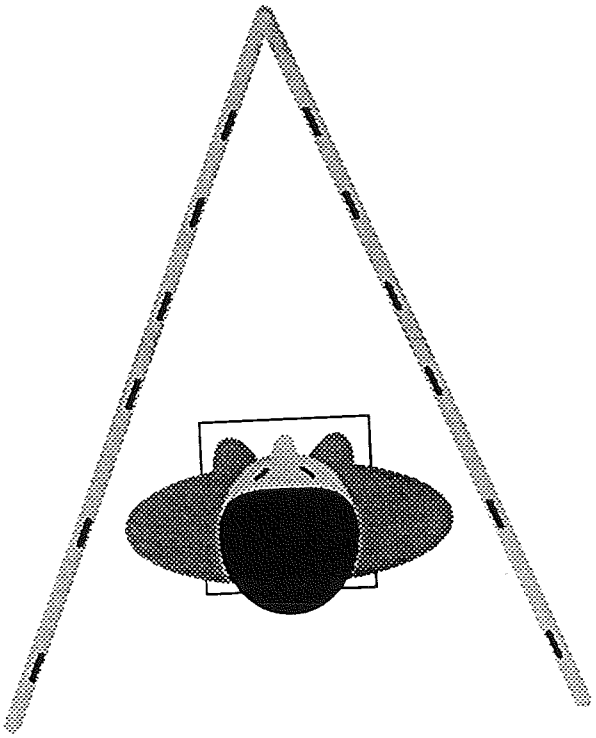
CODEN: LUTFD2/(TFRT-5436)/1–349/(1991)

Visual Contribution in Human Postural Control



Per-Anders Fransson
Patrik Sundström

Department of Automatic Control, Lund Institute of Technology
March 1991

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Master Thesis

Date of issue
February 1991

Document Number
CODEN:LUTFD2/(TFRT-5436)/1-349/(1991)

Author(s)
Per-Anders Fransson, Patrik Sundström

Supervisor
Rolf Johansson, Måns Magnusson

Sponsoring organisation
Dept. of Otorhinolaryngology, Lund University Hospital

Title and subtitle
Visual Contribution in Human Postural Control

Abstract

An important function for the human body is the ability to maintain balance. This has been compared to a sixth sense whose function we would have difficculties to be without. The system of balance is a complex system which uses several different sensors to get a satisfactory image of the position an motion of the body. One of the sensors is vision. Our assignment has been to make an equipment for stimulating the system of balance visually and to study the effect of that stimulation.

The stimulus consists of a vertical patterned screen rotating horisontally. The speed of the screen is controlled by a computer and a number of motion patterns can be programmed. The movements of the test subject are measured by a force platform.

We have found that the equipment provides visual postural stimulation leading to moment reposnses and also that the test subjects feel that their balance is affected. It is likely that other receptor systems give contribution even during powerful visual domination. However, nonparametric identification shows high coherence in frequency ranges around 0.05-0.2 Hz. The results from the analyses within this range can therefore be considered significant. The analysis within the above frequency range show an even proportionality between movements of the surroundings and ankle torque, almost independent of frequency, but the phase may vary with the properties of the stimuli. With a pure sinusoidal stimulus the phase shift was relatively stable at about 180^o, while the other test sequence used (such as the PRBS-stimulus) showed a frequency dependent phase shift starting at 0^o, 90^o, or 180^o. Parametric identification exhibits heavy variations in the time delay between stimulating input signal and response in form of moment changes. Especially active motion patterns with frequent transient elements have short time delays, but continuous motion patterns, like the sinusoid, prolongs the response up to 2-3 s after a while. At occasional times a delay of up to 6 s could be recorded. It is possible to find an ARMAX-model being a good estimation of the real system, giving low residual and correlation values.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
349

Recipient's notes

Security classification

VISUAL CONTRIBUTION IN HUMAN POSTURAL CONTROL

Per-Anders Fransson
&
Patrik Sundström

LUND february 1991

Contents

Appendices

Summary

An important function for the human body is the ability to maintain balance. This has been compared to a sixth sense whose function we would have difficulties to be without. The system of balance is a complex system which uses several different sensors to get a satisfactory image of the position and motion of the body. One of these sensors is vision. Our assignment has been to make an equipment for stimulating the system of balance visually and to study the effect of that stimulation.

The stimulus consists of a vertical patterned screen rotating horisontally. The speed of the screen is controlled by a computer and a number of motion patterns can be programmed. The movements of the test subject are measured by a force platform.

We have found that the equipment provides visual postural stimulation leading to moment responses and also that the tests subjects feel that their balance is affected. On the other hand, the tests show that the control system is difficult to identify with the visual stimulation as only recorded input. It is especially hard to know when vision dominates the control of posture. It is likely that other receptor systems give contribution even during powerful visual domination. However, nonparametric identification shows high coherence in frequencies around 0.05-0.2 Hz. The results from the analyses within this range can therefore be considered significant. The analysis within the above frequency range show an even proportionality between movements of the surroundings and ankle torque, almost independent of frequency, but the phase may vary with properties of the stimuli. With a pure sinusoidal stimulus the phase shift was relatively stable at about 180°, while the other test sequences used such as the PRBS-stimulus showed a frequency dependent phase shift starting at 0°, 90° or 180°. It is therefore likely that different motion patterns affect the phase differently.

Parametric identification exhibits heavy variations in the time delay between stimulated input signal and response in form of moment changes. Especially active motion patterns with frequent transient elements have short timedelays, but continuous motion patterns, like the sinusoid, prolongs the reponse time up to 2 - 3 s after a while. At occasional times a delay of up to 6 s could be recorded.

It is possible to find an ARMAX-model being a good estimation of the real system, giving low residual and correlation values.

1 Introduction

An important function for the human body is the ability to maintain balance. The system of balance is a complex system which uses several different sensors to get a satisfactory image of the position and motion of the body. The body is mechanically unstable construction with a highly placed centerpoint of gravity and only two legs. This creates the need for a more complex control system than needed by most animals. The postural control must therefore be fast, which demands several levels of decision, from the spinal cord up to the central nervous system. The human solution to the control problem is interesting out of several reasons.

It is a hierarchical control system, where every level tries to solve the control problem the best way possible, with regard to ability and access to information on the body situation. The system also has the ability to collect information from several receptor systems without conflicts arising. The receptor system with the best quality and most quantity of the information is allowed to be dominating in postural control. The situation of the system is however continuously verified and complemented with feedback from the other receptor systems. The control system also contains different forms of adaptation, e.g., learning motions, patterns and different situations of balance like, e.g.,walking which is completely automatically after learning.

Another form of adaptation is the continuous mapping of motion patterns during stimulation, which reduces instability.

But the system of balance can get out of order. If the information from the different receptorsystems are too contradictory, it is not possible for the signals to be put together into a uniform image of position and motion of the body. This causes dizziness. Dizziness can also be due to diseases or injuries in connection to the receptor systems. From a medical point of view it is therefore of interest to study postural control, and maybe be able to get some diagnostic information from possible defects.

In this study we have chosen to concentrate on the visual influence on postural control.

Previous studies in this area have been made with relatively large test settings, such as moving rooms and spinning tents. The visual stimulus has also often been combined with postural motion stimulus by letting the test subjects stand on a moveable platform. With the present setup

one can study how the visual stimulus is related to the other receptor systems.

We have chosen to use only visual stimulus, with the intention to make vision dominating in postural control. The movement of the body then show how the stimulus is interpreated by the postural control. As it is known that the perturbing stimulus is only visual, it is possible to see how postural control adapts to the stimulus after a while.

When we designed the equipment we wanted to get an as effective stimulus as possible with an equipment of moderate size. It should be possible to move the equipment without too much trouble.

As the equipment is intended for further use by persons other than ourselves, it was also essential to make it easy and safe to handle.

The study also includes a series of measurements made on a small group of testsubjects with the equipment. These tests where made to verify the function of the equipment and also to see if it was possible to make a control model of visual influence on postural control.

2 A brief biological background

The system of postural control is made up of several partly independent receptor systems. Signals from these systems gives different aspects on the situation of the body and can be put together to form a full image of the position of the body and of ongoing changes. The system can be modelled as shown in figure 2.1.

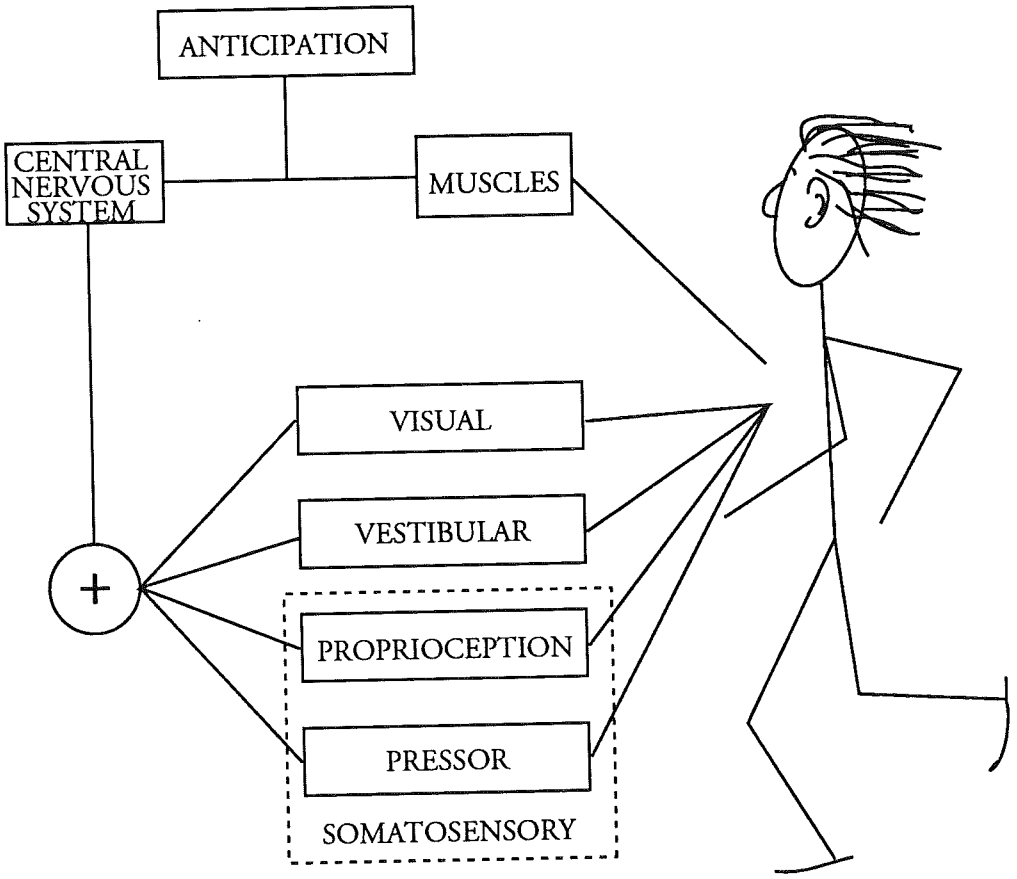


Figure 2.1 Physical model of the system for maintaining posture.

The different receptor systems are described shortly in the following summary.

1.1 Vision

The visual system has mainly three important functions concerning motion [15].

1 Visually triggered movements: Fast, unexpected changes of the visual

image can trigger trained behaviours with aim to correct for a presumed movement of the body. Thus the effect of reflexes, can more efficiently be guided towards a proper correction (see also anticipation).

2 Ballistic determination: We have the ability to determine, the speed of and distance to a target. From this information we can then decide with which force and direction a movement has to be performed with to reach the target. Visual input thus contributes with information necessary for planning of movements.

3 Tracking movements: If you want to visually track an object , you can instantly recognize if the path of the target changes. In this case you use the visual information for continuous corrections of the tracking motion, which is necessary to maintain sharp vision.

Hence, one may expect the contribution from vision in postural control to depend on how the subject observes the surroundings. If you simulate moving surroundings directed towards a standing person you may expect two principally different responses.

1 Conscious countermovement (cortical): The subject perceives that no forces are affecting the body, but interpretes the visual input as the body may be in motion without accelerating (compare, standing in a moving bus), and (by experience) prepare to meet a disturbance of opposite direction.

2 Unconscious countermovement (subcortical): In this case the subject interprets the surroundings to be still while the own body is moving. This induces a movement counteracting the imagined motion. The response will thus be directed in the same direction as the visual stimulus.

It should also be mentioned that anticipation affects the responses, especially if the stimulation has continued for a while.

In terms of stability, vision gives a horizontal reference, making it possible to detect small changes in the position of the body. The other receptor systems may require larger changes in position before reacting. Visual input detects low frequency changes ($0.05<\Delta f<0.1$ Hz) in body position, and contributes with extra stability within this frequency range.

Furthermore, one can visually determine how fast one is moving and

when one will reach the goal, thus interrupting the motion in time [9].

Vision can be complementary to other systems of postural control. In patients who have lost the vestibular function visual input may suffice to maintain posture. Even small linear and rotational movements can be detected by the retina and contribute to postural control. On the other hand the subjects may show impaired balance, when exposed to violent motions or loosing visual information by closing the eyes. This is due to the fact that vision works within a limited frequency range and that a certain visual information is needed to give a contribution to stability in thoose subjects [7].

2.2 The vestibular organ

The vestibular organ in the inner ear, is especially adapted to detect changes in posture and senses motion and position of the head [7]. It also has the ability to influence the visual information and the position of the eye. If one didn't have a mechanism automatically correcting the eye, it would be impossible to maintain a steady image at rapid head movements. To accomplish these corrections, signals from the vestibular organ triggers reflexes that causes eye movements counteracting the head movements. The vestibular organ is also used to trigger predictive movements. The CNS interprets the vestibular signals and finds that a motion is heading towards loss of stability, counteracting movements are then automatically invoked before balance is lost [7].

The vestibular organ consists of three semicircular canals and two sacs (the otolith organ). The inner ear also contains structures for the function of hearing. The semicircular canals sense the angular acceleration of the head. They are situated in three different planes to detect movements in three dimensions.

Angular acceleration of the head will induce a flow of the endolymph fluid contained in the semicircular canals. The flow will act upon the cupula, a swingdoor-like structure, bending it. Simultaneous the hairs (or cilies) of the sensory epithelium embedded in the cupula will be bent as well. The more these cilies are bent, the higher the frequency of de-polarization of the afferent vestibular neurons. Thus, a neuromechanical transduction is achieved. After a velocity step of head movements, the neural impulses will decay exponentially with a time constant of about 6 - 7 seconds in the vestibular nerves, see figure 2.2. The central nervous system will delay the exponential decay to reach a time constant of about 10 - 15 s, which corresponds to a frequency of 0.05 Hz. The otolith

organ (Macula) are especially adapted to sense the orientation of the head in reference to linear acceleration such as gravitation. They contain crystals (otoliths) surrounded by a gelatinous layer on ciliated sensory cells. Linear movements of the head induce movements of the otoliths and bends the cilies which increases the frequency of impulses in the afferent nerves [15].

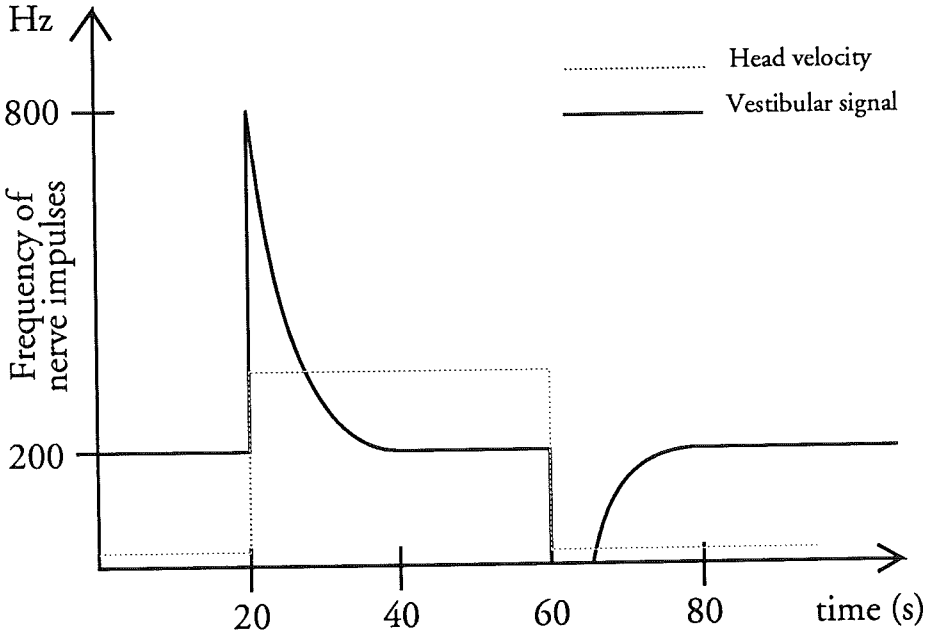


Figure 2.2 Stimuli response of the nerves in the semicircular canals.

The difference between the otolith organs and the semicircular canals is that the canals are individually one-dimensional, while the otolith organ, utricle and possibly saccula, are mainly two- or three dimensional [4].

Adaptation of vestibular influence is possible. If the subject is frequently exposed to rotation or motions of low frequencies as e.g. sailors and ballet dancers, the responses decreases. Ballet dancers can, by training, lower the decay time of the central nervous system to about six seconds, which is the same as the swiftness of the vestibular organ itself.

In monkeys [12] it has been found that movements around 0,005 - 0,05 Hz causes vestibular adaptation. On the other hand, one did not find such tendencies for frequencies above 0.1 Hz. It is likely that adaptation acts in the same low frequency range also in man.

2.3 Proprioceptive sensors

These receptors, muscle spindles, Golgi tendon organs and mechano receptors in joints and tendons, sense the position and tension in

muscles, tendons and joints Thereby detecting the position of the different body segments, and the load on these segments [8].

The proprioceptive receptors in the neck seems to be of special importance, since one must know the position of the head relative to the abdomen if the signals from the vestibular organ is to be correctly interpreted. E.g., when the neck is flexed the vestibular signals has to be reinterpreted.

2.4 Pressor receptors

These pressure sensors or mechanoreceptors are situated in the human skin. When one is standing up the sensors in the soles are of special importance, they sense the pressure force affecting the feet, and thereby how the weight is distributed. By sensing the position of the centerpoint of pressure, one will get feedback of the lateral and sagittal moments of the body.

2.5 Anticipation

Learned experiences make it possible to reduce the effects of a disturbance by changing the centerpoint of gravity in advance, e.g, when one is approaching a stair, one leans forward before the first step. This is done to maintain balance when the body is moving forwards and upwards the stairs. Babies haven't learned this, to adults unconscious, displacement, but they fall backwards when they try to ascend the stairs [H C Diener 1988].

To activate anticipation you have to be able to interpret the coming changes, e.g., that you are about to ascend the stairs. In this case you use the visual clues to identify the stairs, how fast you are approaching and how steep the stairs are. You can also use hearing or memory to anticipate changes, e.g., when you want to change from walking to running. Anticipation can be looked upon as a kind of feedforward or predictive feedback [7].

2.6 Central nervous system (CNS)

The central nervous system integrates all data from the different receptor systems involved in maintaining posture. Somatosensory, vestibular and visual information affect each other at several different levels of the CNS in postural control. Signals from proprioceptive

sensors and the vestibular organ interact already in the spinal cord, affecting simple movements and muscle reflexes [7].

All afferent information from proprioception, vision and the vestibular organ converges with output from the motor cortex in the cerebellum. It is considered that the cerebellum, primarily through feedback, controls ongoing movements, and maybe also passes on the results of a specific movement for future improvements (adaptation). The signals from cerebellum are considered to affect the suppressing of vestibular and spinal reflexes and the time coordination of motoric preparation and executions [7].

Data from cats have suggested that subthalamus may have the ability to store the motion data of simple movements, e.g., walking. Walking is generated by special programmes for normal rhythmic movements. But if any obstacle appears, subthalamic control cannot compensate for this, but continues to execute the "walking program" [7]. Similar properties in man has suggested this function to be depending on the basal ganglia [15].

The function of the basal ganglia is partly unknown, but it has close connections to the cerebral cortex both concerning body mechanics and sensor activity. It is also known that prior to muscle activity, control signals are activated in the basal ganglia before it is possible to measure any activity in motor cortex. It is therefore likely that the basal ganglia initiates most of the motor activity [7]. The basal ganglia is considered the largest area involved in motor control in the CNS with motor programmes and the ability to learn complex motion patterns. The fact that the somatosensory information often is of a specific nature, limited in space, has suggested that the cortical signal from the basal ganglia could be used for initiation, control or supervision of movements. The information is indirectly collected through sensor cortice. Both the cerebellum and the basal ganglia use the motor cortex and the motoric structures of the brain stem to execute their output. It is considered that the motor cortex is relatively low in the hierarchy as it is used to initiate movements both from the cerebellum and the basal ganglia [12]. Stimulating different areas in the motor cortex will induce impulses to different groups of muscles, causing contraction or relaxation. Proprioceptive sensors in the muscle sense the change and provide afferent information about the induced muscle activity.

As a summary, the system for maintaining posture uses several different systems and sensors to get a satisfying image of the position of body segments and the centerpoint of gravity. It has been suggested that

the different systems, with some overlapping, cover different ranges of frequencies in the dynamic of balance [6]. Vision seems to work in the range below 0.1 - 0.2 Hz. In the vestibular organ the semicircular canals have a range above 0.1 Hz and the otolith organ a range below 0.1 Hz. The contribution from the proprioceptive sensors are more unknown, but the receptors can sense disturbances up to 200 Hz. However, a proprioceptive disturbance of the calf muscle only gives a coherent postural disturbance below 1 Hz [11].

Much of the control work is initiated at the level of the spinal cord (and cerebral cortex). But other centers like cerebellum supervise and induce corrections when needed. Receptor systems who get disturbed or get wrong indications could take control from time to time, but they are eventually overridden due to the contradiction to input from the other systems [2]. It is known that blind subjects utilize information from the other organs to compensate for the loss of vision. When comparing the balance between blind and blindfolded seeing persons, the balance of the blind where better. The ability to compensate may hide a lesion to the postural control. It would therefore be beneficial if the control function of each of the different balance systems could be assessed by itself, to find a possible abnormal function. It would then be possible to get an image of the patients physical condition by a series of balance tests with special stimuli to each of the sensory systems. As a part of such a mapping we have studied how to affect the balance system visually.

2.7 Visual influence on postural control (A literature review)

Studies have been made on how the human body reacts to a postural PRBS-motion [9]. Subjects were placed on a laterally movable, servo-controlled, platform, which was moved randomly. The person's response to the movement was measured as an increase in the correcting moment. The tests showed that:

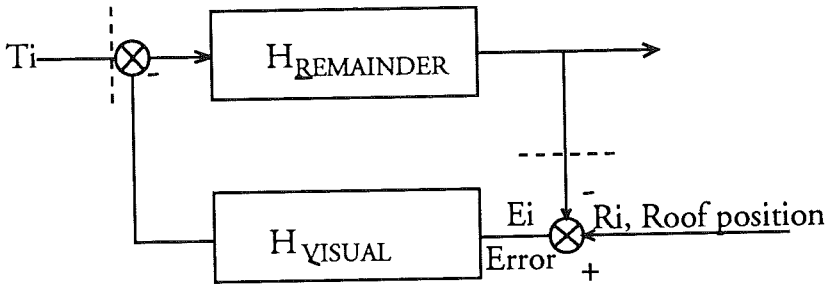
1 The phase delay in the low frequency range was less with eyes closed than with eyes open.

2 The correcting moment increases considerably with eyes closed.

Further experiences are that vision, due to its relative slowness, primarily gives active contribution at low frequencies (< 0.2 Hz). With

eyes closed the visual contribution is compensated by increased consideration to vestibular and proprioceptive sensors.

The feedback effects of vision have been studied with tests on dogs [17] according to the model in figure 2.3.



$$H_V H_R = \frac{Pi\ (optokinetics)}{Ei} = \frac{Pi}{(Ri - Pi)}$$

Figure 2.3 Control model when testing posture on dogs [17].

The dogs were placed on a laterally movable platform with mobile surroundings. The stimulation was performed by moving the platform and surroundings with varying frequencies. The measurements showed that the moment of correction increases significantly without visual information. This difference is comparable to control systems working in closed-loop and open-loop operation, respectively. Vision has a clear tendency to reduce the error if you have a stable surrounding. The tests were repeated with the surroundings moving in phase with the platform. The results corresponded to the moment response of tests with a blindfolded dog. This means that the responses referring to control can be manipulated by changing the quality and quantity of the visual information.

A series of tests with the platform describing a mixed sinusoid motion with two different frequencies, were also performed by these authors. A stable surrounding gave a moment of correction containing a sum of the platform frequencies, while a moving surrounding affected the frequency term with the same frequency as the motion of the surrounding. When using different frequencies with the platform and surroundings, the visually interpreted frequency of the surrounding gave remarkably larger (7 times) responses in the moment of correction than the frequency of

the platform. This indicates that if the visual information differs from the information from the other sensors, the control response due to visual input dominates the total response. Only visual stimuli on the other hand give weaker reactions. If the motion of the table is described as a white noise signal, i.e., contains an equal amount of all frequencies, then the system response will be a sinusoidal motion.

Bronstein *et al* have tested persons standing on a fixed platform in a mobile room [1]. When moving the visual surroundings it was shown that the test subjects made a reflex motion about 0.6 seconds after the moving of the room commenced. The compensating movement is first directed in phase with the stimuli followed by a counterphase movement. Repeated stimuli gave significantly weaker responses, if the subject had full information from the proprioceptive receptor systems in the lower part of the body.

Patients without the vestibular organ can also adapt their postural control function if the stimulation is repeated. It is therefore considered that the proprioceptive receptors have their sensitivity in a frequency range low enough to sense the stimuli (motion speed 3.76 °/s), which is not the case with the vestibular organ. The ease with which this change between receptor systems seem to happen, indicates that changes in priority between the normally dominating vision and the vestibular-, proprioceptive systems are often made. The conclusion is that vision seems to dominate if the signals from proprioceptive systems and the vestibular organ are ambiguous. This hierarchy can however quickly be readapted.

Tests on visual influence of postural control on patients with Parkinson's disease (PD) and cerebellar diseases (CD) have been performed, and compared to normal subjects [2]. It is known that an erraneous visual influence on postural control can be adapted away quickly, especially if other receptor systems are undisturbed and gives contradictory stimuli responses [1]. This means that the influence of vision is continuously CNS- controlled. In the referred study test subjects were placed on a fixed force platform in a mobile room. The measurements showed that CD-patients swayed more than controls, but they could suppress the sway after repeated stimuli. PD-patients also swayed more than normal. Furthermore, the response of PD-patients is remarkably larger to repeated stimuli than that of the controls and the CD-patients. Another observation was that postural recovery after a

stimuli sequence was very weak among the PD-patients. This indicates that that they are more dependent of vision. The ability of adaptation at repeated stimuli and the ability to use several receptor systems are weak.

The visual information is also affected by the vestibular input [21]. Partly by the reflexation used to stabilise the visual image when moving, but also when a subject tries to fixate an object, which is fixed in relation to the head, during rotation of the body. In the latter case the object is perceived to move against the direction of the body rotation.

An important aspect on postural control is if it varies depending on the character of the motion stimuli i.e. if it's continuous or transient [13]. The postural control system works as an open-loop system or as a feedback system. To study this, test were made with a moving platform and mobile surroundings moving in phase with the platform. With this the test subjects response to postural changes in motion were studied. It was shown that the subjects learn a pattern quite fast and thereby adapt to a faster response when the stimuli has the form of a continuous waveform. Even in a PRBS-disturbance it is possible to find clues to a method of operation that improves the function. E.g. it is possible to recognise the amplitude of the motion and that it changes to a motion in the opposite direction. It is on the other hand considered that vision doesn't have much influence once the pattern is learned. This is independent of whether it's a transient or continuous disturbance. However, vision can be used to create a better model when the motions pattern is mapped. There wasn't any difference between transient and continuous control, referring to vision.

3 Physical models

Several attempts to form a control model for the postural control system have been made [8]. An often reappearing suggestion is the presence of an internal model in the brain, estimating the dynamics of the semicircular canals or the dynamics of the body in rotation. In the model below, the function of balance is compared to a filter with positive feedback and a time constant close to that of the semicircular canals. The influence of the internal model can be seen if the signals from a receptor system are interrupted e.g. by turning off the lights. It appears that body movements or optokinetic stimuli will be following the latest estimation of the internal model according to a simple instantaneous model. The

internal model can be seen as a continuously generated best estimation of head velocity based upon the most resent receptor signals, especially vision, weighted by vestibular corrections. If reliable receptor information is lost or if the different receptor signals are in conflict, the state variables of the model are updated depending on the model of dynamics estimated this far [Borah *et al* 1980].

A number of models describing how the visual - vestibular cooperation could be working have been presented [7].

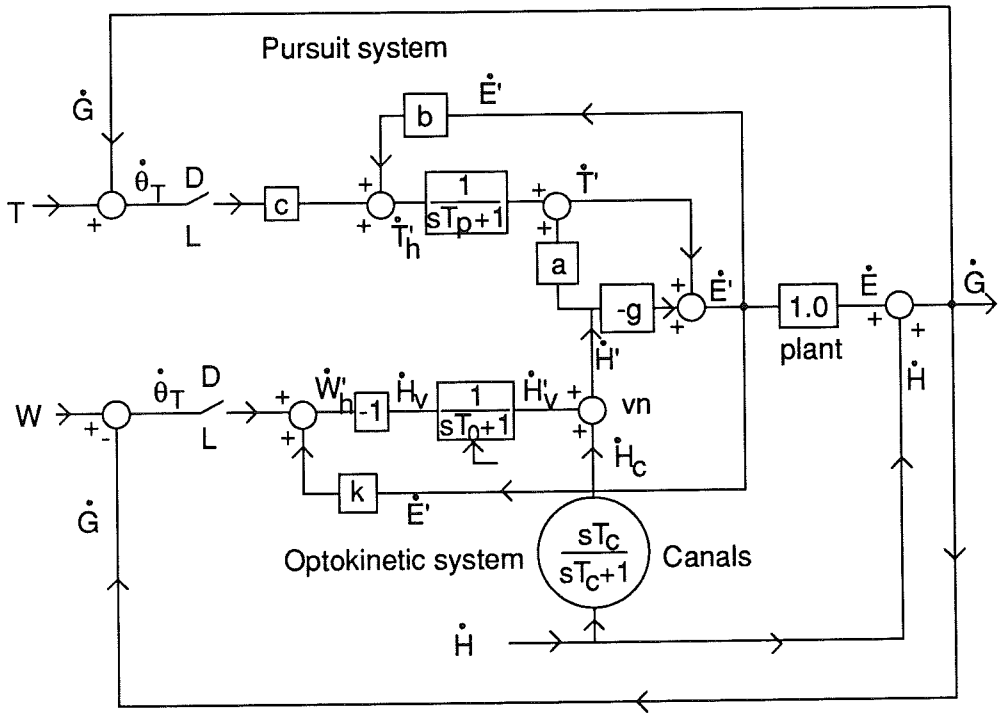


Figure 3.1 Model of visual-vestibular interaction [Robinson] combining the vestibulo-ocular reflex (VOR) with the optokinetic and pursuit systems. H = head velocity, W = world velocity and T = target velocity. Switches D and L corresponds to dark and light [6].

The models of Robinson *et al* and Raphan *et al* are specialised for slow eye movements. They share a switch for fixation suppression, adding of optokinetic and vestibular commands and an internal model of, alternatively filtration with a dominant time constant longer than that of the semicircular canals to resist speed commands. With this method vision complements the vestibular organs incapacity to handle low frequencies by utilising the velocity of the eye. This feedforward gives a high gain and long time delays between stimulus and response (up to 16 seconds in darkness). Another suggestion is given by Zacharias and Young [6].

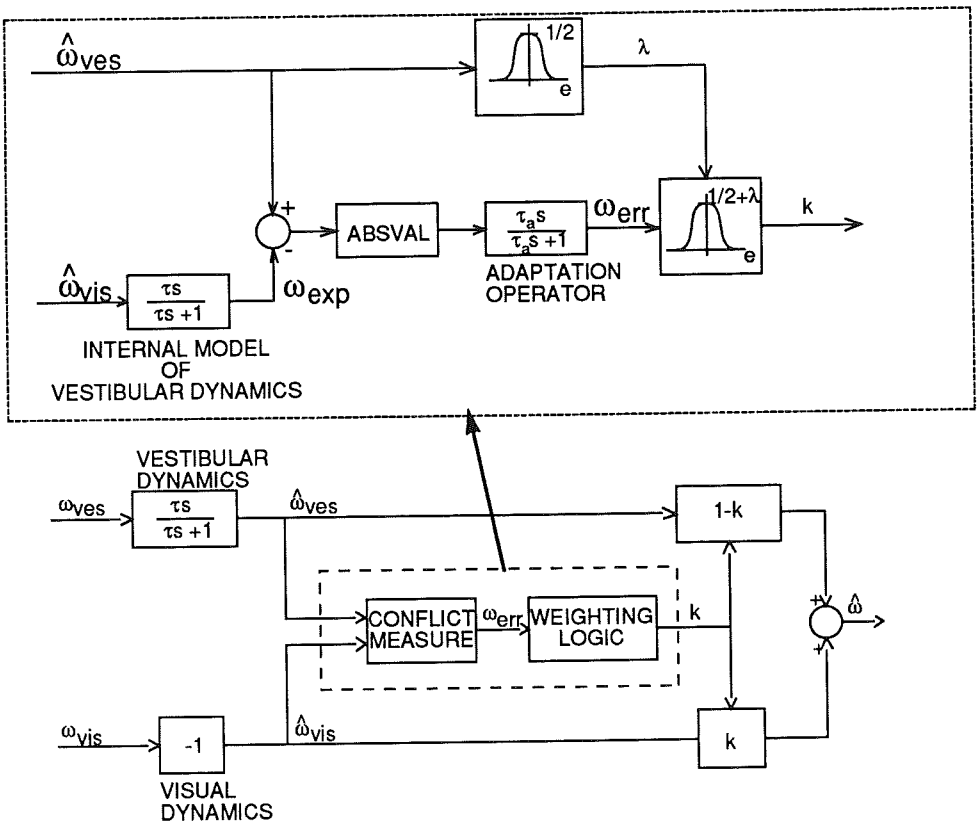


Figure 3.2 Control model by Zacharias and Young [6].

This model is based on the idea that the visual input first is weighted in comparison to an internal model of vestibular dynamics with the same transfer function as the topical vestibular system. The resulting weighting constant decides how much of the result that are to be collected from each of the systems when controlling. This model has some adaptivity to minimize the number of conflicts.

3.1 The model used in our identification

In stance it is possible to form a more complete model. We have decided to look at the model below when we made our identification. The advantage with this model is that you create a simple mechanical model of the system to be controlled by the balance system. If the body only performs small movements, it can be assumed that all torque is around the ankle [11]. This means that the body can be seen as an inverted pendulum according to fig 3.3.

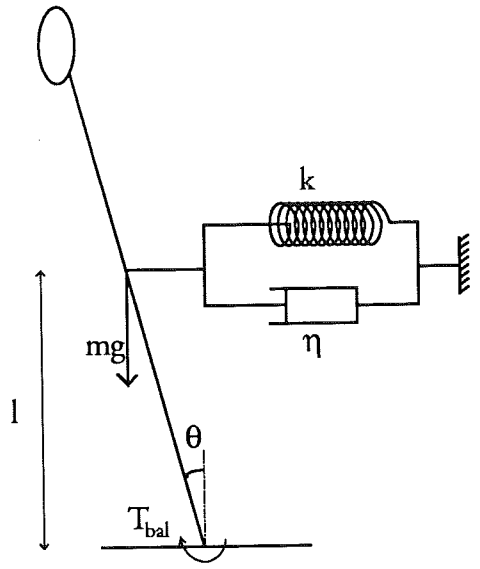


Figure 3.3 Inverted pendelum model of human postural dynamics.

If the model is to be valid the following assumptions must be made.

1 The body is stiff and has a mass m (kg).

2 The body center of mass is located at distance l (m) from the platform surface.

3 The body can be modelled as an inverted pendulum, whose moment of inertia around the ankle is given by

$$J\frac{d^2\theta}{dt^2} = mgl\sin\ \theta(t).\ \ J = ml^2$$

4 There is a stabilising ankle torque T-bal(t)

5 There is a disturbance torque Td(t) from the environment This gives

$$J\frac{d^2\theta}{dt^2} = mgl\sin\ \theta(t) + T_{bal}(t) + T_d(t).\ \ J = ml^2$$

6 T-bal stabilises with PID-control with the components P, I and D determined by coefficients k, n and p

$$P : -mgl\sin\ \theta(t) - kJ\theta(t)$$

$$D : -\eta J\theta(t)$$

$$I : -\rho J\int_{t_0}^{t} \theta(t)dt$$

7 The visual impression gives a misperception of the position and speed according to

$$P : -mgl\sin\ \theta(t) - kJ\theta(t) + b_1 v(t)$$

$$D : -\eta J\theta(t) + b_2 v(t)$$

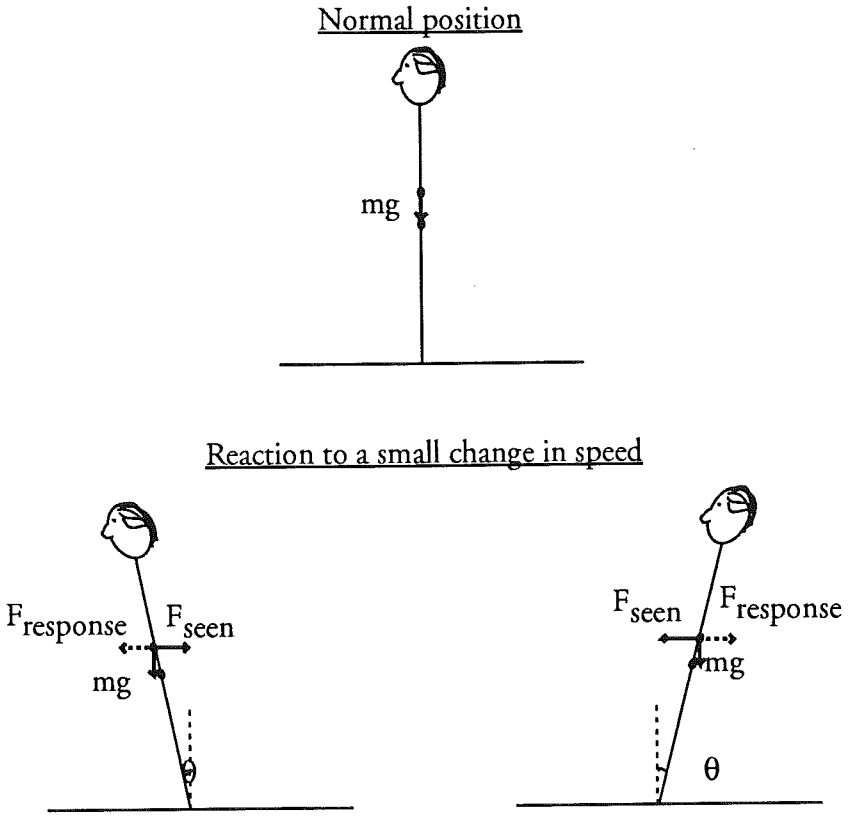


Figure 3.4 Ankle strategy as a response to visual stimulation.

An alternative to the model above could be a model with flexing motions in ankle and waist. Previous studies have however showed that such a model is most common in large moment responses [14]. Shear forces in the horizontal plane can also be observed from the platform in these cases. This model gives us a system of the fourth degree. Our

observations on the movements of the testsubjects, the size of the shear forces and the parametric estimations give insufficient evidence to suggest the latter model to be valid.

Considering the assumptions above our transfer function will be

$$J\frac{d^2\theta}{dt^2} = mgl\sin\ \theta(t) + T_{bal}(t) + T_d(t)$$

The ankle torque t-bal as a PID-regulator gives

$$T_{bal}(t) = -mgl\sin\ \theta(t) - kJ\theta(t) - \eta J\dot{\theta}(t) - \rho J\int_{t_0}^{t}\theta(t)dt$$

By transforming the function with the Laplace transform we will get

$$s^2J\theta + sJ\eta\theta + Jk\theta + \rho J\frac{1}{s} = T_d(t)$$

$$\theta(s) = \frac{\frac{1}{J}s}{s^3 + \eta s^2 + ks + \rho}T_d(s)$$

When the system is affected by disturbances it will transform to

$$\frac{Jd^2\theta}{dt^2} = -\eta J\frac{d\theta}{dt} - kJ\theta(t) + (b_1+b_2)v(t) - \rho J\int_{t\theta}^{t}\theta(t)dt + T_d$$

$$\Rightarrow$$

$$s^2J\theta + sJ\eta\theta + Jk\theta + \rho\frac{J}{s}\theta = (b_1 + b_2)V(s) + T_d$$

$$\Rightarrow$$

$$\theta(s) = \frac{\frac{1}{J}(b_1 + b_2)s}{s^3 + \eta s^2 + ks + \rho}V(s) + \frac{\frac{1}{J}s}{s^3 + \eta s^2 + ks + \rho}T_d(s)$$

If the deflection around the motion axis are small, the approximation $\sin(\theta)\approx\theta$ can be made.

$$T_{bal}(s) \approx (Js^2 - mgl)\theta(s) + T_d(s)$$

$$\Rightarrow$$

$$T_{bal}(s) \approx \frac{1}{s^3+\eta s^2+ks+\rho}\left[(Js^2-mgl)\left(\frac{1}{J}(b_1+b_2)sV(s)+\frac{1}{J}sT_d(s)\right)-T_d(s)(s^3+\eta s^2+ks+\rho)\right]=$$

$$=[J=ml^2]=\frac{1}{s^3+\eta s^2+ks+\rho}\left[(b_1+b_2)\left(s^2-\frac{g}{l}s\right)V(s)-\left(-s^3+s^3+\eta s^2+ks+\rho+\frac{g}{l}s\right)T_d(s)\right]=$$

$$=\frac{1}{s^3+\eta s^2+ks+\rho}\left[(b_1+b_2)\left(s^2-\frac{g}{l}s\right)V(s)-\left(\eta s^2+\left(k+\frac{g}{l}\right)s+\rho\right)T_d(s)\right]$$

This model has been suggested by Johansson *et al* [11] when stimulating directly on the muscle. Unfortunately assumption 6 and 7 are uncertain in our case since it has been found that the visual input can be made of minor importance compared to other receptor input if it is disturbed and thereby gives incorrect information [1]. It is also necessary to consider if the system will be working with open-loop or closed-loop control, or if the system will adapt to the stimulus and change between the two modes of control [19].

The example below illustrates how a system may change between different modes of control, depending on input signals.

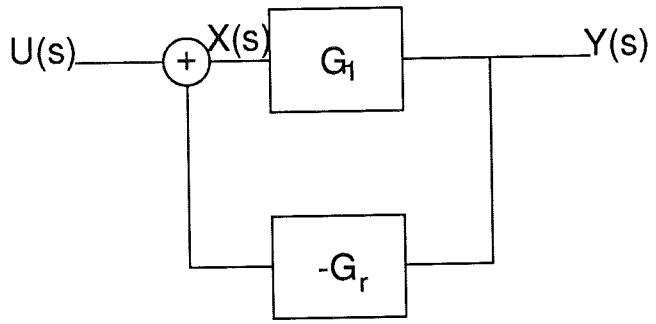


Figure 3.5 Simple control model.

A system with strong input signals will affect a control system of the type shown in figure 2.5 to change transfer function to a system of type .

$$U(s) >> |-Y(s)G_r| \Rightarrow Y(s) \approx U(s)G_1$$

If the stimulation, on the other hand is of the same level as the feedback signal, the system will be of type

$$X(s) = U(s) - G_rY(s) \Rightarrow Y(s) = G_1U(s) - G_1G_rY(s)$$

$$\Rightarrow$$

$$Y(s) = \frac{G_1}{1-G_1G_r}U(s)$$

3.2 Linearizing the relationship between stimulus and moment response

A linear relationship between input and output is desired for identification. In the model described in the previous section, the input signal will be given as a visually interpreted velocity. The output will be the moment response invoked by the postural control system. A model of the type

$$\tau = M(\theta)\ddot{\theta} + V(\theta,\dot{\theta}) + G(\theta)$$

is often used in mechanical systems of segmented rigid links [3].

τ	=	moment
$M(\theta)$	=	moment of inertia matrix
$V(\theta,\dot{\theta})$	=	centrifugal and Coriolis terms
$G(\theta)$	=	gravity term
θ	=	flexing angle
$\dot{\theta}$	=	angle velocity
$\ddot{\theta}$	=	angle acceleration

This model may also be used in a simple mathematico-geometric model of the segmented, articulated human body [3]. As our input signal has the form of $\dot{\theta}$, we will get a linear relation to θ-terms in the expression above. But our equipment has not the ability to measure the position (θ) and accelerations ($\ddot{\theta}$). One possibility to compensate for the acceleration, is to linearize the expression for acceleration.

In our case the test subjects are exposed to a visual stimulus, giving an illusion of velocity. The subjects react to the stimulus with movements around the ankle, see figure 3.4, where the ankle torque is given by

$$M = F \cdot x$$

where x is the distance from feet to estimated centerpoint of body mass.

At an acceleration the force F is given by

$$F = m \cdot a = m\frac{dv}{dt}$$

$$\Rightarrow$$

$$F\,dt = m\,dv \Rightarrow \int_{t}^{t+t_0} F\,dt = \int_{v}^{v+v_0} m\,dv$$

At a short time interval the acceleration can be seen as being linear, which gives

$$F \cdot t_0 = m \cdot v_0$$

This gives the following equation for the moment

$$M = F \cdot x = x \cdot m\frac{dv}{dt}$$

$$\Rightarrow$$

$$M = x \cdot mdv \Rightarrow x \cdot \int_{t}^{t+t_0} F\,dt \Rightarrow x \cdot \int_{v}^{v+v_0} mdv$$

$$\Rightarrow$$

$$M \cdot t_0 = x \cdot m \cdot v_0$$

Which gives us a linear relation between the velocity input and moment output.

4 The test equipment

To perform the visual stimuli we wanted to use something that was not too big or too expensive. Previous tests with visual stimuli has been performed with moving rooms, spinning tents and similar big settings. The reason for this is that the stimuli should cover up the visual field of the test subjects, and that a real object moving is better than a projected image [18]. We also wanted the stimuli to give the illusion of movement forwards and backwards, not only sideways or rotation. To achieve this we decided to move an irregular pattern past the test subject on both sides. This would hopefully give the subject an impression of movement, and by changing the speed and direction we should get the subject to compensate for these changes. The compensation movements could then be measured by the force platform described in appendix III.

All this gave us the following guidelines for the equipment:

* Move a real object, with a pattern, past the tested person on both sides.

* Make it big enough to cover the visual field.

* Control the stimulus pattern with a computer, thus making it easier to compare the measured results with the stimulus.

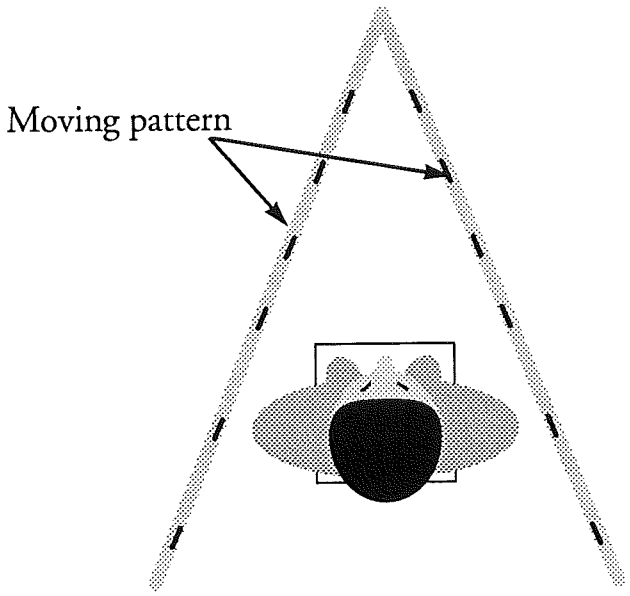


Figure 4.1 The principle for the stimulation equipment

Finally we decided to use a vertical screen moving horisontally at one side of the person. On the opposite side we placed a mirror instead of another screen.

4.1 The device for stimulation

The screen is 1.2 m high and 1.5 m wide, placed in level with the subjects eyes. The screen and the mirror should be as close together as possible in front of the person, and the mirror should be placed parallel to the subject's sagittal plane. Obviously this combination is impossible as it requires to place the mirror through the center of subject. It is however, important to minimize the gap between the screen and its mirror image as otherwise the subject might find a non moving object to act as a reference. This means that he could disregard the stimulus as long as he concentrates on the same non moving object. The compromise made, is that the mirror is slightly angled, but placed as close to the subject as possible without touching him. As a result of this the stimulus are not completely identical on both sides. A little difference in the angular velocity of the pattern will probably induce a small rotating movement in the subject.

The moving screen is made of a 3.0 m long and 1.2 m wide rubberised fabric. On this we put black vertical stripes of tape with irregular distances. The tape causes the screen to make a little more noise and should preferably be replaced with paint. The screen is stretched around two steel rollers and guided by two belts, vulcanised to the back side. One of the rollers is attached to a DC-motor which is controlled by a servo system connected to the computer. To control the speed of the screen you simply give the servo system an appropriate DC-voltage from the computer. A diagram showing the relationship between the control voltage and the speed of the screen is shown in appendix III.

Because of the relatively large screen and the heavy rollers, together approximately 48 kg, the frame had to be made rather heavy. It is also a fact that people are of different height, therefore we had to make it adjustable in height. A drawing of the frame is shown in appendix I.

For the mirror we made two separately adjustable legs that can be placed anywhere at the mirror. We did this to make it easier to place the mirror where we wanted. The setting is quite big but it is rather easy to handle and adjust.

