350E-01,  3.59439E-01, -1.50146E-01,  0.00000E+00,
-9.45481E-04,  4.14947E-03, -1.81101E-03,  0.00000E+00,
 8.45892E-04,  0.00000E+00,  0.00000E+00,  0.00000E+00,

-1.53861E-01,  7.19883E-01, -3.08034E-01,  0.00000E+00,
 7.03175E-05,  0.00000E+00,  0.00000E+00,  0.00000E+00,
-5.88139E-02,  9.10559E-02, -4.02095E-02,  0.00000E+00,
-4.07650E-03,  7.66015E-03, -5.73094E-03,  0.00000E+00,
 1.13350E-01,  3.59439E-01, -1.50146E-01,  0.00000E+00,
-9.45481E-04,  4.14947E-03, -1.81101E-03,  0.00000E+00,
 8.45892E-04,  0.00000E+00,  0.00000E+00,  0.00000E+00,

-1.06276E-02,  4.20930E-02,  0.00000E+00,  0.00000E+00,
-3.09662E-04, -3.51086E-04,  2.46367E-04,  0.00000E+00,
-1.11283E-02,  5.70060E-03,  0.00000E+00,  0.00000E+00,
 7.83933E-03, -2.56483E-04,  1.02111E-02,  0.00000E+00,
-2.95686E-02,  2.10816E-01, -7.71641E-02,  0.00000E+00,
-4.82056E-04,  1.94540E-03, -7.86290E-04,  0.00000E+00,
 6.05011E-04, -1.20095E-04,  0.00000E+00,  0.00000E+00,

-1.06276E-02,  4.20930E-02,  0.00000E+00,  0.00000E+00,
-3.09662E-04, -3.51086E-04,  2.46367E-04,  0.00000E+00,
-1.11283E-02,  5.70060E-03,  0.00000E+00,  0.00000E+00,
 7.83933E-03, -2.56483E-04,  1.02111E-02,  0.00000E+00,
-2.95686E-02,  2.10816E-01, -7.71641E-02,  0.00000E+00,
-4.82056E-04,  1.94540E-03, -7.86290E-04,  0.00000E+00,
 6.05011E-04, -1.20095E-04,  0.00000E+00,  0.00000E+00,

-4.17196E-01,  3.95345E-01, -3.26333E-01,  0.00000E+00,
-3.58206E-03,  2.19911E-03, -6.03413E-04,  0.00000E+00,
 1.29300E-03, -4.85527E-03,  2.59230E-03,  0.00000E+00,
 1.15425E-01, -1.52808E-01,  1.16439E-01,  0.00000E+00,
-2.13535E-02,  5.46972E-02, -1.49099E-02,  0.00000E+00,
-1.19488E-04,  0.00000E+00,  0.00000E+00,  0.00000E+00,
-8.58883E-05,  3.00620E-04, -1.28705E-04,  0.00000E+00,

-4.17196E-01,  3.95345E-01, -3.26333E-01,  0.00000E+00,
-3.58206E-03,  2.19911E-03, -6.03413E-04,  0.00000E+00,
 1.29300E-03, -4.85527E-03,  2.59230E-03,  0.00000E+00,
 1.15425E-01, -1.52808E-01,  1.16439E-01,  0.00000E+00,
-2.13535E-02,  5.46972E-02, -1.49099E-02,  0.00000E+00,
```

```
          -1.19488E-04,  0.00000E+00,  0.00000E+00,  0.00000E+00,
          -8.58883E-05,  3.00620E-04, -1.28705E-04,  0.00000E+00,

          -2.66681E-02,  9.92401E-03, -2.27310E-02,  0.00000E+00,
          -2.37199E-04,  0.00000E+00,  0.00000E+00,  0.00000E+00,
          -3.22731E-04,  0.00000E+00,  0.00000E+00,  0.00000E+00,
           1.00202E-02, -1.39743E-02,  1.31286E-02,  0.00000E+00,
           1.17278E-03, -3.27962E-05,  4.38514E-03,  0.00000E+00,
           2.65487E-06, -1.72772E-04,  3.49690E-05,  0.00000E+00,
           2.85558E-05, -1.72770E-05,  1.68002E-05,  0.00000E+00,

          -2.66681E-02,  9.92401E-03, -2.27310E-02,  0.00000E+00,
          -2.37199E-04,  0.00000E+00,  0.00000E+00,  0.00000E+00,
          -3.22731E-04,  0.00000E+00,  0.00000E+00,  0.00000E+00,
           1.00202E-02, -1.39743E-02,  1.31286E-02,  0.00000E+00,
           1.17278E-03, -3.27962E-05,  4.38514E-03,  0.00000E+00,
           2.65487E-06, -1.72772E-04,  3.49690E-05,  0.00000E+00,
           2.85558E-05, -1.72770E-05,  1.68002E-05,  0.00000E+00 );



/*****************************************************************************/
/**                                                                         **/
/**                        MAIN BODY OF CALL_RSA()                          **/
/**            (otherwise known as the control module for WSUM())           **/
/**                                                                         **/
/*****************************************************************************/


void call_RSA(input, input_tags, output, output_tags)

  struct inp           *input;
  struct inp_tags      *input_tags;
  struct outp          *output;
  struct outp_tags     *output_tags;

{

  void do_nothing();
  void perform_start_transient_modelling();
  void clear_the_accumulators_average_H_and_average_L();
  void Calculate_the_weighted_sum();
  void add_a_single_sample_to_first_init_point_average();
  void Calc_1st_avg_signature();
  void Load_first_signature_into_model_for_one_point_init();
  void initialize_variables();
  void Transfer_to_ADT();

  indata  = input;   /* AWFUL KLUGE to get passed parameters to be GLOBAL. */
  outdata = output;  /* Sorry folks, but this program was inherited from a */
                     /* BASIC programmer. */

  out_tags = output_tags;

  Transfer_to_ADT(); /* Transfer data from parameter list to a local array */
                     /* for looping, indexing, processing (ADT)            */


  /* Local variables are initialized separately in SAFD.C and/or in the    */
  /* SAFD control module at Hunstville using extern void init(). */
```

```
  /* This is done at the beginning of each new engine test.              */

    if (ADT[90] > 0.8)
         do_nothing();

    if (very_first_data_point == 1)
    {
       perform_start_transient_modelling();
       very_first_data_point = 0;
    }

    if (ADT[90] < time_that_we_begin_sampling)
       Calculate_the_weighted_sum();

    if ((ADT[90] >= time_that_we_begin_sampling) && (ADT[90] < end_of_sampling_period))
    {
       add_a_single_sample_to_first_init_point_average();
       sample_count = sample_count + 1;
       Calculate_the_weighted_sum();
    }

    if ((ADT[90] >= end_of_sampling_period) && (done_sampling == 0))
    {
       Calc_1st_avg_signature();
       Load_first_signature_into_model_for_one_point_init();
       done_sampling = 1;
    }

    if (done_sampling == 1)
       Calculate_the_weighted_sum();

   Pc_old = ADT[34];   /* Preserve history data for later calcs. */
}




/*****************************************************************************/
/**                                                                         **/
/**                          RSA SUBROUTINES                                **/
/**                                                                         **/
/** A subroutine called by one of these belongs to the "E" calling level.  **/
/**                                                                         **/
/*****************************************************************************/


void do_nothing()
{
     /* used for debugging under the PC environment Turbo C */
}



int init()
{
     int i, a, b, c;

     /* Clear out the weighed sum calculation variables: */
```

```
    for (i = 0; i < 11; i++)
      npd[i] = 0.0;
    for (i = 0; i < 22; i ++)
    {
      est[i] = 0.0;
      ynomt[i] = 0.0;
    }

    /* Clear out the accumulators where the samples are stored: */

        for (i=0; i<max_adt; i++)
          average_H[i]=0;
        for (i=0; i<max_adt; i++)
          average_L[i]=0;
        wfax_tot = 0.0;  /* clear out the backpressure sums */
        woax_tot = 0.0;
        for (i=0; i<22; i++)
          deltay_L[i] = 0.0;  /* clear out the 5-second delta-y sum */

   /* Initialize the filtering array (created by Ed Nemeth 9-3-92) */

   for (a = 0; a < 22; a++)
     for (b = 0; b < 10; b++)
        filt_nd[a][b] = 0.0;


    /* initialize working variables: */

    end_of_sampling_period = time_to_wait+time_that_we_begin_sampling;
    sample_count = 0;
    very_first_data_point = 1;
    done_sampling = 0;

    return(1);
}

void Transfer_to_ADT()
{
     ADT[0] = indata->adt0;
     ADT[1] = indata->adt1;
     ADT[2] = indata->adt2;
     ADT[3] = indata->adt3;
     ADT[4] = indata->adt4;
     ADT[5] = indata->adt5;
     ADT[6] = indata->adt6;
     ADT[7] = indata->adt7;
     ADT[8] = indata->adt8;
     ADT[9] = indata->adt9;
     ADT[10] = indata->adt10;
     ADT[11] = indata->adt11;
     ADT[12] = indata->adt12;
     ADT[13] = indata->adt13;
     ADT[14] = indata->adt14;
     ADT[15] = indata->adt15;
     ADT[16] = indata->adt16;
     ADT[17] = indata->adt17;
     ADT[18] = indata->adt18;
     ADT[19] = indata->adt19;
     ADT[20] = indata->adt20;
     ADT[21] = indata->adt21;
```

```
    ADT[22] = indata->adt22;
    ADT[34] = indata->adt34;
    ADT[38] = indata->adt38;
    ADT[39] = indata->adt39;
    ADT[40] = indata->adt40;
    ADT[41] = indata->adt41;
    ADT[42] = indata->adt42;
    ADT[43] = indata->adt43;
    ADT[47] = indata->adt47;
    ADT[48] = indata->adt48;
    ADT[49] = indata->adt49;
    ADT[50] = indata->adt50;
    ADT[51] = indata->adt51;
    ADT[52] = indata->adt52;
    ADT[53] = indata->adt53;
    ADT[54] = indata->adt54;
    ADT[55] = indata->adt55;
    ADT[56] = indata->adt56;
    ADT[57] = indata->adt57;
    ADT[90] = indata->adt90;
}


void perform_start_transient_modelling()

   /* Models startup portion of test from 0.2 to 10.1 seconds by using    */
   /* modified YNOM array with special values in it derived from actual   */
   /* data values. See variable YNOM_START in include file NMOD3.H.       */
   /* Mark D'Valentine - formulas; Robert Van den Heuvel - coding 7/10/91 */
{
    int i;
    float PL;

    PL = ((ADT[19]+ADT[20])/2)/3006;
    for (i=0; i<22; i++)
    {
      YNOM_START[i][0] = ADT[i+1] -
                         (YNOM_START[i][1]*PL) -
                         (YNOM_START[i][2]*(PL*PL)) -
                         (YNOM_START[i][3]*(PL*PL*PL));

      v[i] = 1.0;    /* The v-array tells whether a measurement is valid. */
    }
    /* be sure not to set HPFT speeds to the uncleared, bogus values */
    /* (about 40,000 rpm) from the pre-test sensor checkout          */
    /* (correct speeds should not be > 4000 rpm at 0.2 seconds       */

    if (ADT[9] > 4000)
    {
       YNOM_START[8][0] = 456.0 -
            (YNOM_START[8][1]*PL) -
            (YNOM_START[8][2]*(PL*PL)) -
            (YNOM_START[8][3]*(PL*PL*PL));
    }

    if (ADT[10] > 4000)
    {
       YNOM_START[9][0] = 456.0 -
            (YNOM_START[9][1]*PL) -
            (YNOM_START[9][2]*(PL*PL)) -
```

```
            (YNOM_START[9][3]*(PL*PL*PL));
    }
}
```

```
void add_a_single_sample_to_first_init_point_average()