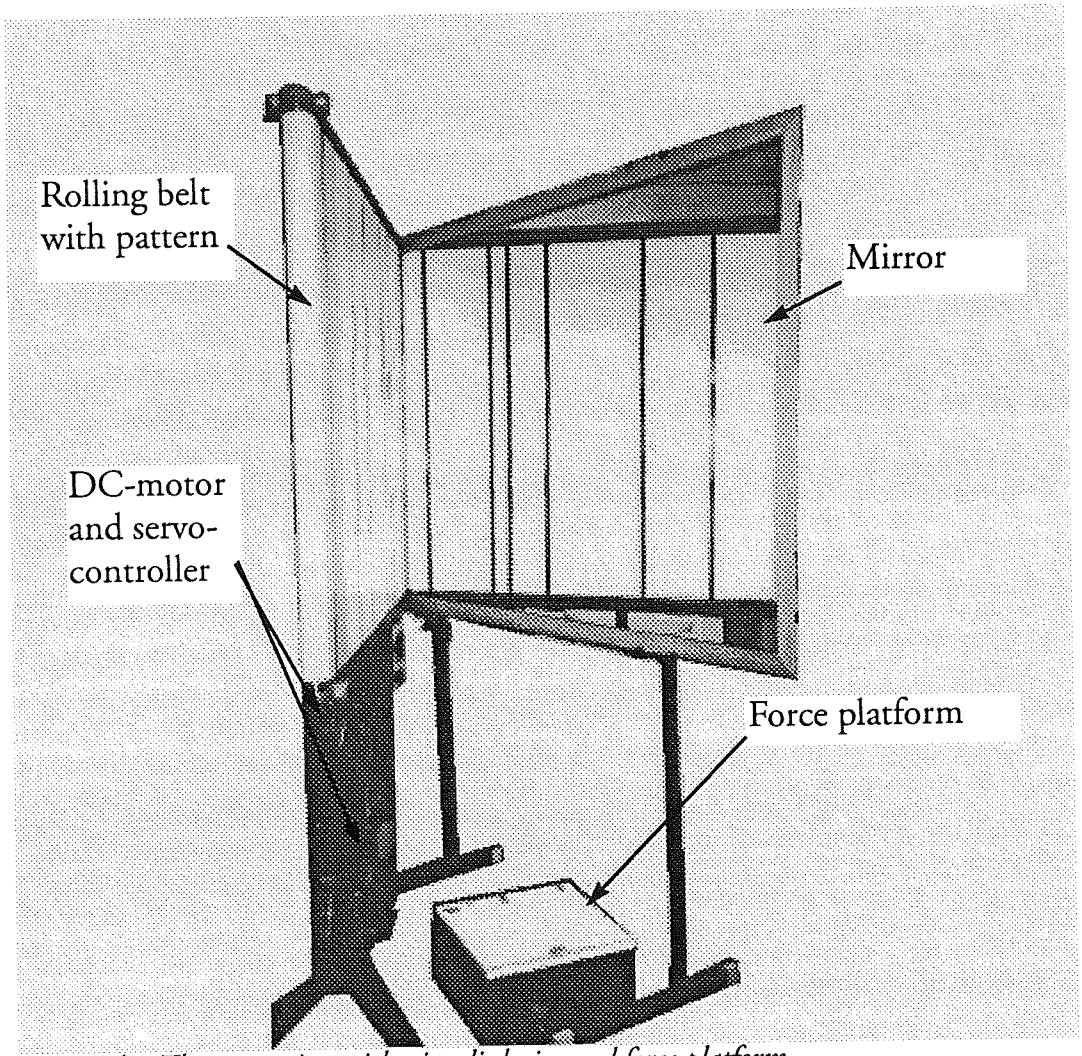


Figure 4.2 The test setting with stimuli device and force platform

4.2 The control program

The control program (VISCON) is written as a real time system with different processes for measuring, generating test sequences, user interface and so on. It is written in Logitech Modula 2 and intended for use with a IBM AT-compatible or higher PC. We have also used a library of menu- and plot routines developed at the Department of Automatic Control, Lund Institute of Technology.

The program has a mouse driven graphical user interface, which is described in the Viscon user guide

There are four main tasks for the program to handle.

1 Let the user control the test equipment.
2 Generate the test sequences.

3 Give control signals to the motor.
4 Collect measure data from the force platform

It is also possible for the program to perform an on-line estimation of the measured control process parameters.

The program consists of a number of processes running in parallel. These are:

* Control
* Measure
* Plot
* Estim
* Testgenerator
* Safe
* Error
* Help

They communicate with each other through a number of monitors (i.e., a monitor is a collection of variables and procedures accessible by one and only one process at a time.):

* Status monitor
* Motor monitor
* Plot monitor
* CLSpec monitor
* Data monitor

Plot, *Estim* and *Measure* also has mailboxes as a mean of communication.

Each of the processes and their way of communicating is described below.

4.2.1 Processes

Measure

Measure is the process that handles all input and output of the system. It collects data from the force platform and generates the output signal to the motor. This is done to simplify the synchronisation of input and output data. *Measure* calls a procedure in *Status monitor* to see if a test is running, if so it obtains the motor speed from *Motor monitor* and gives

the corresponding voltage to the motor. Thereafter it scans the inputs from the platform and writes both input and output data to a file, if it's a new test it opens a new file. The opening and closing of a file is a task that takes a long time compared to the actual writing, therefore the process collects and saves more than one set of data before writing. The number of data scans before writing can be set by the user. *Measure* also calculates the test subject's moment around the ankle in the sagittal plane. When these tasks have been completed the process send the collected data to the mailbox of the plot process and puts them into *Data monitor.* To make it possible to plot and estimate an old test,, *Measure* can load data from a file and treat them as if it where a test running.

Testgen

Testgen is the process that generates the test sequences and provides *Measure* with the value of the wanted speed. Each test begins with period of no stimulus, to let the test subject be stabilised. The user has five different sequences to choose from, and they can also be modified if it is needed. The sequences implemented are:

1 *BeepTest,* begins with an acceleration to a determined speed, keeps running with constant speed for a random time and then retards down to zero speed. It is also possible to get a sound signal at a specific time before the retardation starts.
2 *Ramp random,* accelerates up to a given center speed and then generates accelerations and retardations around this speed. Between the changes there are constant speed for a random time.
3 *Sinus,* which changes the speed as a sinusoid wave around a given center speed.
4 *PRBSSinus,* is the same as above but with 0.7 radians shift in phase according to the PRBS sequence described below.
5 *PRBS,* random changes between acceleration and retardation according to the PRBS sequence described below.

PRBS is a pseudo random binary sequence generated as shown in figure 4.3.

We have chosen to use a length of ten, which gives a random sequence

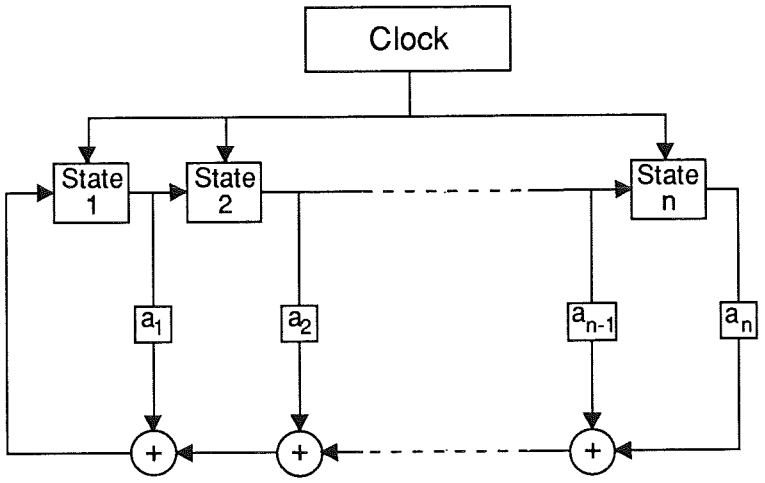


Figure 4.3 Block diagram of the PRBS-generator.

of 1023 clockpulses. The transfer algorithm is given by [16]

$$1 \oplus z^3 \oplus z^{10}$$

If the test needs to be interrupted before completion, the motor is stopped as quickly as possible. The motor cannot be stopped by just giving a zero output, this could cause the motor to burn since the load is quite big. Therefore we have set a maximum value of retardation which is used in these cases.

Estim

Estim implements a function that estimates a polynomial of the form

$$A(q)y = B(q)q^{NK}$$

The polynomial is adapted recursively according to

$$\hat{\theta}(t) = \hat{\theta}(t-1) + K(t)(y(t) - \varphi^T(t)\hat{\theta}(t-1))$$

$$K(t) = P(t-1)\varphi(t)(\lambda + \varphi^T(t)P(t-1)\varphi(t))^{-1}$$

$$P(t) = \frac{(1 - K(t)\varphi^T(t))P(t-1)}{\lambda}$$

For this calculation the measured moment from the measure process and the speed reference from the tachometer is used.

Another feature is that by minimising the loss function

$$V = \frac{1}{n} \sum_{i=1}^{n} |\varepsilon|$$

where

$$\varepsilon = y - \varphi(t)\hat{\theta}(t-1)$$

for a given number of samples, the delay between stimulation and response can be calculated. This function is optional and can be disabled, in that case the delay can be set manually by the user.

The calculated parameters and the loss function can be plotted by the plot process.

Plot

Plot is the process that displays the selected information on screen. The user can choose between three different kinds of plots.

1 **Measure**, shows collected data and the motor voltage.

2 **Estimator**, plots the loss function, if *preestim* is selected, and then plots the estimated parameters.

3 **TestSeq**, shows a selected test sequence without having the motor running.

The process collects message from *Measure* and *Estim* and plots whatever the user has selected to screen. As soon as message containing data from the right process arrives, plot brings up the appropriate window and starts the plot. The size and scale of the windows are adapted to the incoming data. If the data exceeds the upper and lower limits for a longer time, the window is rescaled to an optimum scale with regard to the current structure of the data.

Safe

Safe is a process that doesn't do anything else but checks for a mouseclick in the emergency button. If so it halts the current test immediately.

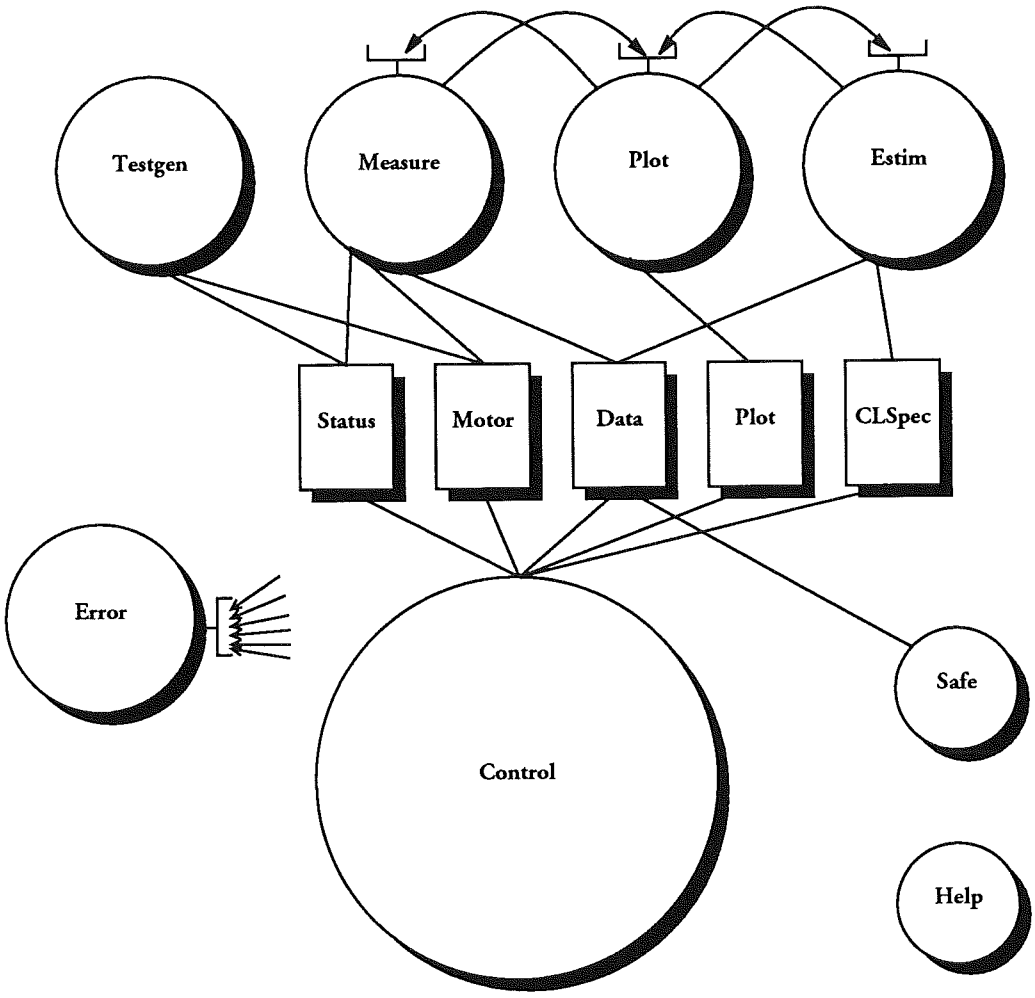


Figure 4.4 Process graph of the control program.

Error

Error receives information and error messages from all the other processes and presents this information in a small window on the screen.

Control

Control handles the communication between the user and the program. It communicates with the other processes through all of the monitors. The interface to the user is a number of buttons and hierarchical menus. These menus gives the user control of the stimulation equipment and its behaviour. The operating of the program is described in the VISCON user guide.

Help

Help is a separate process to implement a help function. When the help button is "pressed", a window with information on the current menu appears.

4.2.2 Monitors

Status monitor

Status monitor contains variables and procedures for managing the behaviour of the system. It gives permission to start a test, holds the name of the file where data is to be saved and handles other general tasks.

Motor monitor

Motor monitor takes care of everything that concerns the motor, e.g. it holds the value of the desired speed and the maximum allowed speed.

Plot monitor

Plot monitor administrates the plotting and keeps track of the selected kind of plot.

CLSpec monitor

CLSpec monitor is the monitor that handles everything concerning the estimator, it contains the parameters and the status of the estimation.

Data monitor

Data monitor handles the flow of measured data between *Measure* and *Estim.* It also holds some of the system parameters that concerns the measuring, such as **delay** and **file const.**

5 Material & experimental procedure

The group of test subjects consisted of 10 healthy subjects, 3 women and 7 men, of the age 20 - 32 years. They were instructed to stand relaxed with feet comfortably separated and with their arms crossed over the chest. To minimise the variation of visual and vestibular input signals, they were told to look straight ahead at the junction of the mirror and the striped belt. They also wore a baseball cap to block off the upper part of the visual field. To avoid sounds from the motor and the belt giving orientational clues, all subjects were equipped with earphones and a walkman playing monotone music to conceal these sounds.

The tests consisted of four different test sequenses with varying stimulation pattern, each lasting for 300 seconds. Between the sequences the subjects stepped down to rest for about two minutes.

The different stimulation sequences were:

1 Ramps of random length and random direction around a defined center speed. This test was used as an introduction to show the test subjects how the tests were performed, and thereby avoiding future surprise effects. The test was not used in the identification.
2 Sinusoid motion. The stimulation consists of a pure sinusoid sequence with a period time of 30 seconds and a selected amplitude (43.6 °/ s). The test was used to see a possible adaptation to a regular motion pattern. In the identification however, no more than two parameters could be estimated due to lack of excitation in the frequency range.
3 PRBS-motion. This motion pattern consists of random changes in direction with a limited amplitude (12.8 °/s) around zero. The PRBS-motion must however be a little deformed by making the changes softer since an instant switch in direction would cause too much strain on the equipment. This sequence is ideal in terms of identification since white noise excitation gives a satisfactory image of the transfer function at all frequencies. It also provides information for a parametric estimation.
4 PRBS-sinus. This motion pattern consists of an underlying sinusoidal signal with a specific speed amplitude (20.5 °/s). The sinusoidal signal is PRBS-modulated with random phase shifts of 0,7 rad. This sequence

also gives information for an estimation. It is also interesting to see a possible adaptation to the underlying sinusoid.

When the tests were completed the subjects were interviewed about their subjective experiences of the tests. They were asked if, and in that case how, they felt affected by the stimuli.

6 Methods of identification

In the identification and verification we have chosen to use both parametric and non parametric methods. We have used the program MatLab to adapt parametric estimations of both ARMAX- and ARX-model. The ARX-model has got a poorer noise model, which might give higher ideal model degree than the corresponding ARMAX-model

The following calculations were made.

6.1 Parametric identification

1 Maximum Likelihood identification of ARMAX- and ARX- function.

2 Validation by simulating the identified model.

3 Residual analysis of measured response and model response.

4 Autocorrelation of the residual (e).

5 Crosscorrelation between the residual (e) and the input signal (u).

6 Step response and impulse response from the identified ARMAX-model.

As there was a considerable variation in time delays, the two latter functions have been implemented in MatLab to find the best parameters. For the ARMAX-model a model degree of type [3 2 2] (degree[A B C] of ARMAX-model) showed to be the most suitable. Therefore an estimation of time delays based on this model degree were made. The function of the ARX-function was to search for other possible models with a parameter degree up to a sixth order system, with the aid of the MatLab - function SelStruc (see appendix I). The best model-order was chosen considering Akaikes weighting constant.

6.2 Nonparametric identification

1 Autospectrum of the input signal (u) and the output signal (T-bal) in form of measured ankle torque.

2 The transfer function between u and T-bal.

3 Coherence between u and T-bal.

4 A Bode plot made by the MatLab function Spa (see appendix V).

Coherence is the relationship between the input and the output at each input frequency. If coherence is equal to 1, then all of the output signals are dependent on the input signals. If coherence is much less than 1, then there are signals that are not being measured, or there is noise in the system, or there is a nonlinear relationship between input and output [8].

As a tool we have implemented an estimation function in our program for the stimulation equipment. This function has two parts. First it calculates the time delay by minimising the loss function of the chosen model order. Secondly it estimates the parameters with a recursive least square method in real time during the test [16],[20].

All test sequences start with 30 seconds stabilization time without stimulation. After that the visual stimulation starts. The identification has been performed at intervals, 50 - 100, 100 -150, 150 - 200 seconds after the beginning of the test, and at a time interval around the times 180 - 250 seconds when motion is stable. A stable motion means that there are no violent motion changes, which could be depending on fatigue phenomena or external disturbances. At the identification we have particulaly studied the results of four test subjects to see possible changes during the duration of the test. Identification on the results of the other persons have been made at the time intervals 50 - 100 and 180 - 250 seconds in a stable state.

7 Results

This section contains a number of plots and results from the four tests specially studied.

7.1 Sinus-stimulation

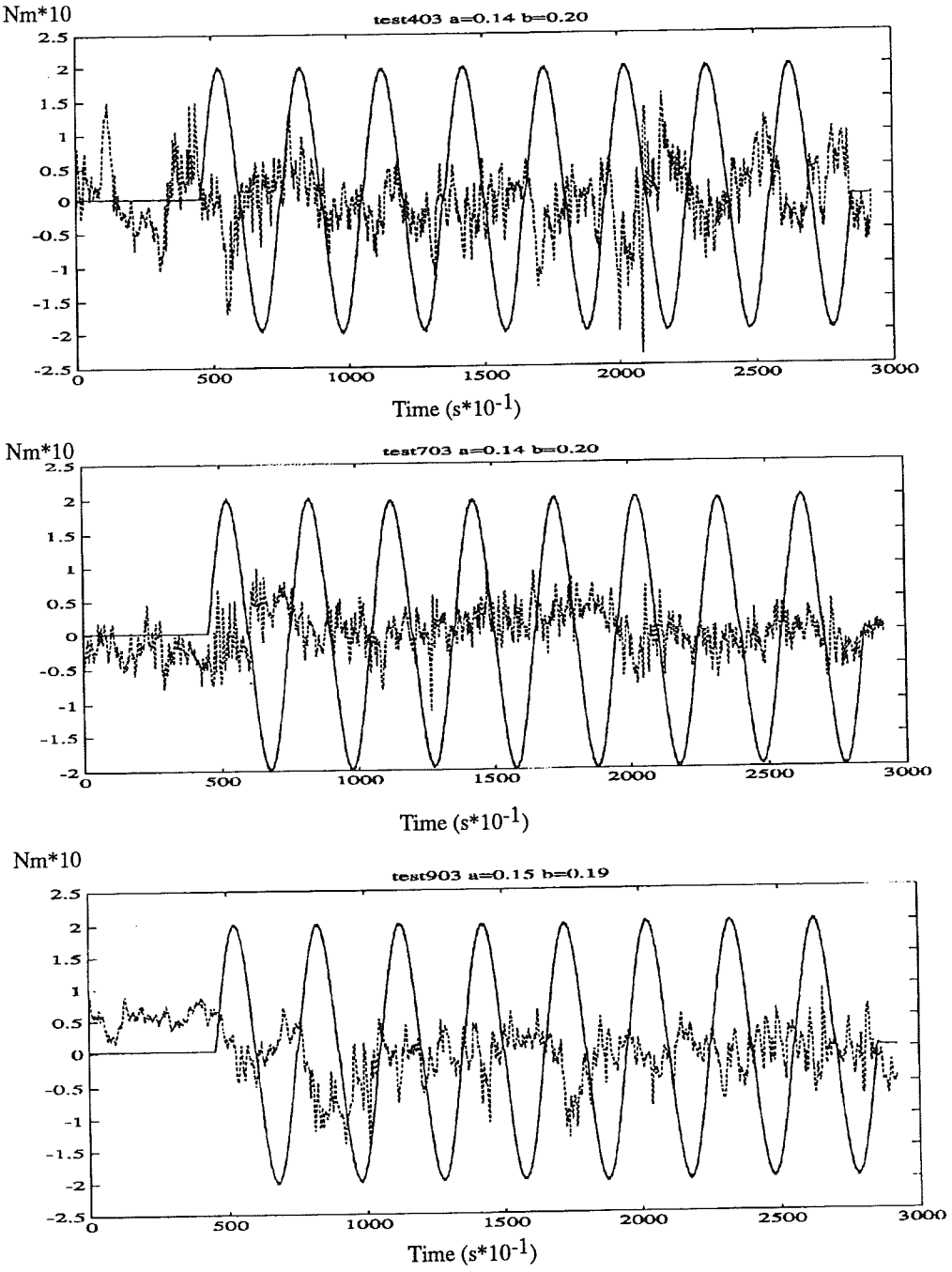


Figure 7.1 Measured results from tests with purely sinusoidal visual stimuli.

Figure 7.1 shows the stimulus velocity, and the body response in terms of increased moment. The scale on the Y-axis refers to the response. The moment response is most powerful initially, and decreases rather quickly. The moment response also shows a tendency to sway in counterphase with the stimulus.

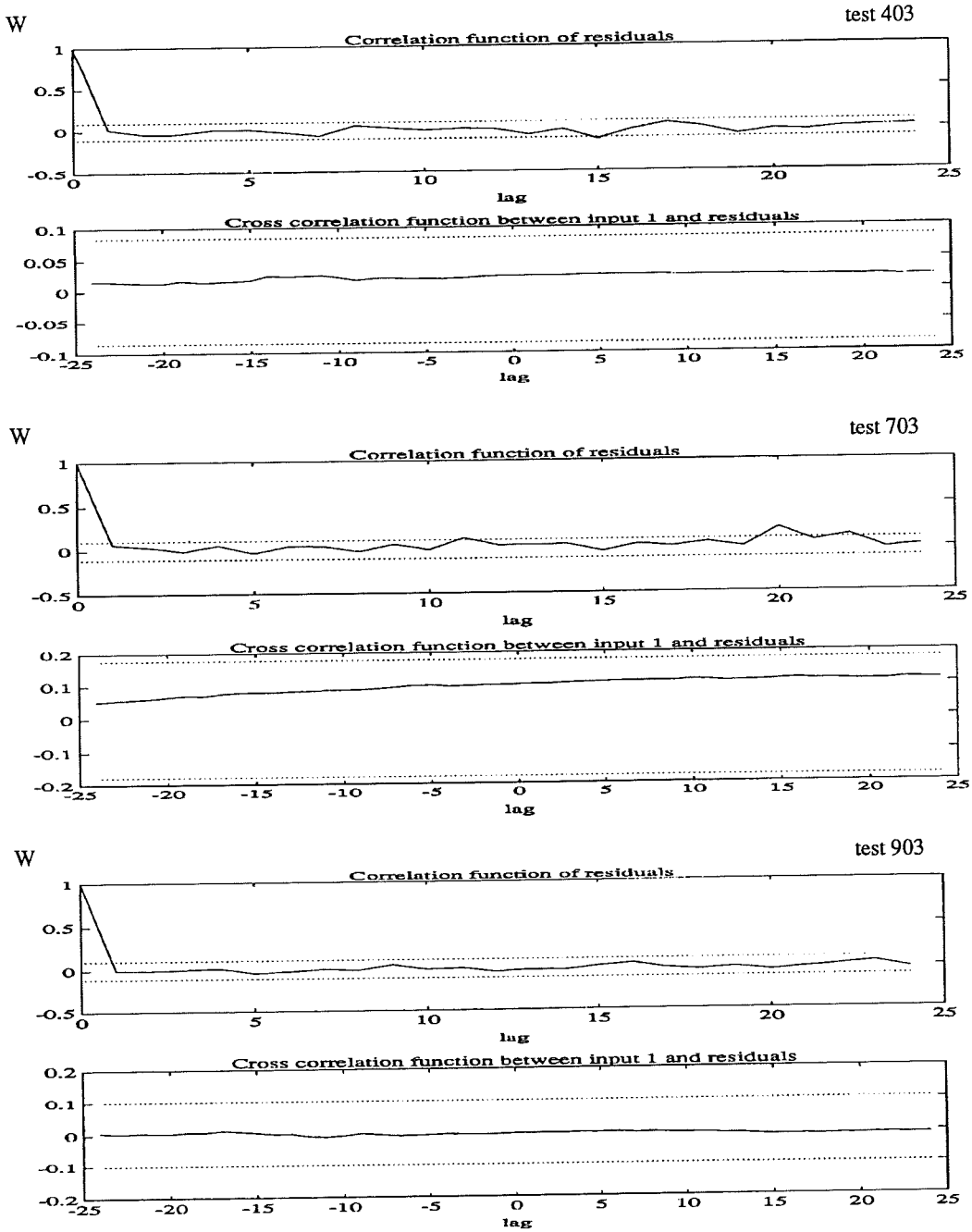


Figure 7.2 Correlation between input and output. Sinus-stimulation.

The next plot, figure 7.2, shows the correlation between input signal and output signal, i.e. the connection between input and output. The result shows that both correlation of the residual and correlation between the residual and the input signal are small and within the 95% confidence limits. This means that the input-output relation can be well explained by the model.

The estimated time delays at different time intervals are shown in table 7.1, first the delays between input stimulus and response of the four test subjects specially studied and the rest of the tests. Table 7.2 shows the polynomials estimated with the ARMAX-model.

Test \interval (s)	50:100	100:150	150:200	180:250
403	0	0.3	0.3	3.1
703	0	3.8	4.9	0.4
903	0	4.9	0.6	0
1203	0.3	0	0.3	1.6

\interval (s)	50:100	180:250
503	0.8	1.6
603	1.3	0.4
803	3.2	0.2
1003	0.3	1.5
1103	0	5.7
1303	1.4	4.2

Table 7.1 Estimated time delays. Sinus-stimulation.

Test	\interval	50:100				150:200			
403	A	1.0000	0.9079	0.8292	0.0550	1.0000	1.6454	0.7191	0.0352
	B	0.0442	0.0872	0.0710		0	-0.0220	-0.0187	
703	A	1.0000	1.3209	1.2798	0.1285	1.0000	0.4896	0.1316	0
	B	0.0245	0.0540	0.0465	-0.0012	0	0.0001	-0.0002	
903	A	1.0000	4.1788	4.2190	0.0997	1.0000	2.2574	1.1277	0.0717
	B	0.0104	0.0472	0.0506		0	0.0095	0.0268	-0.0013
1203	A	1.0000	1.0000	1.1806	0.0201	1.0000	2.5383	0.0599	0
	B	0	-0.0323	-0.0707	-0.0016	0	0.0415	-0.0041	

Table 7.2 Estimated polynomials. ARMAX-model, sinus-stimulation.

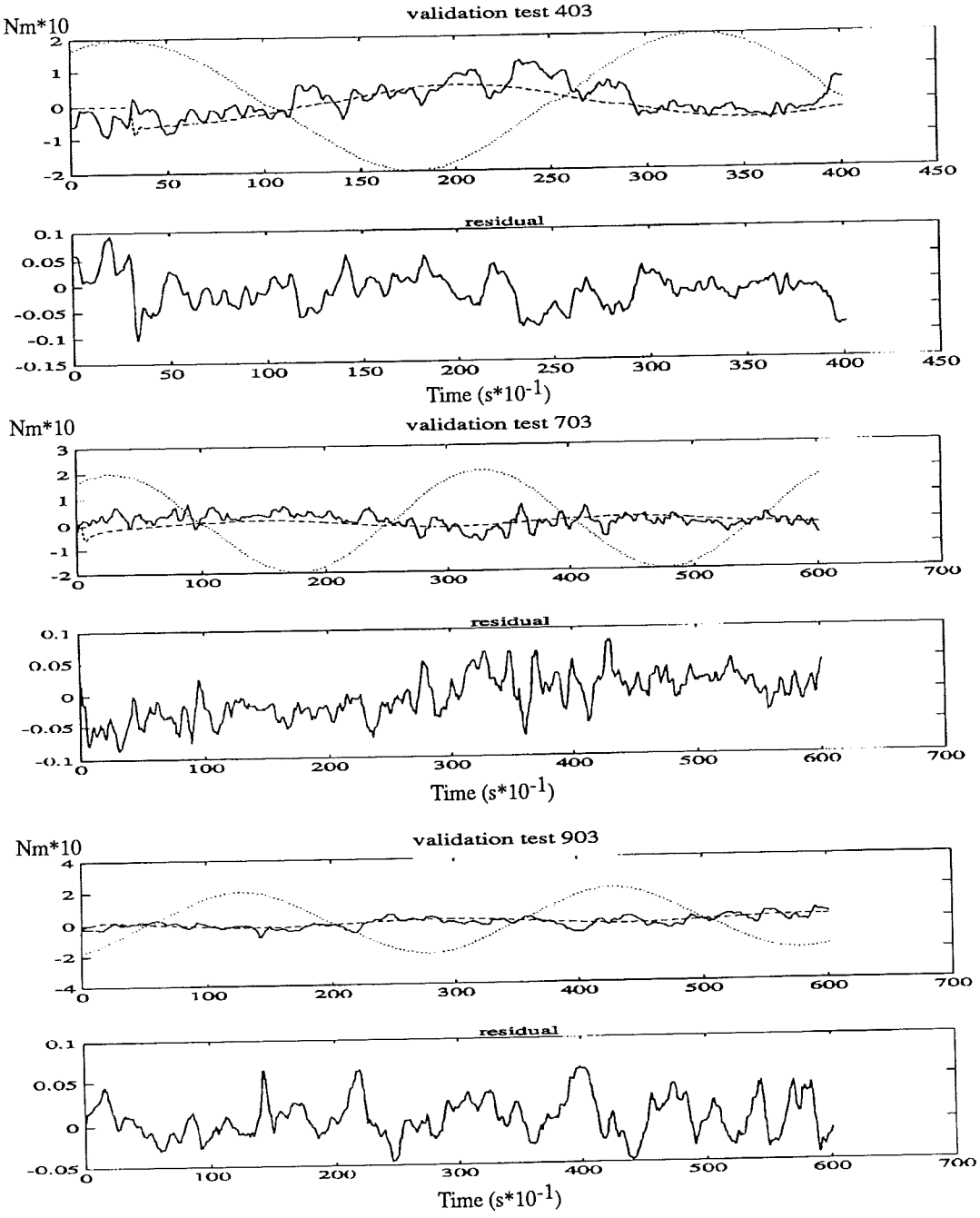


Figure 7.3 Validation of ARMAX-model. Sinus-stimulation.

The curves in the validation plot in figure 7.3 are:

.....	stimulus (velocity of the screen)
- - - - - - - - - - - - - - - -	response from the estimated model (ARMAX)
————————	measured ankle torque

The estimated model follows the measured values relatively well. However, some test subjects had very small responses, especially towards the end of the test, which again indicates the presence of adaptation.
After transferring the discrete ARMAX-polynomials to a continuous form, the pole - zero locations have been plotted, see Figure 7.4.

The results from the nonparametric estimation, coherence and Bode-diagrams, have been plotted in figure 7.5.

The identification results of all test subjects at the intervals 100 - 150 and 180 - 250, have been plotted in a pole - zero plot, see figure 7.6.

For the corresponding ARX-identification, we obtain the model orders and timedelays shown in table 7.3.

Test \ interval	50:100	100:150	150:200	180:250
403	4 3 1	4 3 1	3 2 0	4 3 32
703	4 5 60	4 3 31	4 3 23	3 2 4
903	3 2 0	3 2 48	3 4 5	3 2 2
1203	4 3 2	3 2 2	4 3 62	4 3 15

Table 7.3 Model order and time delay, ARX-model. Sinus-stimulatinon.
The figures read, degree A degree B Time delay (measured in number of samples).

The same set of results for PRBS-motion and PRBS-sinus as stimuli are found in figures 7.7 - 7.12 and tables 7.4 -7.6, and figures 7.13 - 7.18 and tables 7.7 - 7.9, respectively.

The pole - zero plots have special marks for the different times, these are:

	pole	zero
50:100 s	+	○
100:150 s	×	♡
150:200 s	✳	Δ
200:250 s	✙	⬠

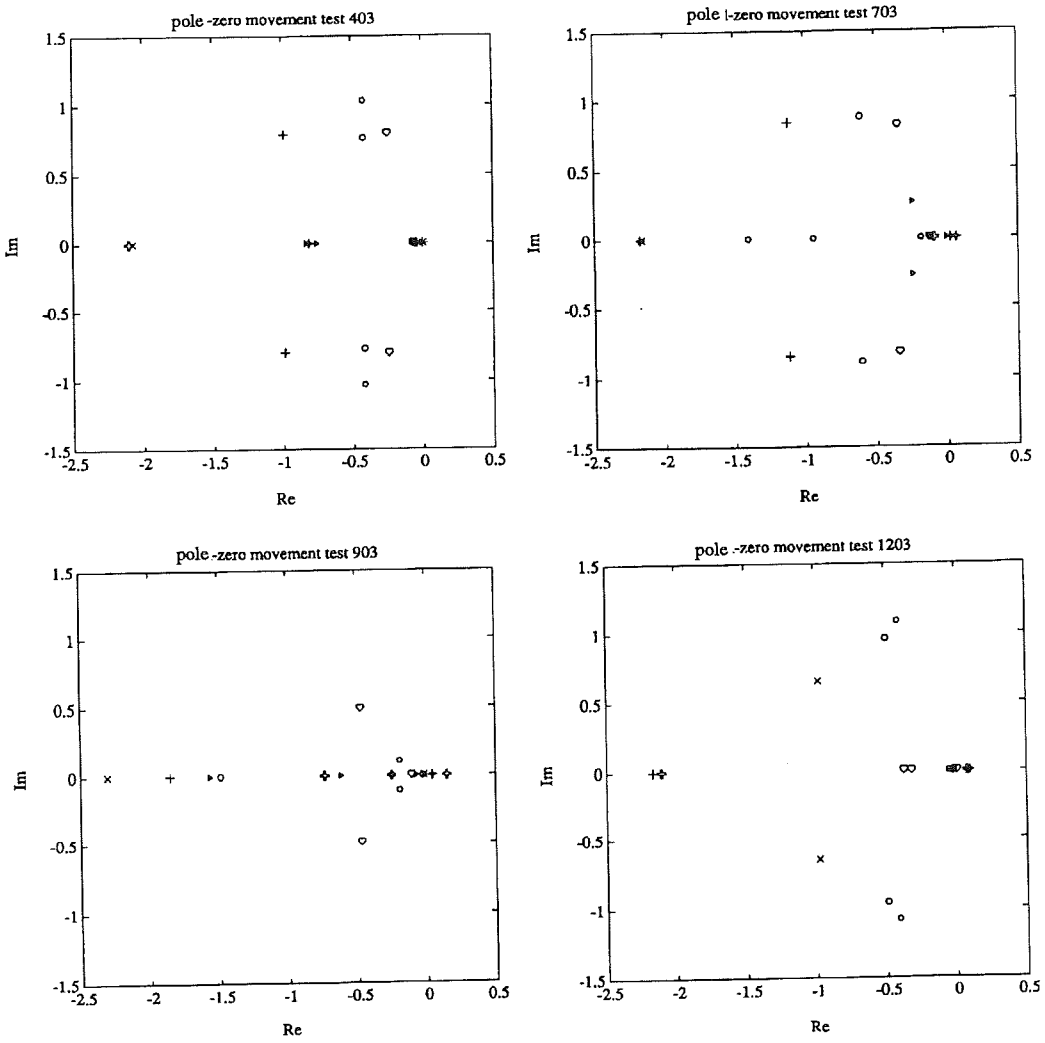


Figure 7.4 Pole - zero placement, ARMAX-model, sinus-stimulation.

The pole-zero plots shows similarities between the test subjects. A specific pole, placed as an integrator, was found in all the estimated models independent of which interval the estimation was made in. Furthermore one can find a large variation in the placement of the other poles and zeroes in the course of the duration of the test. It is important to observe the fact that a pure sinusoidal stimulus gives insufficient data for a reliable parametric model. However, a better frequency excitation than expected was found in a frequency response analysis (see appendix VII).

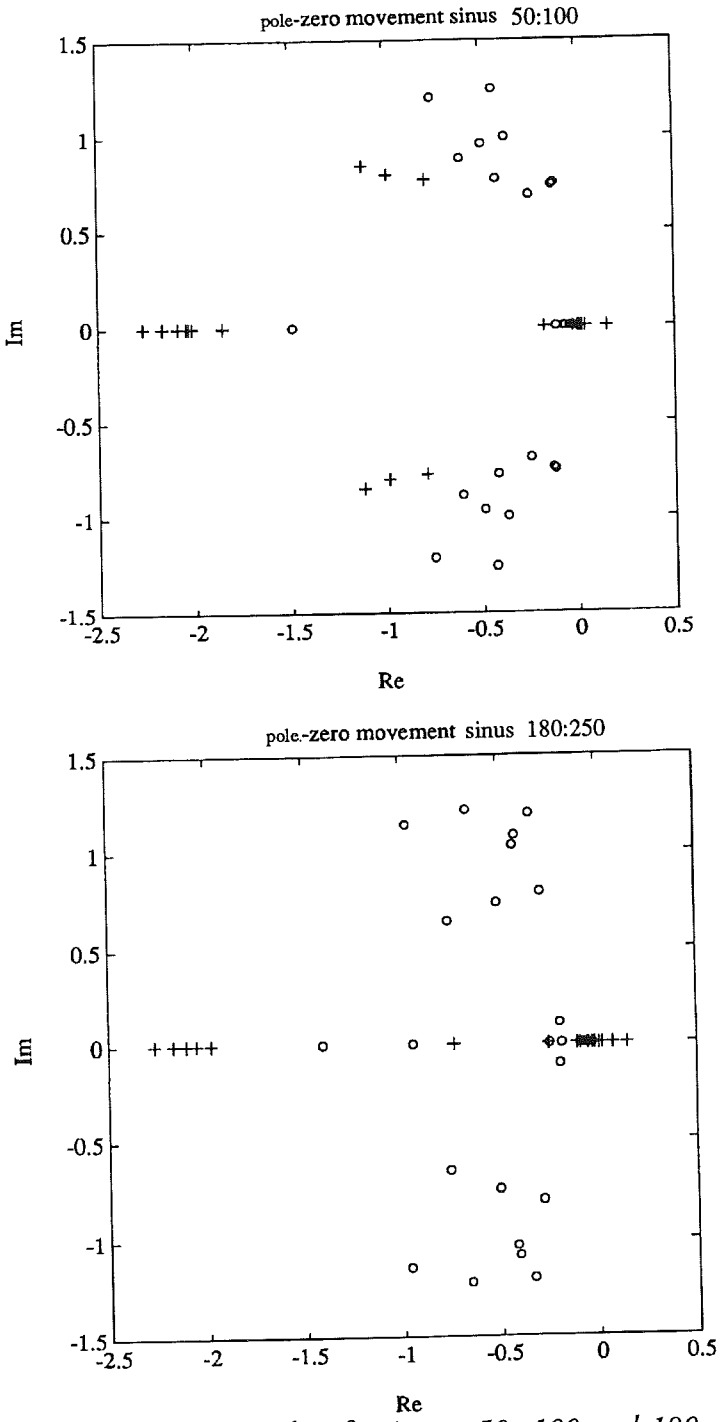


Figure 7.5 Pole - zero placement. Identification at 50 -100 and 180 - 250 s, Sinus-stimulation, all test subjects.

In the composed pole-zero plot, one can find a correspondence between the test subjects, especially the presence of an integrator, but also in the placement of zeroes. It is also possible to find models where the poles in the right halfplane make the system unstable.

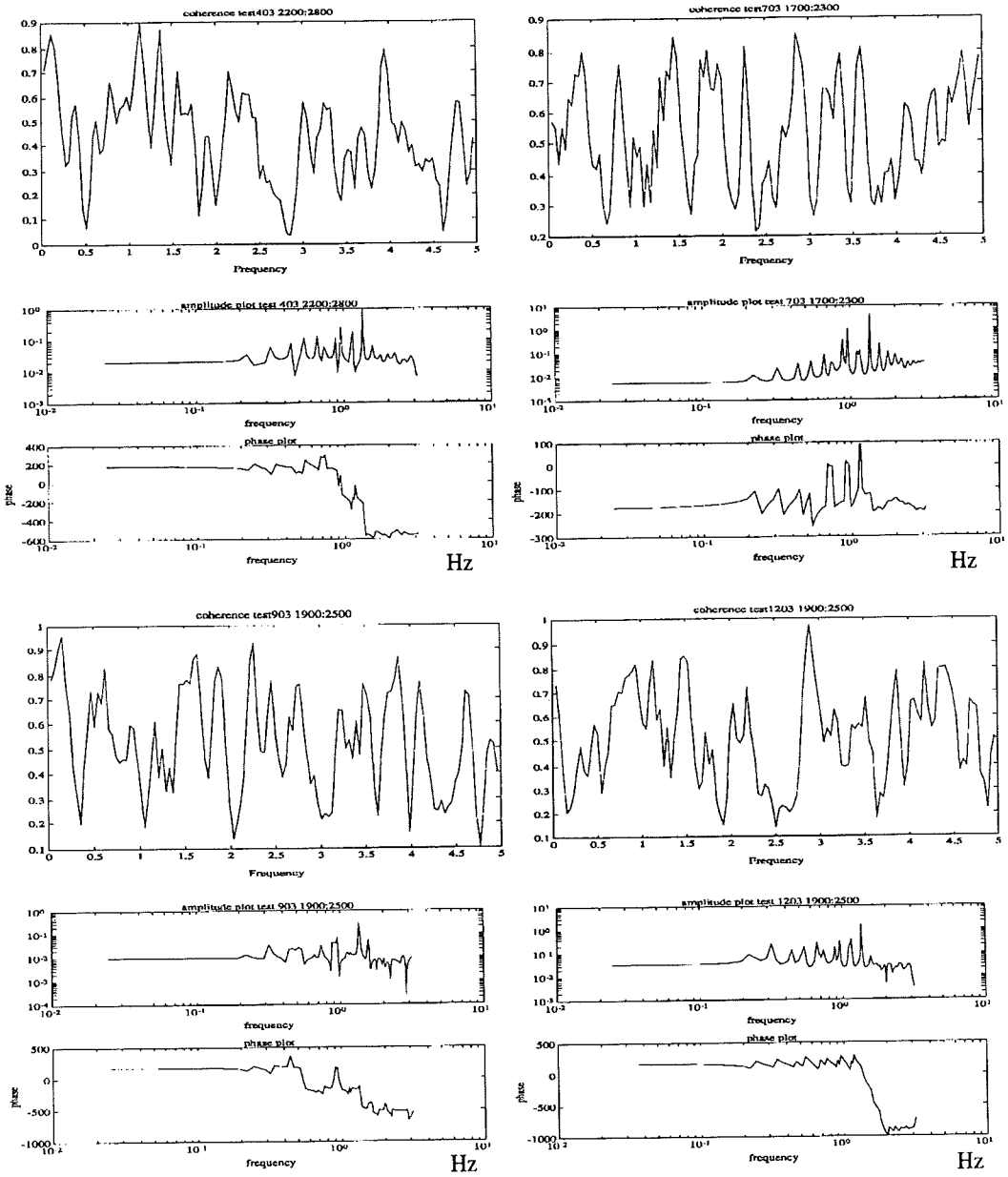


Figure 7.6 Results from the nonparametric identification for. sinus-stimulation. The frequencies are measured in Hz

Unfortunately the coherence is irregular, but there is often a range of low frequencies where it is relatively high. Thus, the Bode- diagram can be wieved as reliable within this range. The Bode-diagrams exhibits stable gain in the low frequency range, with 180° phase shift.

7.2 PRBS-stimulus

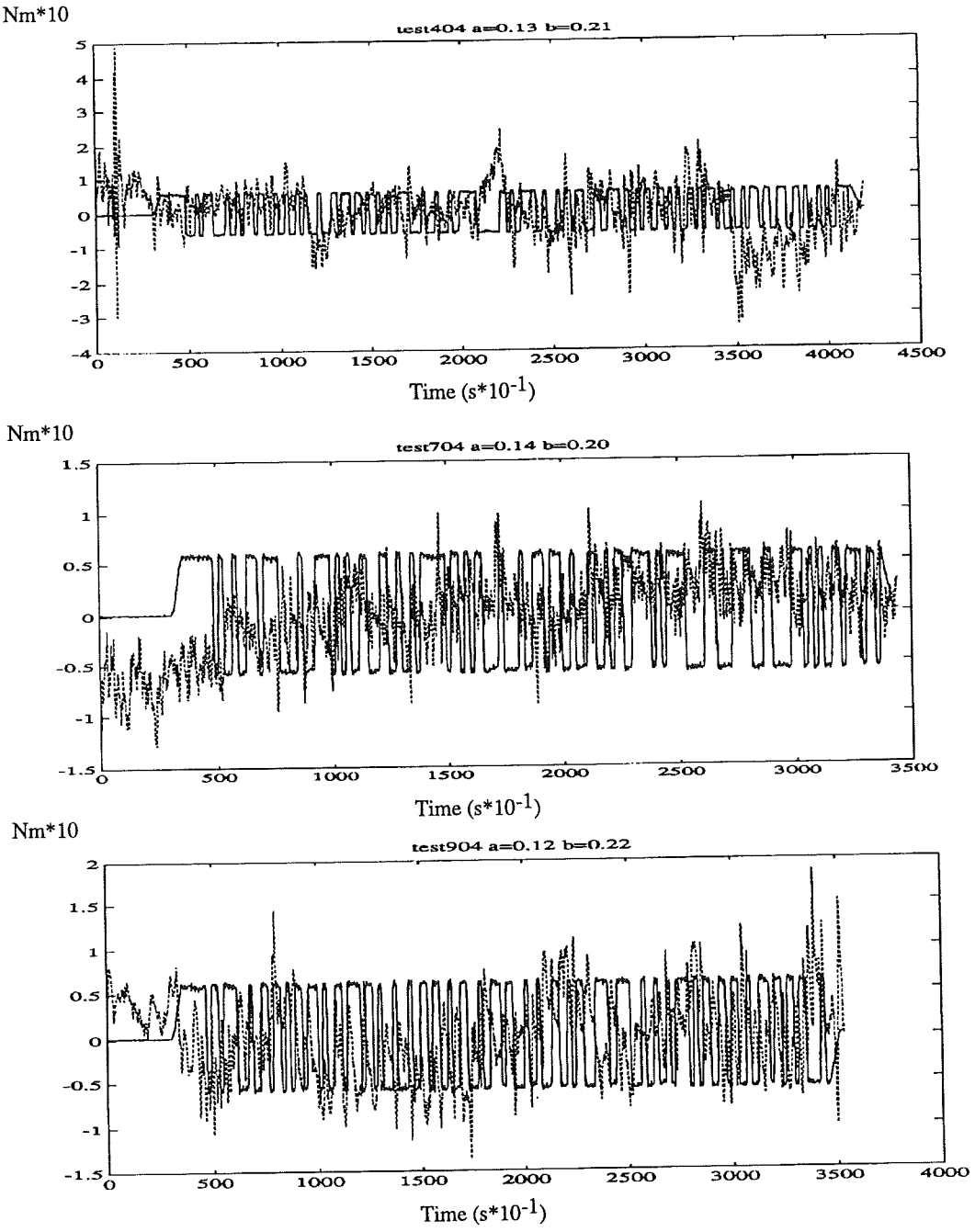


Figure 7.7 Measured results from tests with PRBS-motion as visual stimuli.

The PRBS-stimulation induces large responses with most of the subjects. There are also more pronounced peaks, compared to the other forms of stimuli.

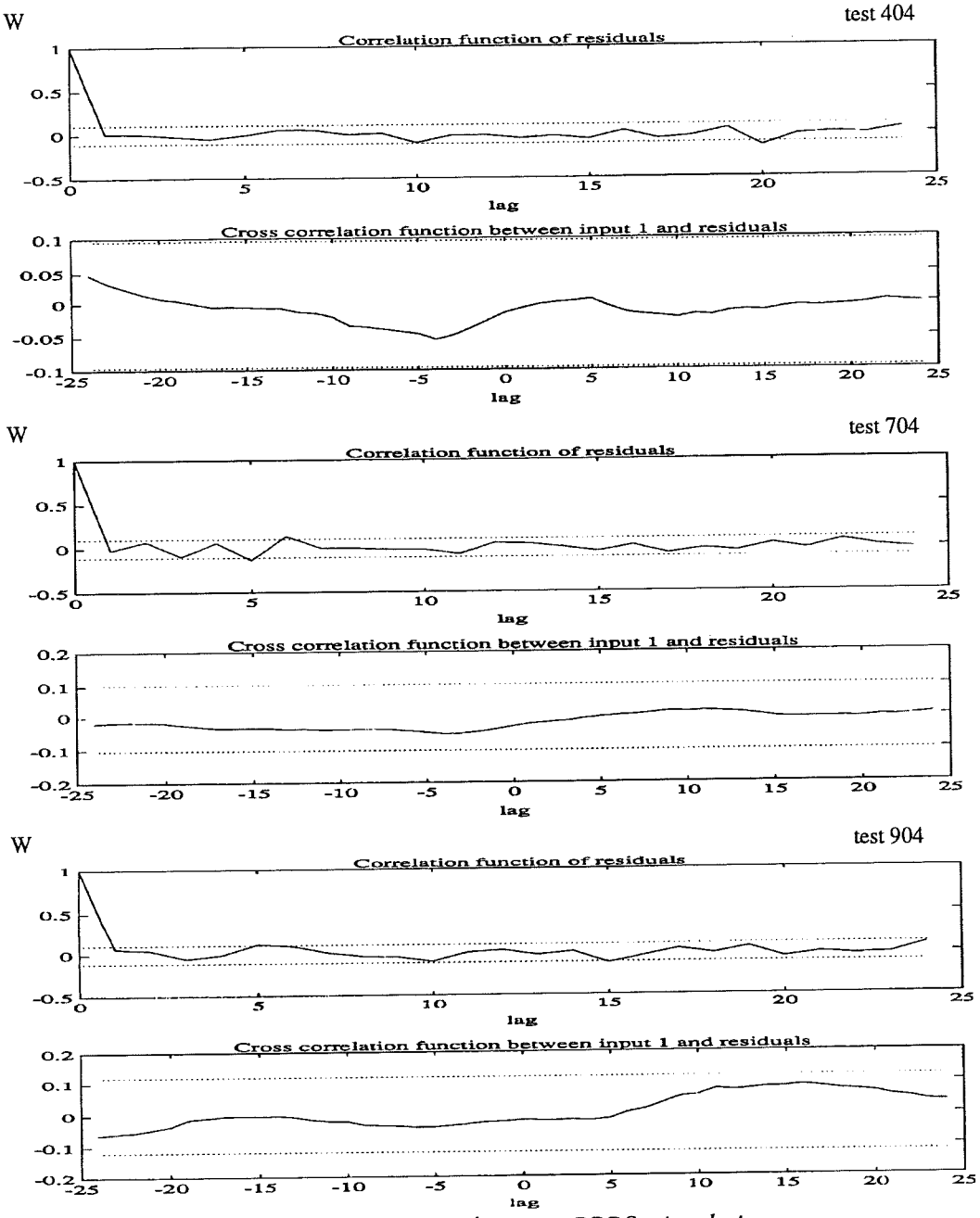


Figure 7.8 Correlation between input and output. PRBS-stimulation.

This test also shows correlation values well inside the 95% confidence limits. Although the level is higher in the correlation between residual and the input signal as compared to the sinusoidal stimulation.

Test \interval	50:100	100:150	150:200	180:250
404	2.9	0.1	1.0	0.3
704	1.0	2.2	0.6	1.4
904	0.8	1.0	0.7	0.5
1204	1.2	2.5	1.9	1.0

\interval	50:100	180:250
504	3.0	0.1
604	3.2	1.2
804		
1004	2.5	0.9
1104	2.8	1.0
1304	1.0	1.9

Table 7.4 Estimated time delays. PRBS-stimulation.

Test	\interval	50:100				150:200			
404	A	1.0000	0.3582	0.0832	-0.0001	1.0000	1.2843	0.9532	0.0860
	B	0.0002	0.0004	0		-0.0123	-0.0293	0	
704	A	1.0000	1.7825	4.1009	0.3056	1.0000	0.6424	0.7228	0.0732
	B	0	-0.0391	-0.0796	0.0085	0	-0.0020	-0.0047	-0.0011
904	A	1.0000	1.0423	0.2889	0.0184	1.0000	1.0792	0.8417	0.0459
	B	0	-0.0157	-0.0083		0	-0.0125	-0.0293	-0.0025
1204	A	1.0000	0.6733	0.8315	0.0331	1.0000	0.6815	1.1672	0.0516
	B	0	-0.0073	-0.0156		0	-0.0127	-0.0275	-0.0045

Table 7.5 Estimated polynomials. ARMAX-model, PRBS-stimulation.

Test \ interval	50:100	100:150	150:200	230:270
404	3 3 10	4 3 2	4 6 10	4 3 4
704	4 3 9	4 3 24	4 4 8	4 3 8
904	4 5 8	3 5 8	4 6 7	4 6 7
1204	4 3 10	4 4 23	3 2 20	4 3 10

Table 7.6 Model order and time delay, ARX-model. PRBS-stimulatinon.
The figures read, degree A degree B Time delay (measured in number of samples).

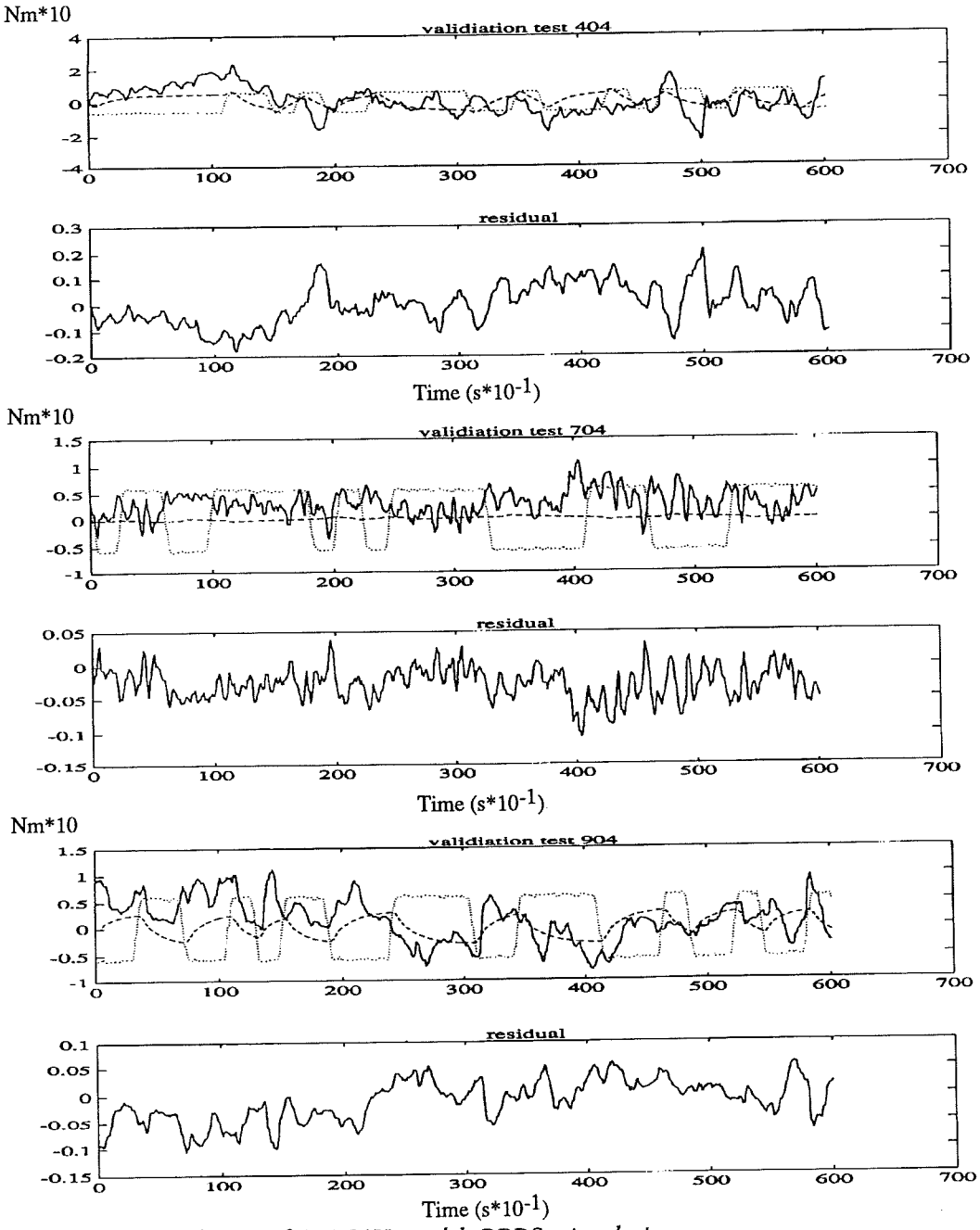


Figure 7.9 Validation of ARMAX-model. PRBS-stimulation.

The validation shows that the estimated model is not sufficient to explain the entire postural control system. The residual values are therefore high.

The curves in the validation plot in figure 7.9 are:

..... stimulus (velocity of the screen)

---------- response from the estimated model (ARMAX)

———— measured ankle torque

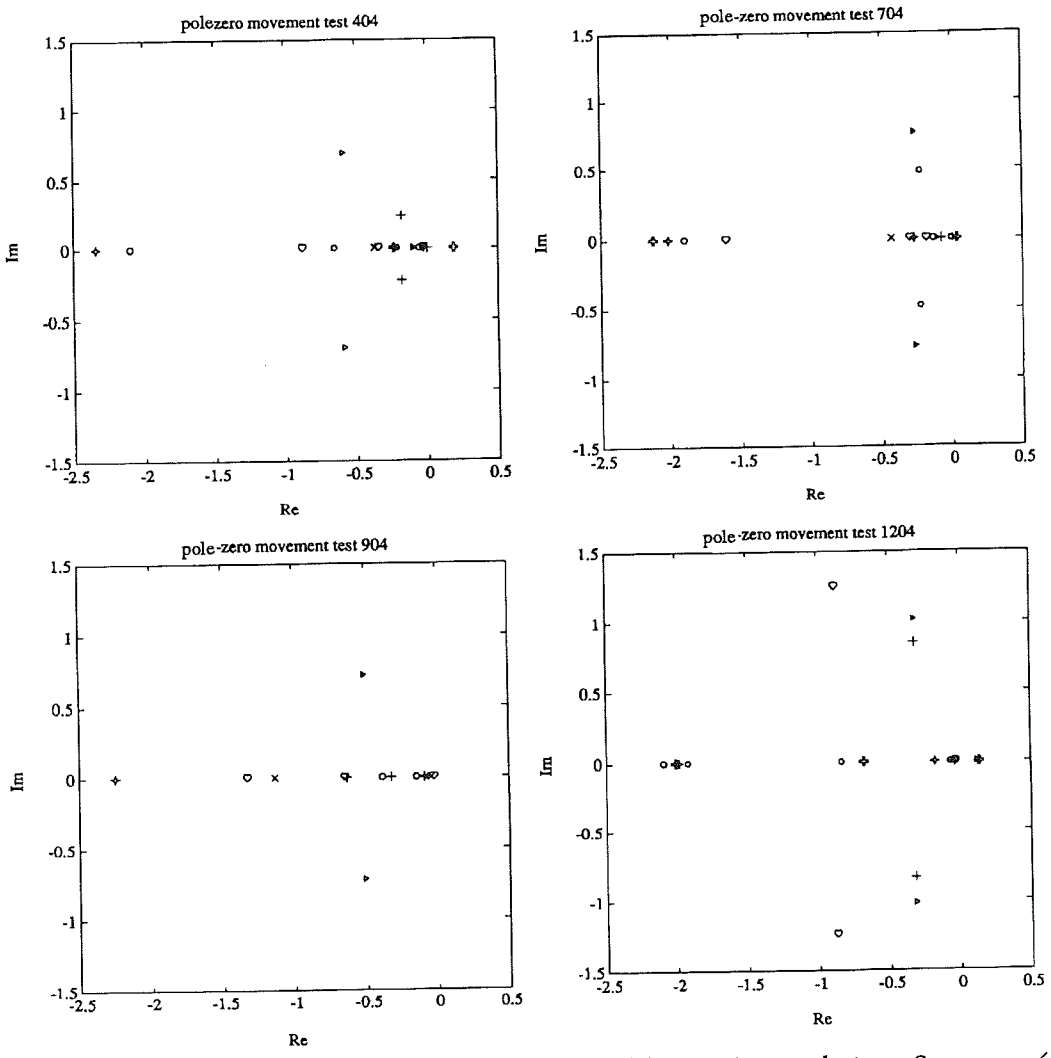


Figure 7.10 Pole - zero placement, ARMAX-model, PRBS-stimulation. See page 40 about the marks.

The diagrams show big variations of pole and zero placements. From an initial location on the real axis, the zeroes move to an imaginary location in the end of the test. The poles behave in a direct opposite way, starting imaginary and ending on the real axis. This movement leads to a more stable form of control towards the end of the test.

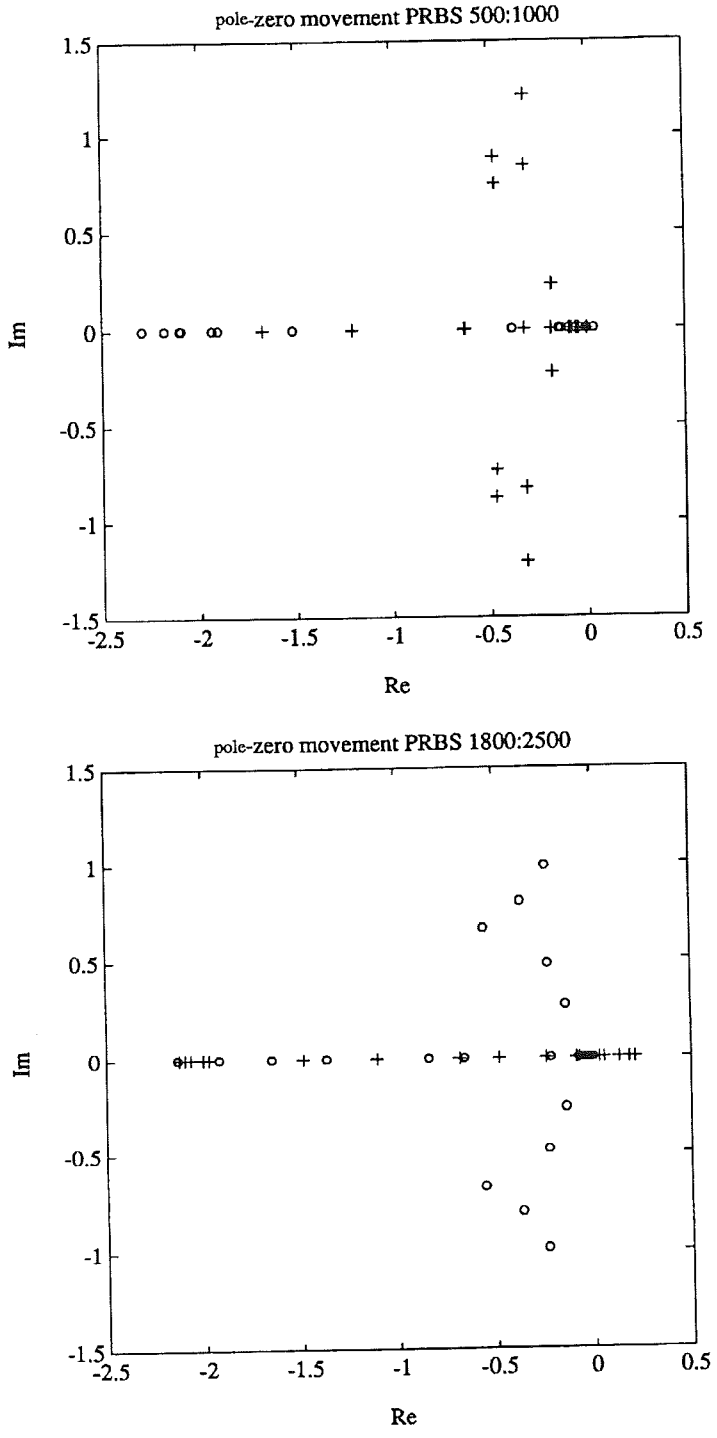


Figure 7.11 Pole - zero placement. Identification at 100 -150 and 180 - 250 s, PRBS-stimulation, all testsubjects.

In the composed pole-zero plot, one can find a correspondance between the testsubjects, especially the presence of an integrator, but also in the placement and changes of the poles and zeroes.

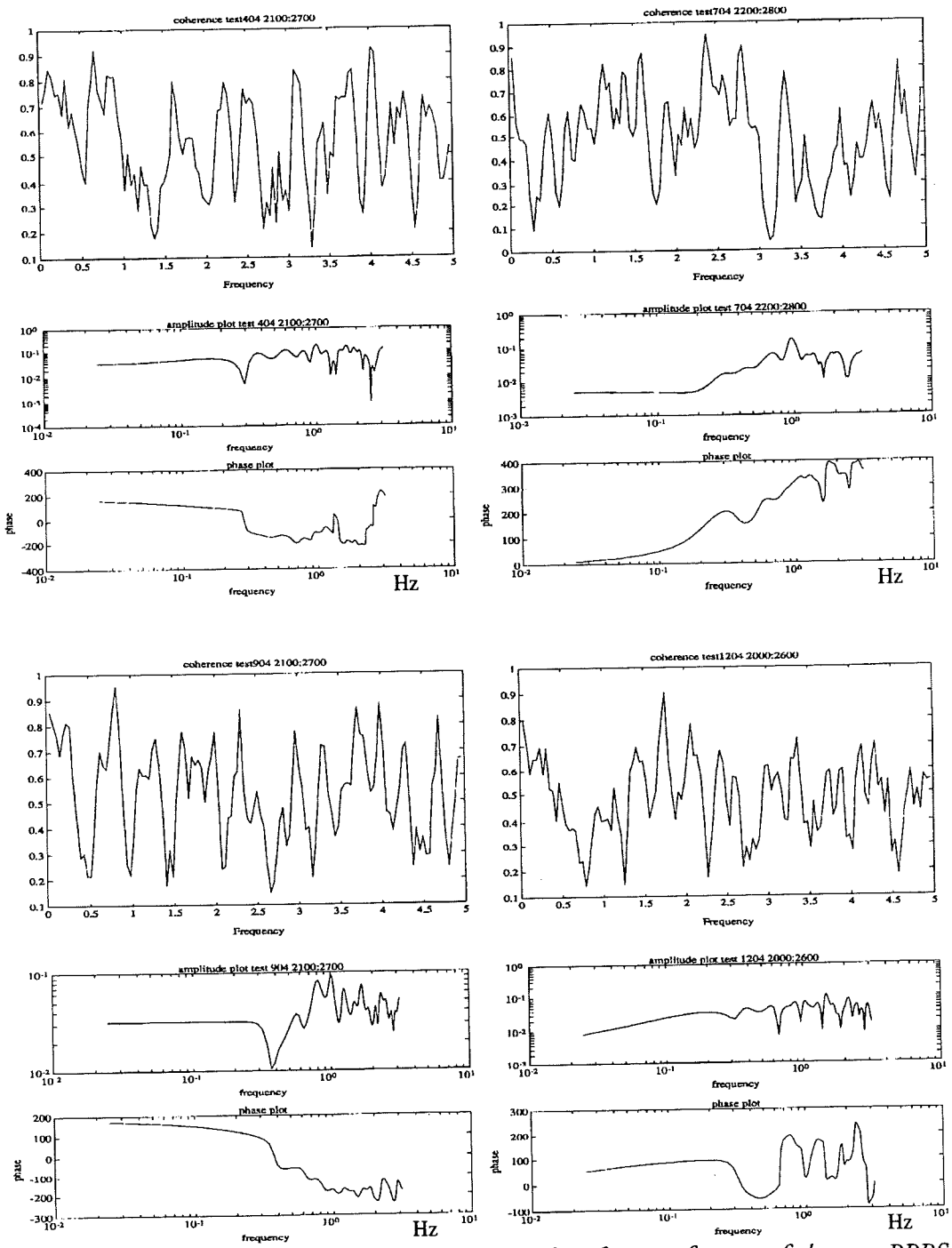


Figure 7.12 Results from the nonparametric identification for one of the tests. PRBS-stimulation.

The values of coherence vary, but they are often high in the low frequency range. The Bode-diagram shows a proportionality between stimulus and response, but the phase shift varies a lot between different subjects.

7.3 PRBSsinus-stimulation

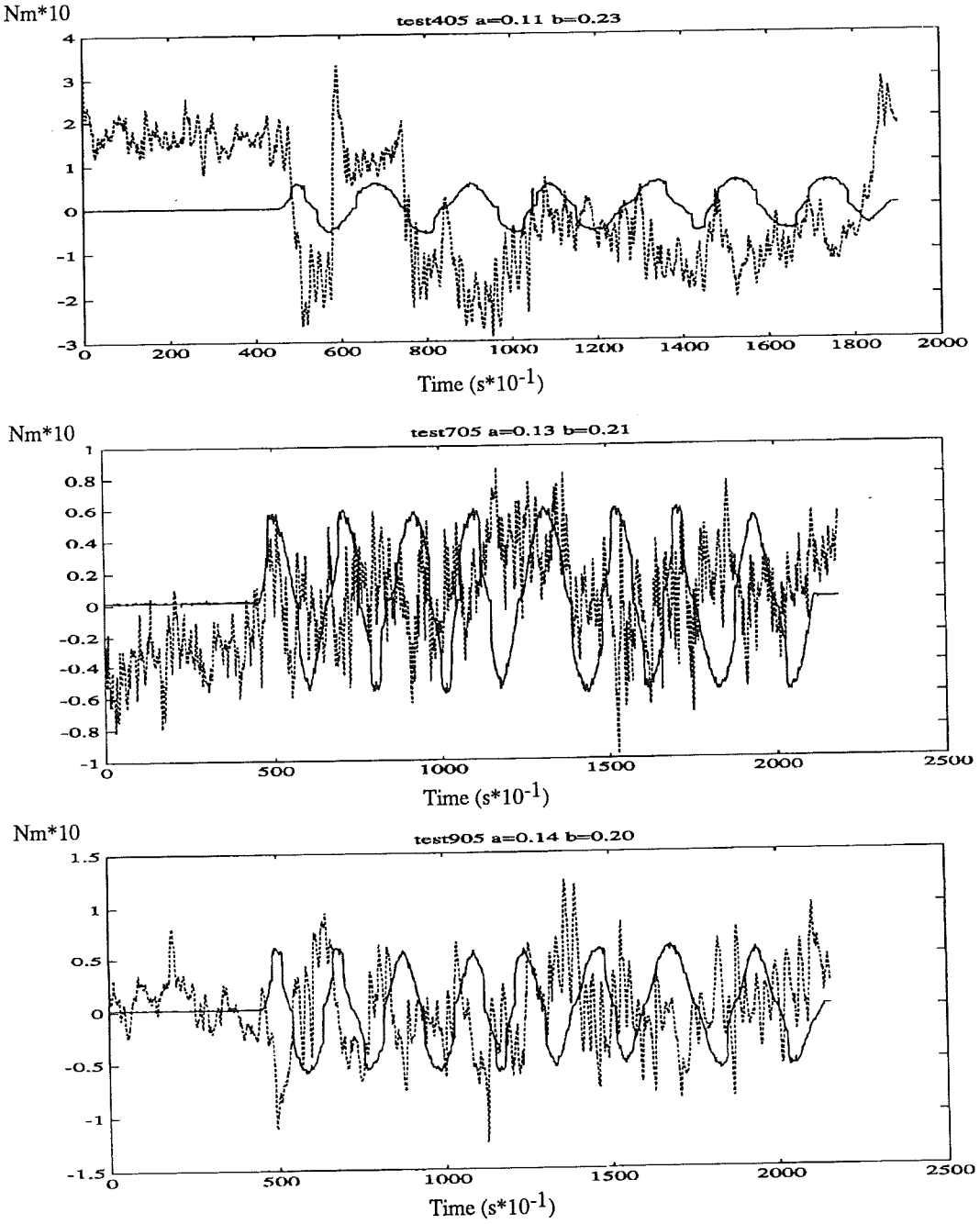


Figure 7.13 Measured results from tests with PRBSsinus as visual stimuli.

PRBSsinus induces large responses in the beginning of the test, but they decrease towards the end of the test.

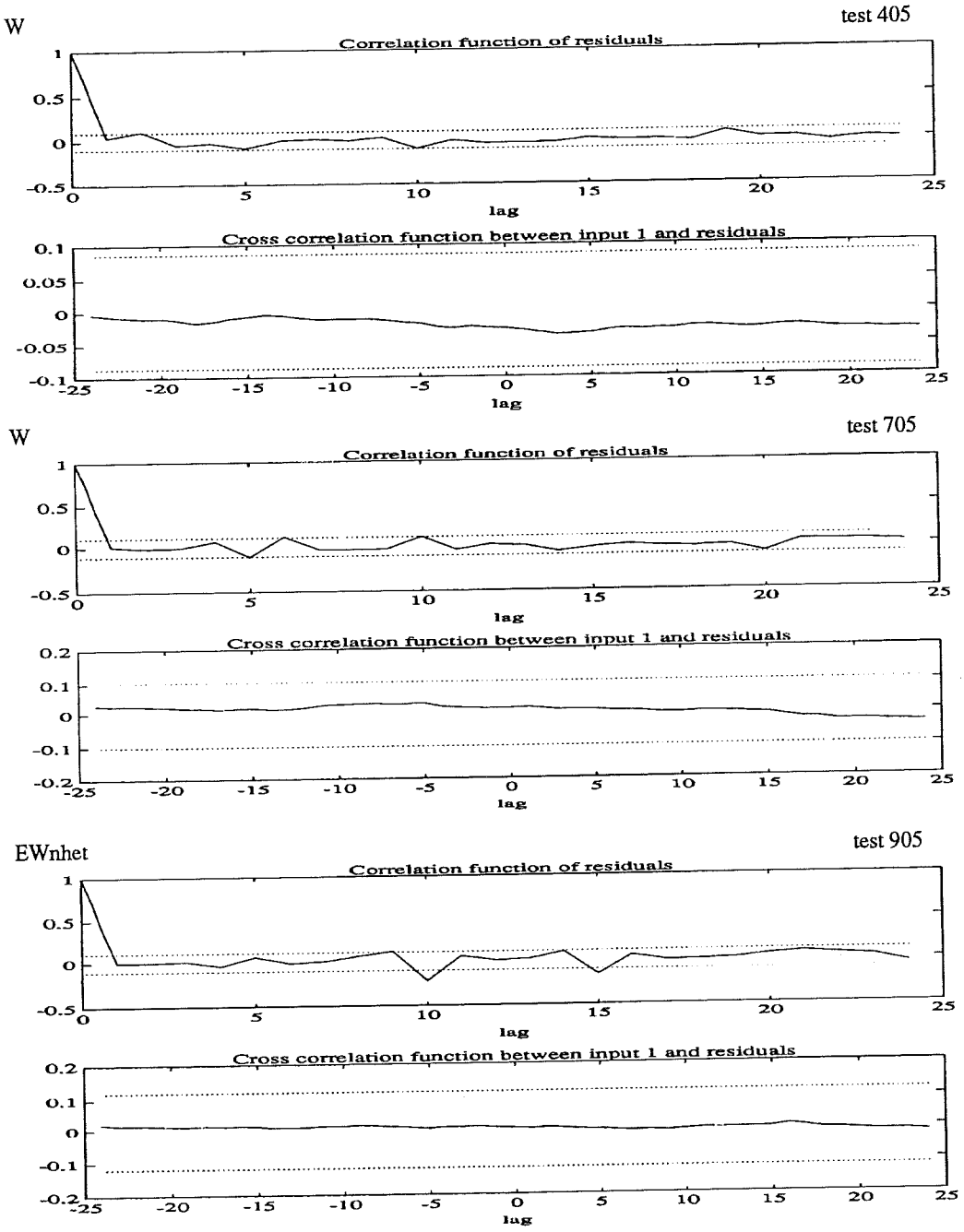


Figure 7.14 Correlation between input and output. PRBSsinus-stimulation.

In this test the correlation values are still small and mostly inside the 95% confidence limits. But the residual correlation values are close to the limits. Still the correlation between residual and input signal are well inside the limits.

Test \interval	50:100	100:140	120:190
405	3.3	1.3	2.8
705	0.2	6.7	1.2
905	0.9	0.7	0
1205	0.3	0.3	0.4

\interval	50:100	140:180
505	0	0
605	1.7	0.5
805	0	7.7
1005	1.1	2.5
1105	4.3	0.7
1305	6.7	1.3

Table 7.7 Estimated time delays. PRBSsinus-stimulation.

Test	\interval	50:100				120:190			
405	A	1.0000	0.2263	0.15551	0.0019	1.0000	0.3754	0.1455	0.0014
	B	0	-0.0067	-0.0139		0	-0.0048	-0.0103	
705	A	1.0000	0.6333	0.7167	0.0773	1.0000	0.5399	0.7732	0.0616
	B	0	-0.0047	-0.0102		0	-0.0171	-0.0353	
905	A	1.0000	2.2784	0.9692	0.0683	1.0000	2.0822	0.4315	0.0729
	B	0	0.0135	0.0381	-0.0032	0.0055	0.0143	0.0060	-0.0033
1205	A	1.0000	0.0085	0.1516	0.0017	1.0000	0.5637	1.1100	0.0072
	B	0	-0.0015	-0.0035		0	-0.0336	-0.0700	-0.0076

Table 7.8 Estimated polynomials. ARMAX-model, PRBSsinus-stimulation.

Test \ interval	50:100	100:140	120:190
405	3 2 27	5 4 9	4 6 50
705	4 3 1	5 4 44	4 5 12
905	5 4 7	3 4 5	4 4 15
1205	3 3 0	5 6 4	5 4 5

Table 7.9 Model order and time delay, ARX-model. PRBSsinus-stimulatinon.
The figures read, degree A degree B Time delay (measured in number of samples).

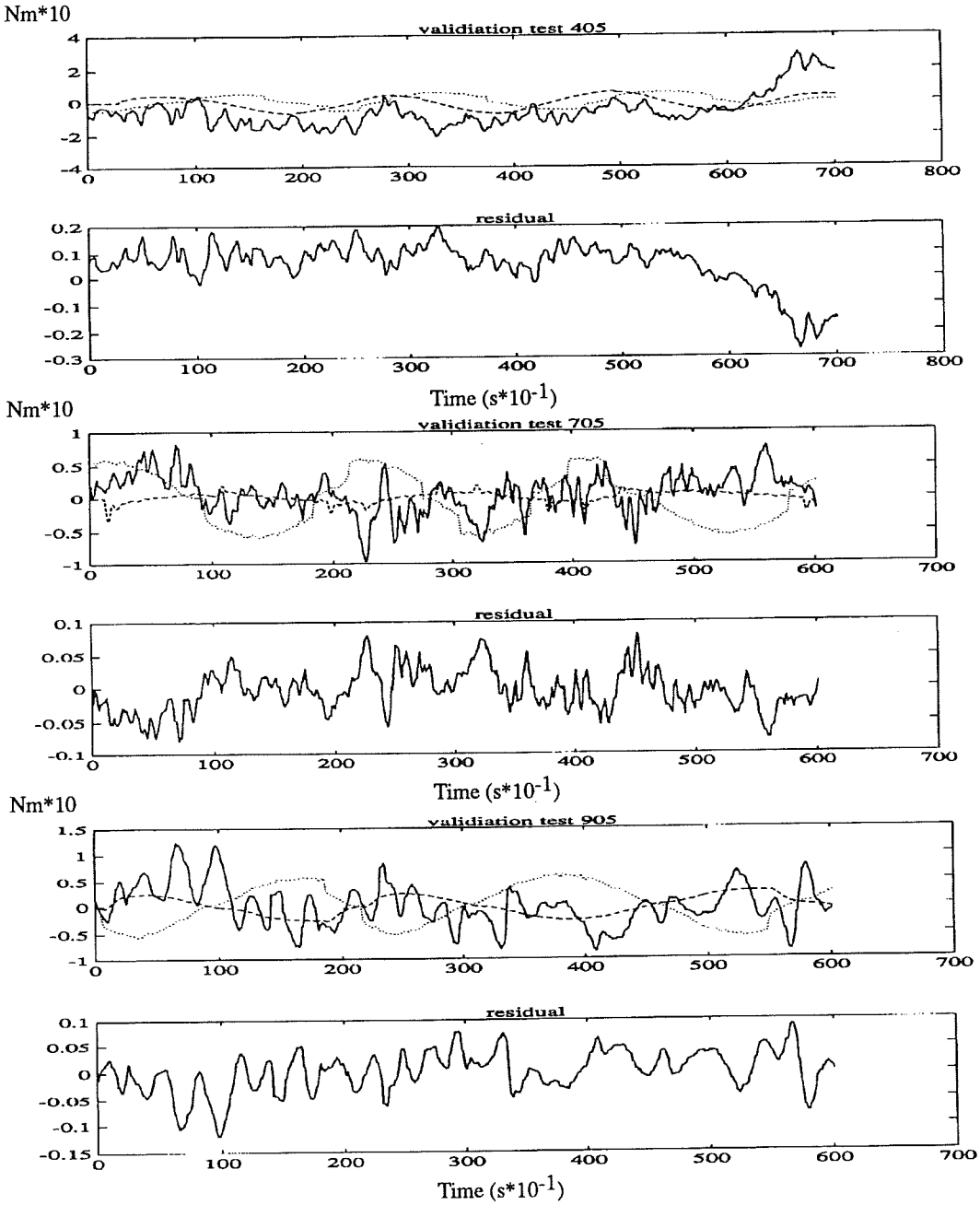


Figure 7.15 Validiation of ARMAX-model, PRBSsinus-stimulation.

The validation indicates the estimated model to be quite well adapted to the moment response from postural control, even if the residual from time to time could be rather high.

The curves in the validation plot in figure 7.15 are:

..... stimulus (velocity of the screen)

- - - - - - - - - - - - - - - - - - response from the estimated model (ARMAX)

——————— measured ankle torque

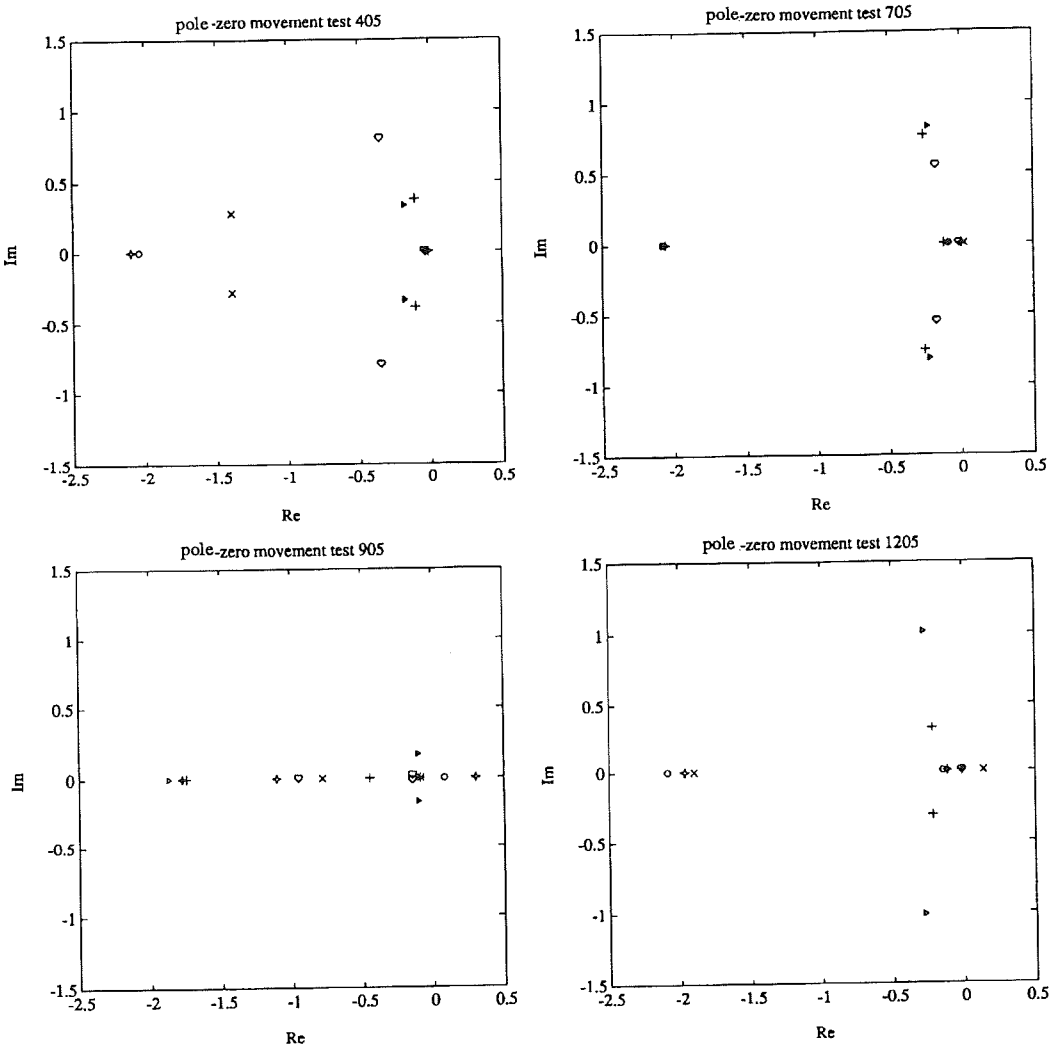


Figure 7.16 Pole - zero placement, ARMAX-model, PRBSsinus-stimulation. See page 40 about the marks.

Initially the pole-zero placement makes the model quite unstable, but in the end the poles have moved to the real axis while the zeroes have moved to an imaginary placement. A pole placed as an integrator is found during the whole test.

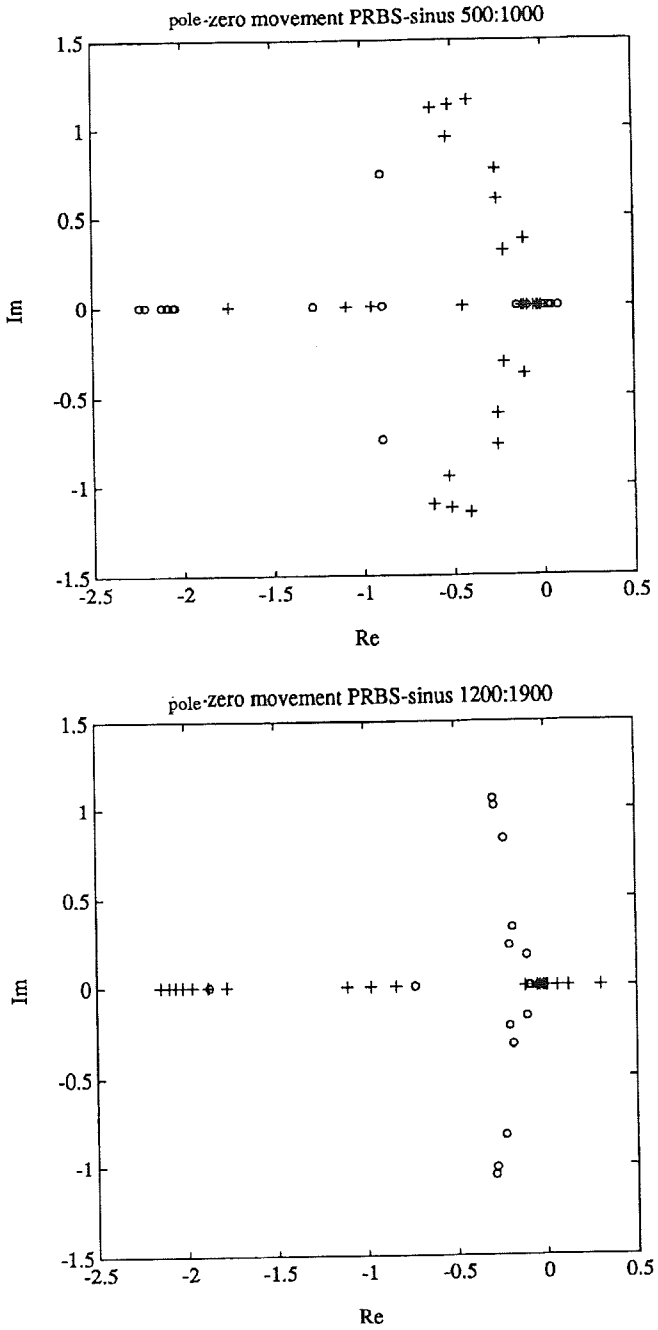


Figure 7.17 Pole - zero placement. Identification at 100 -150 s and 180 - 250 s, PRBSsinus-stimulation, all test subjects.

A large similarity in pole and zero placement between the different subjects is found during the test. They also change their models in the same way towards a more stable control principle.

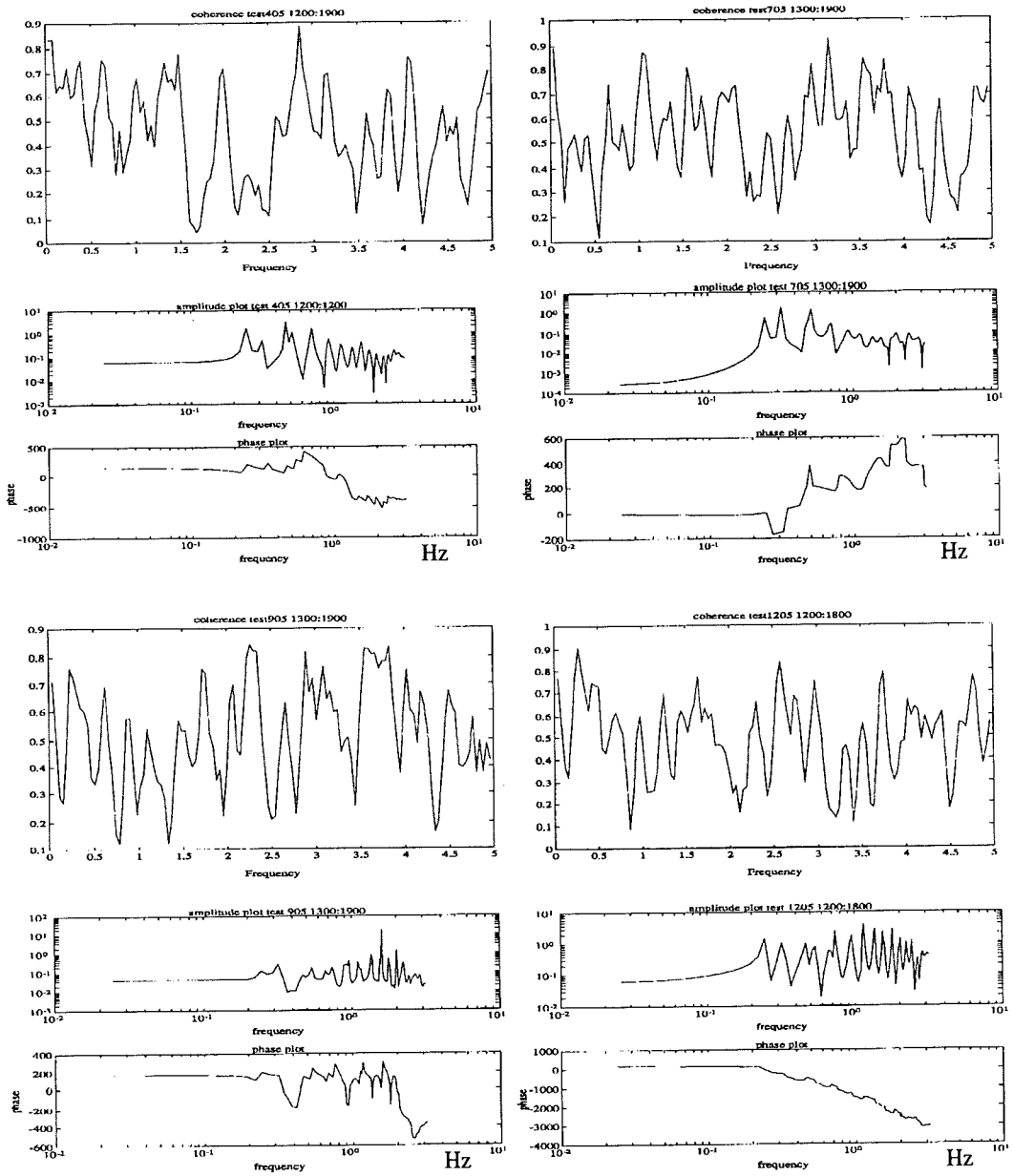


Figure 6.18 Results from the nonparametric identification. PRBSsinus-stimulation.

The coherence shows similarities to the previous stimulus, with very varying level of coherence, but with a rather high value in the low frequency range. The Bode-diagram shows differences between the response from different subjects, mostly as different phase shifts. The gain shows more frequency dependence than with previous types of stimulus.

8 Discussion

The main goal of our work was to design an equipment for visual stimulation and to investigate if it could affect postural control. Previous tests with visual stimuli has been performed with moving rooms, spinning tents and similar big settings. The reason for this is that the stimuli should cover up the visual field of the test subjects, and that a real object moving is considered better than a projected image [18]. We also wanted the stimuli to give the illusion of movement forwards and backwards, not only sideways or rotation. Finally we decided to use a vertical screen moving horisontally at one side of the person. On the other side we placed a mirror instead of a screen.

The results of our measurements clearly indicate that an actual visual stimuli has been created. We have obtained moment responses up to 20 Nm but the responses decrease towards the end of the test.

From previous studies is it known that the visual information stabilize the moment response in postural control [9]. This could, e.g., be seen in measurements of body movements with closed and open eyes, respectively. It has been shown in experiments in dogs, that if visual and proprioceptive information are contradictory, visual influence will be more important than proprioceptive [17]. On the other hand, visual influence could be adapted away relatively fast if the input is contradictionary to other receptor systems, and if these other receptor systems are undisturbed [1],[2]. It is important to remember the strong interconnection between vision and vestibular input.

In the present experiments the visual input to postural control was distorted while information from other receptors where correct concerning the body's position and movements. This causes the other receptor systems to dominate the postural control, but if another receptor system would be disturbed simultaneously as the visual stimuli, it is likely that the visual input would cause the strongest response.

The importance of the pattern of the visual stimulus is also interesting. Maki et al and Ostrovski et al have tested the influence of diffent motion stimuli, i.e., if continuous or transient patterns affect postural control differently [13]. They reported that visual information didn't have much influence once the pattern was learned, but it could be of importance when the model for the stimuli movement was being estimated. However, in contrary to the present experimental design, the visual stimuli was given together with body perturbations. Our results

showed that the postural response are depending on the stimuli pattern. This was seen both in gain variations and phase shifts.

Sinusoidal-stimulation:

The results from the sinusoidal stimulus show that vision is of high influence initially, inducing high moment responses. During the course of the test the influence of vision however decreases or adapts to the stimulus which results in weaker moment responses. A spontaneous movement in counterphase to the stimulus is observed relatively early in the recordings. Some subjects had however very weak responses, especially towards the end of the test.

Due to the lack of excitation in the frequency plane caused by a pure sinusoid, estimations of polynomials of higher complexity than second order models were unreliable. It is, however, interesting to see changes in time delays and parameters.

The results from the parametric estimation in the sinusoidal test must, due to the lack of excitation, be looked upon with some doubt. It is however possible to find a violent dynamic activity in the form of great variations, especially in the pole zero placement.. It is also possible to study the changes in time delays during the course of the test. In the nonparametric identification a strongly varying coherence function is seen. The coherence is relatively stable at a high level only in the frequency range 0.05-0.2 Hz. The Bode-diagram can therefore only be considered reliable within this range. This result is valid for all the different tests, independent of their properties. The identified system has a proportional gain between stimuli and moment response with a phase shift of about 180 degrees. This is more or less valid for all the tested subjects.

PRBS-stimulation:

This sequence doesn't follow any visually identifiable motion pattern which one can adapt to. Compared to the other tests it can be seen that the time delay between stimulus and response is shorter, and it seems as if the visual impression is more important throughout the test, but the moment response is less powerful. It is possible that the input signal is too unreal an impression to be interpreted as a motion. The test sequence is ideal from the identification point of view, since we obtain an excitation over a wide range of frequencies.

The nonparametric analysis shows a more powerful and wider excitation in the frequency domain. Even with this stimulus the

coherence was high in the range 0.05-0.2 Hz. Also in the Bode-diagram similarities to those of the sinus-stimulation are seen. The phase shift is however displaced with 180° and 90° depending on the test subject. There is also a frequency dependence in the phase shift.

PRBSsinus-stimulation:

This sequence contains both the regularity of an underlying sinusoid motion and the irregularity of the PRBS-added phase shifts. This stimulus induces the most powerful moment responses of all stimulation sequences. The result is unexpected because one may have expexted the subjects to learn to distrust the visual input over the repeated tests, and therefore induce the smallest response. One explanation might be fatigue, another one that powerful readings are encountered when the motion isn't following the easier recognizable sinusoidal pattern imagined by the person. Therefore, the direction of the moment response could be unpredictable and controlled by other principles such as anticipation. The phase shift is however displaced with 180°,90° or 0° (depending on the subject), and it is also frequency dependent.

The results from the ARMAX- and ARX-identifications shows significant variations both in the number of parameters and the time delay between visual disturbance and correcting ankle torque. But it is possible to estimate a relatively good model with small residual which is able to follow our recorded values of postural response.

The long time delays that was estimated from ARMAX identification are confusing. One explanation might be found in the models suggested by Raphan et al and Robinson et al [6]. In these models, vision complements the incapability of the vestibular organs to handle low frequencies, by utilizing the velocity of the eye. This feedforward gives a high gain and long time delay between stimulus and response (up to 16 seconds). An often repeated suggestion is the presense of an internal model in the brain estimating the dynamics of the semicircular canals or the dynamics of the body in rotation. The influence of the internal model could be recognized if the signals from a receptor system are interrupted, e.g. ,by turning off the light. An interesting parallel have been found when testing dogs. When stimulating visually and with a platform motion of the same frequency as the visual stimuli, the results corresponds to the results from stimuli with platform motion on a blindfolded dog.

As a summary, test subjects with small moment responses were quite

independent of the appearance of the visual stimuli. The most powerful moment responses were found with the PRBS-sinus, which could be due to fatigue, since it was the last of the sequences, but also to the composition of the sequence with regularity and random changes. Furthermore it is felt that if the subjects had learned to distrust the visual input over the repeated tests, this last test should have induced the smallest responses.

One difficulty at the verification of the results is to determine when to consider vision and its impression to be of significant importance in postural control, or at least when it is dominating in the control function. The large variations in the coherence function may be due to other systems, such as the proprioceptive system, taking control over the postural adjustments. But it could also be a result of the linearisation not being correct or of the model being nonlinear. On basis of the different identifications made and the different references referred to, we can say that the system contains both feedback and feedforward control , the latter in form the of anticipation. With pure visual stimulation the response could well be adapted to a simple inverted pendelum with correspondance to an ARMAX-model of order [3 2 2] (deg(A B C)). Both validation and residual give small values and the measured error could be considered as white noise most of the time.

The results of pole-zero placement show a connection between postural control principles of the test subjects. Especially in the last two tests it is easy to see the movement from a less stable form of control toward a more stable one. The pole-zero movement leads towards a blocking of stimulus influence (zeros), and a higher stability (poles). Another result is that in all identifications it is possible to find a pole placed as an integerator or very close. However, this placement could be caused by low frequency dynamics and therefore it should be studied further at a lower sample rate.

The nonparametric identifications indicate that vision has it's strongest influence in the low frequency range, which is also verified by the coherence analysis. Due to great variation in the coherence values outside the range 0.05-0.2 Hz, no conclusion can be drawn about the function outside this range. In the Bode-diagram one can see a smooth proportional gain at low frequencies, but a greatly varying phase lag depending on the appearance of the stimulation.

9 Conclusions

* Our studies have shown that pure visual stimuli do affect postural control and that the responses from our subjects in the test group show similarities in their control principles.

* The visual pattern affects the dynamics of the postural response in different ways. This is shown as responses with different phase lag and different proportional gains.

* There is a pattern in the way postural control adapts to stimuli. In all tests there is a movement from a less stable form of control in the beginning towards a more stable form of control, as the test progresses.