   /* Takes one raw data point and adds it to a running average which will   */
   /* be used later to produce an engine signature at the current pwr level. */
{
    int i;
    float wfax(), woax();
    for (i = 0; i < max_adt; i++)
      average_H[i] = average_H[i]+ ADT[i];
    wfax_tot = wfax_tot + wfax();            /* these are for same purpose  */
    woax_tot = woax_tot + woax();
}
```

```
void Calc_1st_avg_signature()

   /* "sample_count" counts the number of data points sampled so far.  */
   /* We average the last 5 seconds of raw data to get an average for  */
   /* each parameter (average_H[x]). Each average is a "signature" for */
   /* the engine at the current power level.                           */
{
  int i;
  int item[7] = {0,38,47,41,50,0,0};
  float PL, PL2, PL3;

  for (i=0; i<max_adt; i++)
    average_H[i] = average_H[i] / sample_count;
  PL = average_H[34]/3006;
  PL2 = PL * PL;
  PL3 = PL * PL2;

  /* Set baseline indep. params to their 1st init. point average values */



  XNOM[0][0] = 0.0;
  for (i=1; i<7; i++)
  {
        ED_NOM[i] = XNOM[i][0] +
                    XNOM[i][1] * PL +
                    XNOM[i][2] * PL2 +
                    XNOM[i][3] * PL3;
  }
  for (i=1; i<5; i++)
  {
        XNOM[i][0] = (XNOM[i][0] * average_H[item[i]])/ED_NOM[i];
        XNOM[i][1] = (XNOM[i][1] * average_H[item[i]])/ED_NOM[i];
```

```
        XNOM[i][2] = (XNOM[i][2] * average_H[item[i]])/ED_NOM[i];
        XNOM[i][3] = (XNOM[i][3] * average_H[item[i]])/ED_NOM[i];
  }
  i = 5;
        XNOM[i][0] = (XNOM[i][0] * (wfax_tot/sample_count))/ED_NOM[i];
        XNOM[i][1] = (XNOM[i][1] * (wfax_tot/sample_count))/ED_NOM[i];
        XNOM[i][2] = (XNOM[i][2] * (wfax_tot/sample_count))/ED_NOM[i];
        XNOM[i][3] = (XNOM[i][3] * (wfax_tot/sample_count))/ED_NOM[i];
  i = 6;
        XNOM[i][0] = (XNOM[i][0] * (woax_tot/sample_count))/ED_NOM[i];
        XNOM[i][1] = (XNOM[i][1] * (woax_tot/sample_count))/ED_NOM[i];
        XNOM[i][2] = (XNOM[i][2] * (woax_tot/sample_count))/ED_NOM[i];
        XNOM[i][3] = (XNOM[i][3] * (woax_tot/sample_count))/ED_NOM[i];



/* debug 7-24-92 for Ed Nemeth: Failure investigation. Rewrite of XNOM
   calculation to custom specs for debugging. RVDH

  for (i=1; i<5; i++)
    XNOM[i][0] = average_H[item[i]] - XNOM[i][1]*PL -
                 XNOM[i][2] * PL2 - XNOM[i][3] * PL3;
  XNOM[5][0] = wfax_tot / sample_count - XNOM[5][1] * PL -
               XNOM[5][2] * PL2 - XNOM[5][3] * PL3;
  XNOM[6][0] = woax_tot / sample_count - XNOM[6][1] * PL -
               XNOM[6][2] * PL2 - XNOM[6][3] * PL3;
*/

}




void Load_first_signature_into_model_for_one_point_init()
{
  float PL1, PL2, PL3;
  int i;

  PL1 = average_H[34]/3006;  /* define the current power level & powers of */
  PL2 = PL1 * PL1;
  PL3 = PL1 * PL1 * PL1;

  /* LOAD THE 104% SIGNATURE INTO THE YNOM MATRIX: */
  for (i=0; i < 22; i++)
  {
    /* YNOM[i][0] IS THE 1ST COLUMN OF A 22x4 MATRIX. THE 1ST COLUMN     */
    /* DEFINES THE CONSTANT TERMS IN A SET OF 22 3RD DEGREE POLYNOMIALS. */

    /* VH = "V-High Average," a variable which represents the selected,  */
    /* UNADJUSTED V-High's for 1-point equation calculations             */

    VH[i] = average_H[i+1];
    YNOM[i][0] = VH[i] - PL1*YNOM[i][1] - PL2*YNOM[i][2] - PL3*YNOM[i][3];

  }
}
```

```
void Calculate_the_weighted_sum()
{
   float WSUM();
   float w_sum = -100;

   w_sum = WSUM();          /* Redlining sum (weighted) is calculated here  */

   /* TRANSFER ALL PARAMETERS OF INTEREST TO THE OUTPUT ARRAYS: */

    outdata->adt0 = ADT[0];
    outdata->adt1 = ADT[1];
    outdata->adt2 = ADT[2];
    outdata->adt3 = ADT[3];
    outdata->adt4 = ADT[4];
    outdata->adt5 = ADT[5];
    outdata->adt6 = ADT[6];
    outdata->adt7 = ADT[7];
    outdata->adt8 = ADT[8];
    outdata->adt9 = ADT[9];
    outdata->adt10 = ADT[10];
    outdata->adt11 = ADT[11];
    outdata->adt12 = ADT[12];
    outdata->adt13 = ADT[13];
    outdata->adt14 = ADT[14];
    outdata->adt15 = ADT[15];
    outdata->adt16 = ADT[16];
    outdata->adt17 = ADT[17];
    outdata->adt18 = ADT[18];
    outdata->adt19 = ADT[19];
    outdata->adt20 = ADT[20];
    outdata->adt21 = ADT[21];
    outdata->adt22 = ADT[22];
    outdata->adt34 = ADT[34];
    outdata->adt38 = ADT[38];
    outdata->adt39 = ADT[39];
    outdata->adt40 = ADT[40];
    outdata->adt41 = ADT[41];
    outdata->adt42 = ADT[42];
    outdata->adt43 = ADT[43];
    outdata->adt47 = ADT[47];
    outdata->adt48 = ADT[48];
    outdata->adt49 = ADT[49];
    outdata->adt50 = ADT[50];
    outdata->adt51 = ADT[51];
    outdata->adt52 = ADT[52];
    outdata->adt53 = ADT[53];
    outdata->adt54 = ADT[54];
    outdata->adt55 = ADT[55];
    outdata->adt56 = ADT[56];
    outdata->adt57 = ADT[57];
    outdata->adt90 = ADT[90];
    outdata->est0 = est[0];
    outdata->est1 = est[1];
    outdata->est2 = est[2];
    outdata->est3 = est[3];
    outdata->est4 = est[4];
```