* In our identifications of postural control it is always possible to find a pole placed as an integrator, or at least very close to being one.

* The responses of the postural control could be well adapted to a mechanical model of a simple inverted pendelum, in studied conditions.

Acknowledgements

We wish to direct a special thanks to our tutors
Rolf Johansson, LTH & Måns Magnusson, ENT-clinic
for their much good advice and inspiration to complete this work.

We would also like to thank:
Roland Fransson and Ingvar Troedsson, for help and advice with the mechanical construction.
Olof Samuelsson, for help with the programming.
Hans Svensson, for the mirror stand.
Camilla Nilsson, Gunnar Sundström, Karl Skoberne and Kenneth Kristoffersson, for helping with the report and lending us their computers.

References

(1) A M Bronstein, suppresion of visually evoked postrural responses, Exp Brain Res, 63 :655-658. (1986)

(2) A M Bronstein, J D Hood, M a Gresty, C Danage, Visual control of balance in cerebellar and parkinson syndromes, Brain, 113 : 767-779. (1990)

(3) J J Craig, Introduction to robotics, Addison-Wesley . (1986)

(4) R H S Carpenter, Movement of the eyes, Pion London . (1977)

(5) J H Courjon, G Clement, R Schmid, The influence of interstimulus interval on the development of vestibular habituation to repeated velocity steps, Exp Brain Res 59: 10-15. (1985)

(6) H C Diener, J Dichgans , On the role of vestibular, visual and somatosensory information for dynamic postural control in humans, Prog Brain Res 76,: 253-262. (1988)

(7) A C Guyton, Textbook of medical physiology, W B Saunders. : 624-626. (1976)

(8) V Henn, Visual-vestibular interaktion neurosiences, Neurosciens Res, Prog Bull Vol 18 , No 4 :575-617. (1982)

(9) A Ishida, S Imai , Response of the posture-control system to pseudorandom acceleration disturbans, Med. Biol. Eng. Comput. Vol 18 : 433-438. (1980)

(10) R Johansson, M Magnusson, Human postural dynamics, CRC Critical Reviews in Biomedical Engineering. (1991)

(11) R Johansson, M Magnusson, M Åkesson, Identification of human postural dynamics, IEEE Trans. Biomed. Eng. Vol 35 : 858-869. (1988)

(12) J Jäger, V Henn, Habituation of vestibulo-ocular reflex (VOR) in the monkey during sinusoidal rotation in the dark, Exp Brain Res 41 : 108-114. (1981)

(13) B E Maki, G Ostrovski, A comparison of the transient balance recovery and continuous postural regulation, In Disorders of posture and gait. ed. T Brandt, W Paulus, W Bles, M Dietrich S Krafcayk, A Straube. G Thieme verlag München. (1990)

(14) l M Nashner, G McCollum, The organization of human postrual movements, a formal basis and experimental synthesis, Behav. and Brain Sci. 8 : 135-172 . (1985)

(15) J F Stein, An introduction to nuerophysiology, Blackwell Scientific Publications. (1982)

(16) T Söderström, P Stoica, Systems Identifiation, Prentice Hall. (1984)

(17) R E Talbott, J M Brockhart A predictive model study of the visual contribution to canine postural control, A m. J Physiol. 239 : P80-92. (1980)

(18) L R Young, personal communication

(19) K J Åström, Reglerteori, Almquist o Wiksell. (1985)

(20) K. J Åström & B Wittenmark, Adaptive Control, Addison Wesley .(1989).

(21) E Popov, B N Smetana, V Y Shilkov, Visual effekts of electrical vestibular stimulation, In Disorders of posture and gait. ed. T Brandt, W Paulus, W Bles, M Dietrich S Krafcayk, A Straube. G Thieme verlag München. (1990)

(22) H Enbom, Vestibular and somatosensory contribution to postural control : 10-12, Thesis Univ of LUND .(1990)

(23) J L Meriam, L G Kraige, Mechanics, volume I,II, Statics, John Wiley & sons. (1987)

(24) B Östergren, personal communication.

(25) LA-5600, Linear Amplifier Instruction manual, Electro-Craft Corporation. (1986)

Appendix I: Equipment

The mechanical parts of the test equipment are a screen console with stand and an electrical control system, and a mirror stand.

Screen equipment:

The screen equipment can be separated in two parts. The upper part is a screen console with two rollers. The rollers have two guide slits for two thin belts that has been vulcanised to the back of the screen. One of the rollers has a through shaft that is journalled inside the roller. This roller is mounted with two jamb nuts on telescopic tubes into the framework of the console. The mounting is equipped with a spreader making it possible to move the roller sideways. This construction not only makes it possible to mount the screen, but also to adjust the stretch to a desired level. The driver roller is journalled through two external bearings. The motor is connected through a rubber padding to avoid motor breakdown at overload. The motor can easily be dismounted by loosening four mounting bolts. On the screen console there are also four stop screws for adjustments of the height.

The stand is equipped with four movable legs and two long leading tubes for the screen console. Each leg can be fixed in an appropriate angle by locking the wheels. If you wish, it is possible to lock the legs in a transport position with stop screws at the legmounting. The electrical control system is mounted on the framework beam with four mounting bolts. More information on the control system is to be found in the accompanying user manual. CAUTION!! **A certain amount of caution is necessary when working with the control system, since it has open 220V power connectors** CAUTION!!. A lift jack is also situated on the framework beam. With this it is possible to adjust the height of the screen.

When adjusting the height or preparing the equipment for transportation, the following work order should be followed.

Raising the screen:

* lower the lift jack piston to the bottom by opening the strangler.
* Fasten the spacer at the desired level with the stop screw.
* Make sure that the foot of the lift jack is standing securely on the framework beam.

* make sure that the strangler on the lift jack is closed.
* Loosen the stop screws on the legs of the screen console.
* Pump the lift jack to it's upper position.
* Fasten the screen console with all stop screws.

Repeat all steps until the desired height has been reached.

Lowering the screen:

* Position the spacer at the desired level by extracting the lift jack piston, fasten it with the jamb nut and close the strangler.
* Loosen the stop screws on the legs of the screen console.
* Slowly open the strangler until the screen slowly sinks to the bottom position of the lift jack.
* Fasten the jamb nuts on the console.
* Repeat all steps until the desired position is reached or until the screen is in it's lowest position.

Mirror stand:

The mirror stands on two separate telescopic legs. The legs consist of two square tubes, and the mirror is fastened on the upper one. Each leg has two socked head cap screws on the back, just below the mirror. By loosening these, it is possible to adjust the height. Each leg can also be positioned anywhere on the mirror by loosening the bolt on top of the leg.

To get an illusion as real as possible during stimulation, one should keep the mirror clean.

Setup:

To get the best results possible from the stimulus, it is necessary to adjust the height of the screen and of the mirror so that all of the subjects visual field is covered when positioned on the force platform. The upper part of the visual field is blocked off by using an appropriate headgear. Make sure that the screen and the mirror has the same height, and that the mirror is positioned in front of the edge of the screen. The latter is necessary for avoiding a fixed edge in the subjects visual field.

Position the legs of the screen stand to make it stand steady on the ground, and to make room for the force platform.

Attach the cable from the control system to the computer as follows:

cable	computer connection

white	AO 0
purple	AI 0
blue	ground

Avoid running test sequences with the cable connected if you don't want to have the motor running, use the plot function instead.

When the test sequences are to be run, plug in the power cord from the control system. This should make the cooling fan start running. If the output from the computer is 0 V, then the screen should be still. If not so, follow the instructions in the control systems user manual to make the proper adjustments [25]. If everything looks normal, then the test can be started. It is not recommended to run many long test sequences in a row, as this increases the strain on the motor due to generation of heat. The cooling fan should be kept running for about three minutes after finishing the tests.

Appendix II: Mechanical drawings

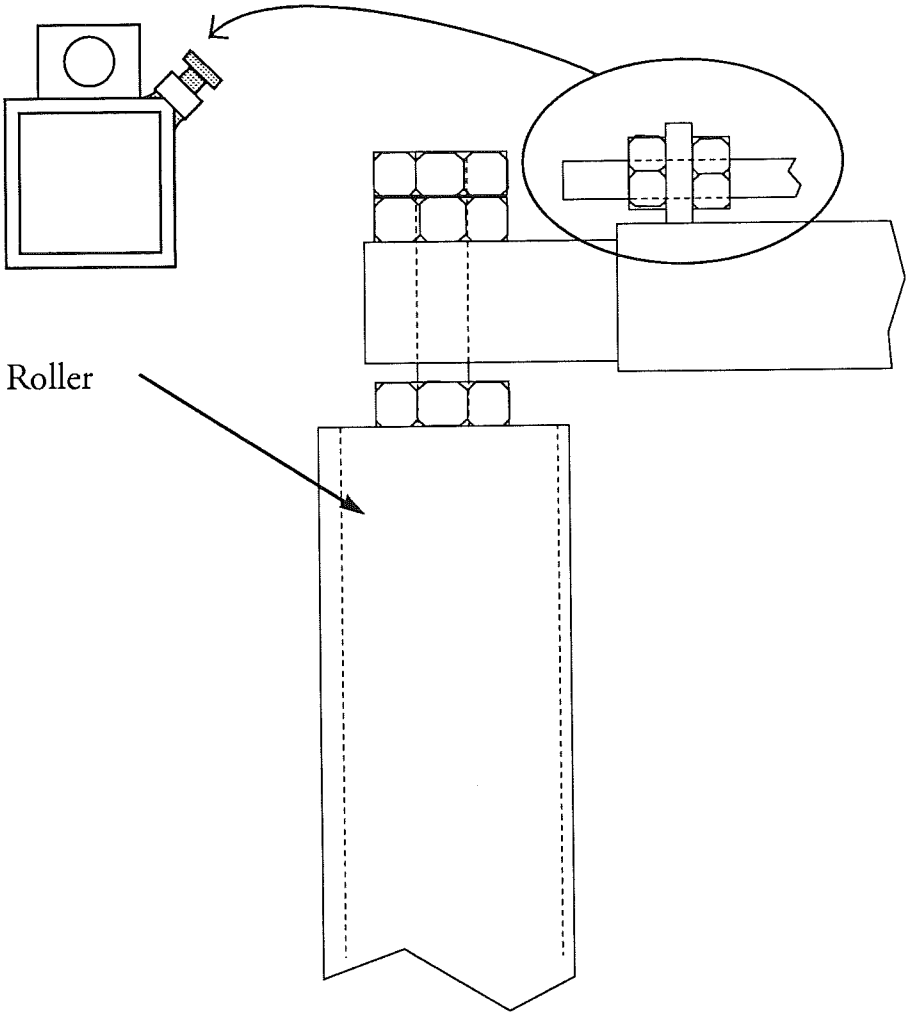


Figure II.1 The band stretching adjuster.

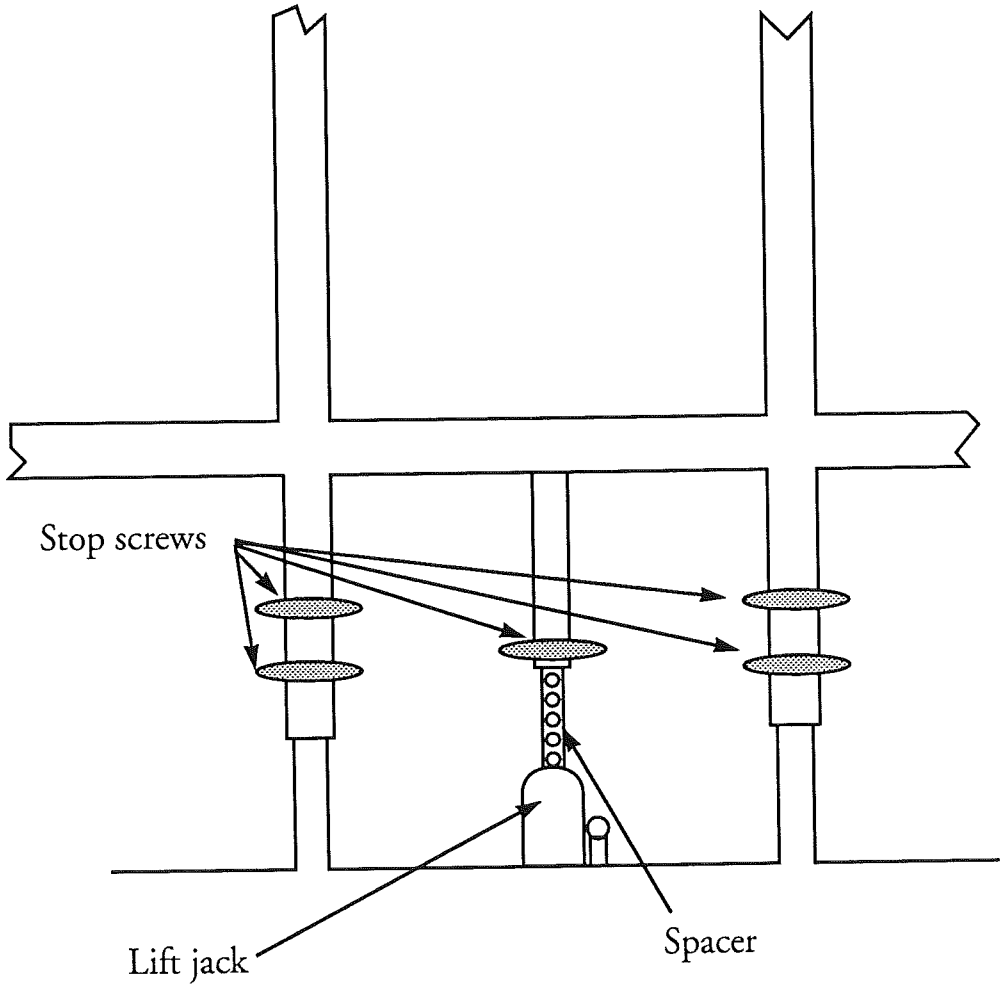


Figure II.2 The height adjustment mechanism.

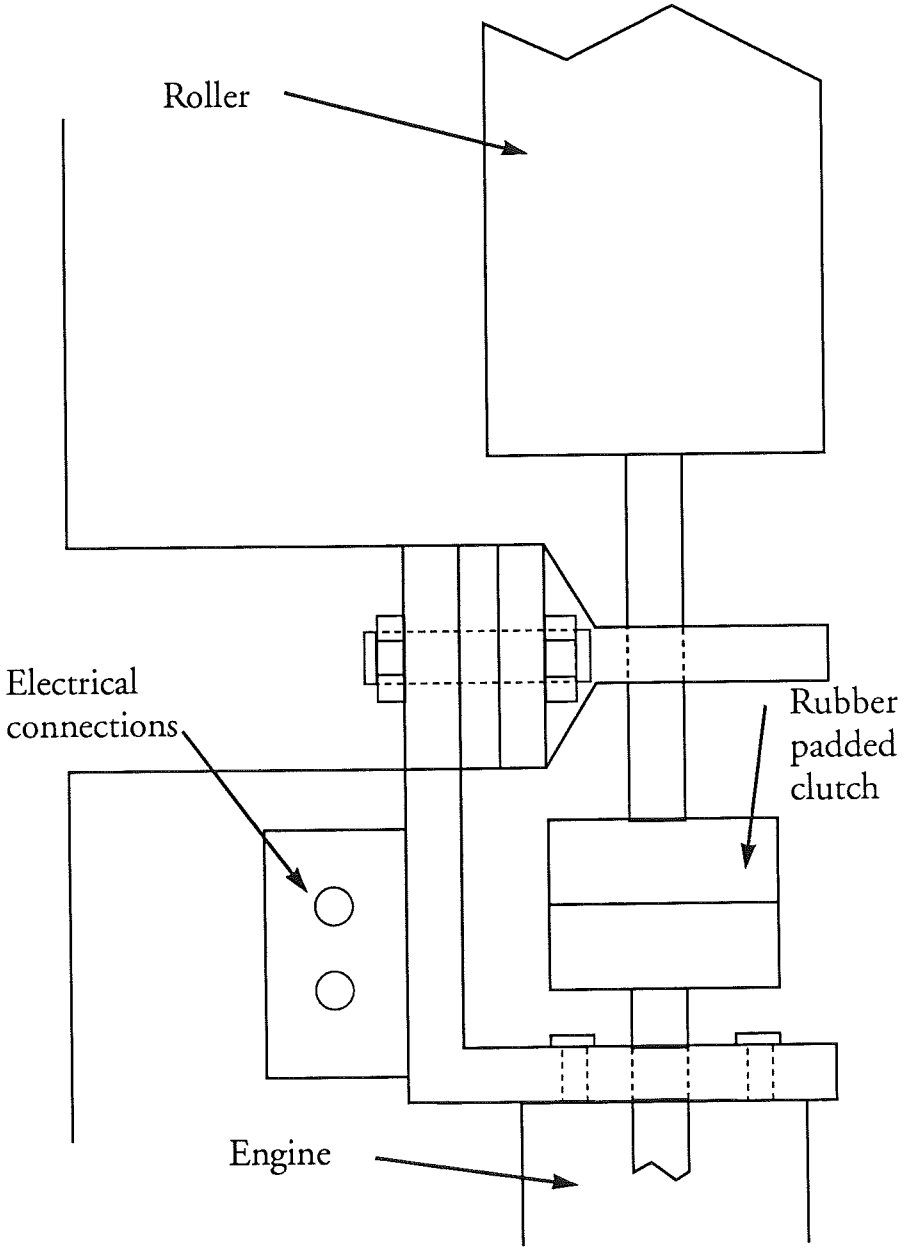


Figure II.3 Motor attachment.

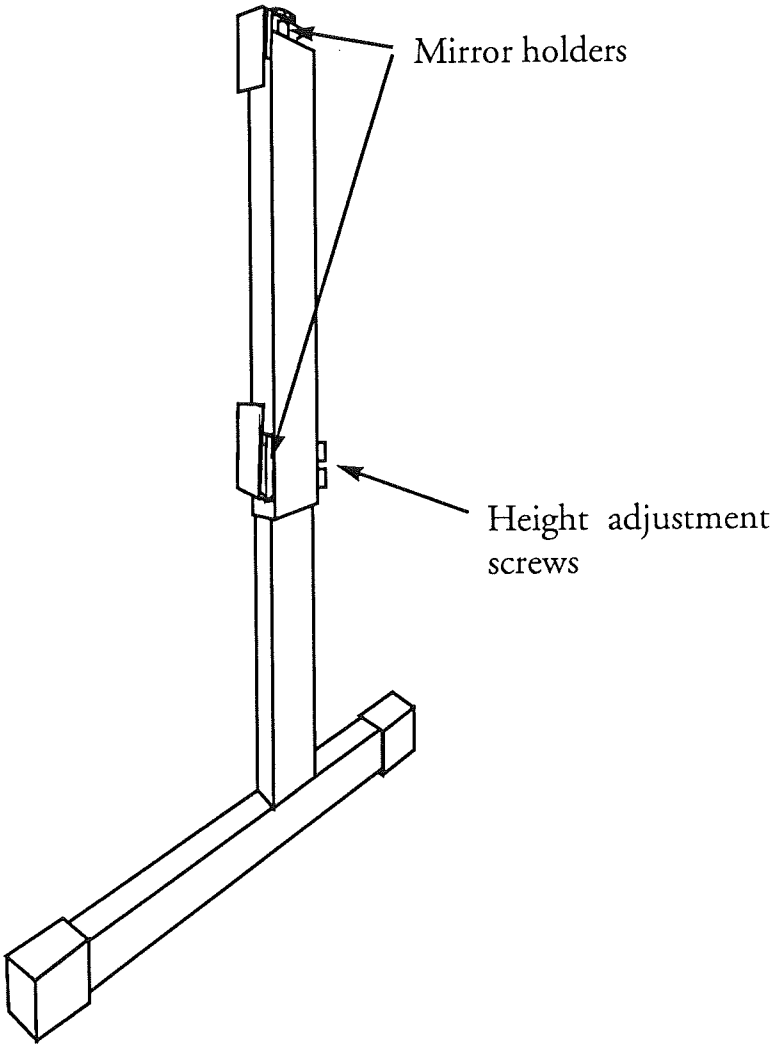


Figure II.4 Mirror stand.

Appendix III: Motor calculations

Our choice of motor has been based upon the maximum load the motor is to be affected by [24],[23]. The biggest force and moment load will be in form of moment of inertia when the belt is accelerated and retarded. The size of the load is determined by

$$M_w = I\alpha$$

M_W =inertia

$$\alpha = \frac{d(\frac{v}{r})}{ds}$$

v=speed

r=radious

$$I = \frac{m}{2}(r_y^2 + r_i^2)$$

Our demands on performance where chosen to be 2 seconds breaktime from a maximum speed of 4 m/s

Data of the rollers: r_y=63 mm r_i=40 mm

mass: m=2*1.5*15.5=37.2 kg

Data of the belt r_d=2 mm

mass: m=3*3.2=9.6 kg

The moment affecting the motor will in this case be 2.0 Nm. With this maximum moment value as a demand we selected the motor. The servo system is suitable for this motor family and it is also adapted to the voltage levels of the computers D/A-ports. Data on the motor and the servo system is to be found in their manuals [25].

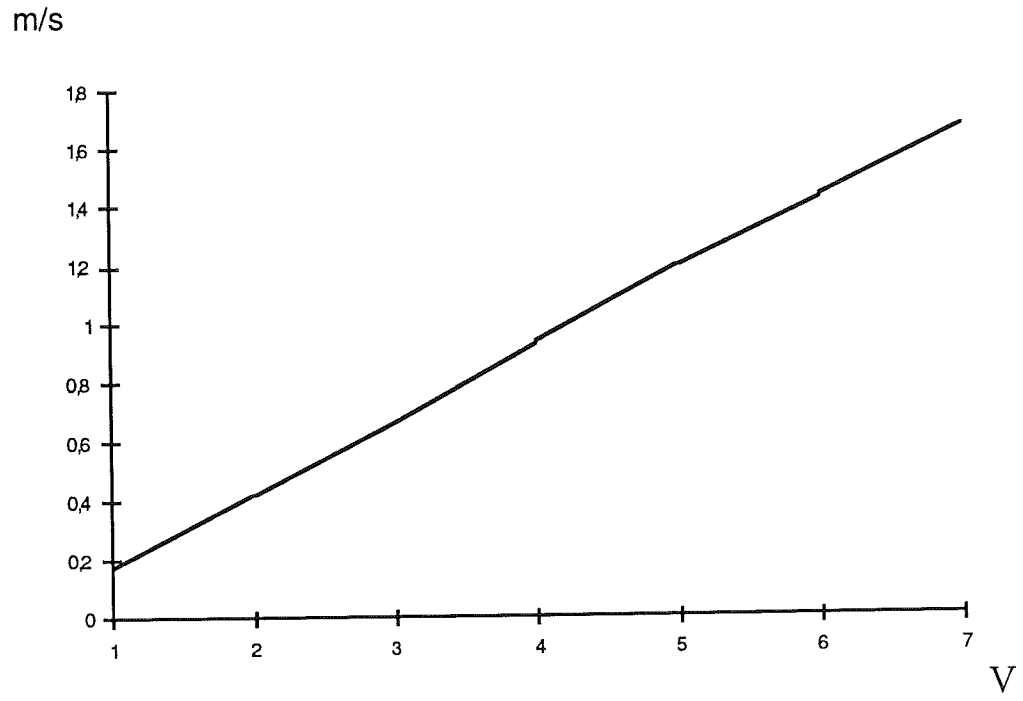


Figure III.1 Relationship between motor input and screen speed

Appendix IV: The force platform

To measure the movements of the person exposed to stimuli, we used a force platform developed at the Lund University Hospital.

When a human is standing upright, mainly vertical forces will affect the footsoles. The distribution of these forces change when the body is swaying which quite easily can be measured by vertical sensors. However, bigger movements also cause shear forces in the platforms plane. Three vertical sensors may therefore not give a satisfactory image of what's happening.

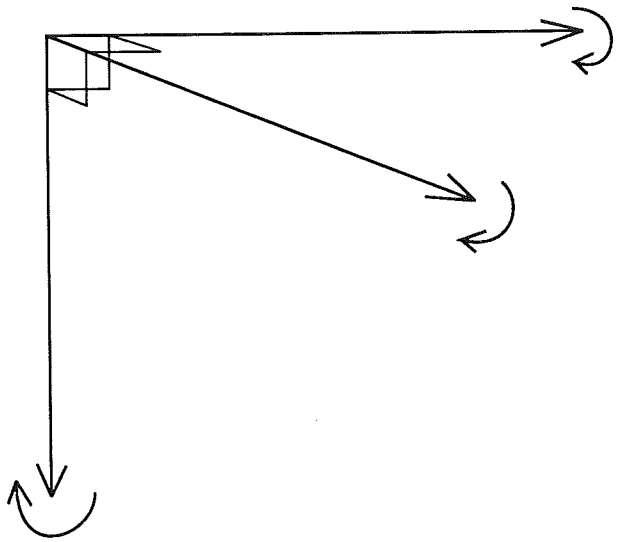


Figure IV.1 Measuring in six degrees of freedom.

To measure in all six degrees of freedom, it is necessary to have at least six sensors, translation and rotation around three axes as shown in figure IV.1 [3].

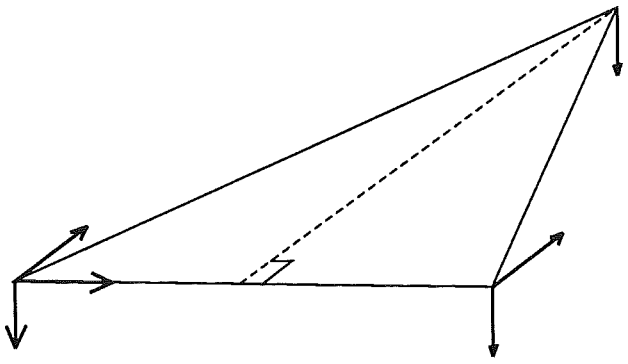


Figure IV.2 Placement of the sensors in the force platform.

The sensors in the platform are placed in a triangle according to figure IV.2.

With the placement of the sensors above the forces can be measured according to

$$Fx = fx$$
$$Fy = fy1 + fy2$$
$$Fz = fz1 + fx2 + fz3$$
$$Mx = (fz2 + fz3)*a/2$$
$$My = fz1*b$$
$$Mz = (fx1 + fx2)*a/2$$

Strain gauges have been chosen as sensors, because they are relatively cheap and one can use simple bridge amplifiers. The measuring range is about 0-10 Hz.

Appendix V: Identification

We have used the following MatLab-functions when estimating appropriate models for our system. We have also implemented a number of programs to speed up the calculation time. A short summary of the most important identification routines and how they are used follows below. Our own MatLab-implemented functions follows in appendix II.

Nonparametric:

Autospectrum: Shows the frequency contents and the power of the frequencies.

Coherence spectrum: Shows how the output signal depends on the input signal

Cross-spectrum: A mix of in- and out-spectrum, This is an intermediate result for further calculations, especially the transfer function.

Transfer function: Gives the connection between input- and output signals for each frequency.

Spa: Estimates a transfer function by a spectral analysis. By using the MatLab-function Bodeplot you will get the transfer function and phase delay for each frequency.

Parametric:

ARMAX: Calculates a prediction error estimation of an ARMAX-model according to

$$A(q)y(t)=B(q)u(t-nk)+C(q)e(t)$$

th=ARMAX(Z,nn)

With $Z=(t_{bal},u)$, i.e. series of output (t_{bal}) and input (u)

nn=(na,nb,nc,nk), nk=timedelay, na,nb,nc=degree of A-,B- and C- polynomials

ARX: Calculates a prediction error estimation of an ARX-model according to

$A(q)y(t)=B(q)u(t-nk)+e(t)$

th=ARX(Z,nn)

With $Z=(t_{bal},u)$

nn=(na,nb,nk), nk=timedelay, na,nb,nc=degree of A- B- and C-polynomials

SelStruc: Chooses the best model degree according to a desired selection rule, in our case AIC.

Present: presents a parametrical model with standard deviation, loss factor and weighing factor according to Akaike.

Idsim: Simulates a system given by Y=IDSIM(Z,th), th from e.g. ARMAX and ARX.

With $Z=(t_{bal},u)$

Polyform: Adapts (a,b,c,d,f)=POLYFORM(th) which gives a standard input-output form according to

$A(q)y(t)=(B(q)/F(q))u(t-nk)+(C(q)/D(q))e(t)$

Resid: Shows the remaining error which the model cannot compensate.

Contin: Transforms ARX- and ARMAX-results to a continuous polynomial of the form

$A(s)y(t)=B(s)u(t)$

Autocorrelation: Shows if the error is white noise, i.e. ,if covariance is 0. This would mean that there is no connection with the error from the previous estimation and the present error.

Crosscorrelation: Shows if the error e(t) is independent of the input signal v(t).

Simulation validation: Is the result of a combined calculation of the estimated model in relation to the measured values.

Continuous estimation:

In our program we have implemeted a function for a recursive least square estimation of the model

$$A(q)y(t)=B(q)u(t)+e(t)$$

with the formula

$$\hat{\theta}(t) = \hat{\theta}(t-1) + K(t)(y(t) - \varphi^{T}(t)\hat{\theta}(t-1))$$
$$K(t) = P(t-1)\varphi(t)(\lambda + \varphi^{T}(t)P(t-1)\varphi(t))^{-1}$$
$$P(t) = \frac{(1 - K(t)\varphi^{T}(t))P(t-1)}{\lambda}$$

P(t)=estimation matrix controlling the behaviour of the estimator

λ=forgetting factor for old values.

The estimator has been provided with a shutting off if the error e(t)is less than an errorsize MinErr choosen by the user. Furthermore, the estimation constant P(t) is controlled to avoid overestimation. To get a good value from the algorithm, an exitation over a frequency range as wide as possible is required. The signals u(t) and y(t) are given by the tachometer signal from the motor respectively signals from the sensors in the force platform combined to an ankle torque through the formula

tbal=gamma(b*(y(1)+y(2))-a*y(3))

Gamma=Transfer constant between the force sensors output voltage and the corresponding force (N/V).

y(1)=Signal from sensor(1) (see appendix IV).

a,b=Placement of the centerpoint of gravity in relation to the back- and front edge of the platform

When we calculate for a MatLab-estimation, we compensate for the fact that the test subject isn't standing in the middle of the platform. This is done by calculating and subtracting the mean value of the sensors normal forces.

At the identification work we have used the following routines and work order. The routines are described in appendix VI.

```
(* data to balance form *)

z=reform(test1205);

title('test1205 ');

(* balanceform limited within given data area *)

z=limitreform(test1205,1200,1800);

( * PARAMETRIC *)

(* estimate the best time delay for ARMAX-model *)

th=fastvalue(z);

(* estimate the best time delay for ARX-model *)

th=fastarx(z);

(* presents the result from ARMAX and ARX- identification *)

present(th);

(* discrete to continuous polynom *)

(ARMAXth1205aT,ARMAXth1205aN)=contin(th,1,0);

(* simulate model *)
```

```
simulate (z,th);

(* validate estimated model and residual *)

valid (z,th);

title(' validation test 1205 ');

(* autocorrelate residual *)

correlate(z,th);

title(' autocorrelation (e) test1205');

(* correlation between stimulation and response *)

p=xcorr(z(:,1),z(:,2));
plot(p);

title(' crosscorrelation (tbal-u) test1205');

(* stepresponse 50*0 200*1 *)
(* impulsresponse 100*0 1*1 299*0 *)
(* fom our estimated model *)

systemresponse(th);

title(' stepresponse test 1205 ');

(* zero-pol plot *)

(A,B,C,D,F)=polyform(th);
pzpl(B,A,1,4,(-2.5 0.5 -1.5 1.5));

title('pol-zero movement test 1205');

(* NONPARAMETRIC *)

(* spectral analysis *)
```

```
q=spectrum(z(:,2),z(:,1),256);

specplot(q,10);
powerspec(q,10);

title('Pxx-u power spectral dencity test1205 ');

coherspec(q,10);

title('coherence test1205 1200:1800');

transferspec(q,10);

title('Txy- transfer function magnitude test1205 ');

(* bode-diagram *)

g=spa(z);
newbode(g);

title(' amplitude plot test 1205 1200:1800 ');
```

Appendix VI: MatLab-listings

```
function z=reform(data);

%REFORM(data);
%
%    makes balanceform from data array "data"
%    into the form z=[tbal,u];
%    where tbal is moment respons and u control signal.
%

z1=mean(data(:,2));
z2=mean(data(:,3));
z3=mean(data(:,4));

mg=z1+z2+z3;
a=0.341*z3/mg;
b=0.341*(z1+z2)/mg;

t=[1:1:length(data(:,1))];
u=data(:,9);
tbal=a*(data(:,4)-z3)-b*((data(:,2)-z1)+(data(:,3)-z2));
tbal1=tbal*20;

z=[tbal,u];
clg
ylabel('Nm*10');
xlabel('s*10E-1');
plot(t,u,t,tbal1);
end
```

```
function z=limitreform(data,lo,hi);

%LIMITREFORM(data,lo,hi);
%
%       makes balanceform from data array "data"
%       within data array data(:,lo) to data(:,hi)
%       into the form z=[tbal,u];
%       where tbal is moment resonse and u control signal.
%

datab=data(lo:hi,:);
z1=mean(data(:,2));
z2=mean(data(:,3));
z3=mean(data(:,4));

mg=z1+z2+z3;
a=0.341*z3/mg;
b=0.341*(z1+z2)/mg;

t=[1:1:length(datab(:,1))];
u=datab(:,9);
tbal=a*(datab(:,4)-z3)-b*((datab(:,2)-z1)+(datab(:,3)-z2));
tbal1=tbal*20;

z=[tbal,u];
clg
ylabel('Nm*10');
xlabel('s*10E-1');
plot(t,u,t,tbal1);

end
```

```
function th=fastvalue(z);

%FASTVALUE(z);
%
%             makes fast ARMAX modell with
%             order [3,2,2] and varying time delay
%             in TH-form which could be used by PRESENT.
%

delay=0;
best=0;
error=10;
newerror=9;
x=0;
while x<20,
 th=armax(z,[3,2,2,delay]);
 newerror=th(2,1);
 if newerror<error ,
   error=newerror;
   x=0;
   best=delay;
 else
   x=x+1;
 end;
 disp(['Loss fcn: ' num2str(th(1,1)) '   Akaike's FPE: ' num2str(th(2,1)) ]);
 disp(['delaytime    ' num2str(delay) '    best ' num2str(best) ]);
 delay=delay+1;
end;
th=armax(z,[3,2,2,delay-21]);
present(th);
disp(['delaytime    ' num2str(delay-21)])
```

```
function th=fastarx(z);

%FASTARX(z);
%
%           Makes  ARX modell with different
%           orders up to [6 6] and varying time delays
%           to find the optimal order and time delay.
%           The result is in form TH which could be used by
%           PRESENT.

td=0;
best=0;
error=10;
newerror=9;
x=0;
while x<20,
 NN=[3 2 td;3 3 td;4 3 td;4 4 td;3 5 td;3 4 td;5 4 td;5 5 td;4 5 td;4 6 td;5 6  td;];
 V=arxstruc(z,z,NN);
 nn=selstruc(V,'aic');
 th=arx(z,nn);
 newerror=th(2,1);
 if newerror<error ,
   error=newerror;
   x=0;
   best=td;
   nn1=nn;
 else
   x=x+1;
 end;
 disp(['Loss fcn: ' num2str(th(1,1)) '   Akaike's FPE: ' num2str(th(2,1)) ]);
 disp(['delaytime    ' num2str(td) '    best ' num2str(best) ]);
 td=td+1;
end;
th=arx(z,nn);
present(th);
nn1
disp(['delaytime    ' num2str(td-21)])
```

```
function simulate(z,th);

%SIMULATE(z,th);
%
%    simulates the model in form TH,
%    reactions on control signal u
%    compared to measured moment tbal.
%


ym=idsim(z(:,2),th);
t=[1:1:length(z(:,1))];
y=z(:,1);
u=z(:,2);
clg
plot(t,y*10,t,ym*10,t,u);
ylabel('Nm*20');
xlabel('s*10E-1');
end
```

```
function valid(z,th);

%VALID(z,th);
%
%    makes a validation by simulation of the
%    model TH with control signal z(:,2). It also plots
%    the residual between model in form TH and the
%    real response measured and stored in z=[tbal,u]
%


ym=idsim(z(:,2),th);
t=[1:1:length(z(:,1))];
y=z(:,1);
u=z(:,2);
clg
e=ym-y;
subplot(211);
plot(t,y*10,t,ym*10,t,u);
ylabel('Nm*10');
xlabel('s*10E-1');
subplot(212);
plot(t,e);
ylabel('Nm*10');
xlabel('s*10E-1');
title('residual');
subplot(211);

end
```

```
function systemresponse(th);

%SYSTEMRESPONSE(th);
%
%    simulates the model in form TH
%    with a step and impuls as
%    control input.
%

clg
t1=450:1:699;
u1=round(t1/1000);
t1=t1-450;
ym=idsim(u1',th);

subplot(211);
axis([0,250,-4,1.5]);
plot(t1,ym*60,t1,u1);
ylabel('Gain');
xlabel('s*10E-1');

t1=490:1:609;
t2=489:1:608;
u2=round(t1/1000)-round(t2/1000);
axis([0,120,-2,1.2]);
t1=t1-490;

subplot(212);
axis([0,120,-2,1.2]);
ym=idsim(u2',th);
plot(t1,ym*60,t1,u2);
ylabel('Gain');
xlabel('s*10E-1');
title('impulsresponse ');
axis;
subplot(211);

end
```

```
function powerspec(P,Fs)

%POWERSPEC(P,Fs);
%
%       Spectrum plot using the output of the SPECTRUM function.
%       uses P, the output of SPECTRUM,  and Fs, the
%       sample frequency:
%
%               Pxx - X Power Spectral Density
%               Pyy - Y Power Spectral Density
%

n = max(size(P));
m = min(size(P));
f = (1:n-1)/n*Fs/2;

clg
subplot(211);
semilogy(f,P(2:n,1)),...
xlabel('Frequency')
ylabel('W');

if m == 1
        return
end
subplot(212);
semilogy(f,P(2:n,2)),...
title('Pyy - tbal Power Spectral Density');
xlabel('Frequency');
subplot(211);

end
```

```
function transferspec(P,Fs)

%TRANSFERSPEC(P,Fs);
%
%       Spectrum plot using the output of the SPECTRUM function.
%       TRANSFERSPEC(P,Fs), uses P, the output of SPECTRUM,  and Fs, the
%       sample frequency, to successively plot:
%
%               abs(Txy) - Transfer Function Magnitude
%               angle(Txy) - Transfer Function Phase
%

n = max(size(P));
m = min(size(P));
f = (1:n-1)/n*Fs/2;

clg
subplot(211);
semilogy(f,abs(P(2:n,4))),...
xlabel('Frequency')
ylabel('Gain')

subplot(212);
plot(f,180/pi*angle(P(2:n,4))),...
title('Txy - Transfer function phase');
xlabel('Frequency'),...
ylabel('Degree')
subplot(211);

end
```

```
function correlate(z,th);

%CORRELATE(z,th);
%
%    Autocorrelates the error between model and
%    measured values in z=[tbal,u]. It also makes
%    crosscorrelation between the error and control
%    signal u for possible connections.
%


ym=idsim(z(:,2),th);
t=[1:1:length(z(:,1))];
y=z(:,1);
u=z(:,2);
e=ym-y;
eac=xcorr(e);
ecc=xcorr(e,u);
clg;
subplot(211);
plot(eac);
subplot(212);
plot(ecc);
title('crosscorrelation (e-u)');
subplot(211);

end
```

```
function coherspec(P,Fs);

%COHERSPEC(P,Fs);
%
%       Spectrum plot using the output of the SPECTRUM function.
%       uses P, the output of SPECTRUM,  and Fs, the
%       sample frequency and average covariace value within length of 3.
%
%               Cxy - Coherence Function
%

Pl=P(:,5);
n=length(Pl);
for i=1:n;
  if i==1,
    pl(i)=(Pl(i)+Pl(i+1))/2;
  elseif i==n,
    pl(i)=(Pl(i-1)+Pl(i))/2;
  else
    pl(i)=(Pl(i-1)+Pl(i)+Pl(i+1))/3;
  end;
end;
pl=pl';
n = max(size(P));
m = min(size(P));
f = (1:n-1)/n*Fs/2;

clg
if m == 1
        return
end

plot(f,abs(pl(2:n))),...
xlabel('Frequency'),
```

Appendix VII: Complementary results & graphs

Under this chapter all our results from estimation and validation made by the previous program listings are collected.

* Estimated continuous polynomial with their respective time delay.

* Estimation and validation of sinus-stimulation.

* Estimation and validation of PRBS-stimulation.

* Estimation and validation of PRBSsinus-stimulation.

VII.1 Estimated continuous polynomials with their respective time delay.

EXJOBB indentifierade polynom

test403 500:1000 : delay = 0

ARMAXth403aN =

0.0442	0.0872	0.0710	0.0000

ARMAXth403aT =

1.0000	0.9079	0.8292	0.0550

1000:1500 :delay = 3

ARMAXth403bN =

0	-0.0073	-0.0152	0.0000

ARMAXth403bT =

1.0000	0.5564	0.7359	0.0460

1500:2000 :delay = 3

ARMAXth403cN =

0	-0.0220	-0.0187	-0.0007

ARMAXth403cT =

1.0000	1.6454	0.7191	0.0352

1800:2500 : delay = 31

ARMAXth403dN =

0	-0.0240	-0.0515	-0.0023

ARMAXth403dT =

1.0000	0.9127	1.3027	0.0927

test703 500:1000delay = 0

ARMAXth703aN =

0.0245	0.0540	0.0465	-0.0012

ARMAXth703aT =

1.0000	1.3209	1.2798	0.1285

1000:1500 :delay = 38

ARMAXth703bN =

```
         0   -0.0063   -0.0133    0.0006

ARMAXth703bT =

    1.0000    0.7988    0.8689    0.0951

 1500:2000 :delay = 49

ARMAXth703cN =

   1.0e-03 *

         0   -0.0959   -0.2023    0.0136

ARMAXth703cT =

    1.0000    0.4896    0.1316    0.0003

 1800:2500 :delay = 4

ARMAXth703dN =

         0   -0.0125   -0.0372   -0.0032

ARMAXth703dT =

    1.0000    2.5273    1.7472    0.2392

test903 500:1000 : delay = 0

ARMAXth903aN =

    0.0104    0.0472    0.0506   -0.0021

ARMAXth903aT =

    1.0000    4.2788    4.2190    0.0997

 1000:1500 :delay = 49

ARMAXth903bN =

         0   -0.0049   -0.0114   -0.0002

ARMAXth903bT =

    1.0000    1.0391    0.5554    0.0470

 1500:2000 :delay = 6

ARMAXth903cN =

         0    0.0095    0.0268   -0.0013
```

```
ARMAXth903cT =

    1.0000    2.2574    1.1277    0.0717

 1800:2500 :delay = 0

ARMAXth903dN =

    0.0044    0.0036    0.0002   -0.0001

ARMAXth903dT =

    1.0000    0.6230    0.1391    0.0114

test1203 500:1000 : delay = 3

ARMAXth1203aN =

         0   -0.0323   -0.0707   -0.0016

ARMAXth1203aT =

    1.0000    1.0000    1.1806    0.0201

 1000:1500 :delay = 0

ARMAXth1203bN =

   -0.0049   -0.0097   -0.0069   -0.0001

ARMAXth1203bT =

    1.0000    0.6780    0.1123   -0.0006

 1500:2000 :delay = 3

ARMAXth1203cN =

         0    0.0415   -0.0041

ARMAXth1203cT =

    1.0000    2.5383    0.0599

 1800:2500 :delay = 16

ARMAXth1203dN =

         0    0.0234    0.0474   -0.0038

ARMAXth1203dT =

    1.0000    0.8644    1.3823    0.0652
```

test503 500:1000 : delay = 8

ARMAXth503aN =

0	-0.0007	-0.0013	0.0002

ARMAXth503aT =

1.0000	0.2737	0.5797	0.0206

1800:2500 :delay = 16

ARMAXth503bN =

0	0.0547	0.1334	-0.0011

ARMAXth503bT =

1.0000	1.9919	2.3608	0.1395

test603 500:1000 : delay = 13

ARMAXth603aN =

0	0.0080	0.0160	-0.0002

ARMAXth603aT =

1.0000	0.8798	1.7729	0.0306

1800:2500 :delay = 4

ARMAXth603bN =

0	-0.0072	-0.0143	-0.0002

ARMAXth603bT =

1.0000	0.7174	1.5858	0.0782

test803 500:1000 : delay = 32

ARMAXth803aN =

0	0.0006	0.0014	0.0002

ARMAXth803aT =

1.0000	0.5099	0.5376	0.0096

1800:2500 :delay = 2

ARMAXth803bN =

```
         0   -0.0043   -0.0093   -0.0009

ARMAXth803bT =

    1.0000    0.5874    0.7312    0.0173

test1003 500:1000 : delay = 3

ARMAXth1003aN =

         0   -0.0108   -0.0224    0.0001

ARMAXth1003aT =

    1.0000    0.7799    1.1568    0.0466

 1800:2500 :delay = 15

ARMAXth1003bN =

         0   -0.0095   -0.0211   -0.0011

ARMAXth1003bT =

    1.0000    1.3611    1.9855    0.0951

test1103 500:1000 : delay = 0

ARMAXth1103aN =

   -0.0018   -0.0029   -0.0022    0.0000

ARMAXth1103aT =

    1.0000    0.3007    0.5702    0.0235

 1800:2500 :delay = 57

ARMAXth1103bN =

         0    0.0058    0.0134    0.0004

ARMAXth1103bT =

    1.0000    1.0941    0.8929    0.0703

test1303 500:1000 : delay = 14

ARMAXth1303aN =

         0   -0.0187   -0.0421    0.0007
```

```
ARMAXth1303aT =

    1.0000    1.5697    2.1249    0.1271

 1800:2500 :delay = 42

ARMAXth1303bN =

         0   -0.0030   -0.0073    0.0002

ARMAXth1303bT =

    1.0000    1.5537    1.0551    0.0421

test404 500:1000 : delay = 29

ARMAXth404aT =

   1.0e-03 *

         0    0.2316    0.4944    0.0161

ARMAXth404aN =

    1.0000    0.3582    0.0832   -0.0001

 1000:1500 :delay = 1

ARMAXth404bN =

   1.0e-03 *

         0    0.5490   -0.7408   -0.3534

ARMAXth404bT =

    1.0000    1.2578    0.3422    0.0093

 1500:2000 :delay = 10

ARMAXth404cN =

         0   -0.0123   -0.0293   -0.0007

ARMAXth404cT =

    1.0000    1.2843    0.9532    0.0860

 1800:2500 :delay = 3

ARMAXth404dN =

         0    0.0171    0.0008   -0.0008
```

```
ARMAXth404dT =

    1.0000    0.9305    0.1930    0.0085

test704 500:1000 : delay = 10

ARMAXth704aT =

         0   -0.0391   -0.0796   -0.0099

ARMAXth704aN =

    1.0000    1.7825    4.1009    0.3056

 1000:1500 :delay = 22

ARMAXth704bN =

         0    0.0095    0.0038   -0.0001

ARMAXth704bT =

    1.0000    2.0845    0.8246    0.0870

 1500:2000 :delay = 6

ARMAXth704cN =

         0   -0.0020   -0.0047   -0.0011

ARMAXth704cT =

    1.0000    0.6424    0.7228    0.0732

 1800:2500 :delay = 14

ARMAXth704dN =

   1.0e-03 *

         0   -0.3336   -0.6978    0.0246

ARMAXth704dT =

    1.0000    0.4589    0.2827    0.0018

test904 500:1000 : delay = 8

ARMAXth904aT =

         0   -0.0157   -0.0083   -0.0009

ARMAXth904aN =
```

```
    1.0000    1.0423    0.2889    0.0184

 1000:1500 :delay = 10

ARMAXth904bN =

         0   -0.0202   -0.0237   -0.0009

ARMAXth904bT =

    1.0000    1.9927    0.8995    0.0200

 1500:2000 :delay = 7

ARMAXth904cN =

         0   -0.0125   -0.0293   -0.0025

ARMAXth904cT =

    1.0000    1.0792    0.8417    0.0459

 1800:2500 :delay = 5

ARMAXth904dN =

         0   -0.0002   -0.0013   -0.0028

ARMAXth904dT =

    1.0000    1.1721    0.8378    0.0514

test1204 500:1000 : delay = 12

ARMAXth1204aT =

         0   -0.0073   -0.0156   -0.0005

ARMAXth1204aN =

    1.0000    0.6733    0.8315    0.0331

 1000:1500 :delay = 25

ARMAXth1204bN =

         0   -0.0257   -0.0574    0.0078

ARMAXth1204bT =

    1.0000    1.7857    2.3600    0.0725

 1500:2000 :delay = 19
```

```
ARMAXth1204cN =

         0   -0.0127   -0.0275   -0.0045

ARMAXth1204cT =

    1.0000    0.6815    1.1672    0.0516

 1800:2500 :delay = 10

ARMAXth1204dN =

    0.0160    0.0406    0.0159   -0.0029

ARMAXth1204dT =

    1.0000    2.8253    1.7941    0.1083

test504 500:1000 : delay = 30

ARMAXth504aT =

   1.0e-03 *

         0    0.4375    0.7818   -0.6449

ARMAXth504aN =

    1.0000    1.0213    0.8559    0.0677

 1800:2500 :delay = 1

ARMAXth504bN =

         0   -0.0008   -0.0019   -0.0007

ARMAXth504bT =

    1.0000    0.7798    0.8217    0.0422

test604 500:1000 : delay = 32

ARMAXth604aT =

         0   -0.0092   -0.0186   -0.0017

ARMAXth604aN =

    1.0000    0.6829    1.6005    0.0804

 1800:2500 :delay = 12

ARMAXth604bN =
```

```
         0    0.0047    0.0086   -0.0021

ARMAXth604bT =

    1.0000    0.5053    1.0461    0.0377

test804 500:1000 : delay =

 1800:2500 :delay =

test1004 500:1000 : delay = 25

ARMAXth1004aT =

   1.0e-03 *

         0   -0.4392   -0.9933    0.0340

ARMAXth1004aN =

    1.0000    0.8045    0.1137   -0.0001

 1800:2500 :delay = 9

ARMAXth1004bN =

         0   -0.0035   -0.0041   -0.0003

ARMAXth1004bT =

    1.0000    2.2439    0.2362    0.0058

test1104 500:1000 : delay = 28

ARMAXth1104aT =

         0   -0.0021   -0.0046   -0.0003

ARMAXth1104aN =

    1.0000    0.9965    1.0561    0.0556

 1800:2500 :delay = 10

ARMAXth1104bN =

         0   -0.0458   -0.0717   -0.0056

ARMAXth1104bT =

    1.0000    3.0360    2.3190    0.0520

test1304 500:1000 : delay = 30
```

```
ARMAXth1304aT =

         0   -0.0145   -0.0219   -0.0001

ARMAXth1304aN =

    1.0000    2.9255    2.1572    0.1013

 1800:2500 :delay = 19

ARMAXth1304bN =

   1.0e-03 *

         0   -0.3154   -0.6431    0.0392

ARMAXth1304bT =

    1.0000    0.2709    0.0872   -0.0003

test405 500:1000 : delay = 33

ARMAXth405aT =

         0   -0.0067   -0.0139   -0.0004

ARMAXth405aN =

    1.0000    0.2263    0.1551    0.0019

 1000:1400 :delay = 13

ARMAXth405bN =

         0   -0.0009   -0.0025   -0.0018

ARMAXth405bT =

    1.0000    0.7459    0.7967    0.0331

 1200:1900 :delay = 28

ARMAXth405cN =

         0   -0.0048   -0.0103   -0.0003

ARMAXth405cT =

    1.0000    0.3754    0.1455    0.0014

test705 500:1000 : delay = 2

ARMAXth705aT =
```

```
         0   -0.0047   -0.0102   -0.0009

ARMAXth705aN =

    1.0000    0.6333    0.7167    0.0773

 1000:1400 :delay = 67

ARMAXth705bN =

         0    0.0069    0.0141   -0.0003

ARMAXth705bT =

    1.0000    0.3647    0.3419    0.0053

 1200:1900 :delay = 12

ARMAXth705cN =

         0   -0.0171   -0.0353    0.0000

ARMAXth705cT =

    1.0000    0.5399    0.7732    0.0616

test905 500:1000 : delay = 9

ARMAXth905aT =

         0    0.0135    0.0381   -0.0032

ARMAXth905aN =

    1.0000    2.2784    0.9692    0.0683

 1000:1400 :delay = 7

ARMAXth905bN =

         0   -0.0119   -0.0103   -0.0008

ARMAXth905bT =

    1.0000    1.2369    0.2919    0.0196

 1200:1900 :delay = 0

ARMAXth905cN =

    0.0055    0.0143    0.0060   -0.0033
```

ARMAXth905cT =

1.0000 2.0822 0.4315 0.0729

test1205 500:1000 : delay = 3

ARMAXth1205aT =

0 -0.0015 -0.0035 -0.0005

ARMAXth1205aN =

1.0000 0.4485 0.1516 0.0017

1000:1400 :delay = 3

ARMAXth1205bN =

0 0.0287 0.0506 -0.0075

ARMAXth1205bT =

1.0000 0.6423 2.1085 0.0275

ARMAXth1205cN =

0 -0.0336 -0.0700 -0.0076

1200:1900 :delay = 4

ARMAXth1205cT =

1.0000 0.5637 1.1100 0.0072

test505 500:1000 : delay = 0

ARMAXth505aT =

0.0105 0.0185 0.0136 -0.0004

ARMAXth505aN =

1.0000 0.5318 0.4305 0.0106

1400:1800 :delay = 0

ARMAXth505bN =

0.0154 0.0145 -0.0005 0.0000

ARMAXth505bT =

1.0000 0.7897 0.0464 0.0007

test605 500:1000 : delay = 17

ARMAXth605aT =

0 0.0055 0.0118 0.0003

ARMAXth605aN =

1.0000 1.0504 1.5545 0.0219

1400:1800 :delay = 5

ARMAXth605bN =

0 0.0313 0.0222 -0.0032

ARMAXth605bT =

1.0000 1.8339 3.9626 0.0363

test805 500:1000 : delay = 0

ARMAXth805aT =

0.0408 0.0882 0.0459 -0.0003

ARMAXth805aN =

1.0000 2.0772 1.1091 0.0361

1400:1800 :delay = 77

ARMAXth805bN =

0 0.0044 0.0094 -0.0001

ARMAXth805bT =

1.0000 0.4459 0.1092 0.0040

test1005 500:1000 : delay = 11

ARMAXth1005aT =

0 -0.0081 -0.0167 -0.0001

ARMAXth1005aN =

1.0000 0.8743 1.5418 0.0805

1400:1800 :delay = 25

ARMAXth1005bN =

0 0.1640 0.3078 -0.0009

ARMAXth1005bT =

1.0000 4.1914 10.4988 0.4121

test1105 500:1000 : delay = 43

ARMAXth1105aT =

0 -0.0111 -0.0245 0.0002

ARMAXth1105aN =

1.0000 1.0865 1.2114 0.0323

1400:1800 :delay = 7

ARMAXth1105bN =

0 0.0032 -0.0016 -0.0001

ARMAXth1105bT =

1.0000 0.7907 0.0481 0.0007

test1305 500:1000 : delay = 67

ARMAXth1305aT =

0 -0.0149 -0.0326 0.0015

ARMAXth1305aN =

1.0000 1.3345 1.7420 0.1719

1400:1800 :delay = 13

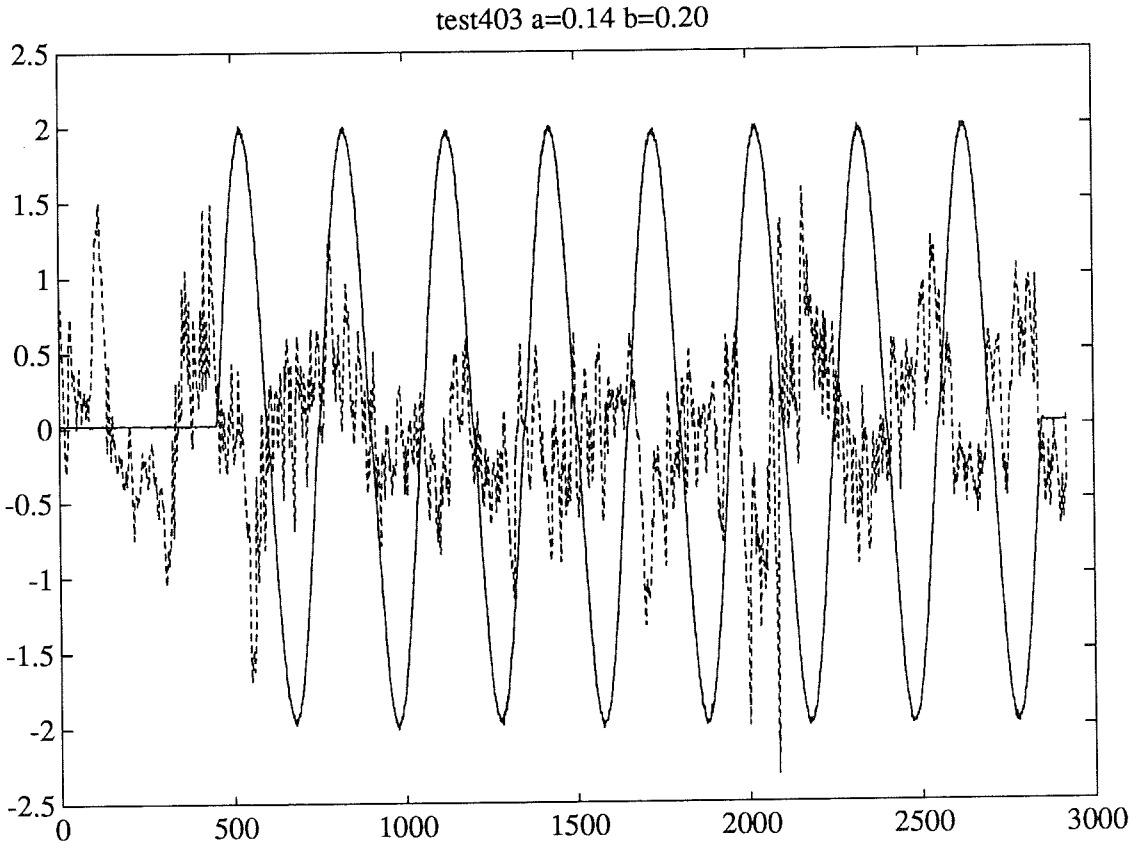
ARMAXth1305bN =

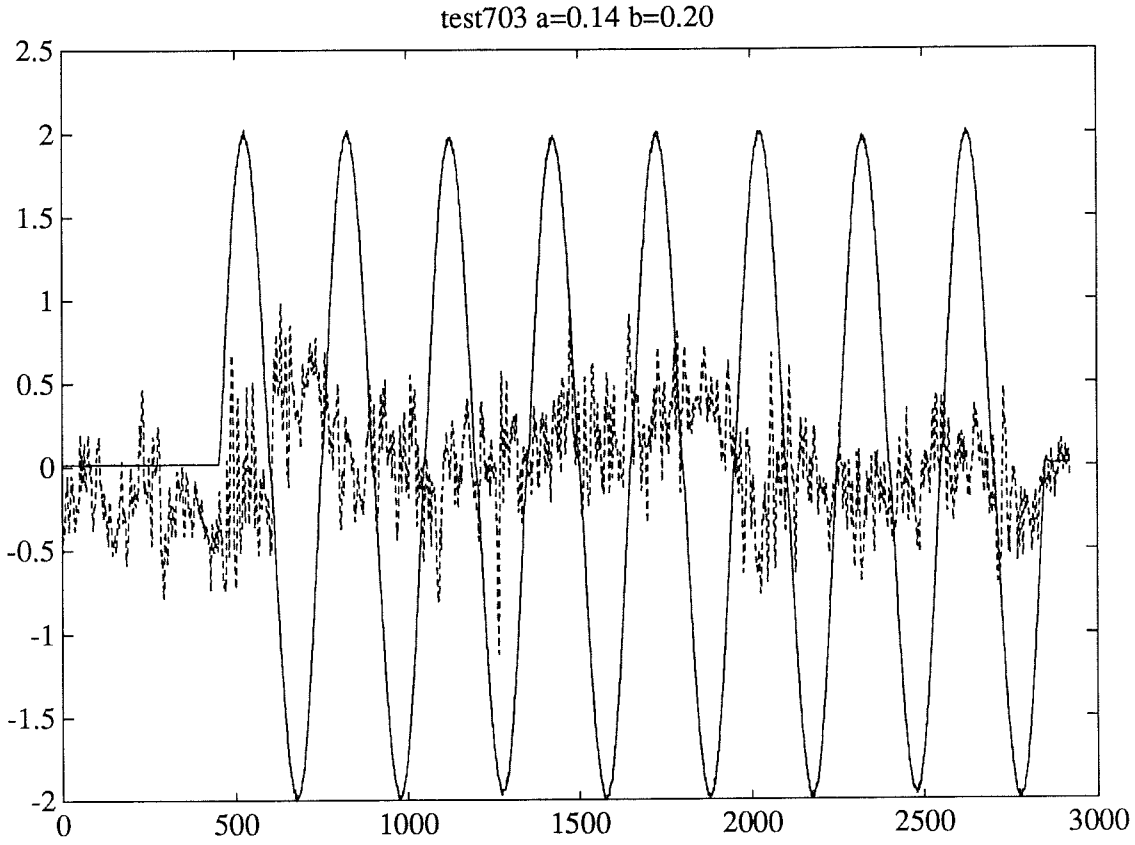
0 -0.0192 -0.0391 -0.0005

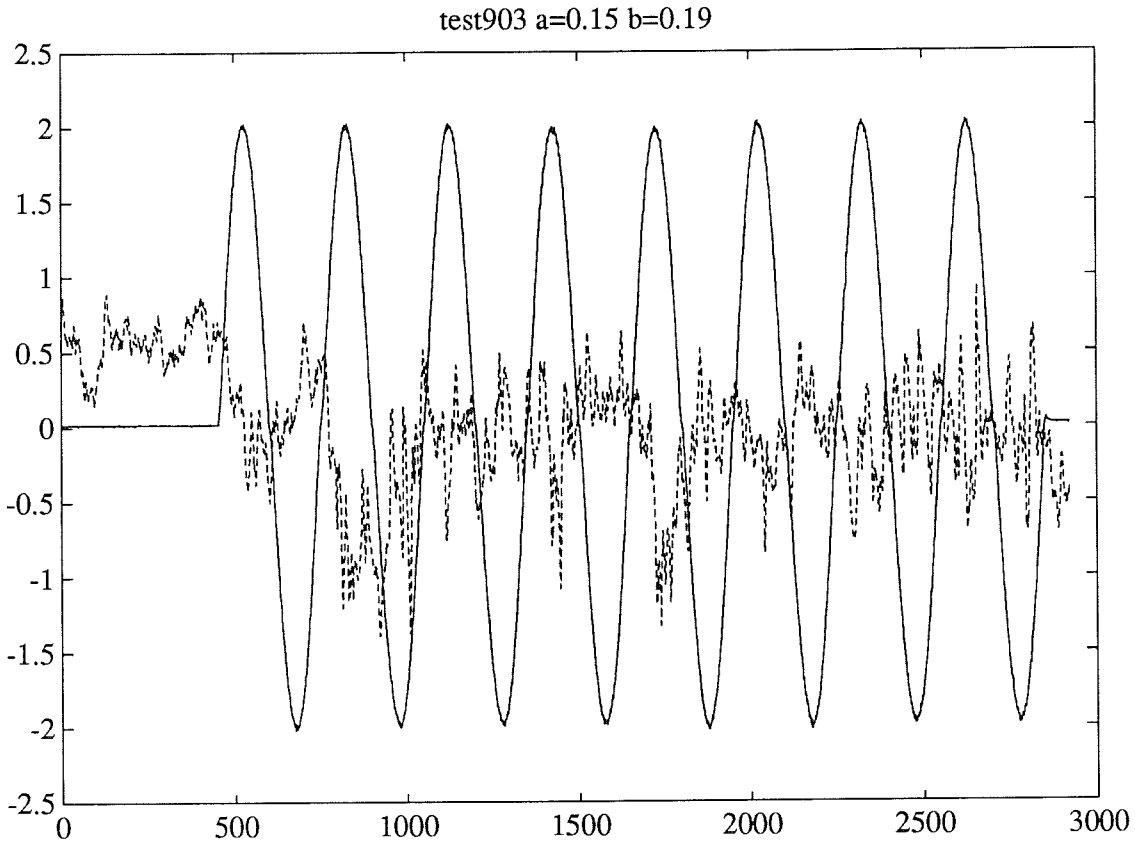
ARMAXth1305bT =

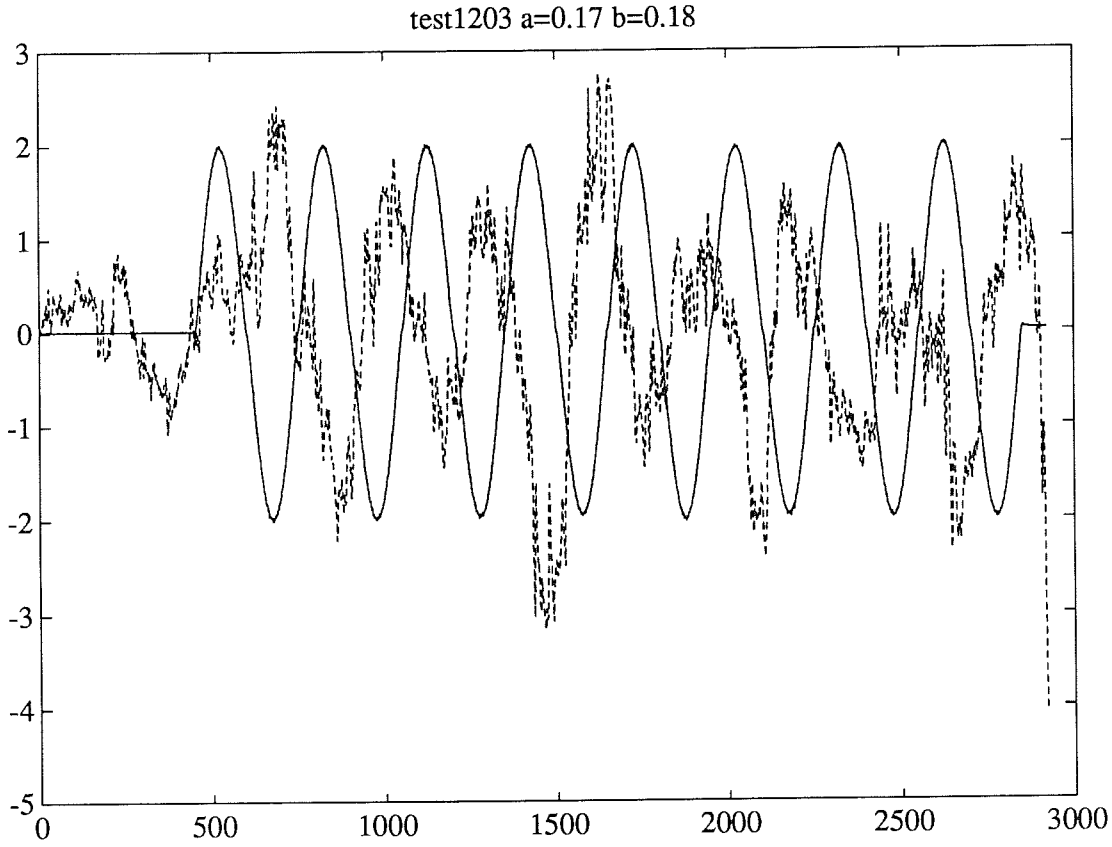
1.0000 0.6644 1.2537 0.1135

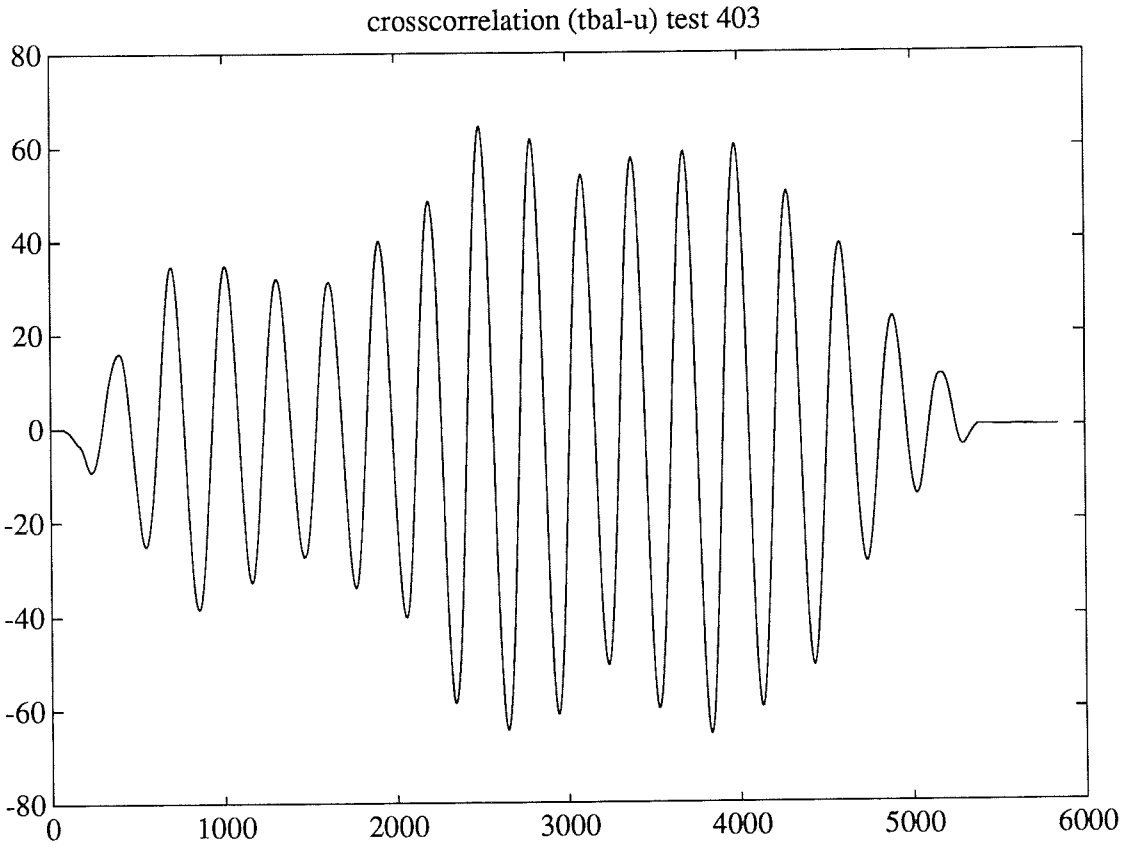
VII.2 Estimation and validation of sinus-stimulation.

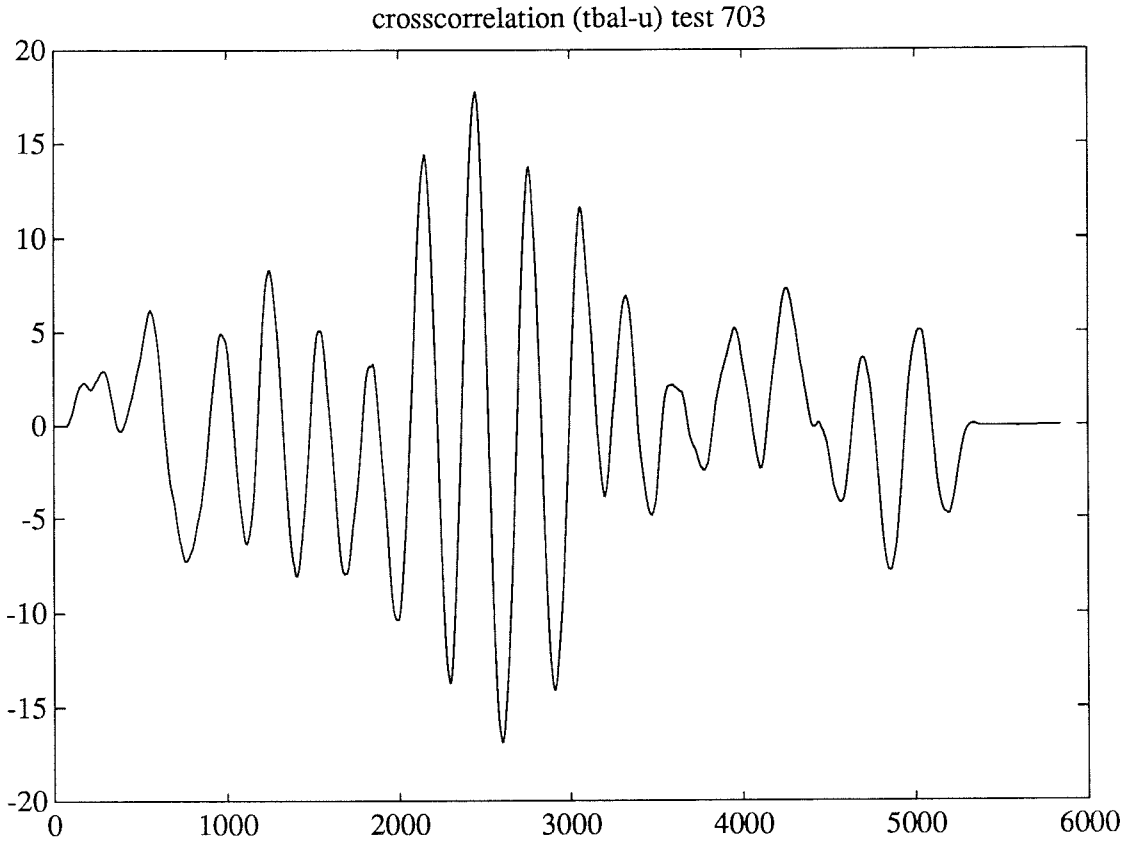

test403 a=0.14 b=0.20
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
500
1000
1500
2000
2500
3000

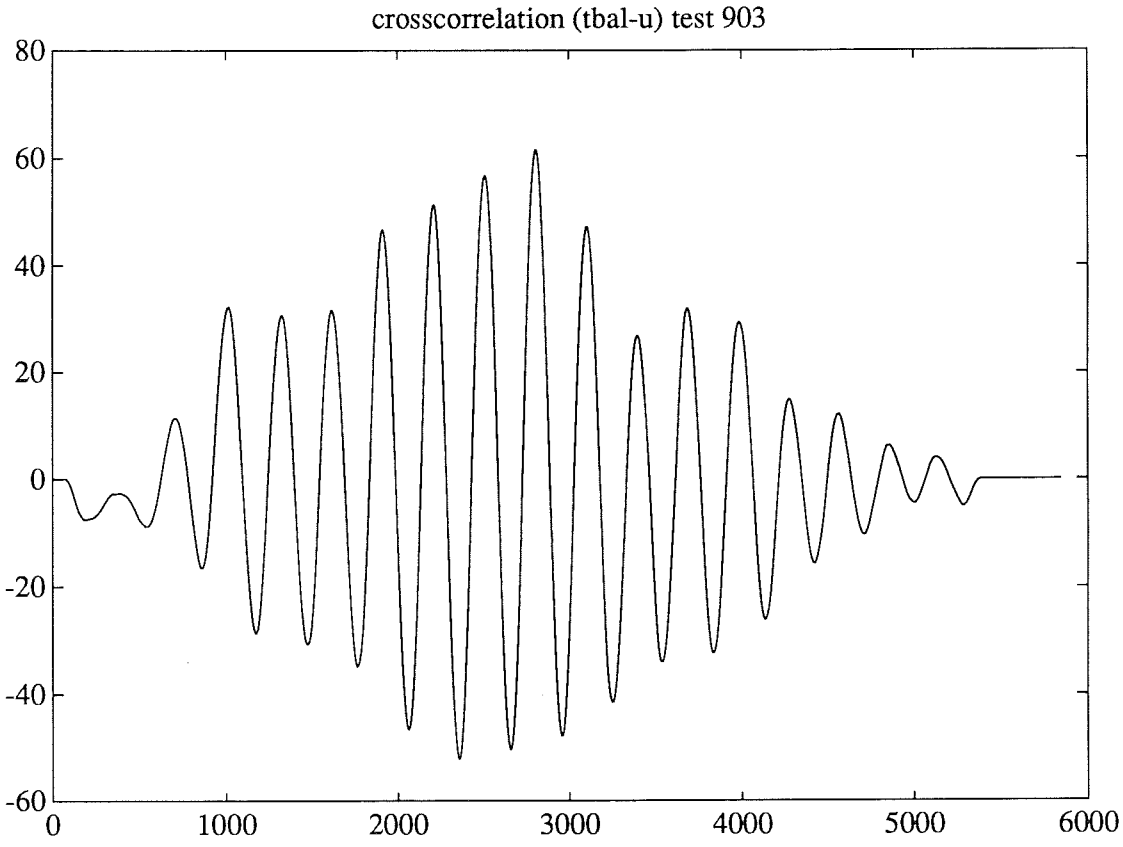

test703 a=0.14 b=0.20
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
500
1000
1500
2000
2500
3000

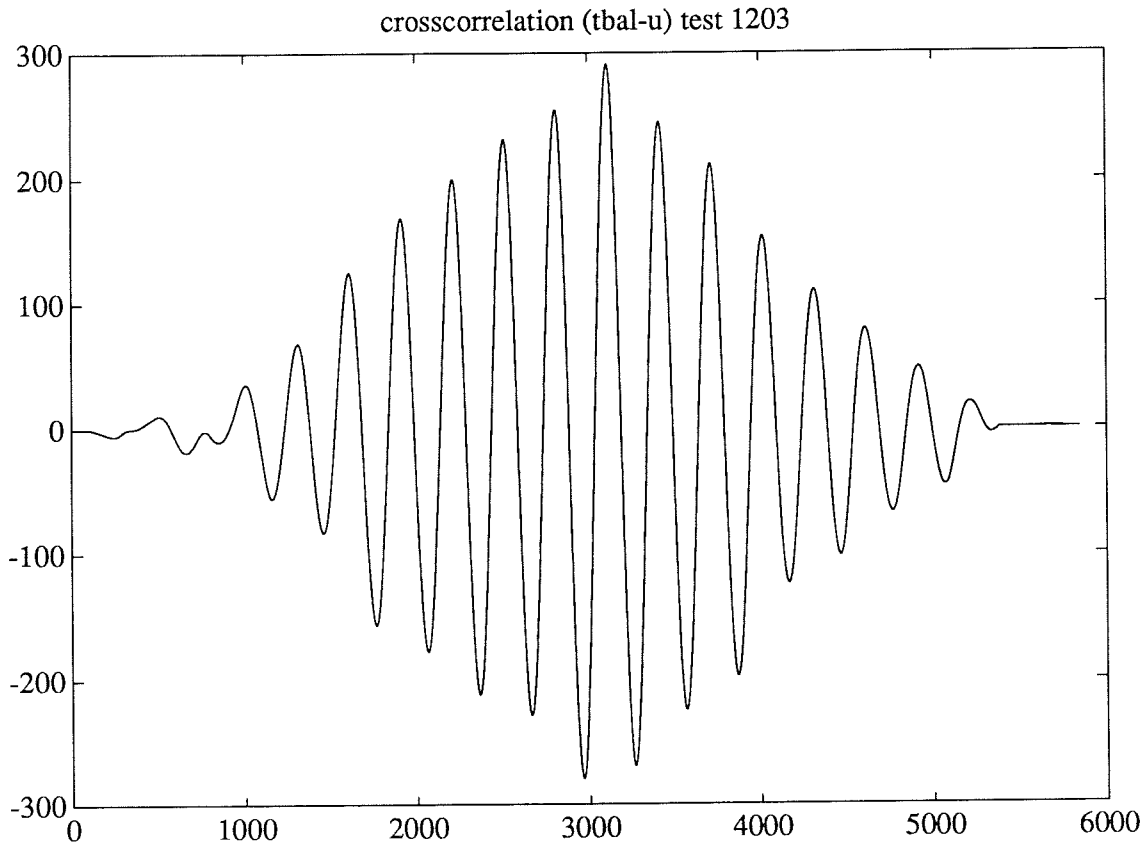

test903 a=0.15 b=0.19
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
500
1000
1500
2000
2500
3000

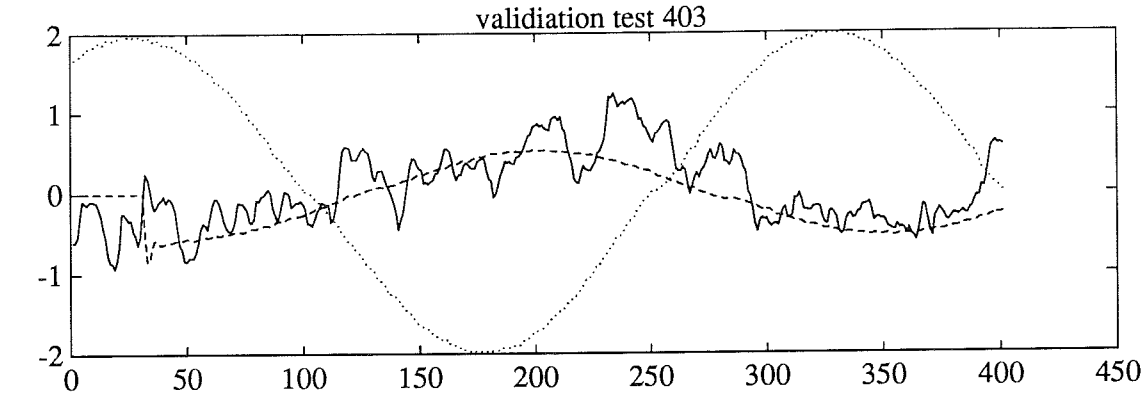

test1203 a=0.17 b=0.18
3
2
1
0
-1
-2
-3
-4
-5
0
500
1000
1500
2000
2500
3000

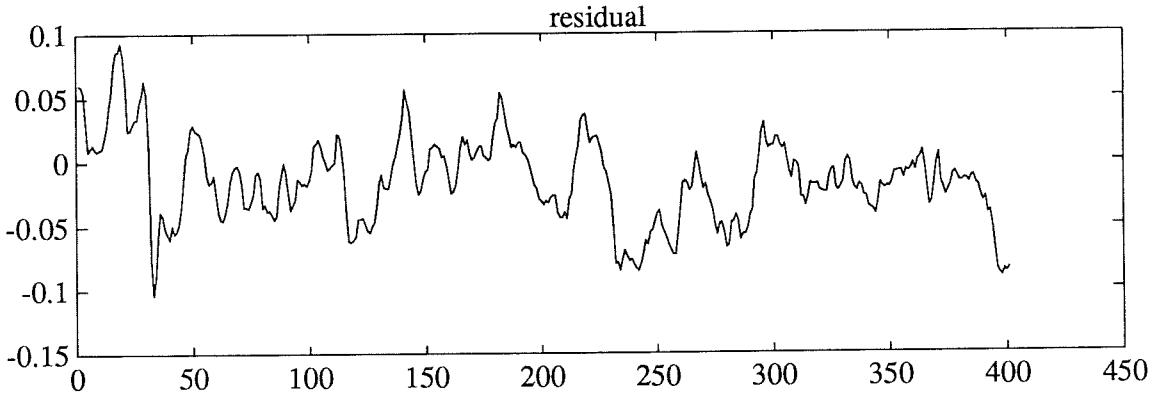

crosscorrelation (tbal-u) test 403
80
60
40
20
0
-20
-40
-60
-80
0
1000
2000
3000
4000
5000
6000

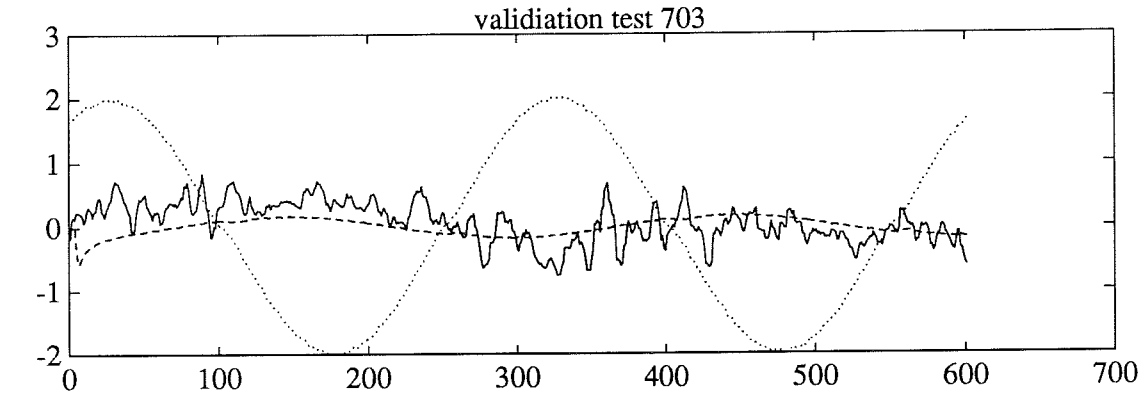

crosscorrelation (tbal-u) test 703
20
15
10
5
0
-5
-10
-15
-20
0
1000
2000
3000
4000
5000
6000

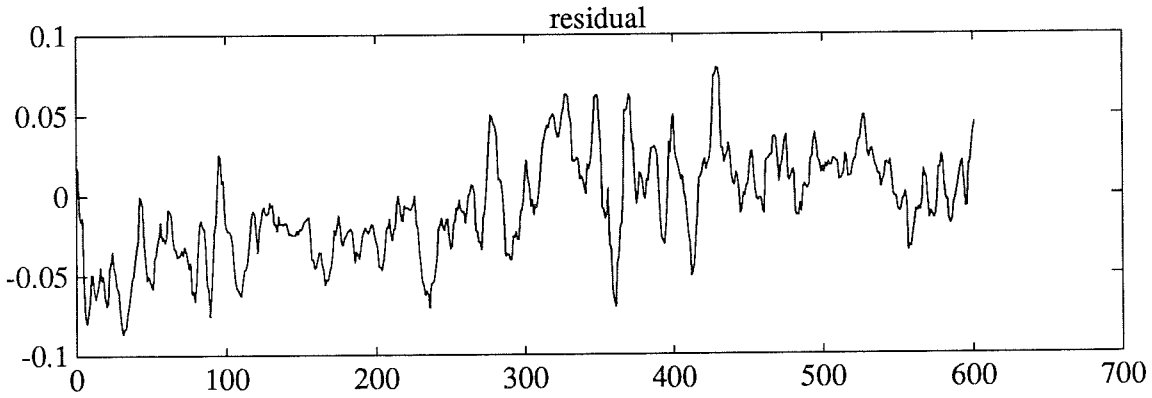

crosscorrelation (tbal-u) test 903
80
60
40
20
0
-20
-40
-60
0
1000
2000
3000
4000
5000
6000

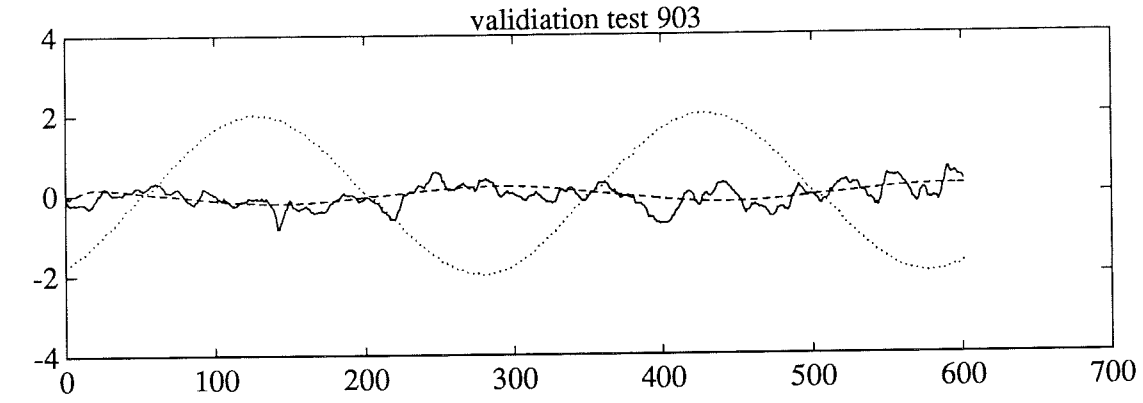

crosscorrelation (tbal-u) test 1203
300
200
100
0
-100
-200
-300
0
1000
2000
3000
4000
5000
6000

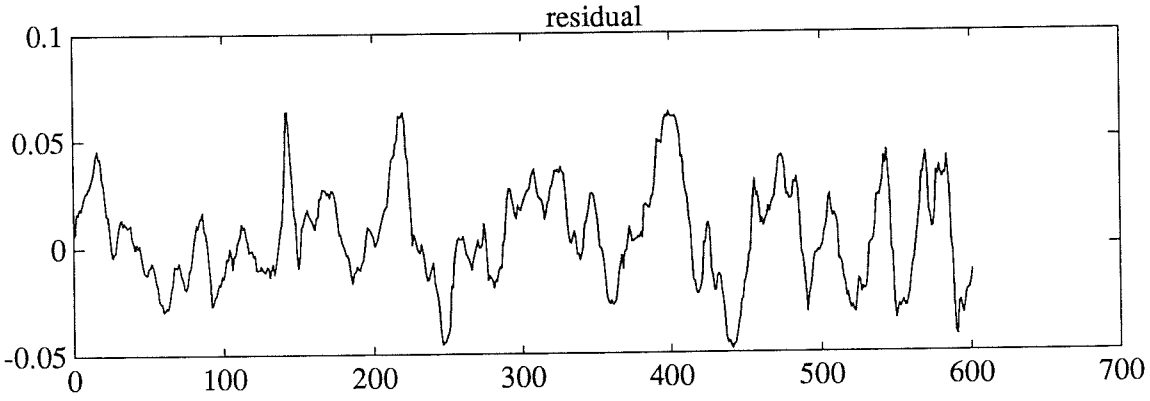

validiation test 403
2
1
0
-1
-2
0
50
100
150
200
250
300
350
400
450

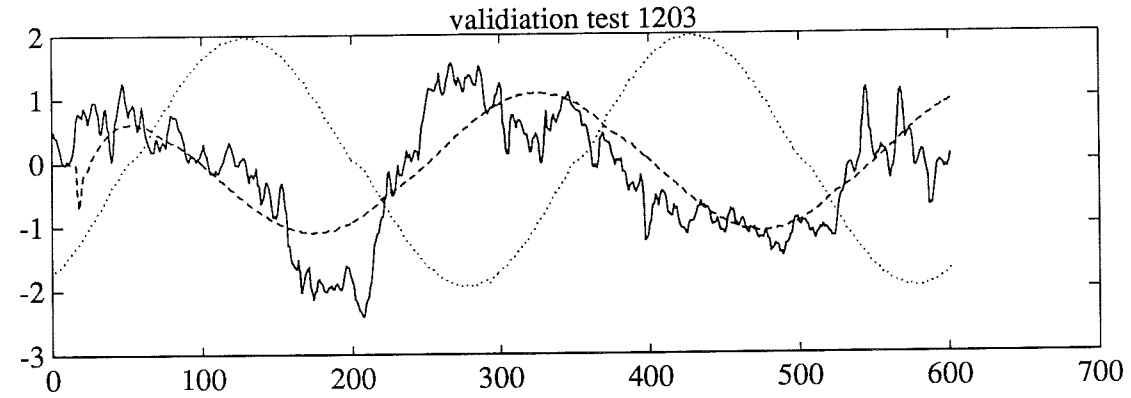

residual
0.1
0.05
0
-0.05
-0.1
-0.15
0
50
100
150
200
250
300
350
400
450

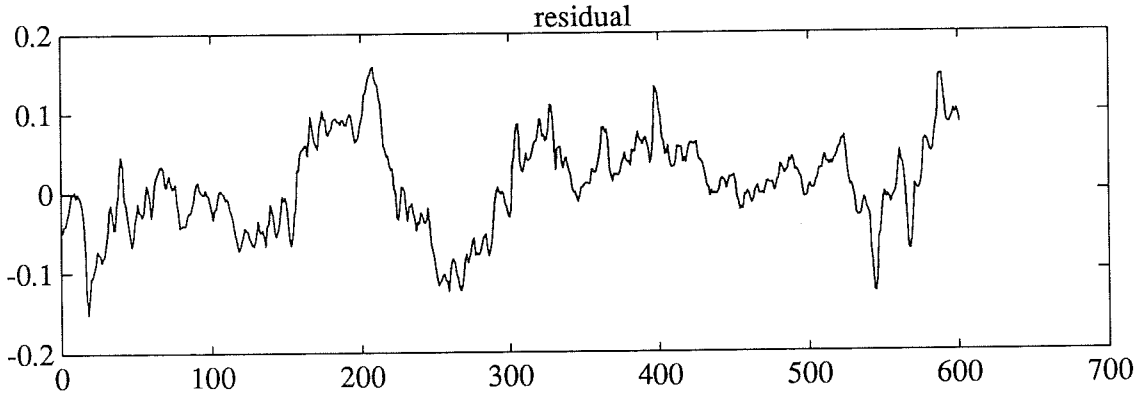

validiation test 703
3
2
1
0
-1
-2
0
100
200
300
400
500
600
700

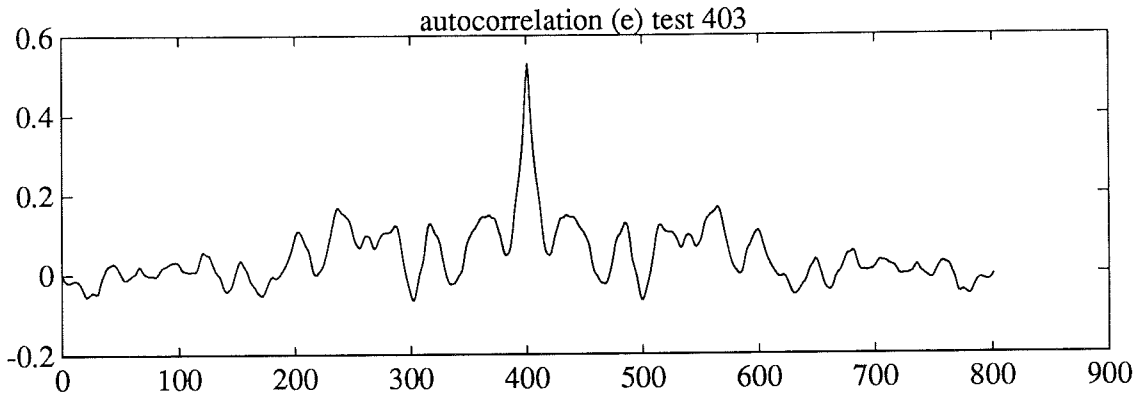

residual
0.1
0.05
0
-0.05
-0.1
0
100
200
300
400
500
600
700

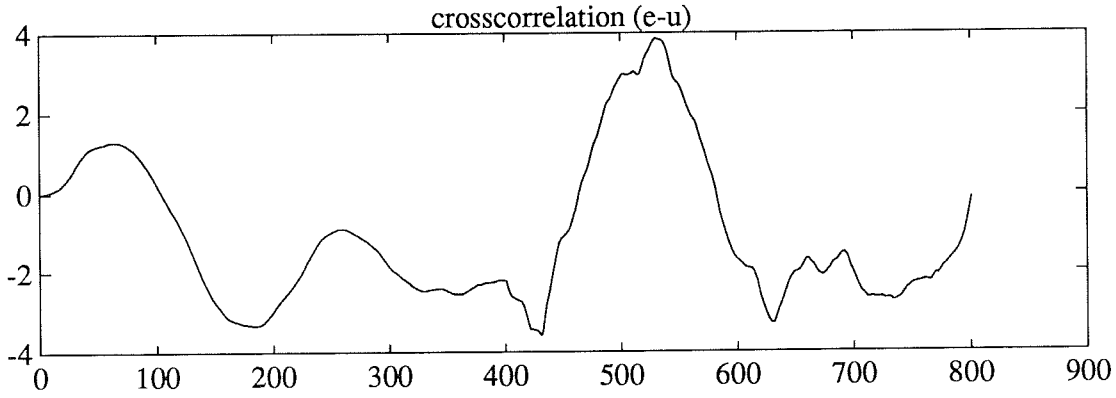

validiation test 903
4
2
0
-2
-4
0
100
200
300
400
500
600
700

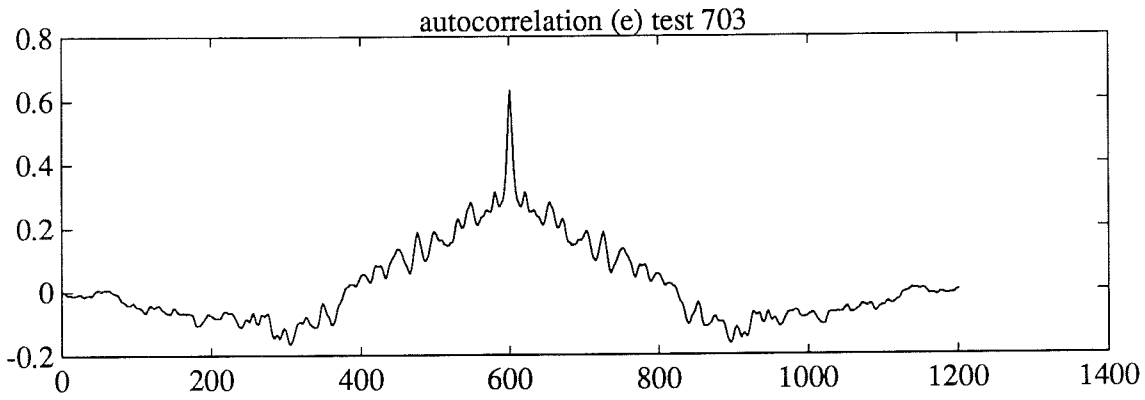

residual
0.1
0.05
0
-0.05
0
100
200
300
400
500
600
700

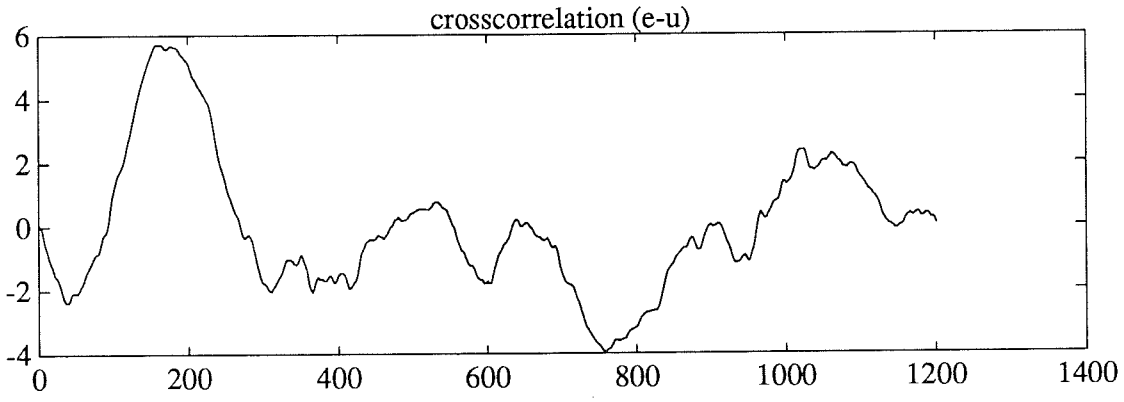

validiation test 1203
2
1
0
-1
-2
-3
0
100
200
300
400
500
600
700

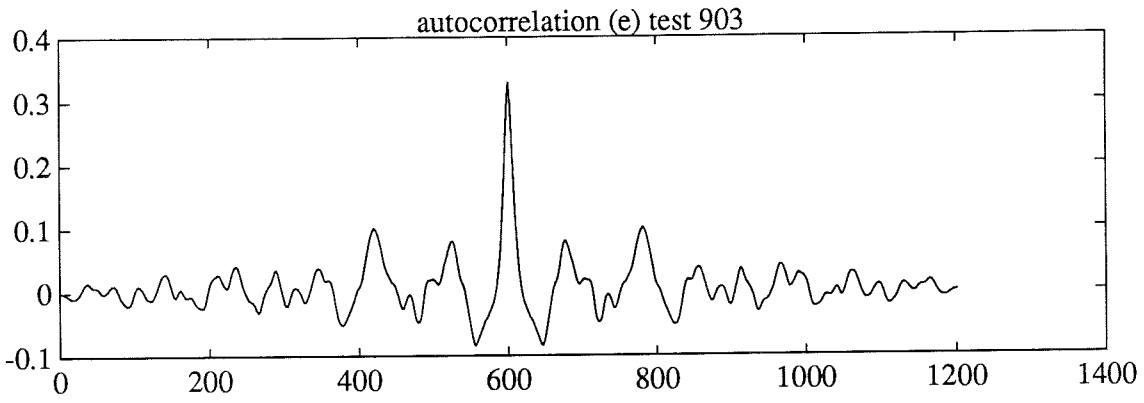

residual
0.2
0.1
0
-0.1
-0.2
0
100
200
300
400
500
600
700

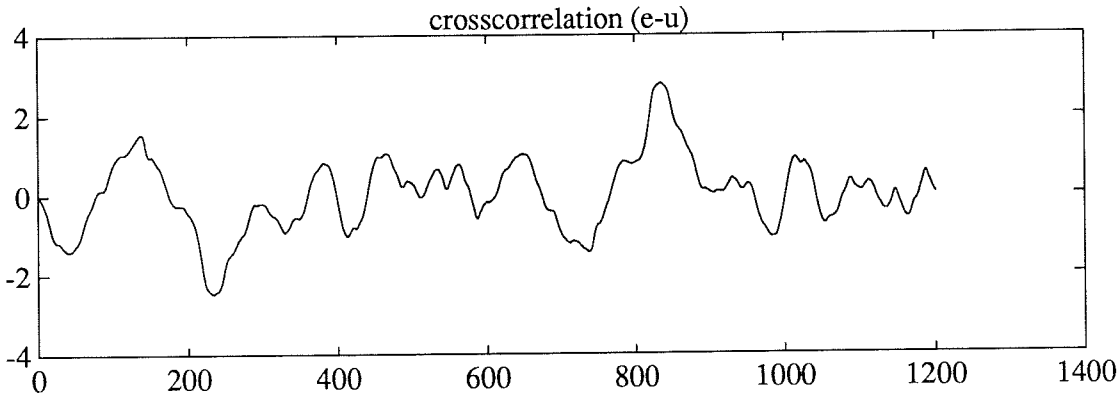

autocorrelation (e) test 403
0.6
0.4
0.2
0
-0.2
0
100
200
300
400
500
600
700
800
900