```
  outdata->est5 = est[5];
  outdata->est6 = est[6];
  outdata->est7 = est[7];
  outdata->est8 = est[8];
  outdata->est9 = est[9];
  outdata->est10 = est[10];
  outdata->est11 = est[11];
  outdata->est12 = est[12];
  outdata->est13 = est[13];
  outdata->est14 = est[14];
  outdata->est15 = est[15];
  outdata->est16 = est[16];
  outdata->est17 = est[17];
  outdata->est18 = est[18];
  outdata->est19 = est[19];
  outdata->est20 = est[20];
  outdata->est21 = est[21];
  outdata->wsum = w_sum;
  outdata->shutdown = 0;

/* SET FLAGS TO HEX ZERO SINCE WE DON'T USE THIS FEATURE: */

  out_tags->adt0 = 0x00000000L;
  out_tags->adt1 = 0x00000000L;
  out_tags->adt2 = 0x00000000L;
  out_tags->adt3 = 0x00000000L;
  out_tags->adt4 = 0x00000000L;
  out_tags->adt5 = 0x00000000L;
  out_tags->adt6 = 0x00000000L;
  out_tags->adt7 = 0x00000000L;
  out_tags->adt8 = 0x00000000L;
  out_tags->adt9 = 0x00000000L;
  out_tags->adt10 = 0x00000000L;
  out_tags->adt11 = 0x00000000L;
  out_tags->adt12 = 0x00000000L;
  out_tags->adt13 = 0x00000000L;
  out_tags->adt14 = 0x00000000L;
  out_tags->adt15 = 0x00000000L;
  out_tags->adt16 = 0x00000000L;
  out_tags->adt17 = 0x00000000L;
  out_tags->adt18 = 0x00000000L;
  out_tags->adt19 = 0x00000000L;
  out_tags->adt20 = 0x00000000L;
  out_tags->adt21 = 0x00000000L;
  out_tags->adt22 = 0x00000000L;
  out_tags->adt34 = 0x00000000L;
  out_tags->adt38 = 0x00000000L;
  out_tags->adt39 = 0x00000000L;
  out_tags->adt40 = 0x00000000L;
  out_tags->adt41 = 0x00000000L;
  out_tags->adt42 = 0x00000000L;
  out_tags->adt43 = 0x00000000L;
  out_tags->adt47 = 0x00000000L;
  out_tags->adt48 = 0x00000000L;
  out_tags->adt49 = 0x00000000L;
  out_tags->adt50 = 0x00000000L;
  out_tags->adt51 = 0x00000000L;
  out_tags->adt52 = 0x00000000L;
  out_tags->adt53 = 0x00000000L;
  out_tags->adt54 = 0x00000000L;
```

```
    out_tags->adt55 = 0x00000000L;
    out_tags->adt56 = 0x00000000L;
    out_tags->adt57 = 0x00000000L;
    out_tags->adt90 = 0x00000000L;
    out_tags->est0 = 0x00000000L;
    out_tags->est1 = 0x00000000L;
    out_tags->est2 = 0x00000000L;
    out_tags->est3 = 0x00000000L;
    out_tags->est4 = 0x00000000L;
    out_tags->est5 = 0x00000000L;
    out_tags->est6 = 0x00000000L;
    out_tags->est7 = 0x00000000L;
    out_tags->est8 = 0x00000000L;
    out_tags->est9 = 0x00000000L;
    out_tags->est10 = 0x00000000L;
    out_tags->est11 = 0x00000000L;
    out_tags->est12 = 0x00000000L;
    out_tags->est13 = 0x00000000L;
    out_tags->est14 = 0x00000000L;
    out_tags->est15 = 0x00000000L;
    out_tags->est16 = 0x00000000L;
    out_tags->est17 = 0x00000000L;
    out_tags->est18 = 0x00000000L;
    out_tags->est19 = 0x00000000L;
    out_tags->est20 = 0x00000000L;
    out_tags->est21 = 0x00000000L;
    out_tags->wsum =     0x00000000L;
    out_tags->shutdown = 0x00000000L;

}
```

```
/*****************************************************************************/
/**                                                                         **/
/**                        WEIGHTED SUM SUBROUTINE                          **/
/**                                                                         **/
/**   (calculates redlining sum of deviations between models and reality)   **/
/**                                                                         **/
/**                                                                         **/
/**   SUMMARY: This module is essentially one big list of equations which   **/
/**            get executed when the value of the redline sum is desired.   **/
/**            The module is commented into modules (A, B, C, etc.) which   **/
/**            each represent some significant sub-process.                 **/
/**                                                                         **/
/**   THIS IS THE ONLY LEVEL "E" SUBROUTINE.                                **/
/**                                                                         **/
/*****************************************************************************/
```

```
float WSUM()
```

```
{

/* VARIABLES USED FOR GENERAL WSUM PROCESSING */

int alsteffensindex, alkey;
float PL;
float sum;
int c,m,n,a,b,i,j;
float B1, B2, r1;
int a1, a2;

/* Ed Nemeth: 9-3-92: variables added for filtering of nd calculation */

float filt_sum = 0;
int n_samp = 0;
int number_of_samples = 0;