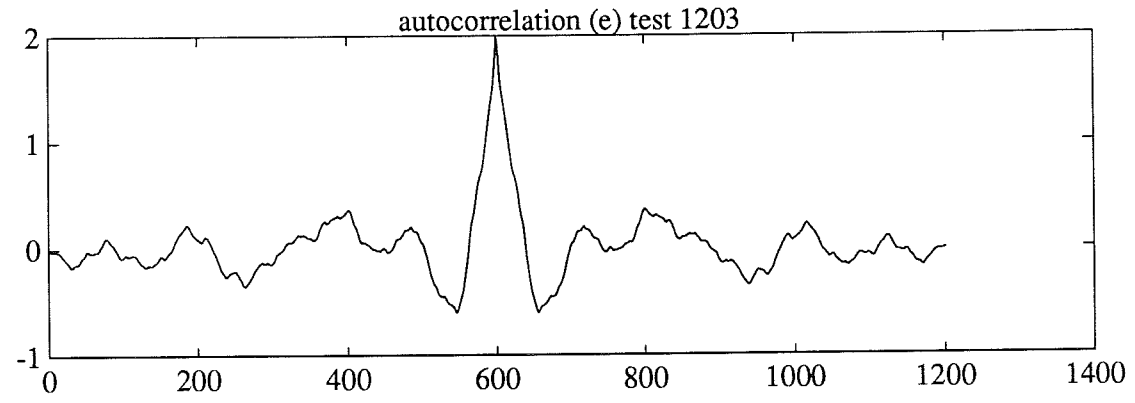

crosscorrelation (e-u)
4
2
0
-2
-4
0
100
200
300
400
500
600
700
800
900

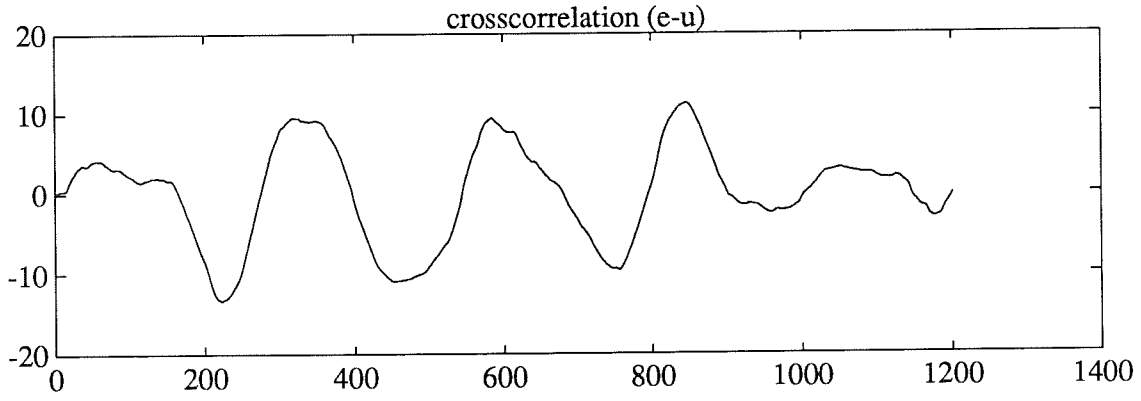

autocorrelation (e) test 703
0.8
0.6
0.4
0.2
0
-0.2
0
200
400
600
800
1000
1200
1400

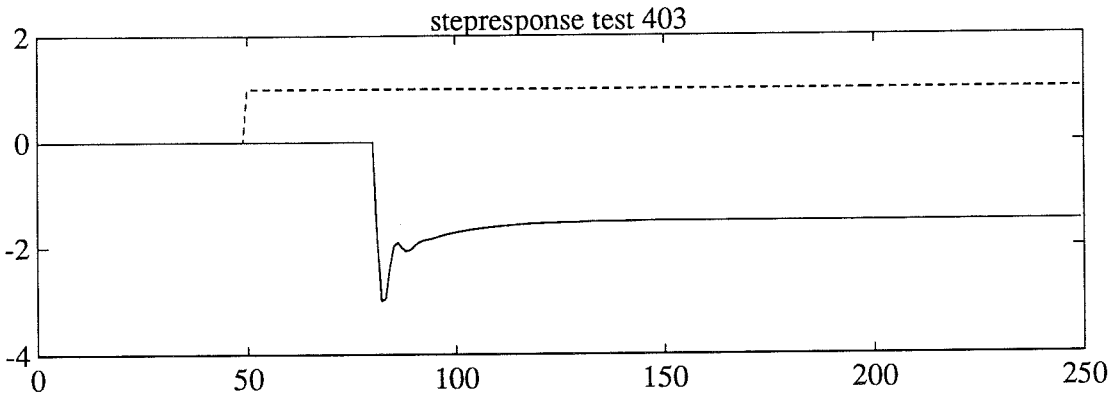

crosscorrelation (e-u)
6
4
2
0
-2
-4
0
200
400
600
800
1000
1200
1400

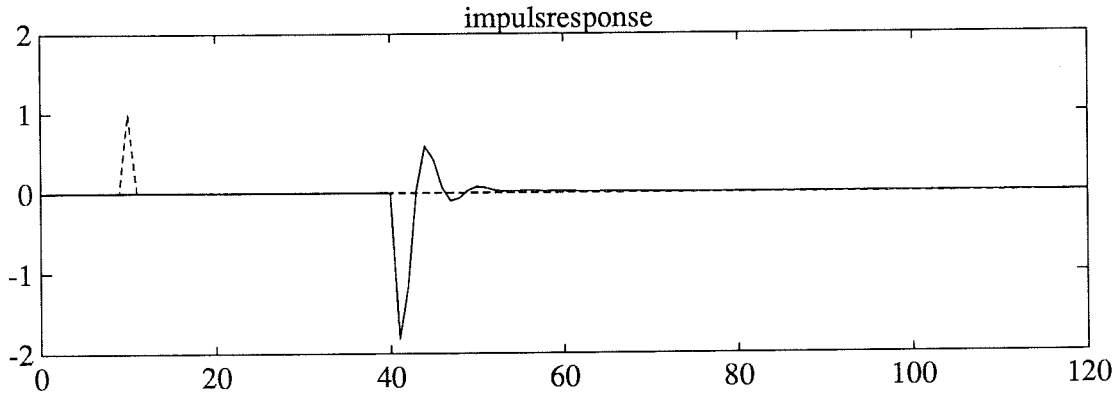

autocorrelation (e) test 903
0.4
0.3
0.2
0.1
0
-0.1
0
200
400
600
800
1000
1200
1400

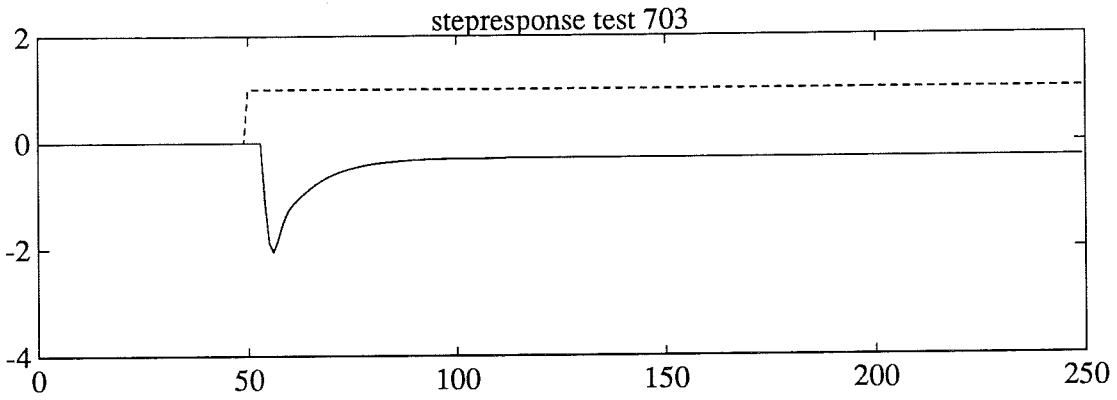

crosscorrelation (e-u)
4
2
0
-2
-4
0
200
400
600
800
1000
1200
1400

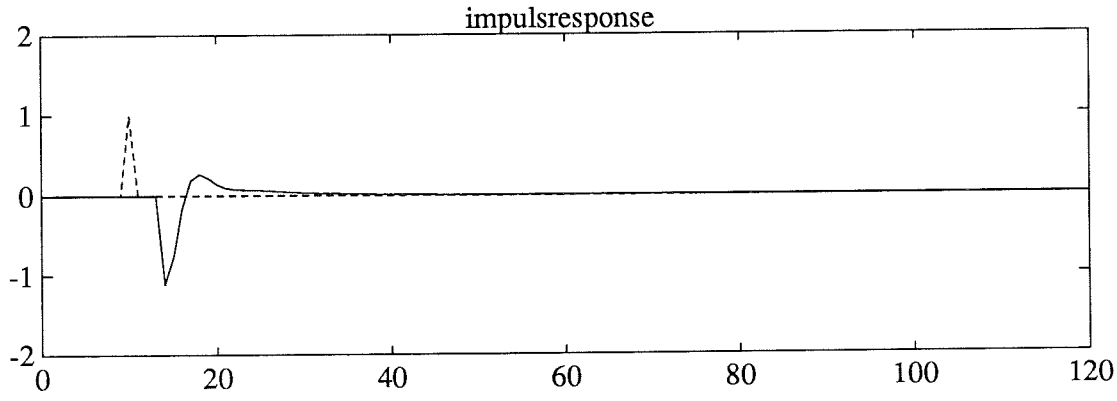

autocorrelation (e) test 1203
2
1
0
-1
0
200
400
600
800
1000
1200
1400

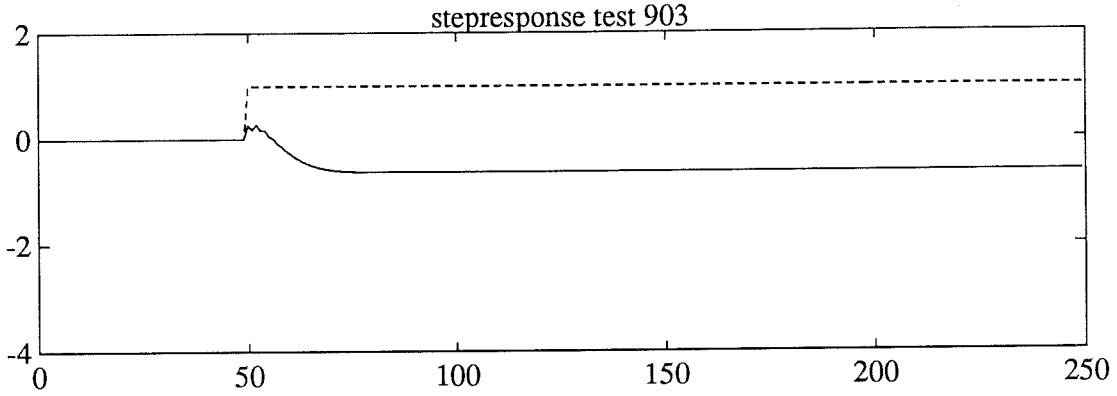

crosscorrelation (e-u)
20
10
0
-10
-20
0
200
400
600
800
1000
1200
1400

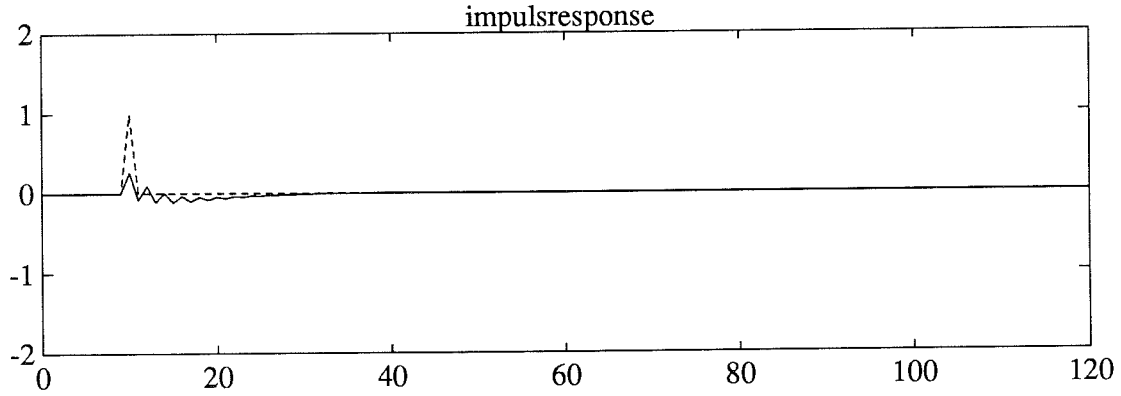

stepresponse test 403
2
0
-2
-4
0
50
100
150
200
250

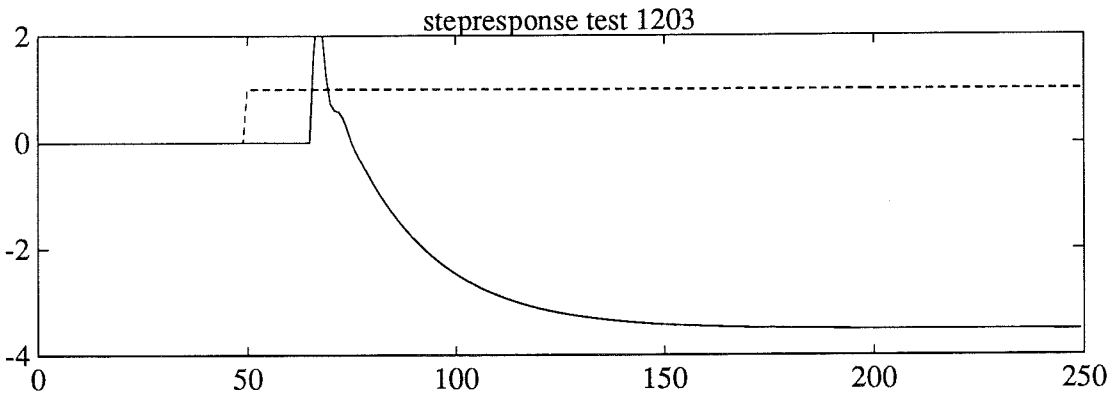

impulsresponse
2
1
0
-1
-2
0
20
40
60
80
100
120

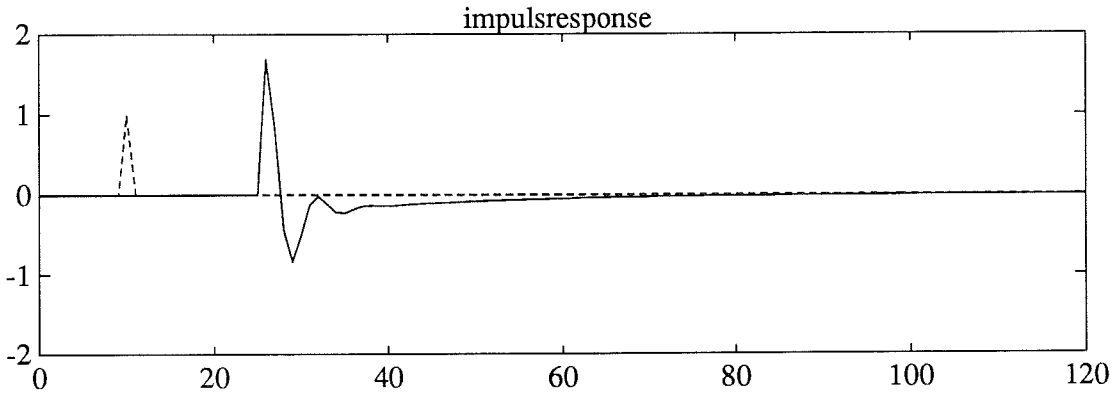

stepresponse test 703
2
0
-2
-4
0
50
100
150
200
250

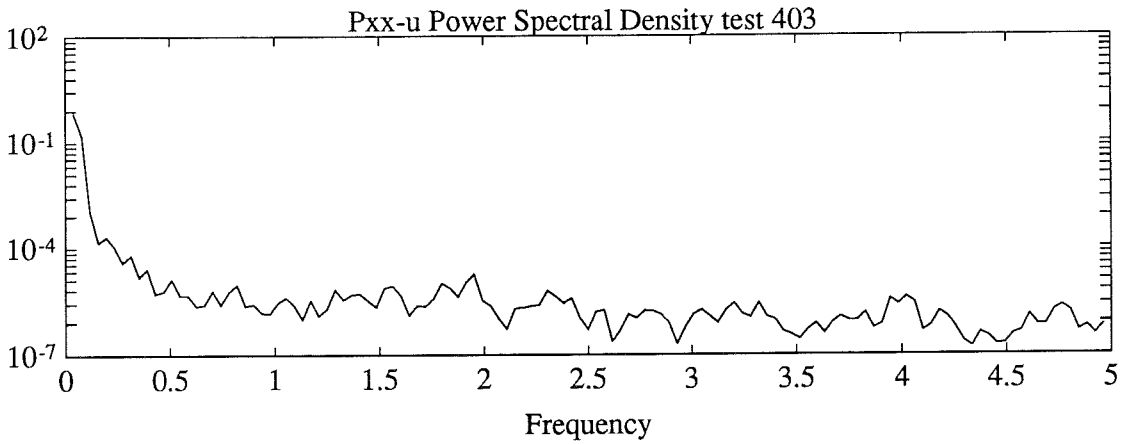

impulsresponse
2
1
0
-1
-2
0
20
40
60
80
100
120

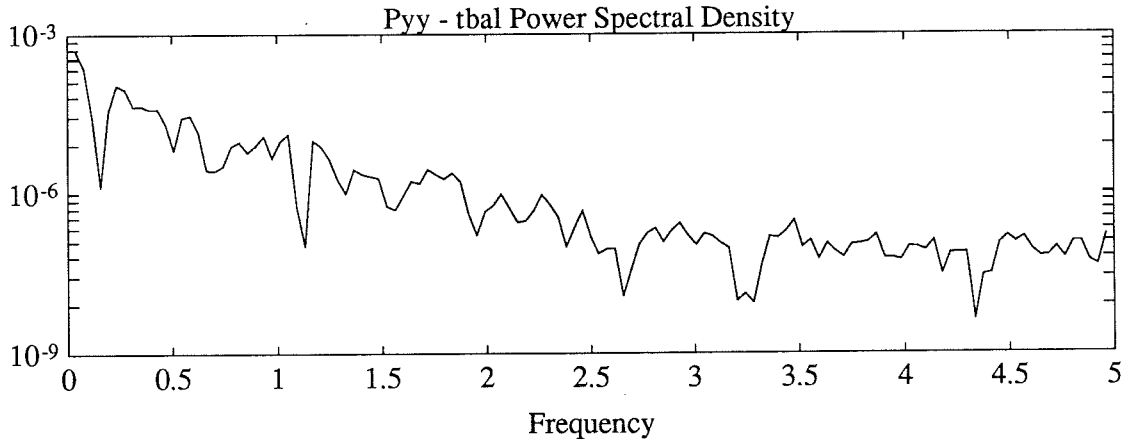

stepresponse test 903
2
0
-2
-4
0
50
100
150
200
250

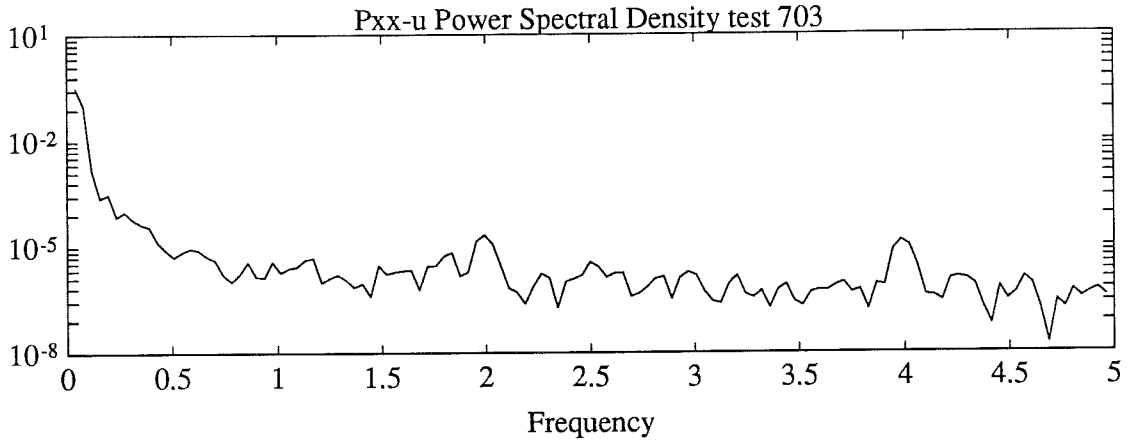

impulsresponse
2
1
0
-1
-2
0
20
40
60
80
100
120

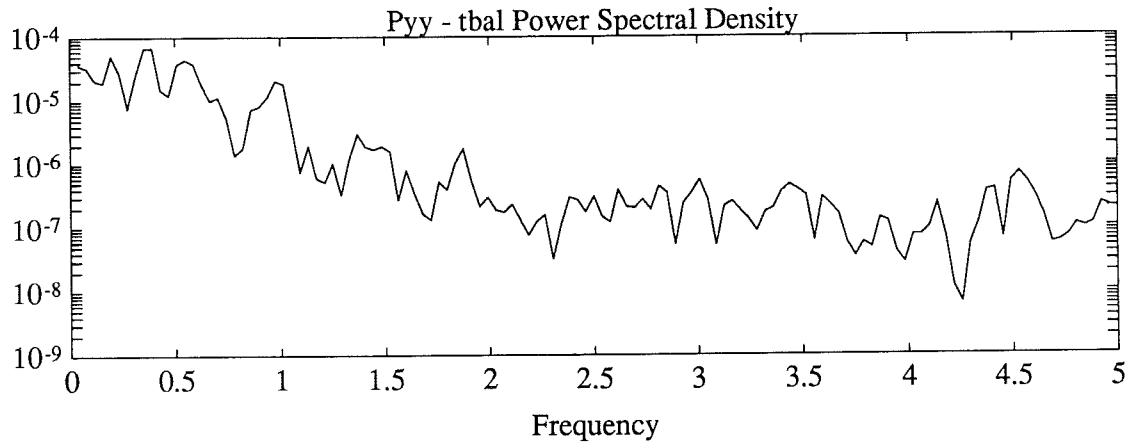

stepresponse test 1203
2
0
-2
-4
0
50
100
150
200
250

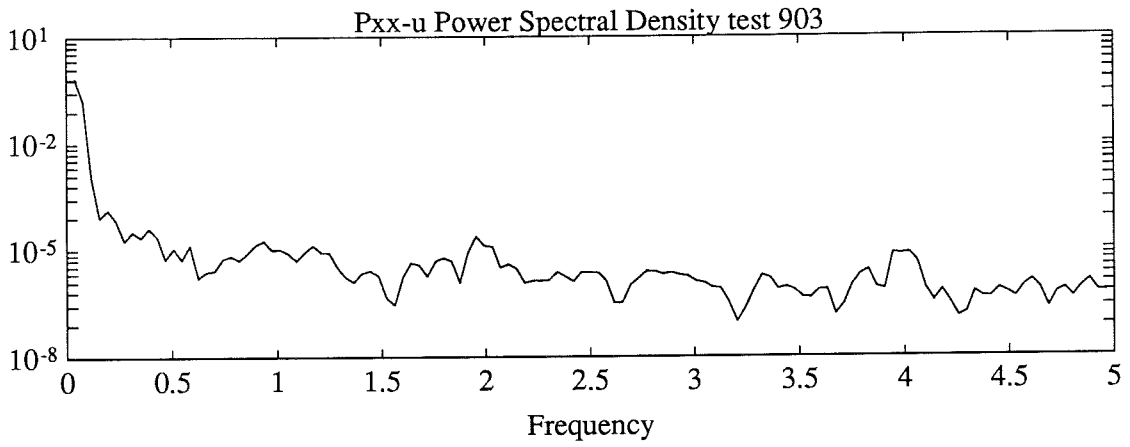

impulsresponse
2
1
0
-1
-2
0
20
40
60
80
100
120

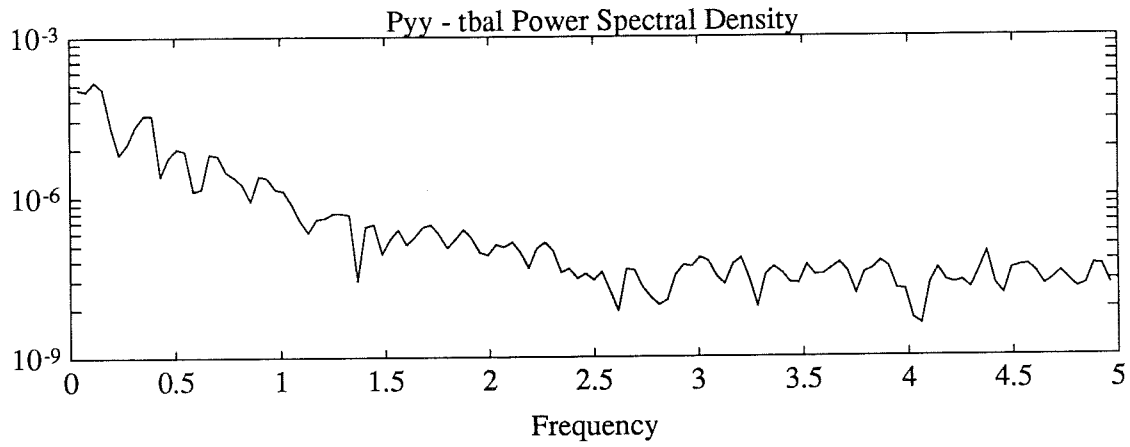

Pxx-u Power Spectral Density test 403
10^2
10^-1
10^-4
10^-7
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

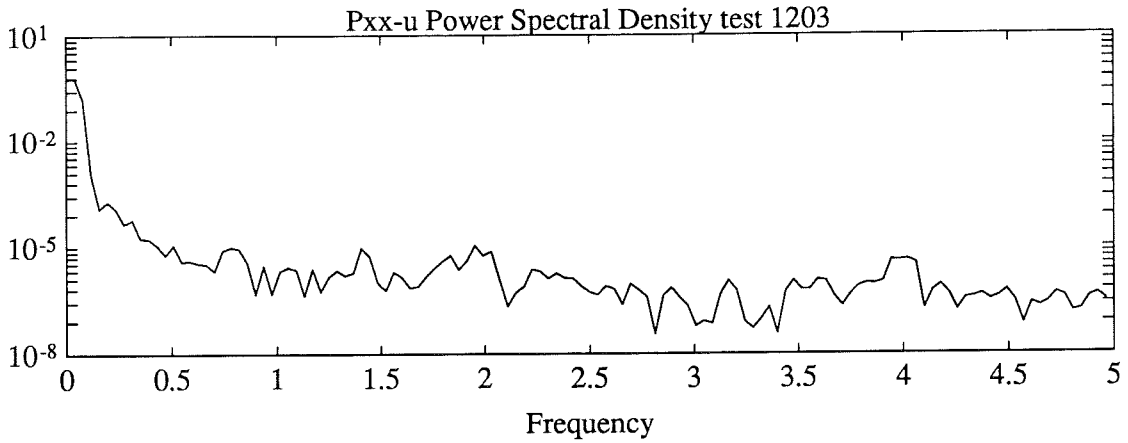

Pyy - tbal Power Spectral Density
10^-3
10^-6
10^-9
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

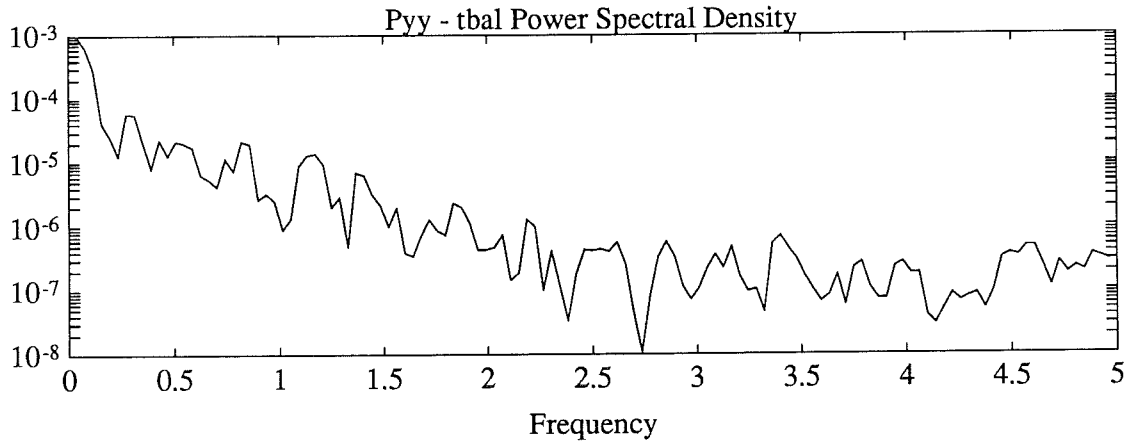

Pxx-u Power Spectral Density test 703
10^1
10^-2
10^-5
10^-8
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

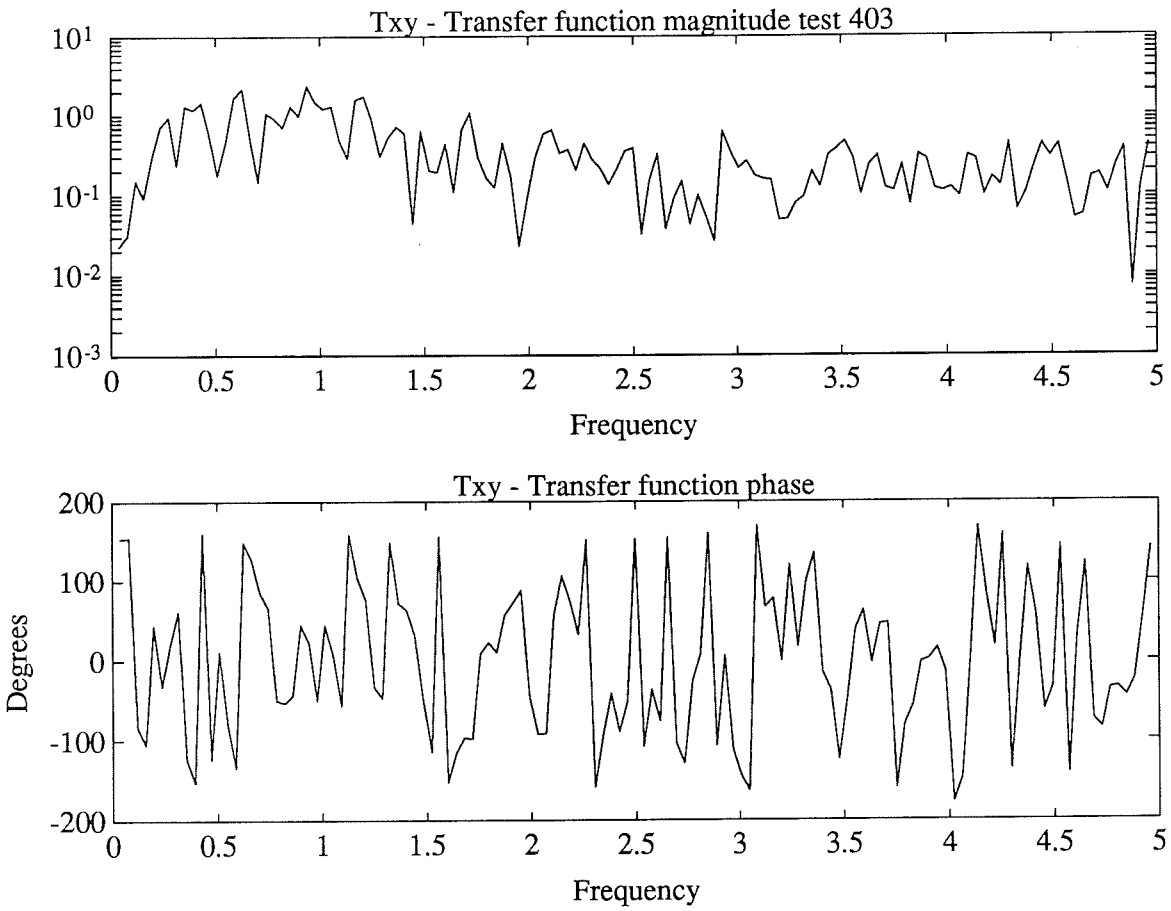

Pyy - tbal Power Spectral Density
10^-4
10^-5
10^-6
10^-7
10^-8
10^-9
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

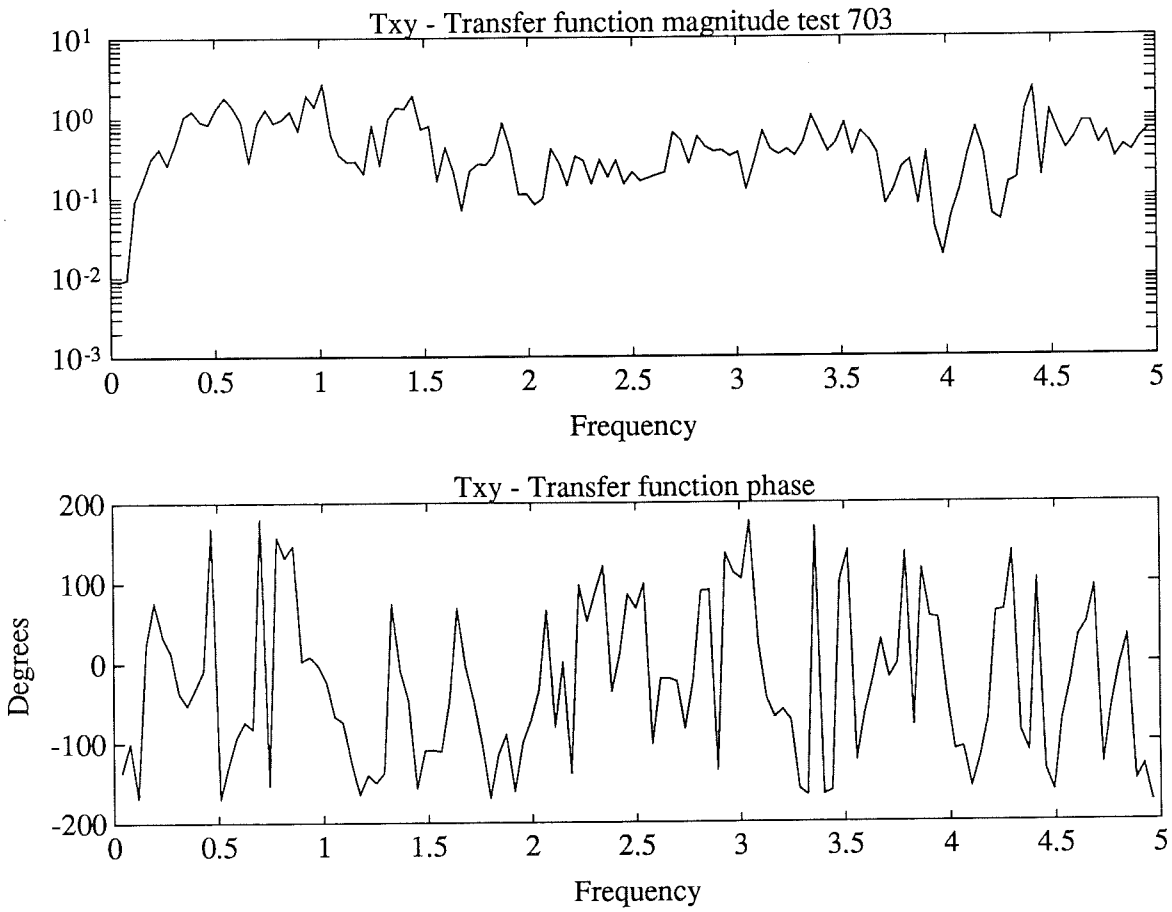

Pxx-u Power Spectral Density test 903
10^1
10^-2
10^-5
10^-8
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

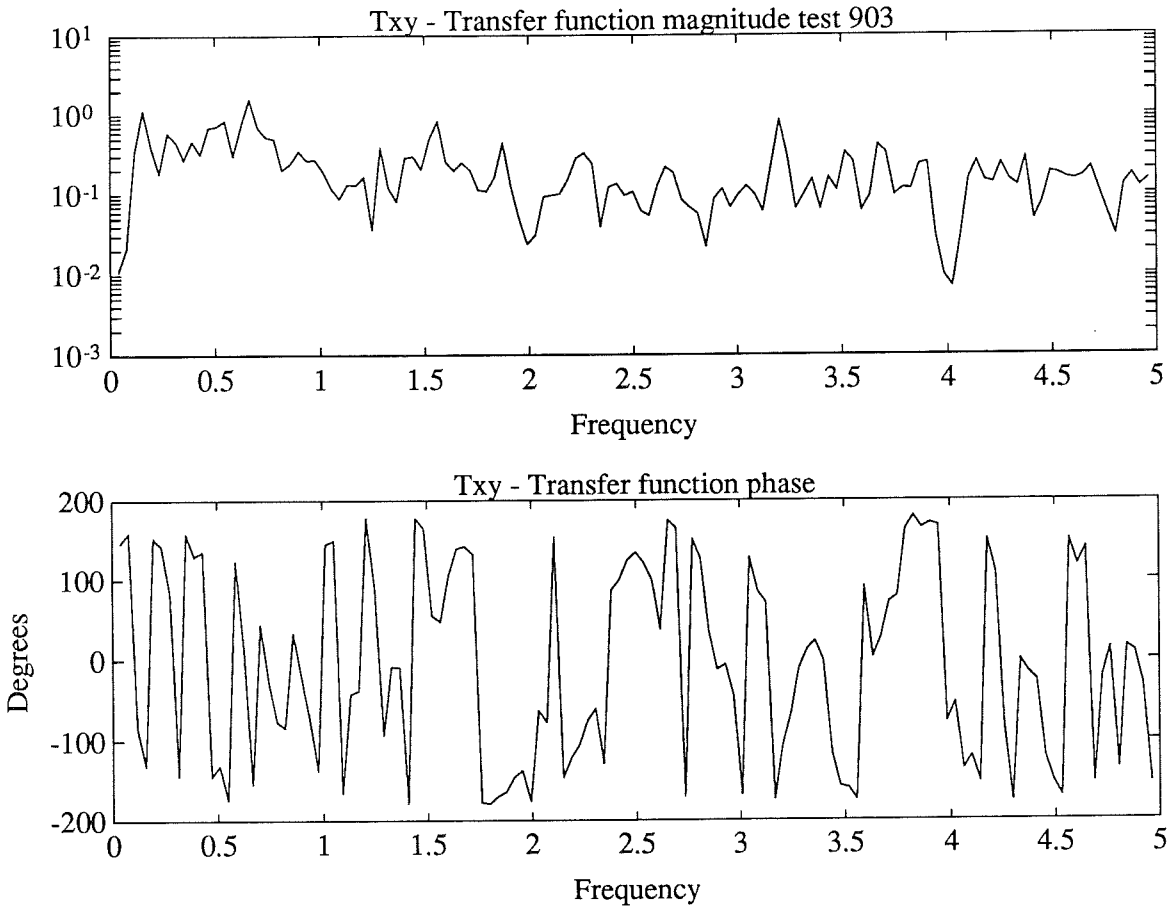

Pyy - tbal Power Spectral Density
10^-3
10^-6
10^-9
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

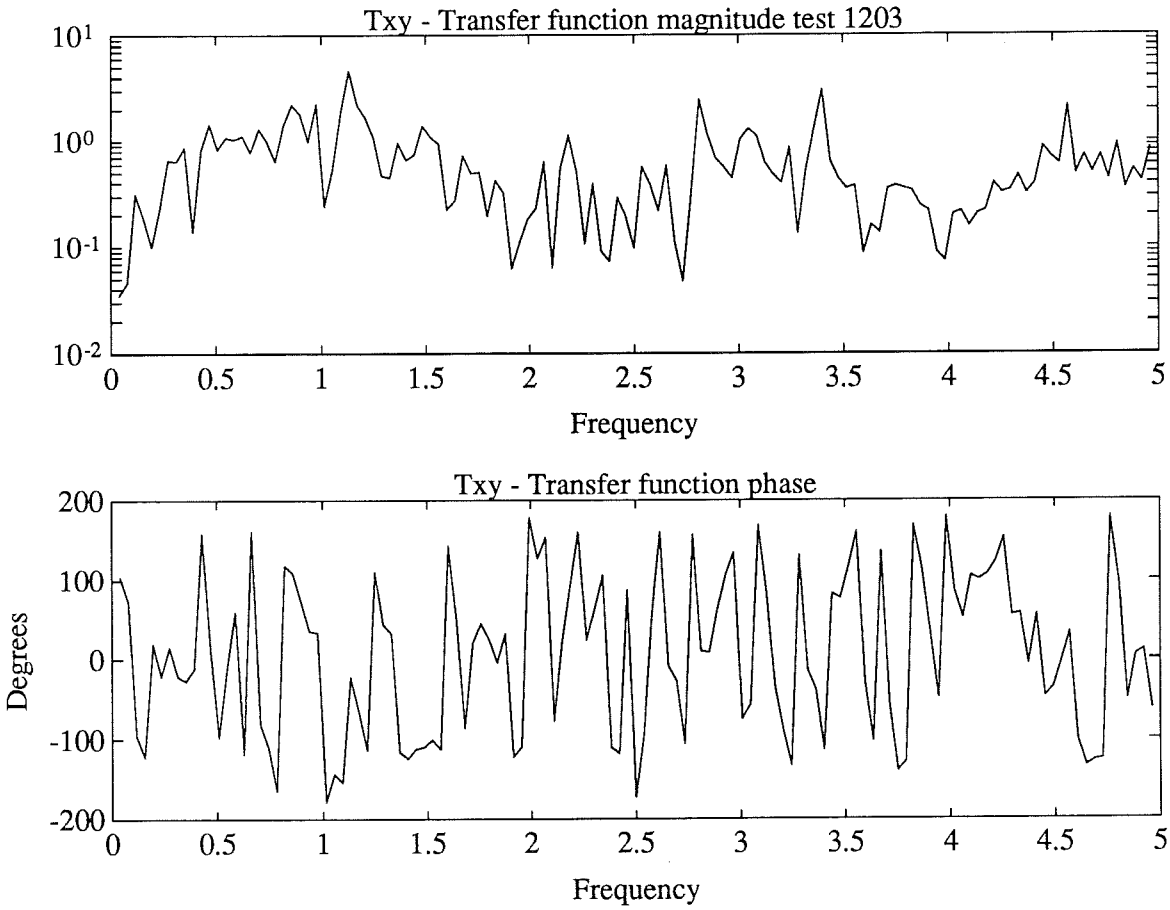

Pxx-u Power Spectral Density test 1203
10^1
10^-2
10^-5
10^-8
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

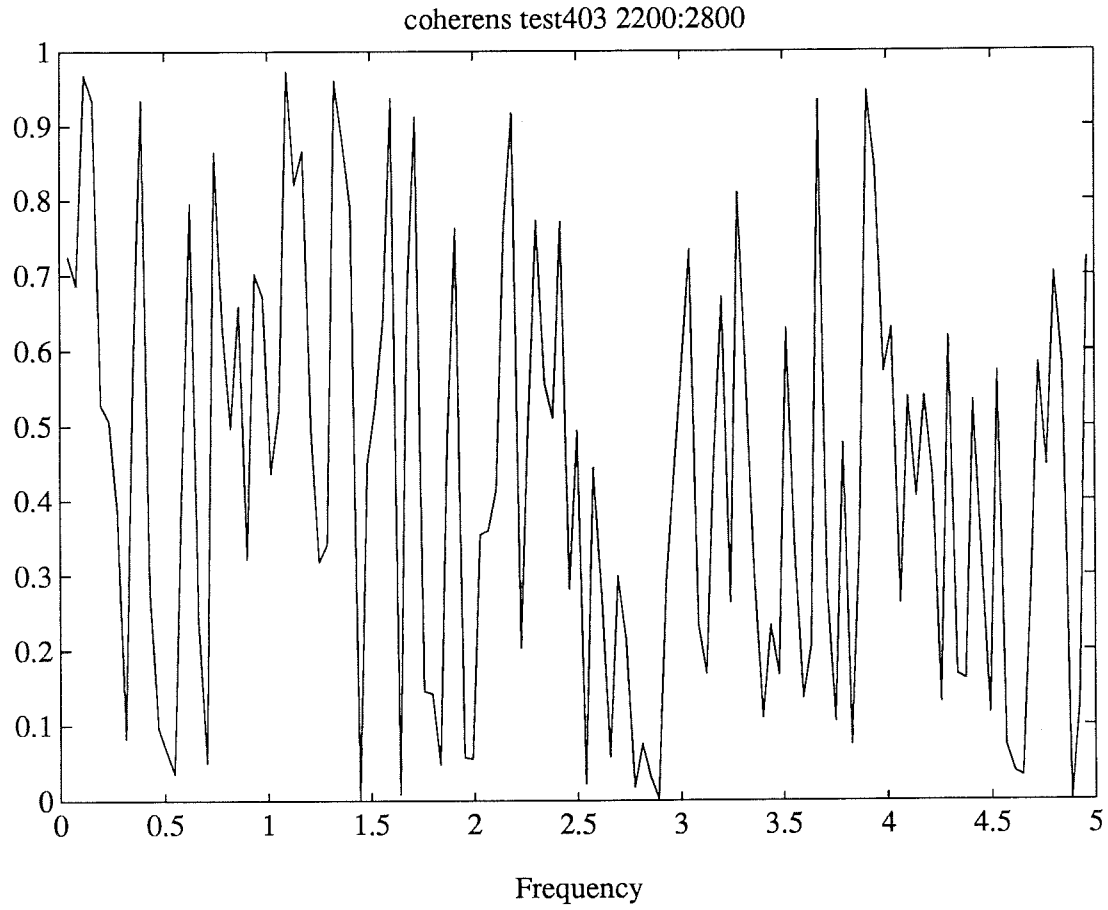

Pyy - tbal Power Spectral Density
10^-3
10^-4
10^-5
10^-6
10^-7
10^-8
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

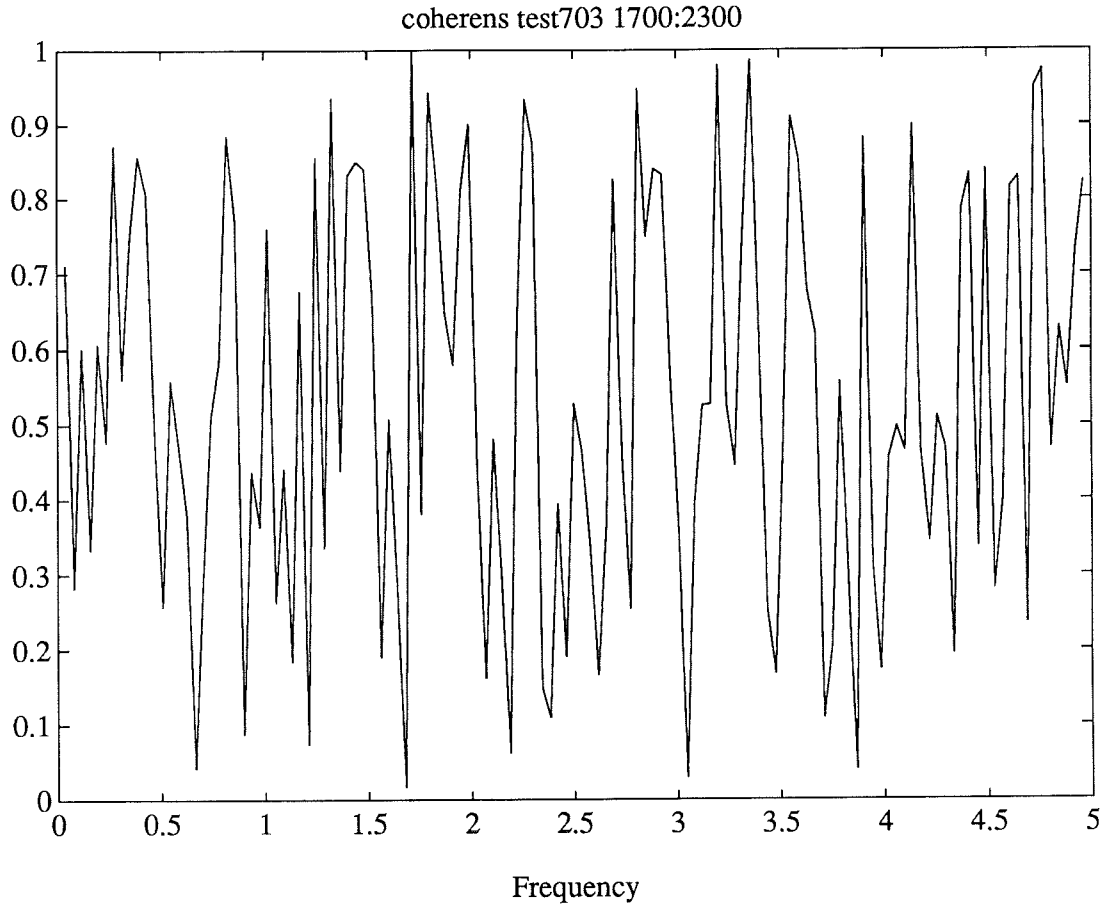

Txy - Transfer function magnitude test 403
10^1
10^0
10^-1
10^-2
10^-3
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency
Txy - Transfer function phase
200
100
0
-100
-200
Degrees
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