/* continuing with old WSUM definitions: */

float AMIN = 0.2;                              /* not defined yet */
float dmax = 0;
float d1 = 0, d2 = 0;
float d[22];
float vmax[11];
int jq, kq, indx[3];  /* added for Al Steffan's modifications of MODULE C */
float adiff[3], asum[3], aminimum = 3e+38;
float nonredundant_pid_limit = 11;  /* Added 2/26/92 */


/* VARIABLES FOR WSUM WEIGHTING ("W" CALCULATION BELOW) added 2/92 */

float sig[22];
float sig_counter = 0;
float k[22] = {
   1, 1, 1.04, 1.13, 1.27, 1.49, 1.82, 2.31, 3.05, 4.20, 6.05, 8.91,
   13.91, 22.40, 37.76, 65.54, 120.15, 226.57, 440.56, 881.11, 1982.50,
   3965.00 };


float wfax();   /* USED TO CALC (FUEL) REPRESSURIZATION FLOWS */
float woax();   /* DITTO (LOX) */


/* NIRD CODE CHANGES (01/16/92)

   we need to wait longer after power level transients are over
   before we begin computing PL based on commanded Pc again.

   if time < 5.9 sec, then compute PL using measured Pc: */


if (ADT[90] < time_main_stage_is_reached)
{
  PL = ((ADT[19]+ADT[20])/2)/3006;
  transient_count = 0;
  transient_flag = 1;
}

/* else if time >= 5.9 sec, then check whether ADT[34] = Pc_old */
```

```
else
{
  if ( ADT[34] == Pc_old )
  {

/* if transient_flag = 1 here, then it's the first sample after a
   transient.  this code calculates PL using commanded Pc during
   steady-state engine operation.  PL is calculated using measured
   Pc during transients.  this code waits 4 samples after a transient
   is over before PL is computed using commanded Pc again. */

    if ( transient_flag == 1 )
    {
      transient_count = 0;
      PL = ((ADT[19]+ADT[20])/2)/3006;
    }

/* else if transient_flag = 0, then determine whether to compute */
/* PL using commanded Pc or measured Pc                          */

    else
    {
      transient_count = transient_count + 1;

/* if transient_count < 4 then compute PL using measured Pc */

      if ( transient_count < 4 )
      {
        PL = ((ADT[19]+ADT[20])/2)/3006;
      }

/* if transient_count >= 4 then compute PL using commanded Pc */
/* (this method is used during the majority of most tests)    */

      else
      {
        PL = ADT[34] / 3006;
      }
    }

/* since ADT[34] = Pc_old here, then re-set transient_flag to 0 */

    transient_flag = 0;
  }

/* else if ADT[34] <> Pc_old then compute PL using measured Pc */

  else
  {
    PL = ((ADT[19]+ADT[20])/2)/3006;
    transient_flag = 1;
    transient_count = 0;
  }
}


/* "VAL" EQUATION CALCULATION:                          MODULE C     */
/* ------------------------------------------------------------------ */
/*  Validate six independent parameter measurements by comparison of  */
/*  three redundant values                                            */
```

```
/*  Rewritten for speed and brevity by Al Steffens, Jr. 9-4-91 up to   */
/*  the statement, "val[n] = r;"                                       */

for (n = 0; n < 6; n++)
{
   for (jq=0;jq<3; jq++) {

      /*** Get the ADT indices -- jq = 1st index, kq = off-index ***/
      kq = (jq+1) % 3;                        /* permute kq */
      indx[jq] = crossref[n][jq];             /* ADT indices */
      indx[kq] = crossref[n][kq];             /* ADT off-indices */

      /*** Get ADT differences (and sums) ***/
      adiff[jq] = fabs(ADT[indx[jq]] - ADT[indx[kq]] );
      asum[jq] = (ADT[indx[jq]] + ADT[indx[kq]])/2;

      /*** Find smallest difference ***/
      if (adiff[jq] < aminimum) {
         aminimum = adiff[jq];
         val[n] = asum[jq];
      }

   } /* for jq */

   /*** Test if difference is within limit ***/
   if (aminimum > PCHK[n])
      val[n] = 0;

} /* for n */

if (ADT[90] < time_main_stage_is_reached)
{
  val[6] = 0;
}
else
  val[6] = wfax();   /* Call fuel repressurization flow calculation */

if (ADT[90] < time_main_stage_is_reached)
{
  val[7] = 0;
}
else
  val[7] = woax(); /* Call LOX/HEX repressurization flow calculation */

/* "INDVAL" EQUATION CALCULATION:                                      */

/* ------------------------------------------------------------------ */
/* Calculates mixture ratio (not yet implemented) and shifts indices/values*/
/* to a new array (indval[7]).  The values in this new array are used later*/
/* in the "DELTA-X" calculation */

for (n = 0; n < 7; n++)
{
if (n == 0)
 {
 indval[n] = 0.0;                         /* mixture ratio */
 }
else
 if ((n == 5) || (n == 6))               /* repress */
  {
```

```
  indval[n] = val[n+1];
  }
else                                        /* all others */
indval[n] = val[n];
}


/* "DEPNOM" EQUATION CALCULATION:                    MODULE D          */
/* ------------------------------------------------------------------ */
/* Calculate the nominal dependent values using PL in a polynomial    */
/* USE SPECIAL MATRIX YNOM_START (stands for YNOM-startup-transient) */
/* THE TIME IS LESS THAN 5 SECONDS INTO THE TEST. MODELS STARTUP      */
/* SEPARATELY TO PROVIDE SMOOTHER WEIGHTED SUM. 6/27/91 RVdH          */

if (ADT[90] < 10.0)
{
   for (m = 0; m < 22; m++)
   {
     depnom[m] = YNOM_START[m][0] + (YNOM_START[m][1]*PL) + (YNOM_START[m][2]
                 * (PL * PL)) + (YNOM_START[m][3] * (PL * PL * PL));
   }
}
else
/* THE REGULAR MATRIX IS USED FOR THE REST OF THE TEST: */
{
   for (m = 0; m < 22; m++)
   {
     depnom[m] = YNOM[m][0] + (YNOM[m][1]*PL) + (YNOM[m][2] * (PL * PL)) + (YNOM[m][3] * (PL * PL * PL));
   }
}

for (m = 0; m < 22; m++)
{
   dep_star[m] = ADT[m+1]/(1+deltay[m]);
}

/* "INDNOM" EQUATION CALCULATION:                                     */
/* ------------------------------------------------------------------ */
/* Calculate the nominal independent values using PL in a polynomial  */

for (n = 0; n < 7; n++)
{
indnom[n] = XNOM[n][0] + (XNOM[n][1] * PL) + (XNOM[n][2] * (PL * PL))
+ (XNOM[n][3] * (PL * PL * PL));
}

/* "DELTA-X" EQUATION CALCULATION:                                    */
/* ------------------------------------------------------------------ */
/* Calculate the % deltas between nominal and measured independent */
/* variable values */

for (n = 0; n < 7; n++)
{
B1 = indval[n];              /* indval array calculated in Module C */
B2 = indnom[n];

if ((B1 == 0.0) || (B2 == 0.0))
r1 = 0.0;
 else
 {
```

```
 r1 = (B1 - B2)/B2;
 }
 deltax[n] = r1;
}

/* "INFLU" EQUATION CALCULATION:                                        */
/* -------------------------------------------------------------------- */
/* Uses 3D influence coefficient array to derive estimates of dependent */
/* variables */
/* SET THE WHOLE COEFF ARRAY TO ZERO IF                                 */
/* THE TIME IS LESS THAN 5 SECONDS INTO THE TEST. MODELS STARTUP        */
/* SEPARATELY TO PROVIDE SMOOTHER WEIGHTED SUM. 6/27/91 RVdH            */

if (ADT[90] < 10.0)
/* IN THE BEGINNING, ZERO OUT THE COEFF ARRAY FOR STARTUP MODELLING: */
{
  for (m = 0; m < 22; m++)
   {
     for (n = 0; n < 7; n++)
     {
       influ[m][n] = 0;
     }
   }

}
else /* FOR THE REST OF THE TEST, USE THE ARRAY AS NORMAL: */
{
  for (m = 0; m < 22; m++)
  {
      for (n = 0; n < 7; n++)
      {
        influ[m][n] = COEF[m][n][0] + (COEF[m][n][1] * PL) +
        (COEF[m][n][2] * (PL * PL)) + (COEF[m][n][3] * (PL * PL * PL));
      }
  }
}

/* "DELTA-Y" EQUATION CALCULATION:                                      */
/* -------------------------------------------------------------------- */
/* Calculate the total influence of independent parameters on each      */
/* dependent parameter */

for (m = 0; m <22; m++)
{
  sum = 0;
  for (n = 0; n < 7; n++)
      sum = sum + (influ[m][n] * deltax[n]);
  deltay[m] = sum;
}

/* "EST" EQUATION CALCULATION:                                          */
/* -------------------------------------------------------------------- */
/* Calculate the final estimate of the nominal dependent values         */

for (m = 0; m < 22; m++)
    est[m] = depnom[m] * (1 + deltay[m]);


/* MODULE E_____________________________________________________________*/
/* "D" EQUATION CALCULATION: Defined later, defaults used for now.      */
```

```
/* No code here at this time (D'Val)                    10/30/91 RVDH  */


/* "V" EQUATION CALCULATION:                            MODULE F       */
/* ------------------------------------------------------------------- */
/* Initialize the valid "flag" array                                   */

/* if v[a] = 1 the sensor is valid and used in WSUM                    */
/* if v[a] = 0.5 the sensor is temporarily invalid & used later       */
/* if v[a] = 0   the sensor is permanently invalid & never used again */

if (ADT[90] < 10.0) {
  for (a = 0; a < 22; a++)
  {
      d[a] = d_st[a];
      if (a < 11) vmax[a] = vmax_st[a];

      if (v[a] > 0)
      {
         v[a] = 1.0;
         if (d[a] < 0) v[a] = 0.5;
         if (ADT[a+1] == 0.0) v[a] = 0.0;
      }
      else v[a] = 0.0;
  }

  /* don't use HPFT speeds if their values are actually the uncleared */
  /* bogus values (-40,000 rpm) from the pre-test sensor checkout     */
  /* (correct speeds should not be > 4,000 rpm at 0.7 seconds         */

  if ((ADT[90] < 0.7) && (ADT[9] > 4000)) v[8] = 0.5;
  if ((ADT[90] < 0.7) && (ADT[10] > 4000)) v[9] = 0.5;

  /* don't use H.G. Inj Pressure (PID 24) if it's less than MCC Pc    */
  /* after 1.5 seconds                                                */

  if ((ADT[90] > 1.3) && (ADT[17] < PL*3006)) v[16] = 0.0;
}

else
{
   for (a = 0; a < 22; a++)
   {
      d[a] = d_ms[a];
      if (a < 11) vmax[a] = vmax_ms[a];

      if (v[a] > 0)
      {
         v[a] = 1.0;
         if (d[a] < 0) v[a] = 0.5;
         if (ADT[a+1] == 0.0) v[a] = 0.0;
      }
      else v[a] = 0.0;
   }
}

/* Mark D'Val changes to ND calculation as of 9/10/91:                */
/* ****************************************************************** */

/* "ND" EQUATION CALCULATION:                                         */
```

```
 /* ------------------------------------------------------------------ */
 /* Calculate normalized measurement deviations */

 for (a = 0; a < 22; a++)
 {
   if (v[a] < 1)
     ed_nd[a] = 0.0;
   else
     ed_nd[a] = (ADT[a+1] - est[a])/d[a];
 }

 /* Added changes 9-3-92 by Ed Nemeth to smooth noise (filtering) */

number_of_samples = 5;
n_samp = number_of_samples - 1;   /* for C numbering system; starts at 0 */