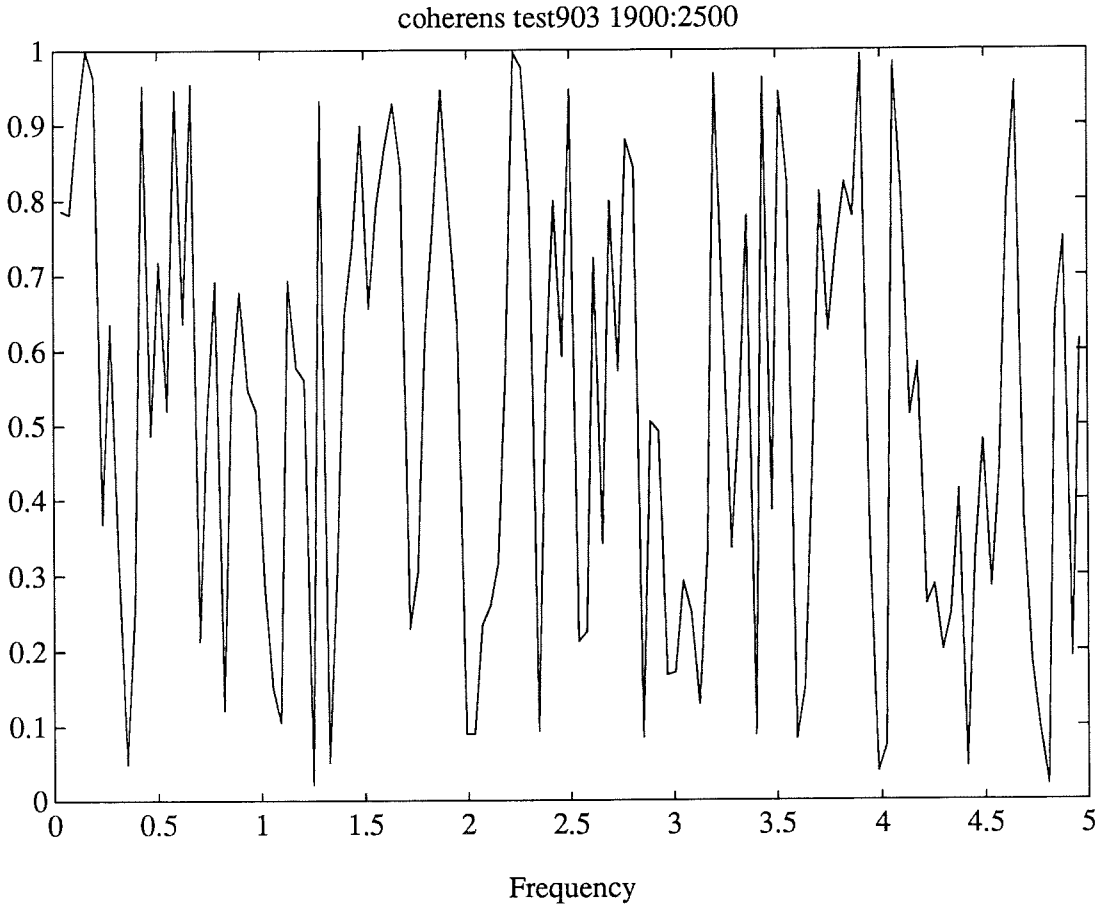

Txy - Transfer function magnitude test 703
10^1
10^0
10^-1
10^-2
10^-3
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency
Txy - Transfer function phase
200
100
0
-100
-200
Degrees
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

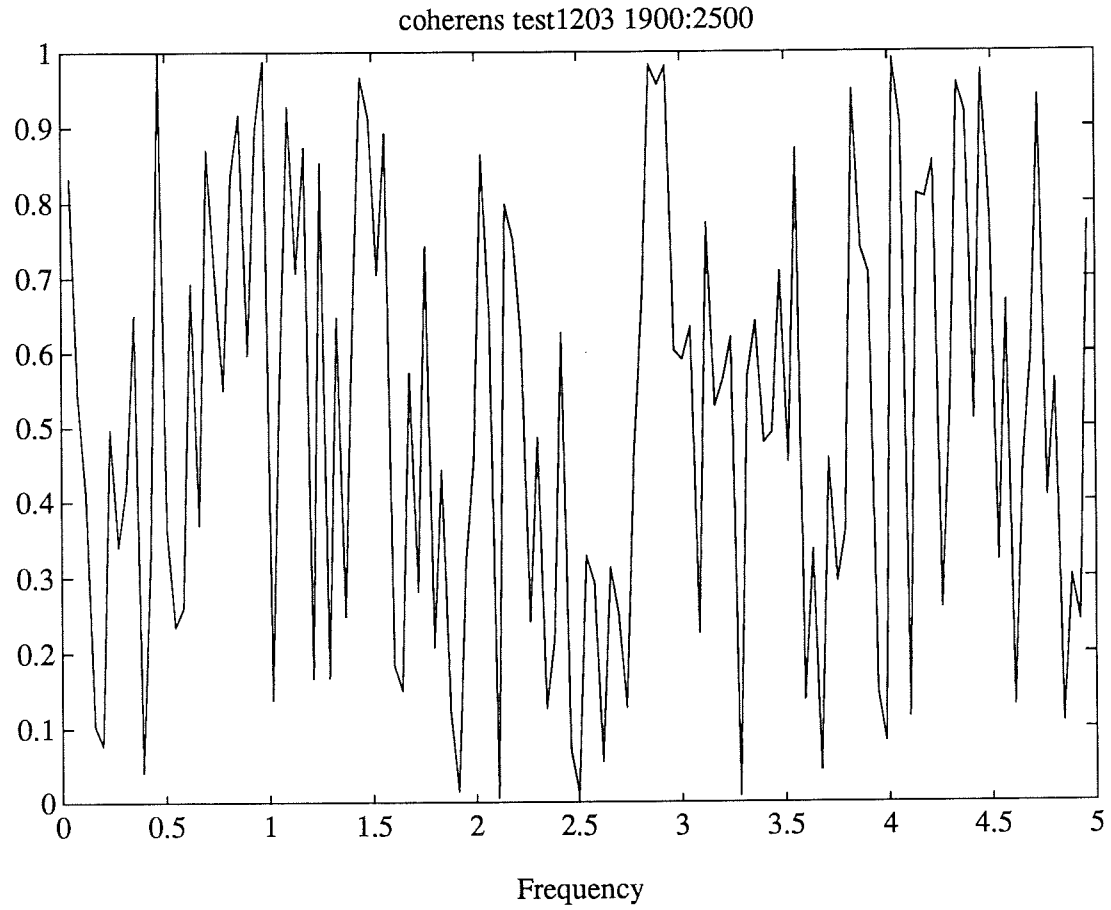

Txy - Transfer function magnitude test 903
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency
Txy - Transfer function phase
Degrees
200
100
0
-100
-200
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

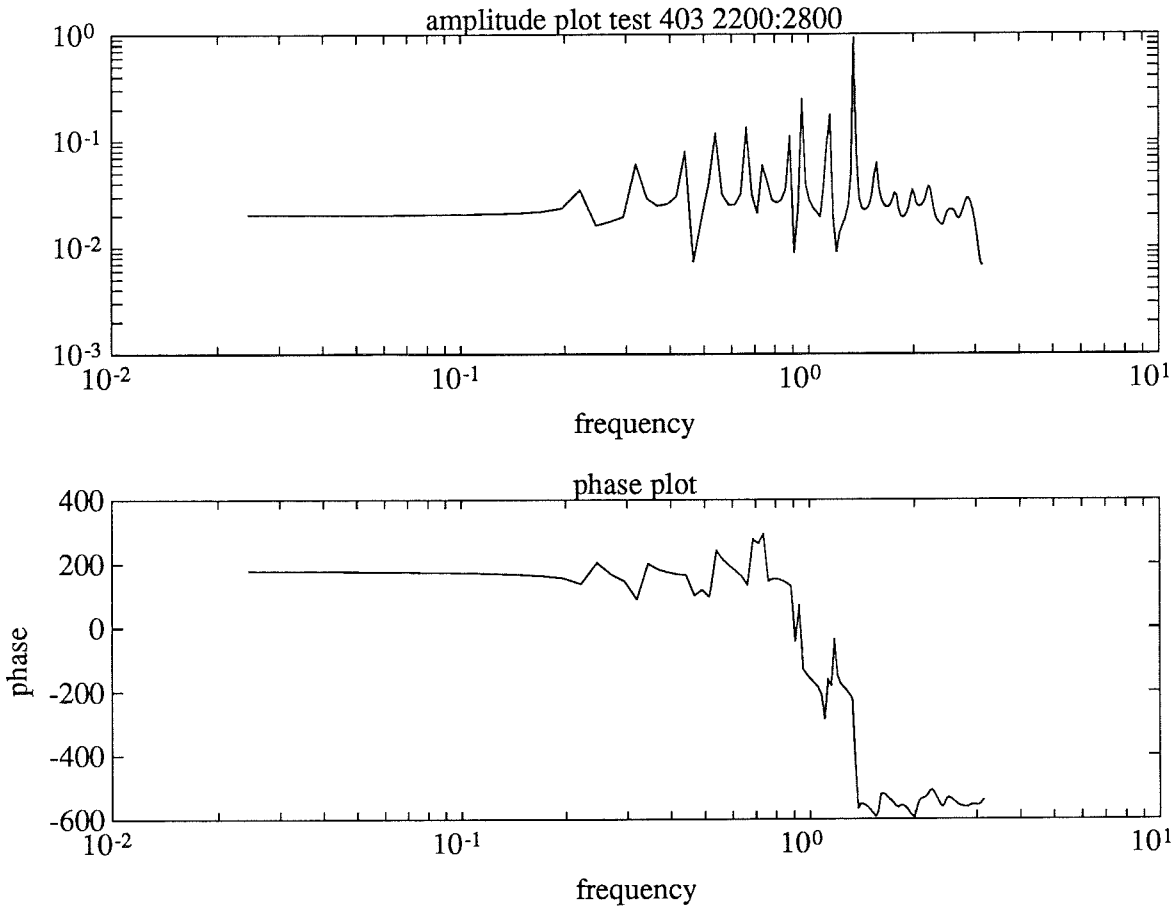

Txy - Transfer function magnitude test 1203
10^1
10^0
10^-1
10^-2
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency
Txy - Transfer function phase
200
100
0
-100
-200
Degrees
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

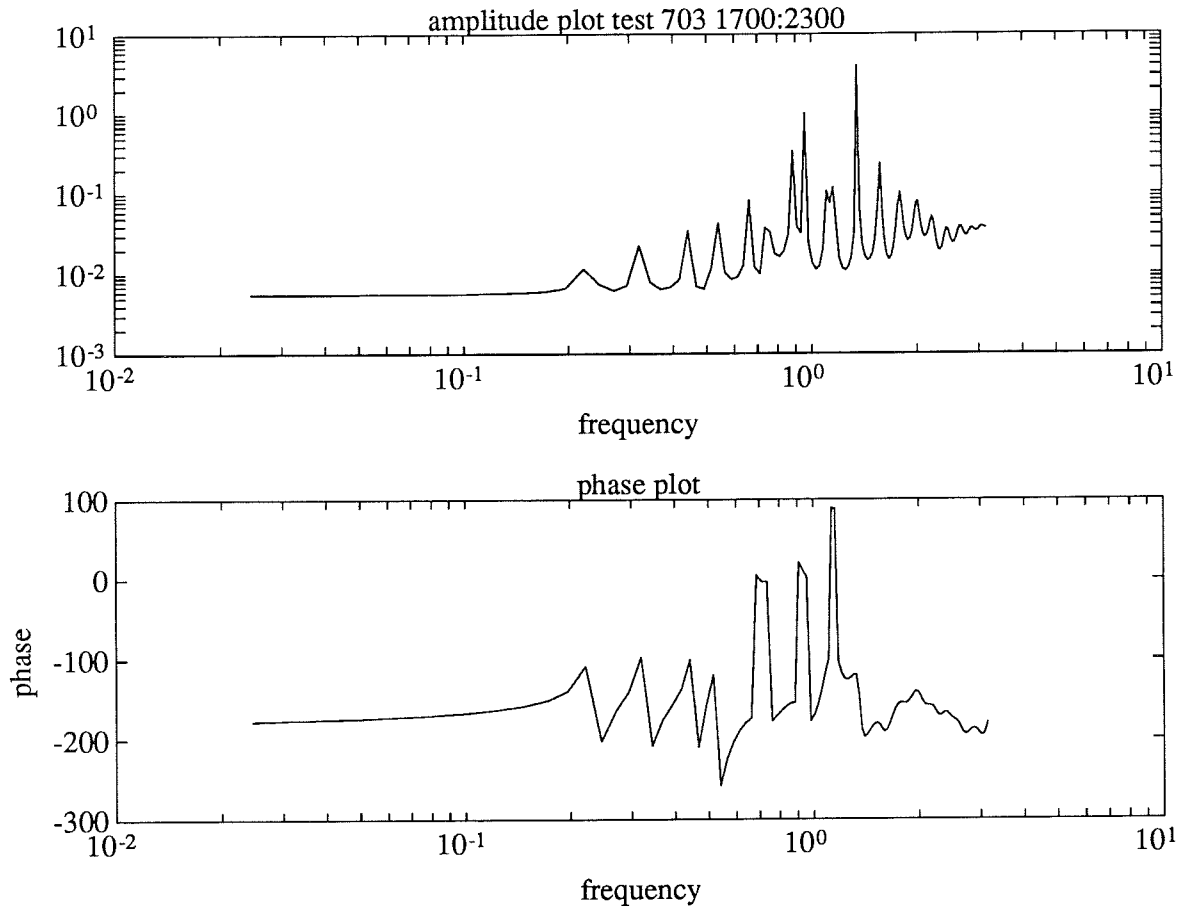

coherens test403 2200:2800
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

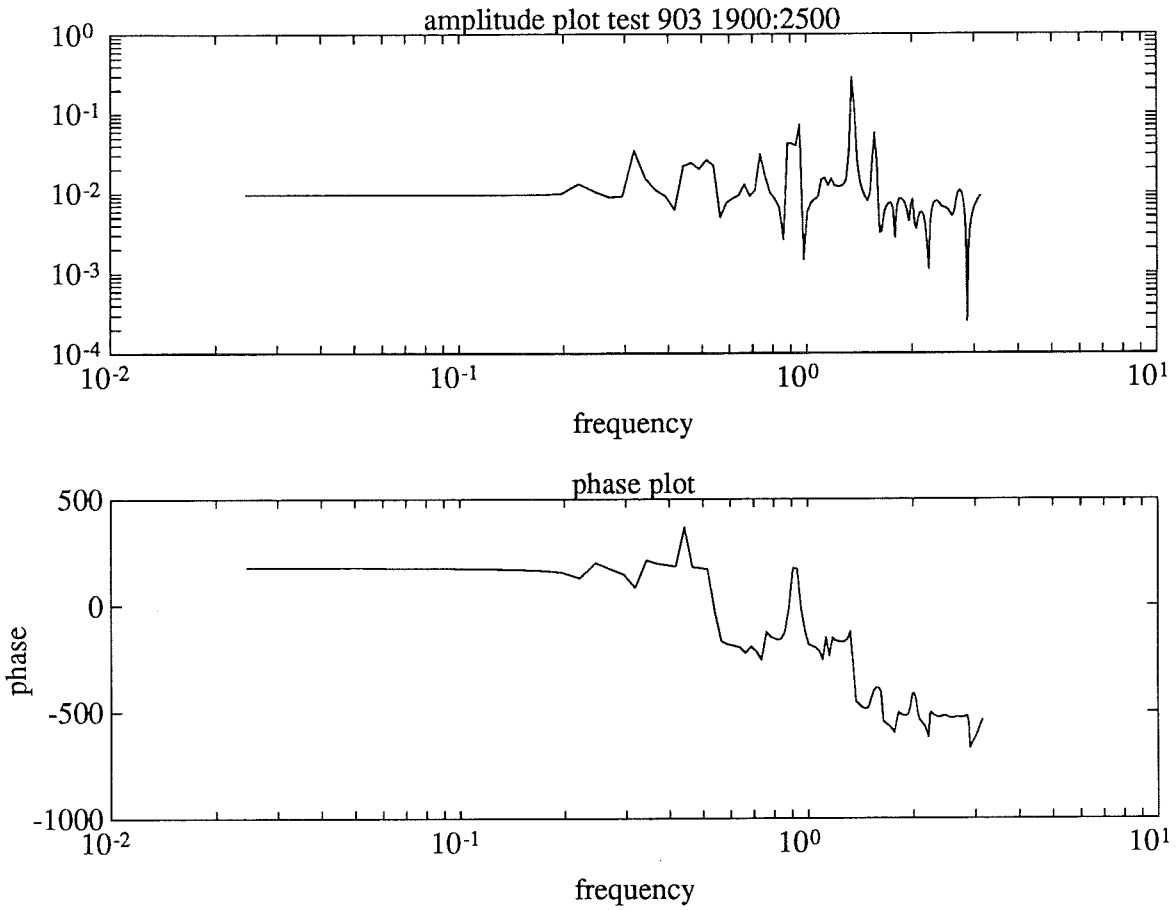

coherens test703 1700:2300
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

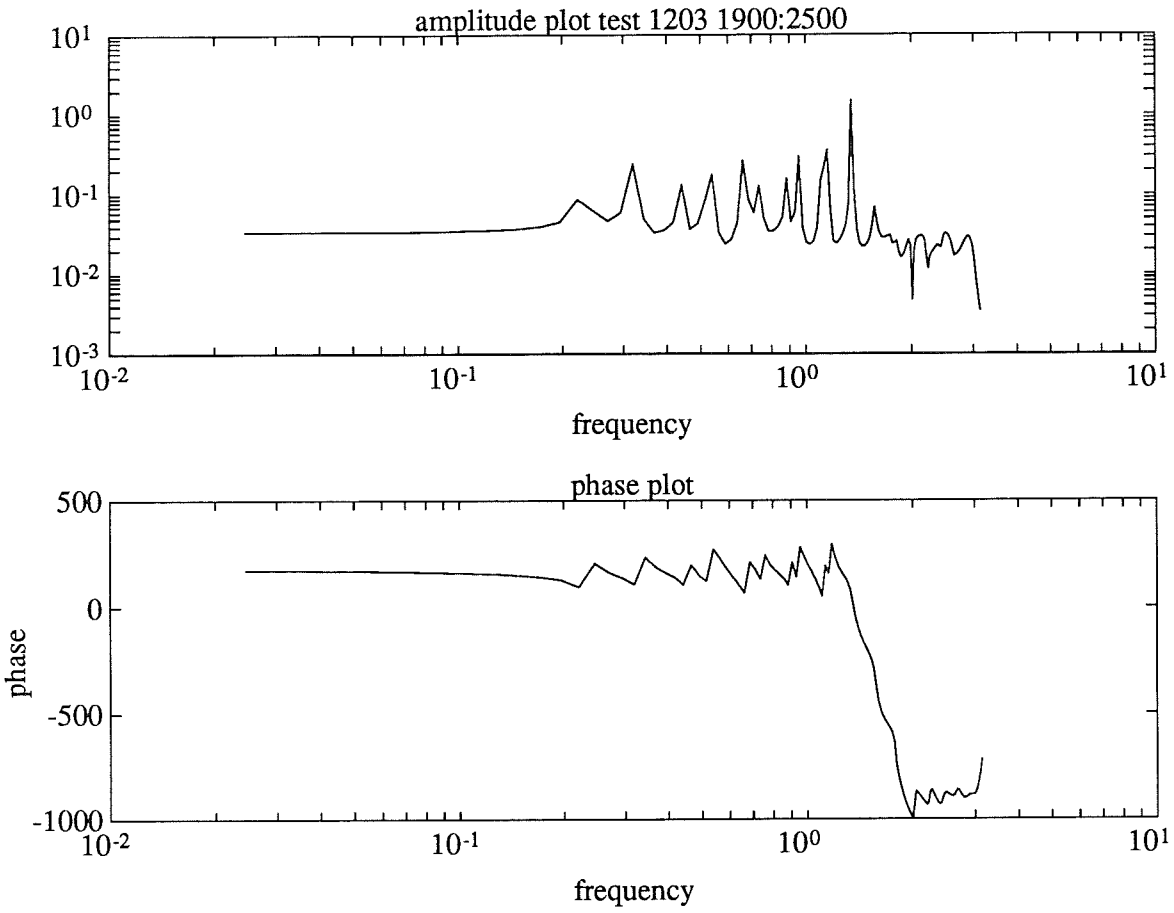

coherens test903 1900:2500
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

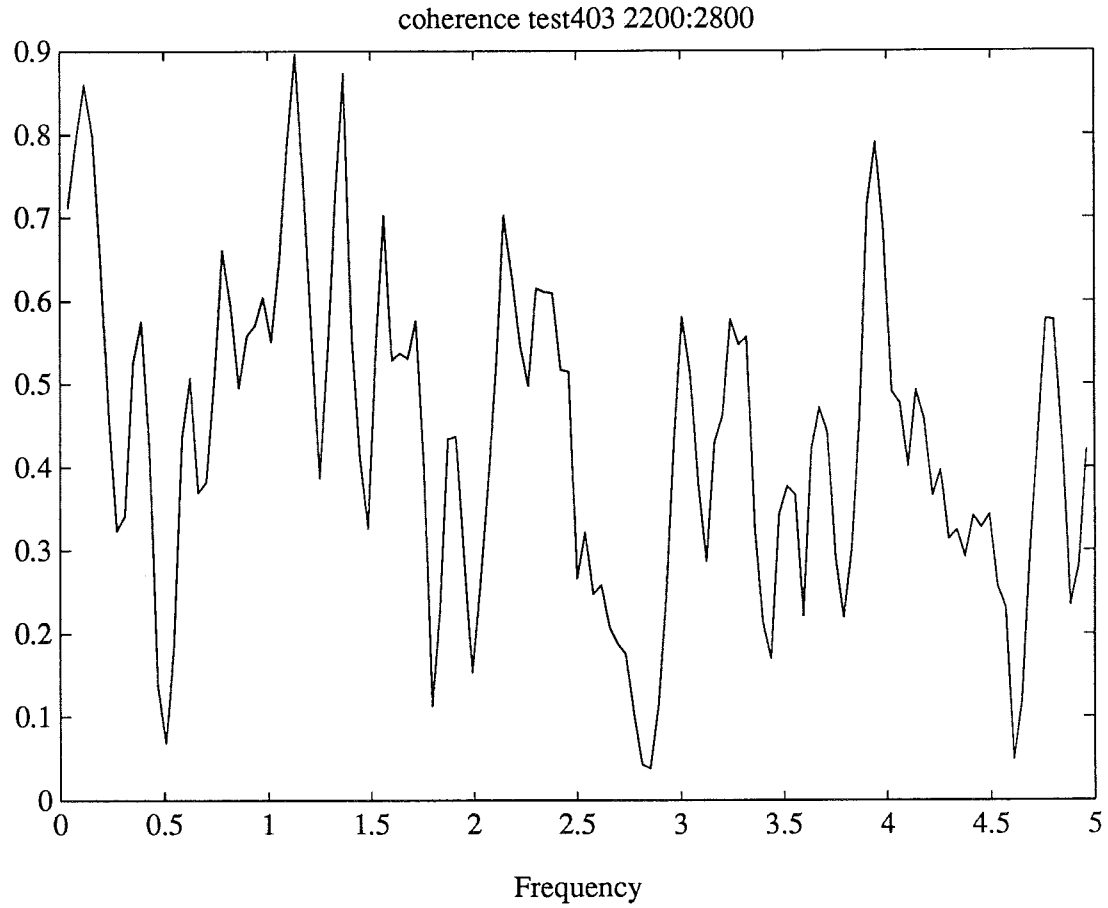

coherens test1203 1900:2500
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

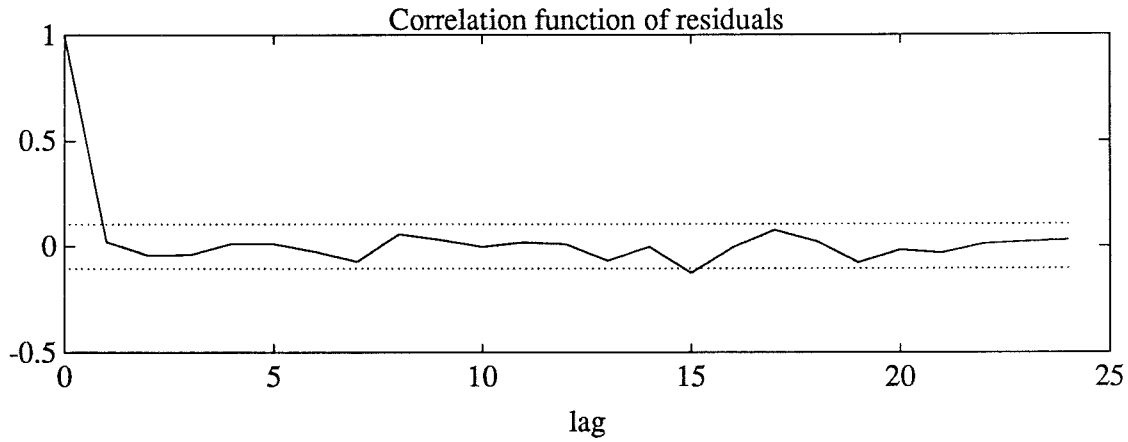

amplitude plot test 403 2200:2800
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-2}
10^{-1}
10^0
10^1
frequency
phase plot
400
200
0
-200
-400
-600
phase
10^{-2}
10^{-1}
10^0
10^1
frequency

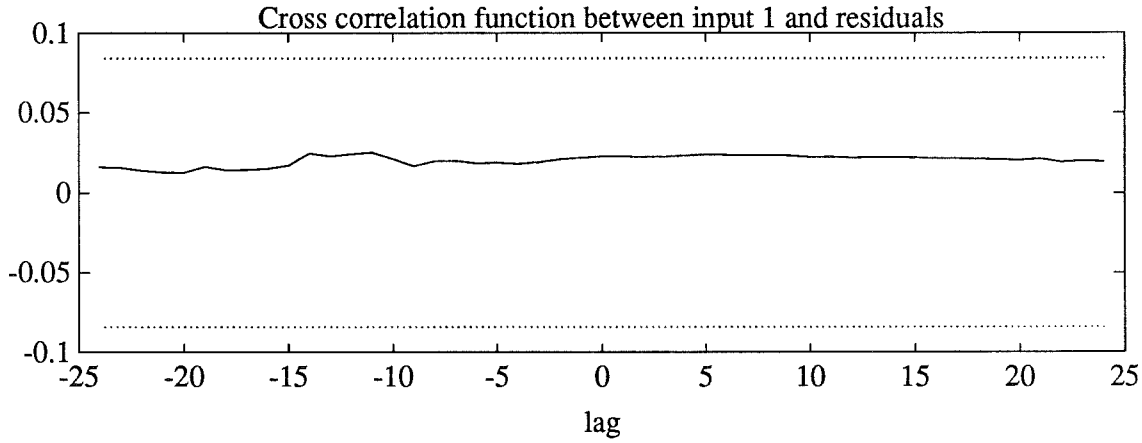

amplitude plot test 703 1700:2300
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
frequency
phase plot
100
0
-100
-200
-300
phase
10^{-2}
10^{-1}
10^{0}
10^{1}
frequency

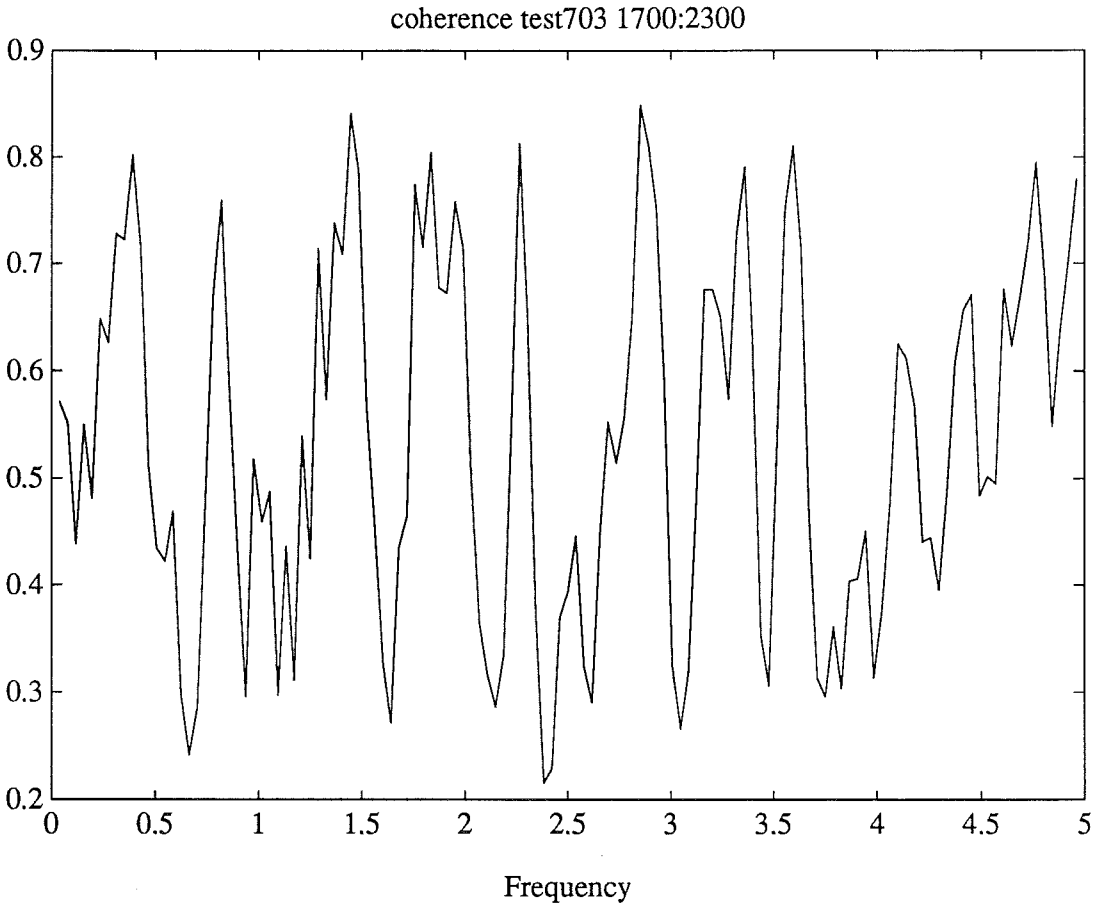

amplitude plot test 903 1900:2500
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-2}
10^{-1}
10^0
10^1
frequency
phase plot
500
0
-500
-1000
phase
10^{-2}
10^{-1}
10^0
10^1
frequency

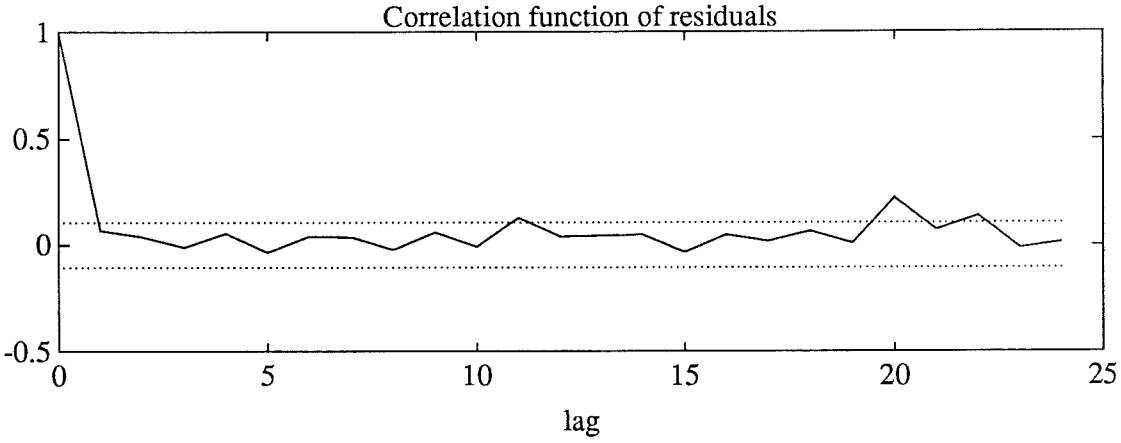

amplitude plot test 1203 1900:2500
10^1
10^0
10^-1
10^-2
10^-3
10^-2
10^-1
10^0
10^1
frequency
phase plot
500
0
-500
-1000
phase
10^-2
10^-1
10^0
10^1
frequency

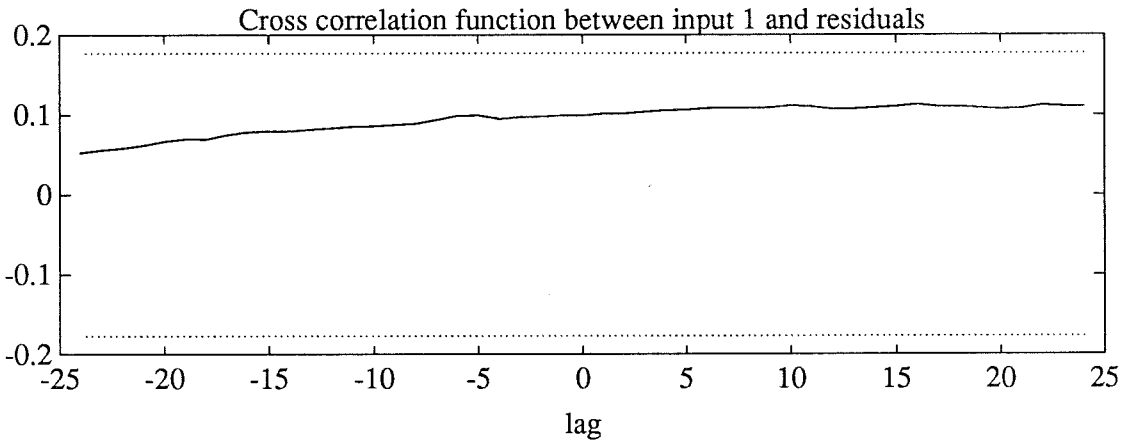

coherence test403 2200:2800
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

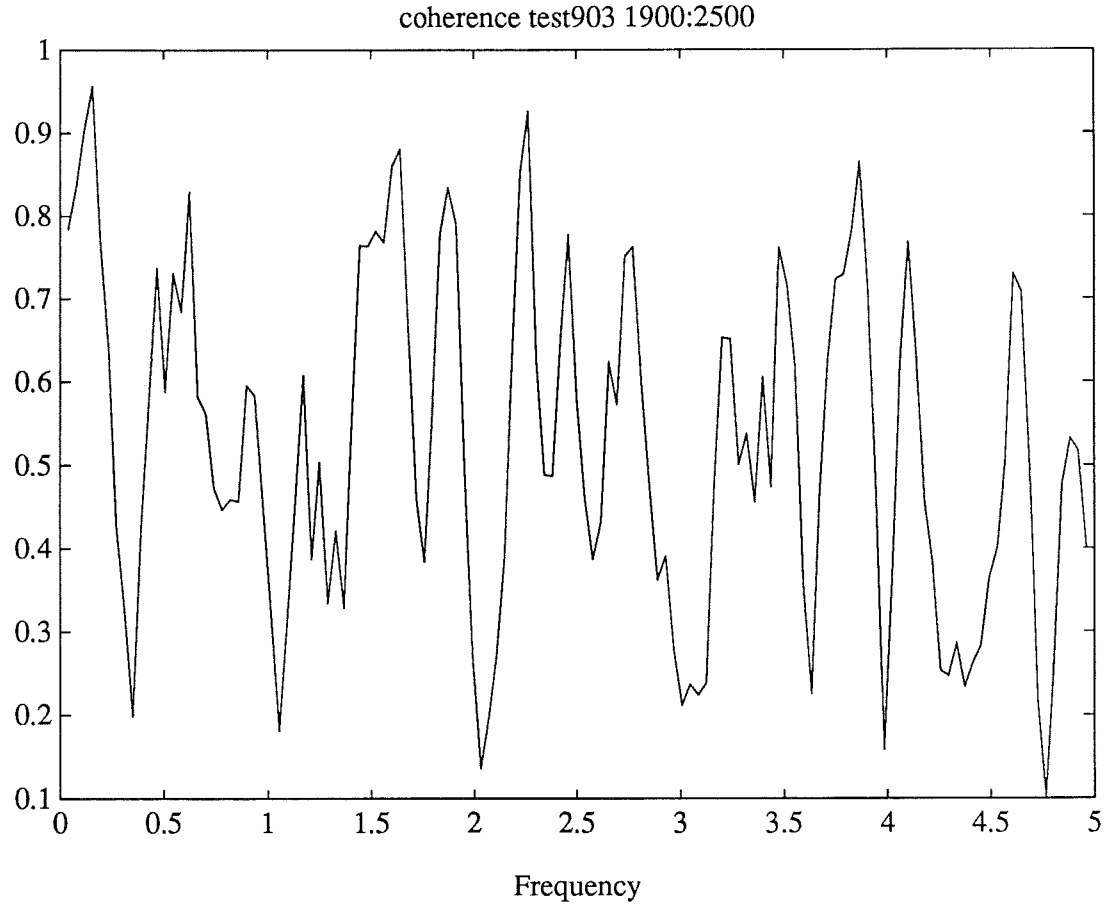

Correlation function of residuals
1
0.5
0
-0.5
0
5
10
15
20
25
lag

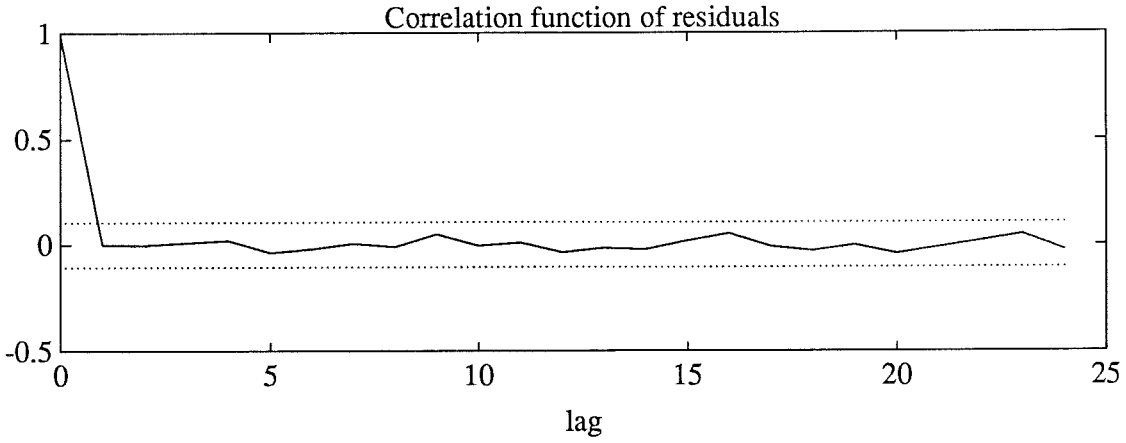

Cross correlation function between input 1 and residuals
0.1
0.05
0
-0.05
-0.1
-25
-20
-15
-10
-5
0
5
10
15
20
25
lag

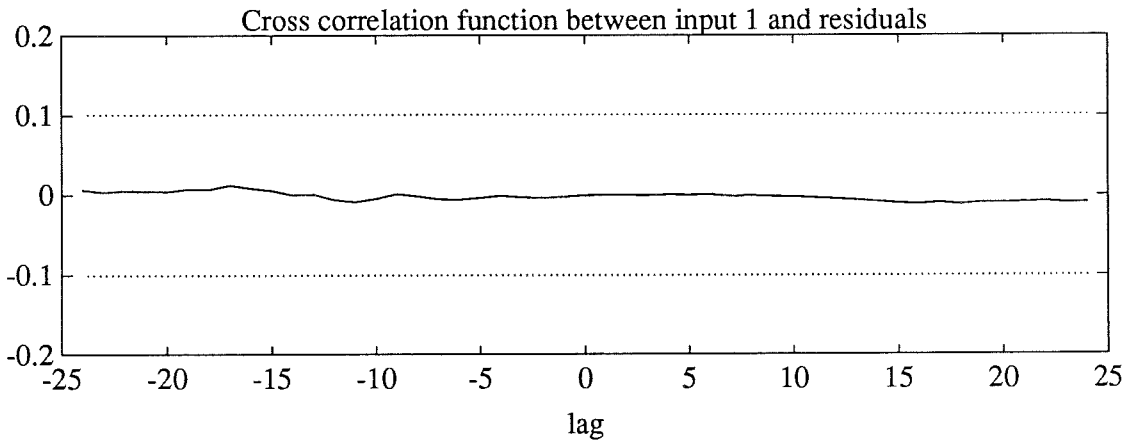

coherence test703 1700:2300
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

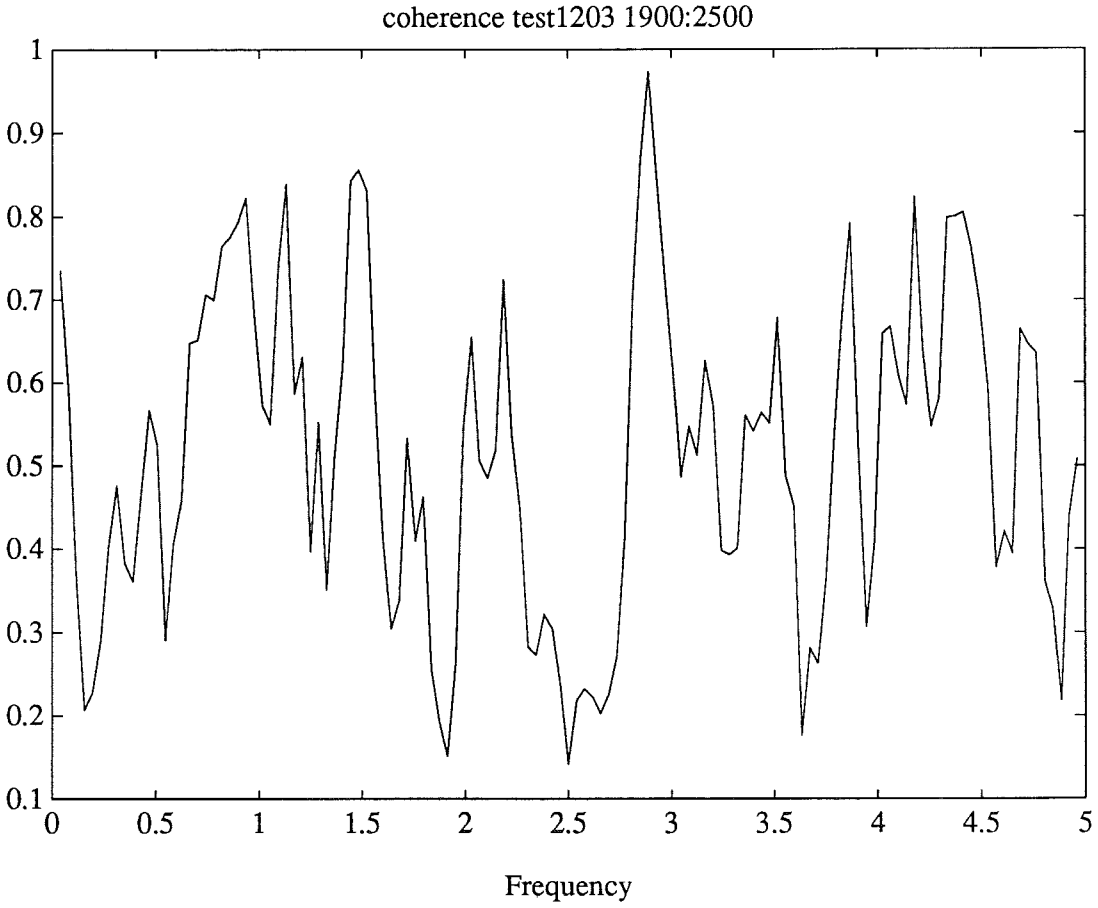

Correlation function of residuals
1
0.5
0
-0.5
0
5
10
15
20
25
lag

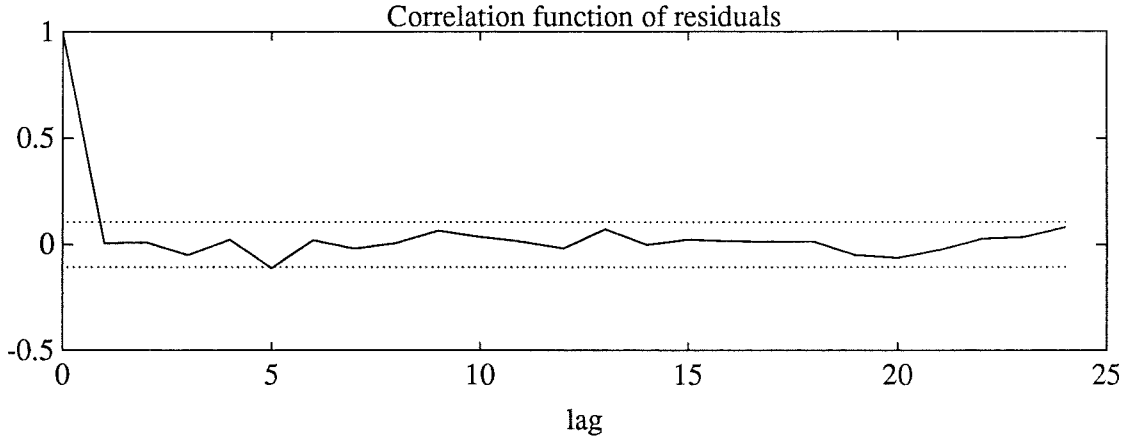

Cross correlation function between input 1 and residuals
0.2
0.1
0
-0.1
-0.2
-25
-20
-15
-10
-5
0
5
10
15
20
25
lag

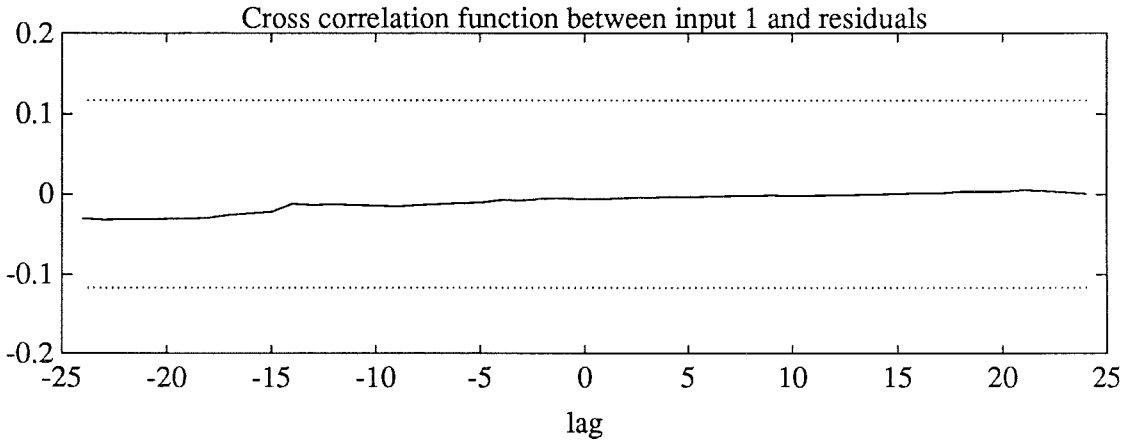

coherence test903 1900:2500
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

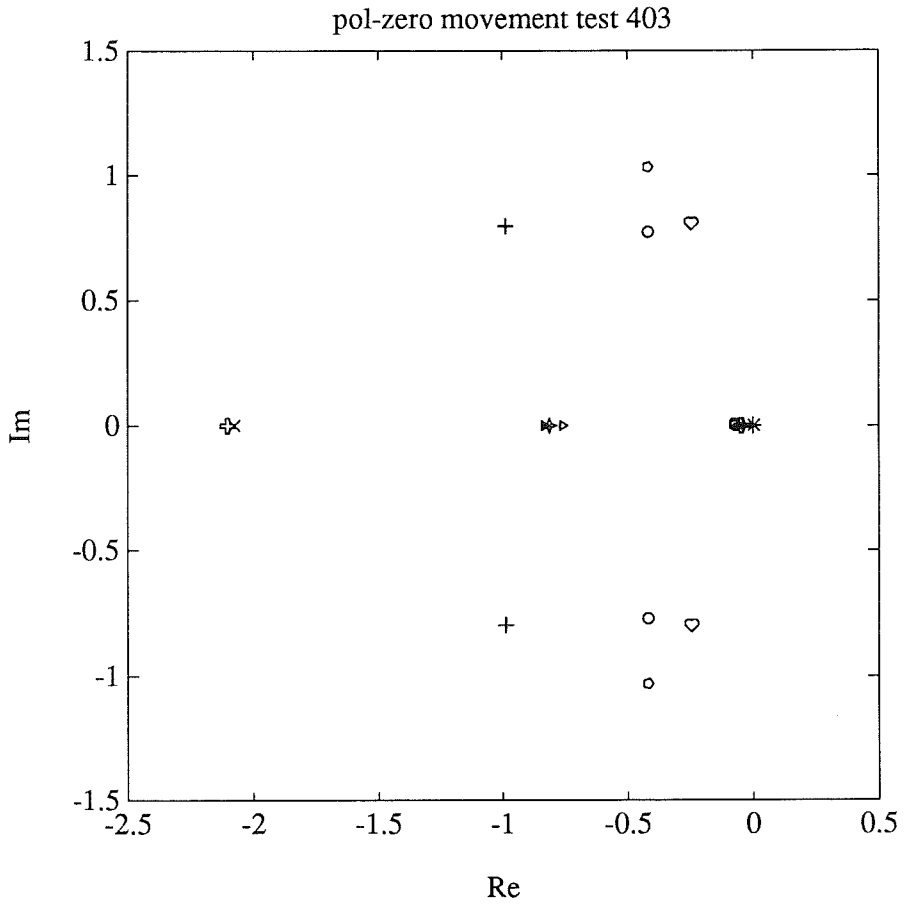

Correlation function of residuals
1
0.5
0
-0.5
0
5
10
15
20
25
lag

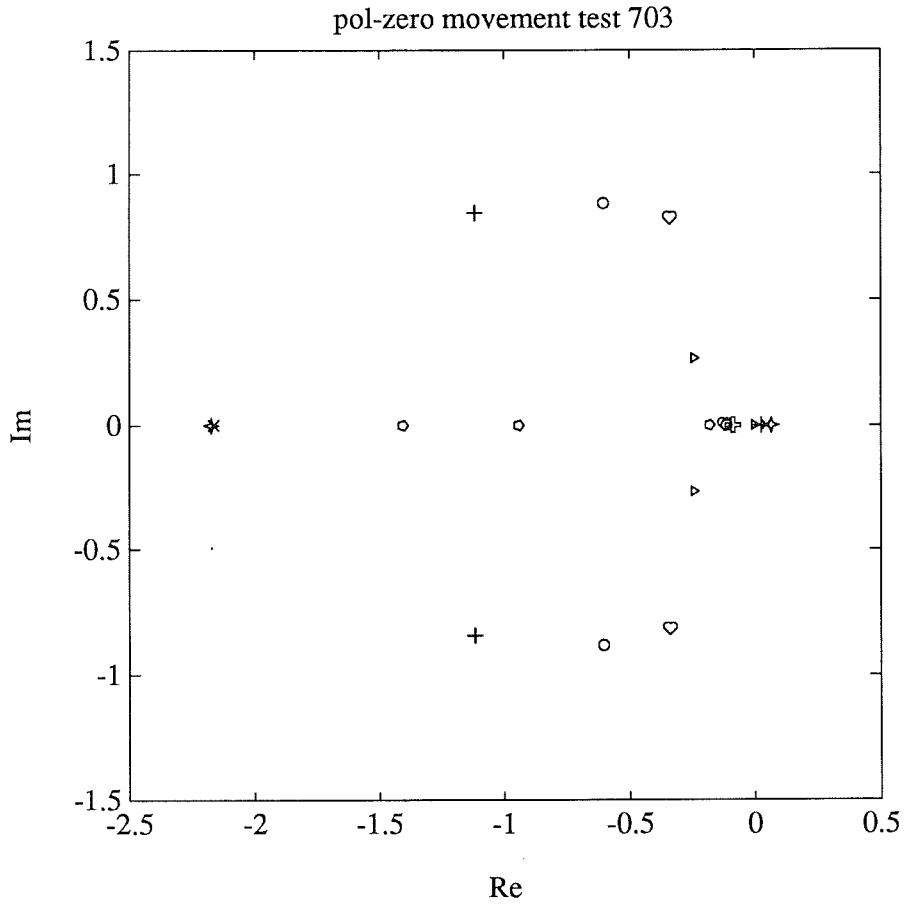

Cross correlation function between input 1 and residuals
0.2
0.1
0
-0.1
-0.2
-25
-20
-15
-10
-5
0
5
10
15
20
25
lag