/* update filter array with latest data. Number_of_samples represents the
   number of buffer cells, each one of which represents a single sample
   in time, progressively back in time each data point observed. */

for (a = 0; a < 22; a++)
  for (b = 0; b < n_samp; b++)
  {
     filt_nd[a][b] = filt_nd[a][b+1];

  }

for (a = 0; a < 22; a++)
{
   filt_nd[a][n_samp] = ed_nd[a];
}

/* This double loop is to calc the average of [number_of_samples]
   buffer cells as mentioned above. This represents the average of
   the last [number_of_samples] number of data points observed. */

for (a = 0; a < 22; a++)
{
  filt_sum = 0;
  for (b = 0; b < number_of_samples; b++)
  {
     filt_sum = filt_sum + filt_nd[a][b];
  }
  nd[a] = filt_sum/number_of_samples; /* nd is assigned the filtered value */
}
```

```
 /* "VND" EQUATION CALCULATION:                                        */
 /* ------------------------------------------------------------------ */
 /* Calculate validated, normalized deviations of each measurement (0 - 21) */

 for (a = 0; a < 11; a++)
```

```
{
  a1 = 2.0 * a;
  a2 = a1 + 1;          /* accounts for dual sensors of some measurements */

  /* dmax is the maximum standard deviation allowed between "good"   */
  /* measured and estimated values                                   */

  dmax = 3.0;   /* set dmax to 3 until a better value is determined  */

  /* if both of the redundant parameters are available then check    */
  /* whether one or the other should be disabled (never disable both */
  /* (if both redundant params indicate a problem use both in WSUM   */

  if ((v[a1] == 1) && (v[a2] == 1))
  {
    /* if P1 and P2 indicate similar trends (i.e., their measured values */
    /* are similar direction and magnitude from their estimated values), */
    /* then use both of them in the WSUM calculation:                    */

    if (fabs(nd[a1] - nd[a2]) < vmax[a]*dmax)
    {
       v[a1] = 1;
       v[a2] = 1;
    }
    else

    /* if P1 checks out but P2 does not, then disable P2 */

    if ((fabs(nd[a1]) < dmax) && (fabs(nd[a2]) > dmax))
       v[a2] = 0;
    else
       /* if P2 checks out but P1 does not, then disable P1 */
       if ((fabs(nd[a1]) > dmax) && (fabs(nd[a2]) < dmax))
          v[a1] = 0;
  }
  vnd[a1] = nd[a1] * v[a1];
  vnd[a2] = nd[a2] * v[a2];

}

/* validate(); */

/* "NPD" EQUATION CALCULATION:                    MODULE G          */
/* ------------------------------------------------------------------ */
/* calculate normalized parameter deviations for final summation of wsum */

for (a=0; a < 11; a++)
{
   a1 = 2.0 * a;
   a2 = a1 + 1;

   /* divide by the number of valid parameters, d1 = 1 or 2    */
   /* (set d1 = 1 if v[a1] + v[a2] <2 to avoid a divide by 0 or 1.5 */

   d1 = v[a1] + v[a2];
   if (d1 < 2) d1 = 1.0;

   npd[a] = (fabs(vnd[a1]) + fabs(vnd[a2]))/d1;
}
```

```
  /* "W" EQUATION CALCULATION:                                          */
  /* ------------------------------------------------------------------ */
  /* Define a weight for each normalized parameter deviation            */

  sum = 0;   /* set wsum to zero initially before calculating it */

  /* INITIALIZE THE SIG[] TABLE FOR WSUM WEIGHTING: (Added 2/92 RVdH): */

  sig[0] = 0.0;
  sig[1] = 0.1;
  sig_counter = 0.1;
  for (i=2; i<22; i++)
  {
    sig_counter = sig_counter + 0.2;
    sig[i] = sig_counter;
  }

  /* CALCULATE THE WEIGHTS FOR WSUM BASED UPON A LINEAR FIT AIMED AT
     INCORPORATING NORMAL DISTRIBUTIONS RVdH/D'Valentine 2/92. */

  /*
    for (j=0; j<11; j++)
    {
       w[j] = 4000.00;
       for (i=0; i<21; i++)
          if ((npd[j] >= sig[i]) && (npd[j] < sig[i+1]))
          {
             w[j] = k[i] + (  ((k[i+1] - k[i])/0.2) * (npd[j] - sig[i]) );
             npd[j] = 0;
          }
    }
  */

   /* CALCULATE THE WEIGHTED SUM OF DEVIATIONS FROM NORMAL MODEL VALUES: */

    for (c=0; c<11; c++)
    {
       a1 = 2*c;
       a2 = a1 + 1;

       w[c] = 1 + (npd[c]*npd[c]*npd[c]);

       if (((v[a1] < 1) || (v[a2] < 1)) && (w[c] > nonredundant_pid_limit))
         w[c] = nonredundant_pid_limit;

       sum = sum + fabs(w[c]);
    }

    return(sum);

} /* exit WSUM() */
```

```
/*****************************************************************************/
```

```
/**                                                                    **/
/**                        LEVEL "F" SUBROUTINES                       **/
/**                                                                    **/
/**       These are the bottom-most routines in the calling tree.      **/
/**                                                                    **/
/************************************************************************/
```

```
float woax()
{
  /* Calculate oxidizer backpressure. Mechanics unknown and assumed to work. */
  /* Robert Van den Heuvel (RSA programmer) 11/26/91 */

  float hxpiv, hxt1v, hxdpv, hxc, betav, hxgamv;
  float gf1, gf2, gf3, gf4, hxy;
  float hxdiv, hxdtv, hxprat, hxb4, hxkc, hxtxf, hxrov;
  float pi,r, r1, r3;
  float sum;

  pi = 3.14159;

  hxpiv = ADT[56] + 14.7;
  hxt1v = ADT[57];
  hxdpv = ADT[58];

  /* Set Woax parameters to reasonable values if the measured parameters */
  /* are missing from the incoming data stream: (12/5/91 change -- D'Val.) */

  if (((hxpiv < 15.0) || (hxt1v < 1)) || (hxdpv < 1))
  {
    r = 1.6;
    return(r);
  }

  hxdiv = 1.5260;
  hxdtv = 0.7183;
  hxc = 0.9870;
  betav = hxdtv/hxdiv;                          /* 0.470707732 */
  hxrov = (hxpiv - 200.0) / 150.0 *
  (2.096434 - 3.12524E-03*hxt1v + 1.492811E-06*(hxt1v*hxt1v)) +
   (2.728433 - 4.00388E-03*hxt1v + 1.893453E-06*(hxt1v*hxt1v));
  hxgamv = (hxpiv - 200.0) / 150.0 *
  (1.007482E-01 - 2.10756E-04*hxt1v + 1.14232E-07*(hxt1v*hxt1v)) +
   (1.543231 - 2.76844E-04*hxt1v + 9.367208E-08*(hxt1v*hxt1v));
  hxprat = (hxpiv - hxdpv)/hxpiv;
  hxb4 = pow(betav,4.0);                         /* 0.049091389 */

  hxkc = hxc * sqrt(1.0/(1.0 - hxb4));          /* 1.01215672 */
  gf1 = (hxgamv - 1.0)/hxgamv;
  gf2 = 2.0/hxgamv;
  sum = pow(hxprat,gf1);
  gf3 = (1.0 - sum)/(1.0 - hxprat);

  sum = pow(hxprat,gf2);
  gf4 = (1 - hxb4)/(1 - hxb4 * sum);
```

```
  hxy = sqrt( sum / gf1 * gf3 * gf4);
  hxtxf = 0.99539 + 8.6957E-06 * hxt1v;          /* 1.51897841 */
  r1 = sqrt(2.0 * 32.176 * hxrov * hxdpv);
  r3 = pi * (hxdtv * hxdtv)/4.0/12.0;
  r = hxkc * r3 * hxy * hxtxf * r1;

  return(r);

 ) /* woax */




float wfax()
{
  /* Calculate fuel backpressure. Mechanics unknown and assumed to work. */
  /* Robert Van den Heuvel (RSA programmer) 11/26/91 */

  float fpiv, ft1v, fdpv, fc, fbeta, fgam, fg1;
  float fg2, fg3, fg4, fy;
  float fdiv, fdtv, fprat, fb4, fkc, ftxf, frov;
  float pi,r, r1,r2;
  float sum;

  pi = 3.14159;

  fpiv = ADT[53] + 14.7;
  ft1v = ADT[54];
  fdpv = ADT[55];

  /* Set WFAX parameters to reasonable values if the measured parameters */
  /* are missing in the incoming data stream (12/5/91 modif. by D'Val):  */

  if (((fpiv < 15.0) || (ft1v < 1)) || (fdpv < 1))
  {
    r = 0.7;
    return(r);
  }

  fdiv = 1.5200;
  fdtv = 0.7663;
  fc = 0.9870;
  fbeta = fdtv/fdiv; /* 0.504144736 */
  fgam = 1.365 + 1.6E-4 * (ft1v-400.0) + 1.75E-5 * (fpiv - 400.0);
  fprat = (fpiv - fdpv)/fpiv;

  fb4 = pow(fbeta,4.0); /* 0.064615194 */

  fkc = fc * sqrt(1.0/(1.0 - fb4)); /* 1.02052112 */
  fg1 = (fgam - 1.0)/fgam;
  fg2 = 2.0/fgam;

  sum = pow(fprat,fg1);

  fg3 = (1.0 - sum)/(1.0 - fprat);
```

```
  sum = pow(fprat,fg2);

  fg4 = (1 - fb4)/(1- fb4 * sum);
  fy = sqrt( sum / fg1 * fg3 * fg4);
  ftxf = 0.99539 + 8.6957E-06 * ft1v; /* 1.513006017 */
  frov = (fpiv - 250.0) / 200.0 *
  (2.457675E-01 - 5.48004E-04*ft1v + 4.07272E-07*(ft1v*ft1v)) +
  (3.102394E-01 - 6.92109E-04*ft1v + 5.140363E-07*(ft1v*ft1v));

  r1 = sqrt(2.0 * 32.176 * frov * fdpv); /* 15717 */
  r2 = pi * (fdtv * fdtv)/4.0/12.0;

  r = fkc * r2 * fy * ftxf * r1;

  return(r);

}  /* wfax */
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1994	Final Contractor Report

4. TITLE AND SUBTITLE

Rocketdyne Safety Algorithm
Space Shuttle Main Engine Fault Detection

5. FUNDING NUMBERS

WU-323-52-84
C-NAS3-25884

6. AUTHOR(S)

Arnold M. Norman, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Rocketdyne Division
Rockwell International
6633 Canoga Ave.
Canoga Park, California 91304

8. PERFORMING ORGANIZATION REPORT NUMBER

E-9009

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135-3191

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA CR-195356

11. SUPPLEMENTARY NOTES

Project Manager, James W. Gauntner, Space Propulsion Technology Division, NASA Lewis Research Center, organization code 5310, (216) 433-7435.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Categories 15, 16 and 20

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

The Rocketdyne Safety Algorithm (RSA) has been developed to the point of use on the TTBE at MSFC on Task 4 of LeRC contract NAS3-25884. This document contains a description of the work performed, the results of the nominal test cases and a table of the resulting cutoff threshold levels, a plot of the RSA value vs. time for each nominal case, the results of the major anomaly test cases and a table of the resulting cutoff times, a plot of the RSA value vs. time for each anomaly case, a logic flow description of the algorithm, the algorithm code, and a development plan for future efforts.

14. SUBJECT TERMS

Health monitoring; Rocket engine fault detection; Space shuttle main engine; Algorithms; Real time operation

15. NUMBER OF PAGES

190

16. PRICE CODE

A09

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	