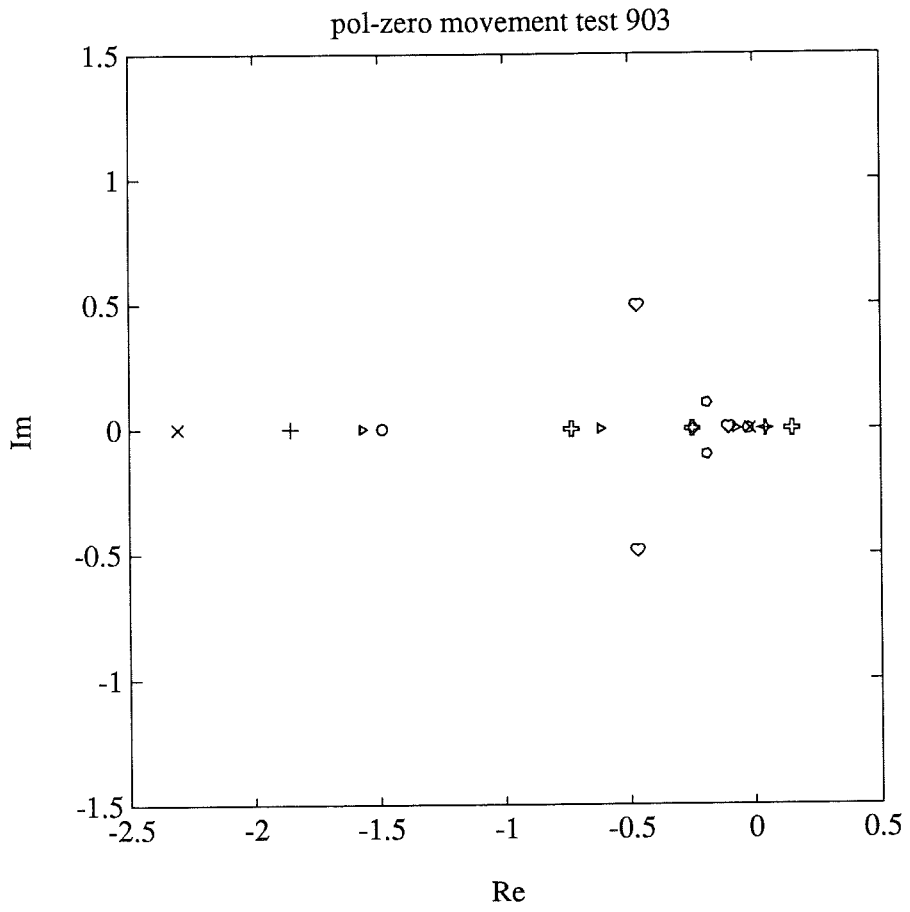

coherence test1203 1900:2500
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

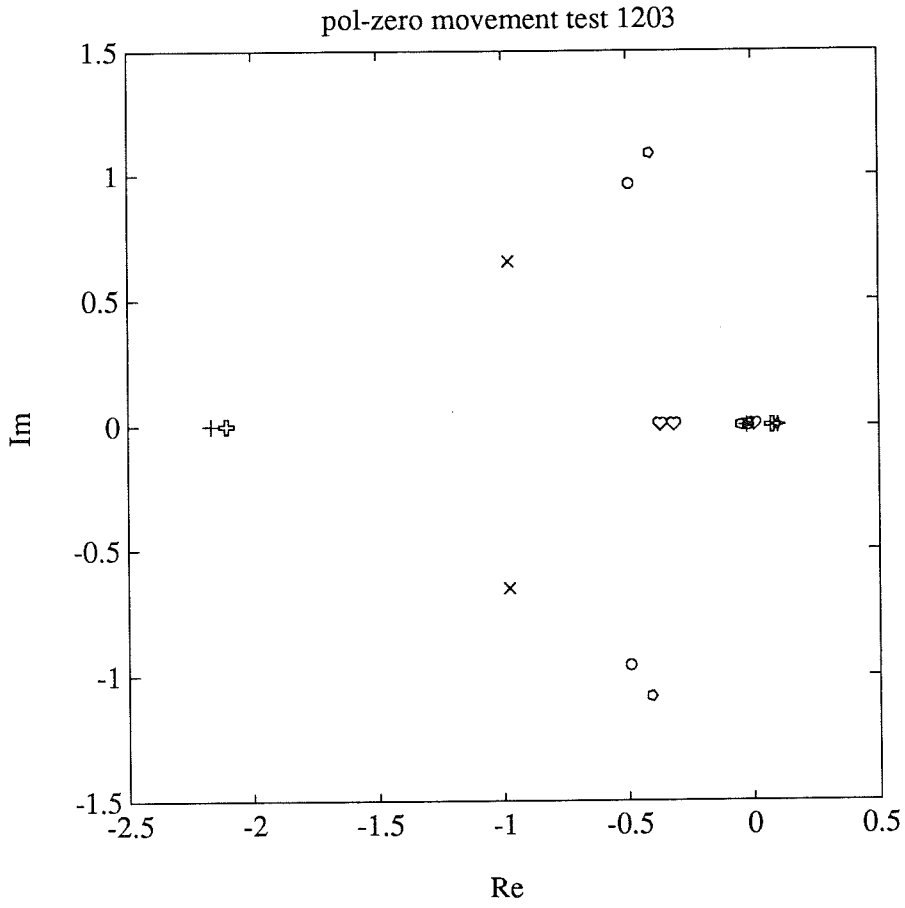

Correlation function of residuals
1
0.5
0
-0.5
0
5
10
15
20
25
lag

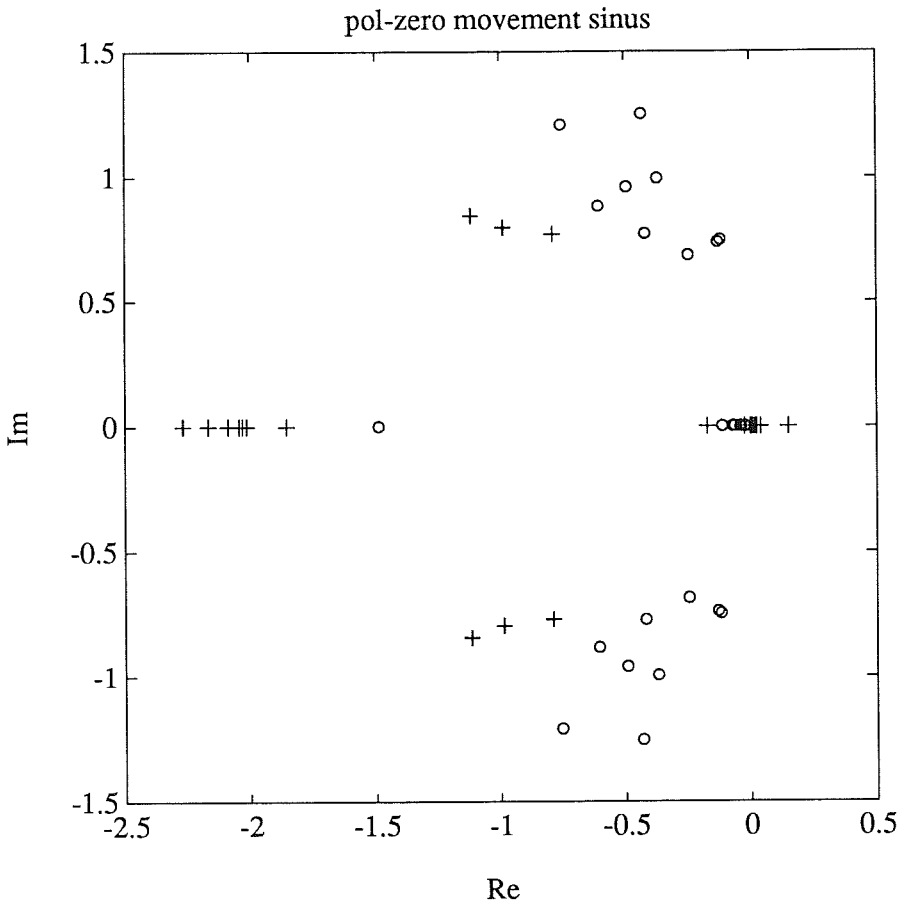

Cross correlation function between input 1 and residuals
0.2
0.1
0
-0.1
-0.2
-25
-20
-15
-10
-5
0
5
10
15
20
25
lag

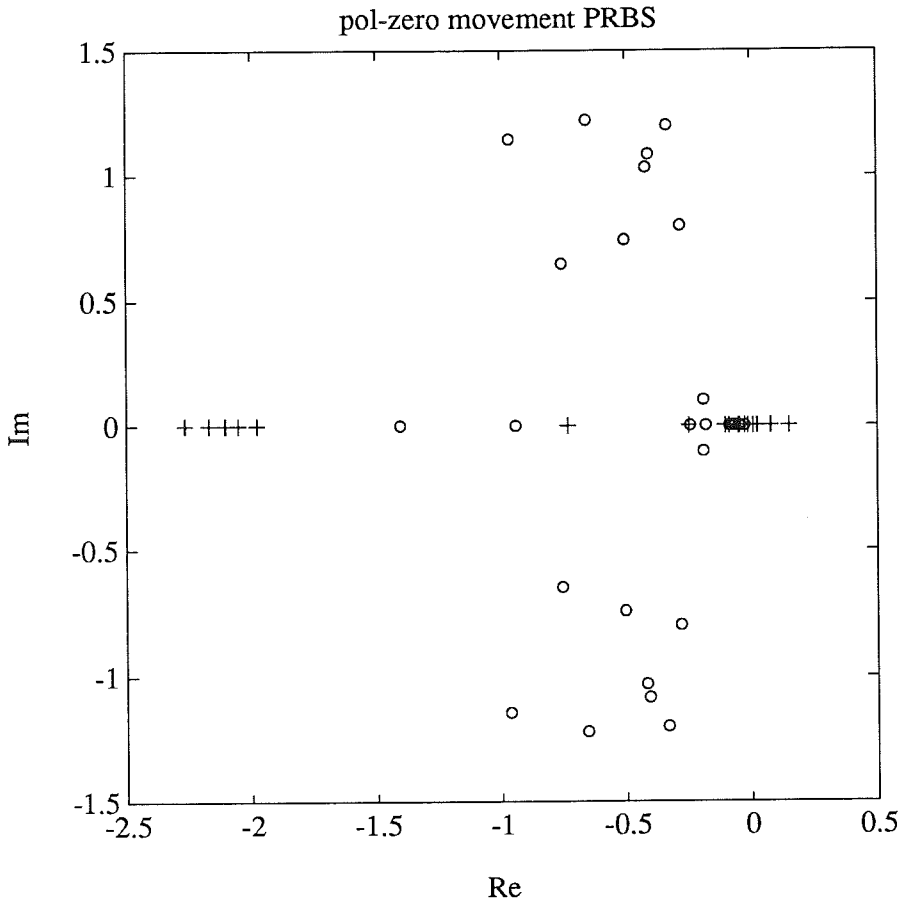

pol-zero movement test 403
1.5
1
0.5
0
-0.5
-1
-1.5
Im
-2.5
-2
-1.5
-1
-0.5
0
0.5
Re


pol-zero movement test 703
1.5
1
0.5
0
-0.5
-1
-1.5
Im
-2.5
-2
-1.5
-1
-0.5
0
0.5
Re

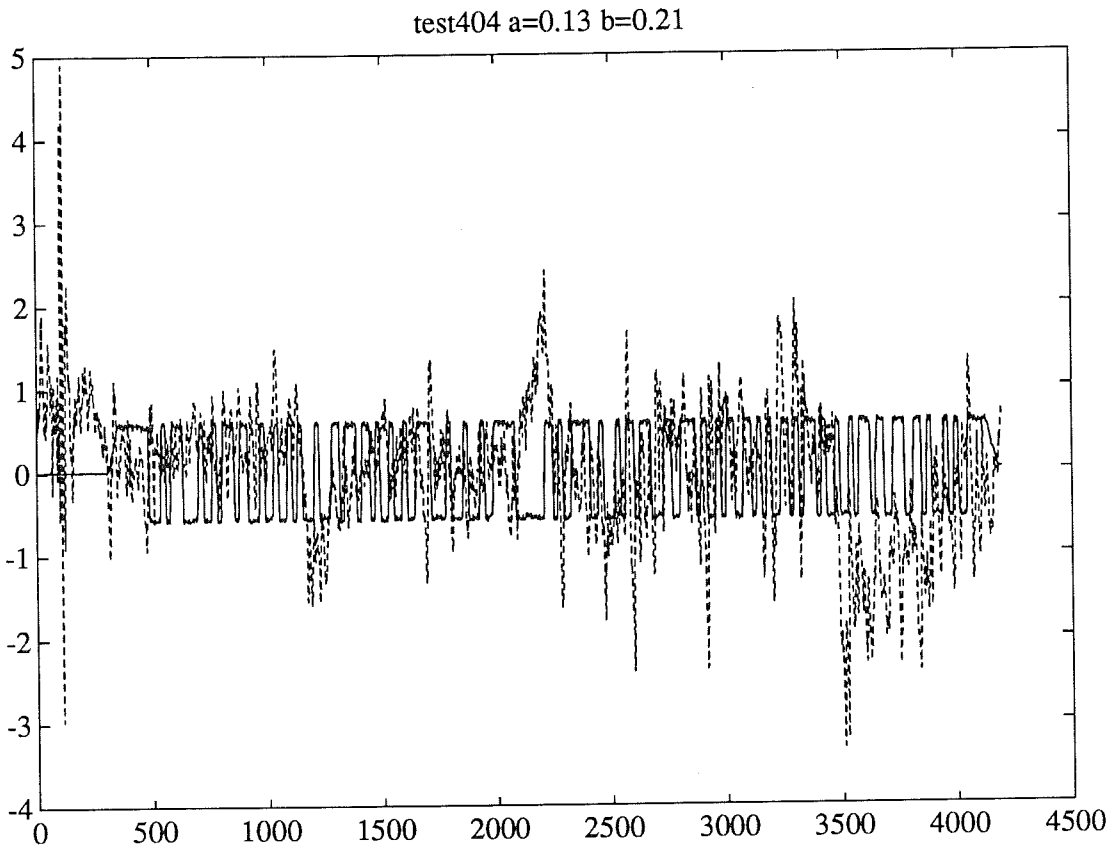

pol-zero movement test 903
1.5
1
0.5
0
-0.5
-1
-1.5
Im
-2.5
-2
-1.5
-1
-0.5
0
0.5
Re

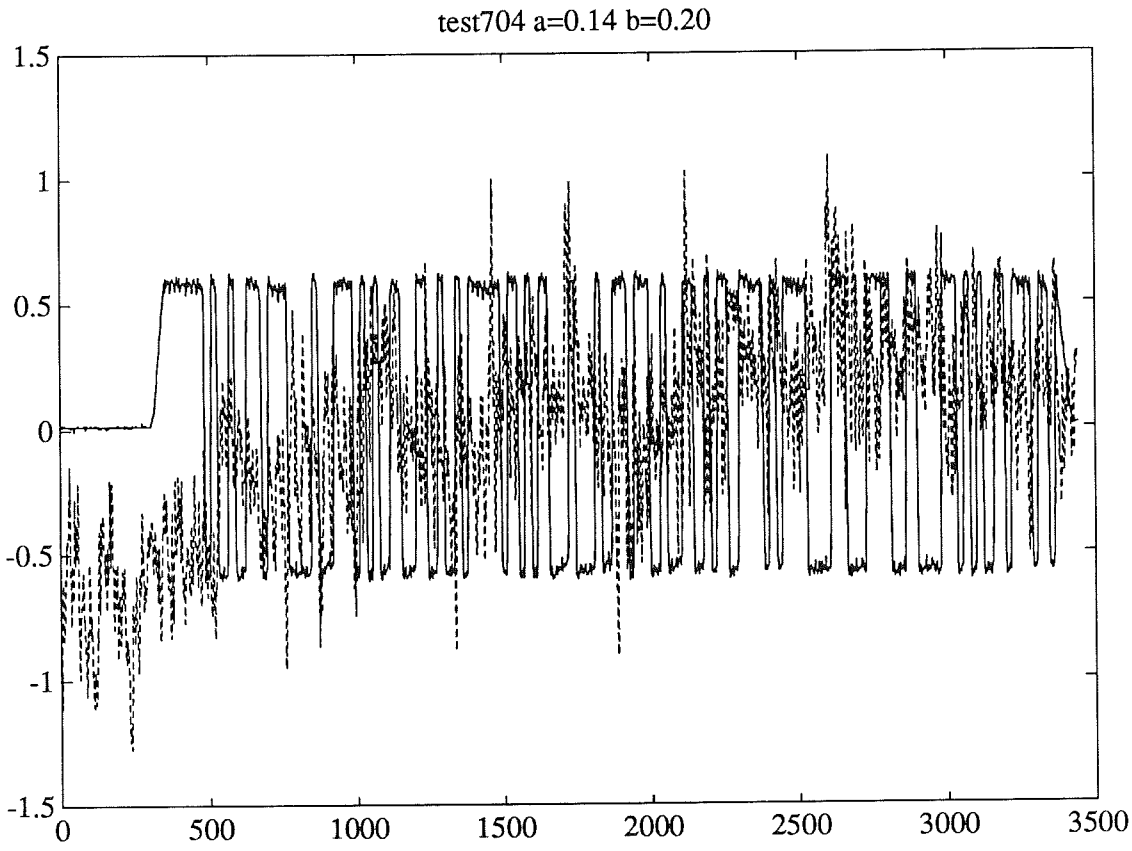

pol-zero movement test 1203
1.5
1
0.5
0
-0.5
-1
-1.5
Im
-2.5
-2
-1.5
-1
-0.5
0
0.5
Re

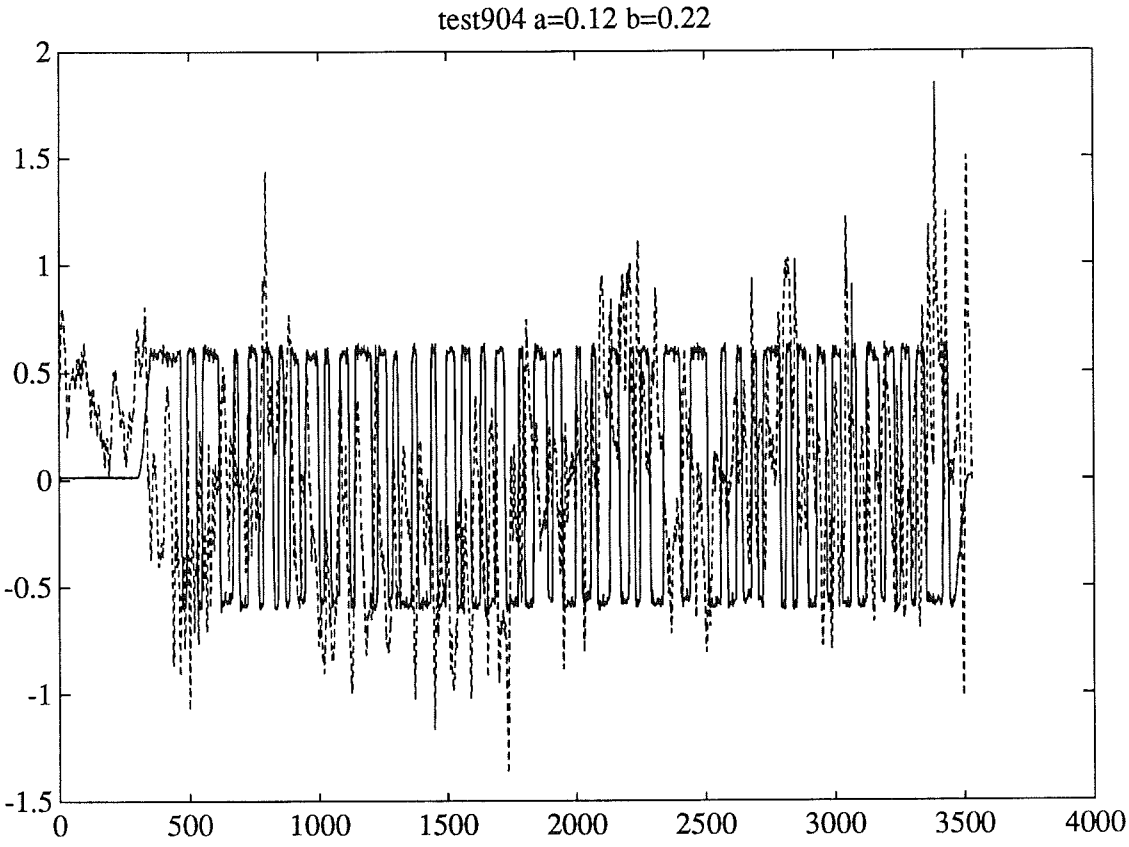

pol-zero movement sinus
1.5
1
0.5
0
-0.5
-1
-1.5
Im
-2.5
-2
-1.5
-1
-0.5
0
0.5
Re

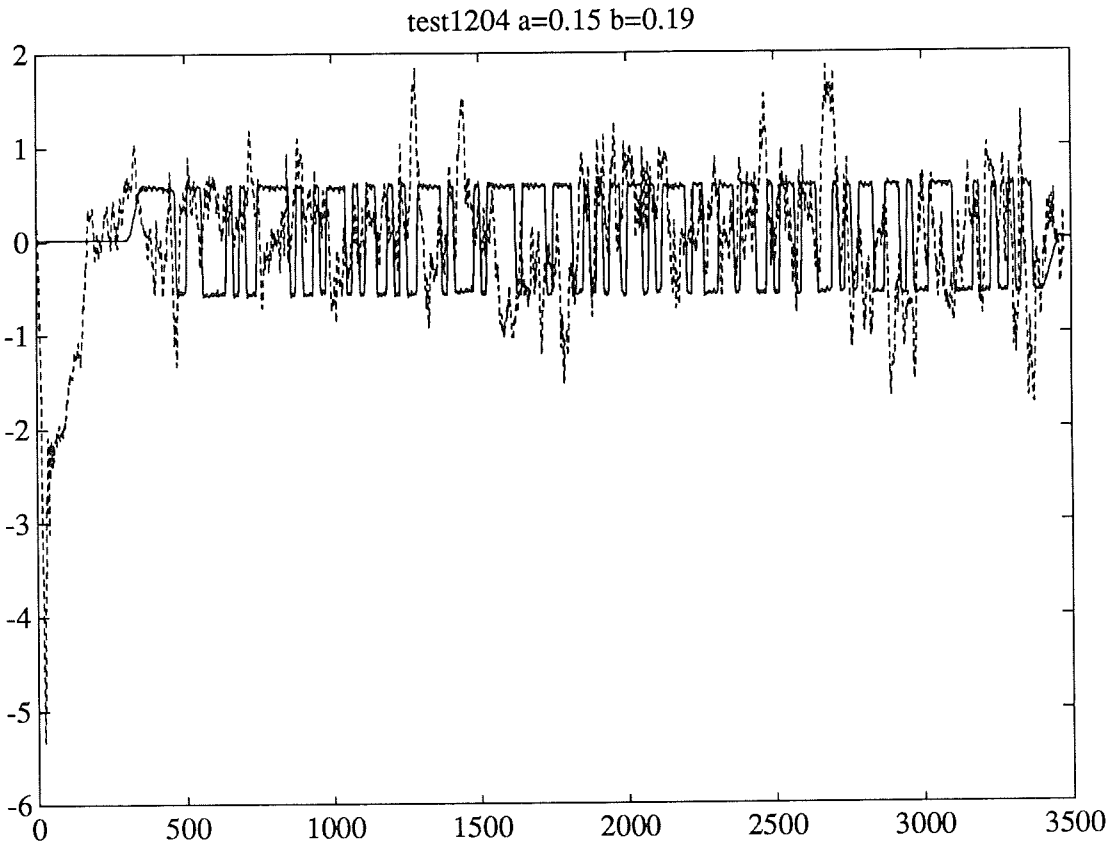

pol-zero movement PRBS
1.5
1
0.5
0
-0.5
-1
-1.5
Im
-2.5
-2
-1.5
-1
-0.5
0
0.5
Re

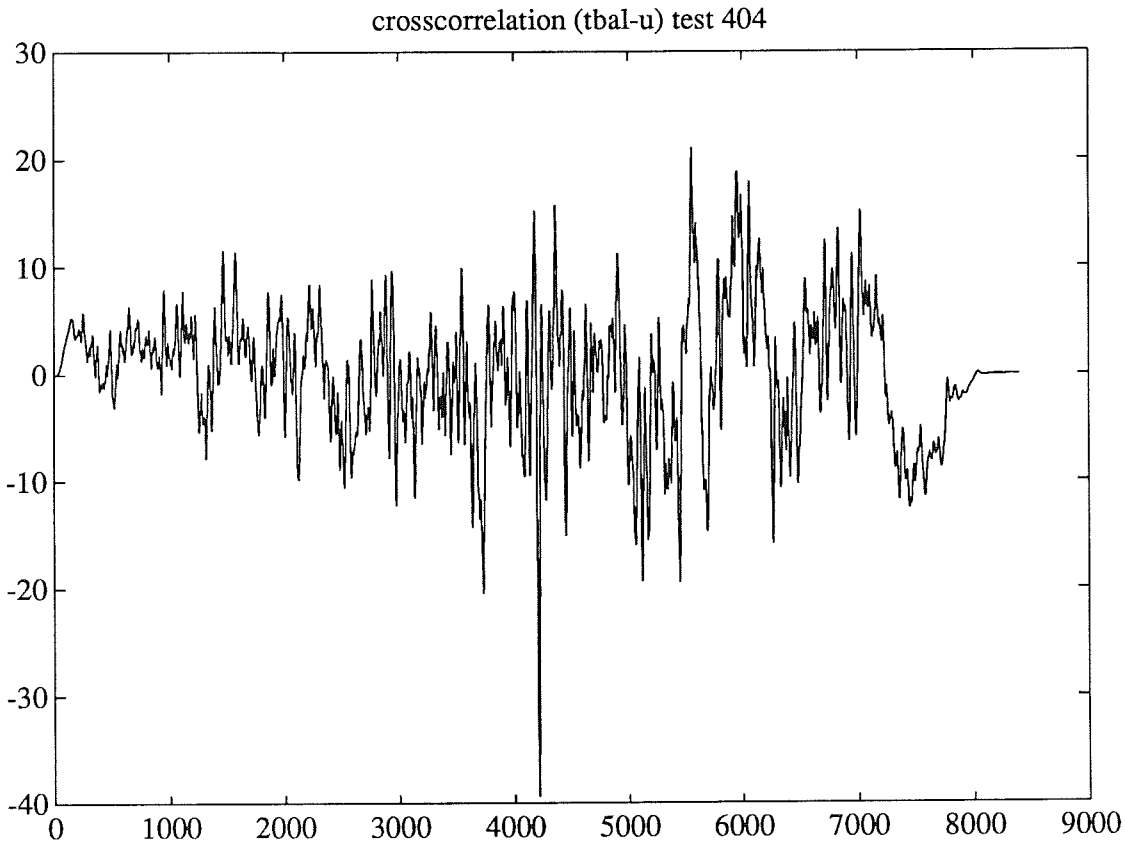
VII.3 Estimation and validation of PRBS-stimulation.

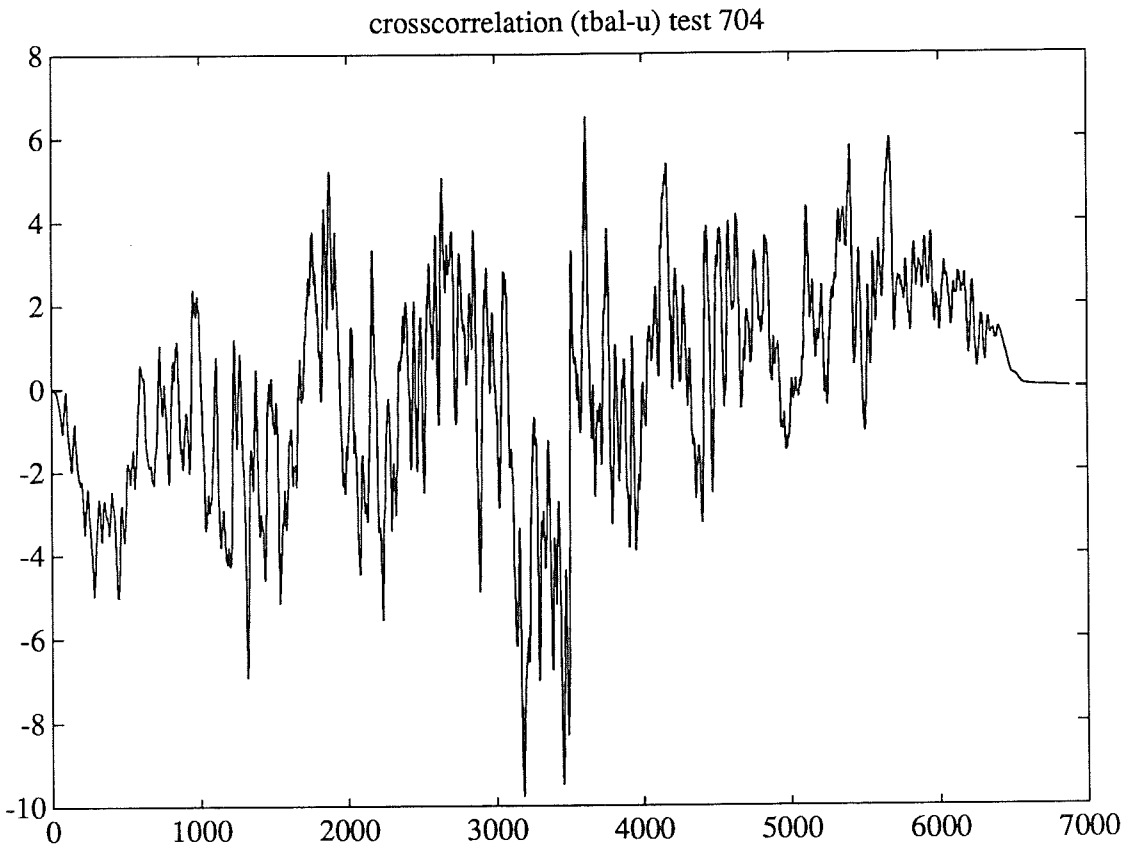

test404 a=0.13 b=0.21
5
4
3
2
1
0
-1
-2
-3
-4
0
500
1000
1500
2000
2500
3000
3500
4000
4500

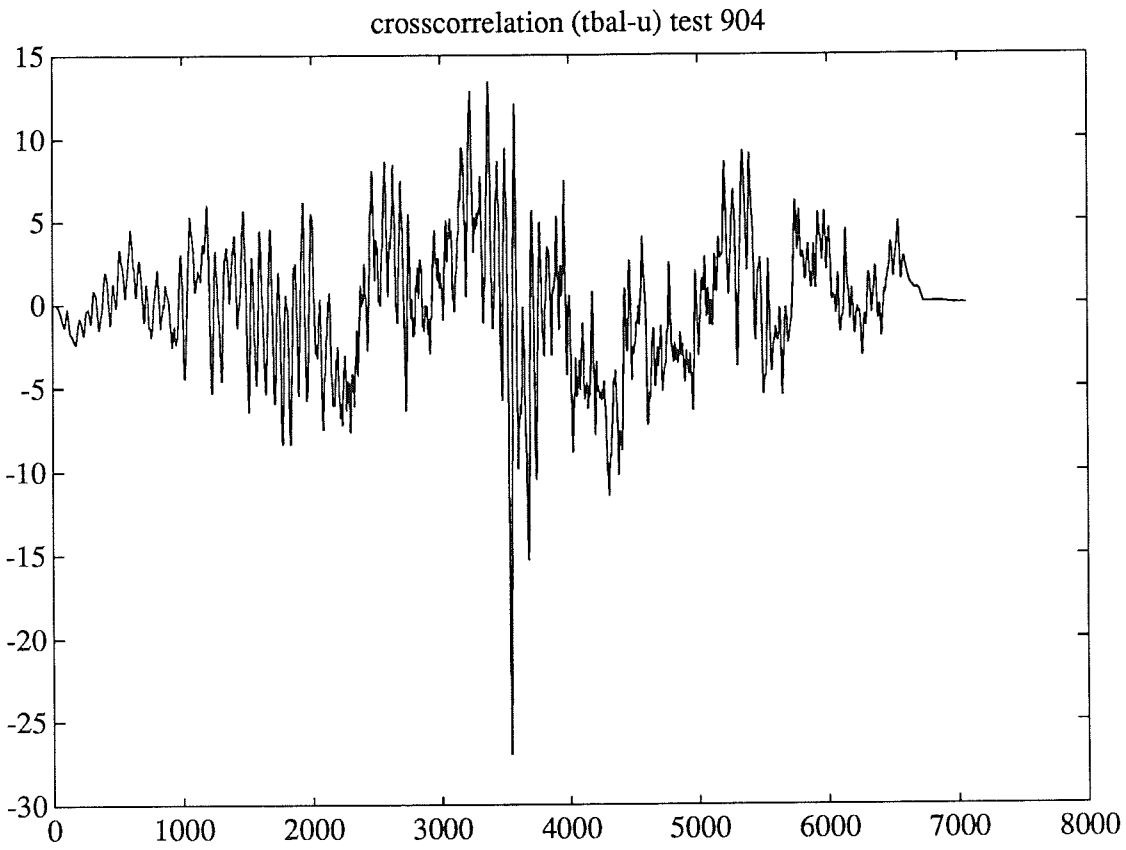

test704 a=0.14 b=0.20
1.5
1
0.5
0
-0.5
-1
-1.5
0
500
1000
1500
2000
2500
3000
3500

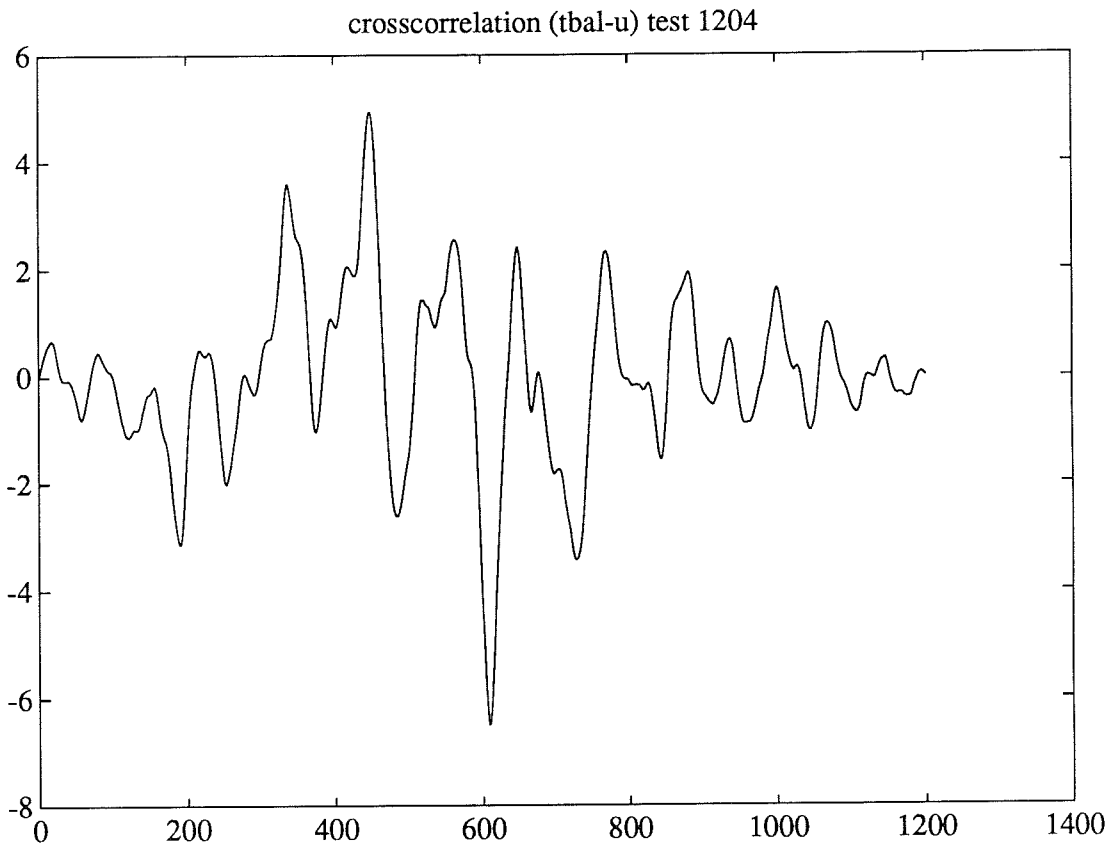

test904 a=0.12 b=0.22
2
1.5
1
0.5
0
-0.5
-1
-1.5
0
500
1000
1500
2000
2500
3000
3500
4000

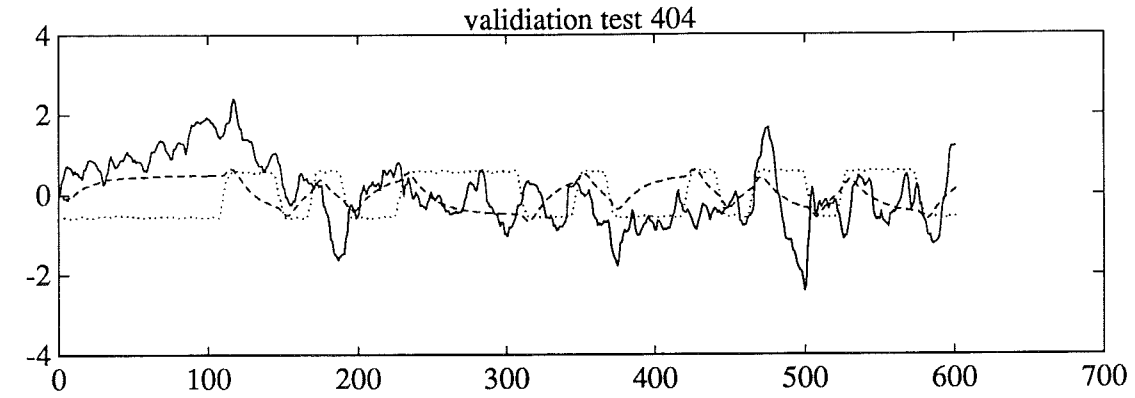

test1204 a=0.15 b=0.19
2
1
0
-1
-2
-3
-4
-5
-6
0
500
1000
1500
2000
2500
3000
3500

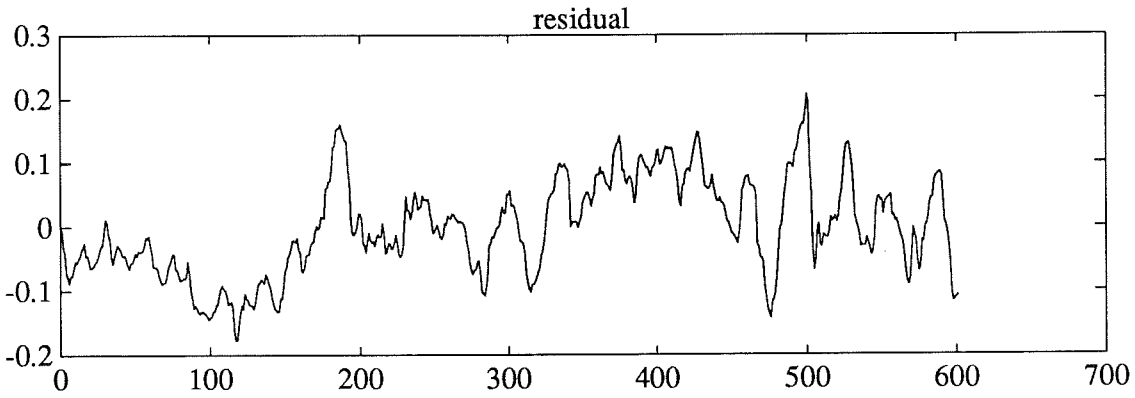

crosscorrelation (tbal-u) test 404
30
20
10
0
-10
-20
-30
-40
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

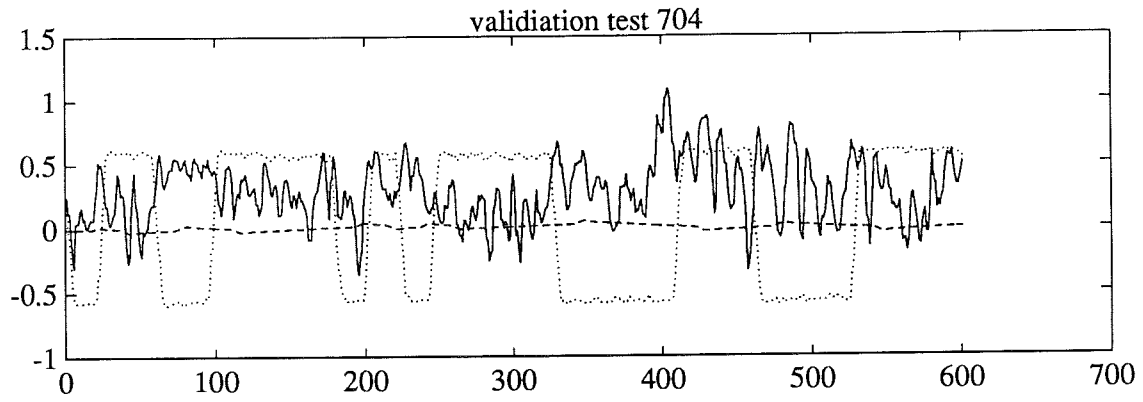

crosscorrelation (tbal-u) test 704
8
6
4
2
0
-2
-4
-6
-8
-10
0
1000
2000
3000
4000
5000
6000
7000

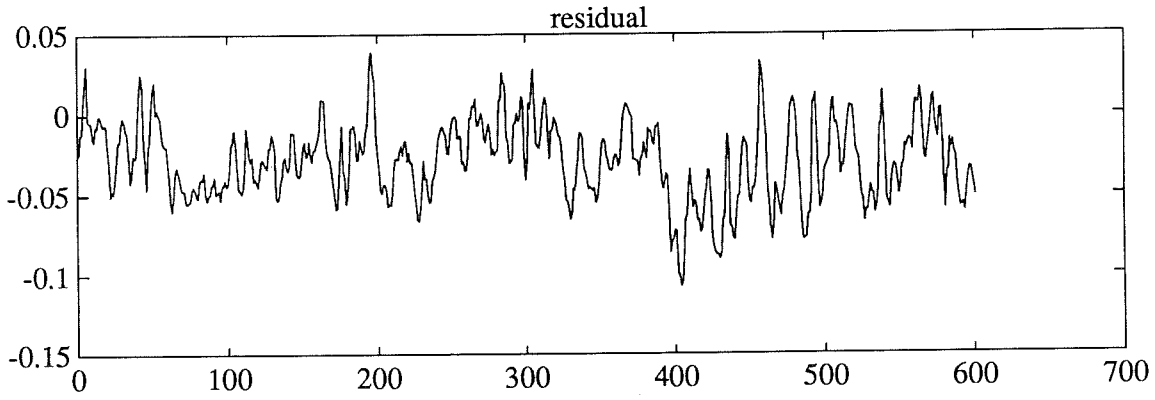

crosscorrelation (tbal-u) test 904
15
10
5
0
-5
-10
-15
-20
-25
-30
0
1000
2000
3000
4000
5000
6000
7000
8000

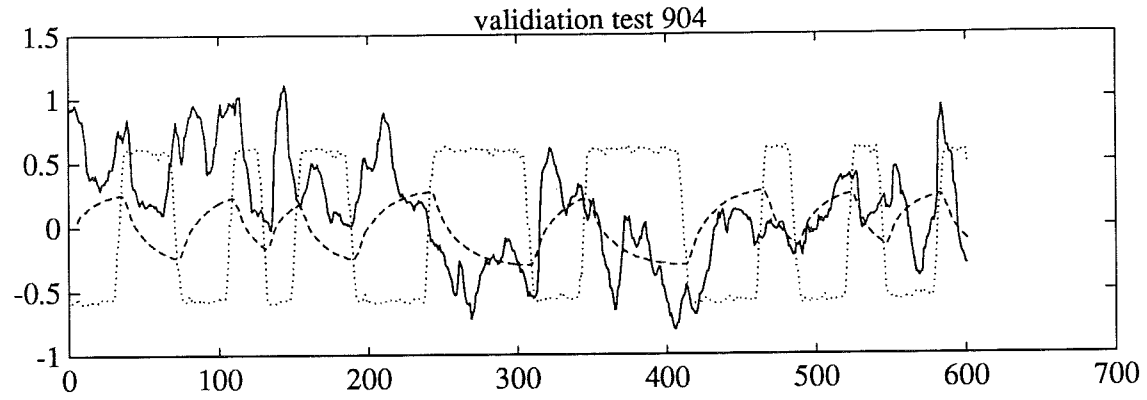

crosscorrelation (tbal-u) test 1204
6
4
2
0
-2
-4
-6
-8
0
200
400
600
800
1000
1200
1400

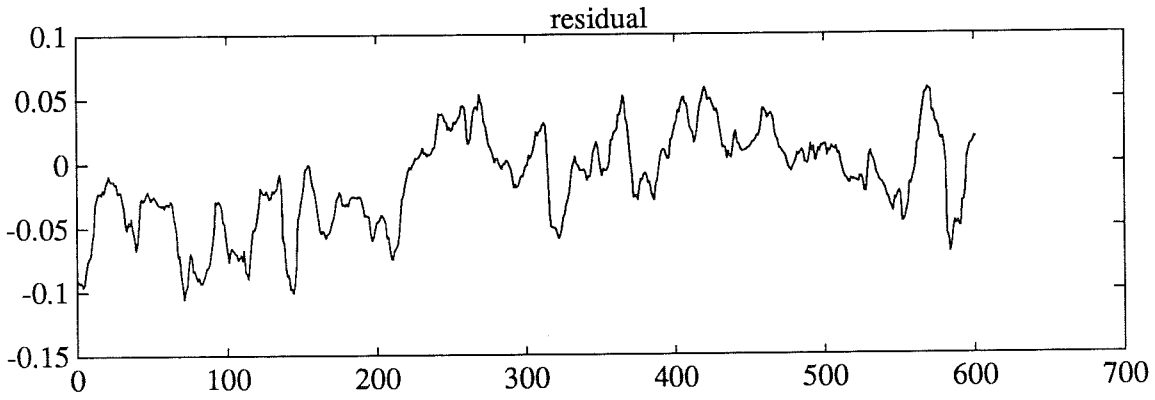

validiation test 404
4
2
0
-2
-4
0
100
200
300
400
500
600
700

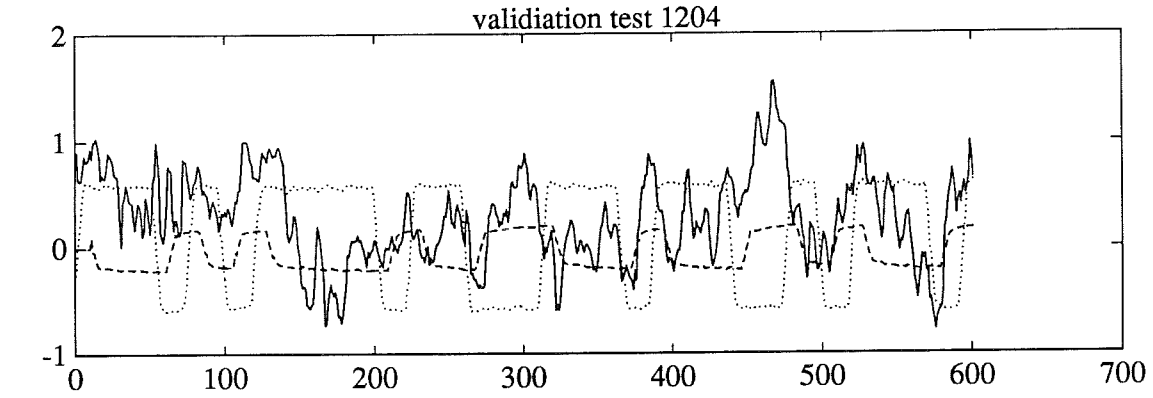

residual
0.3
0.2
0.1
0
-0.1
-0.2
0
100
200
300
400
500
600
700

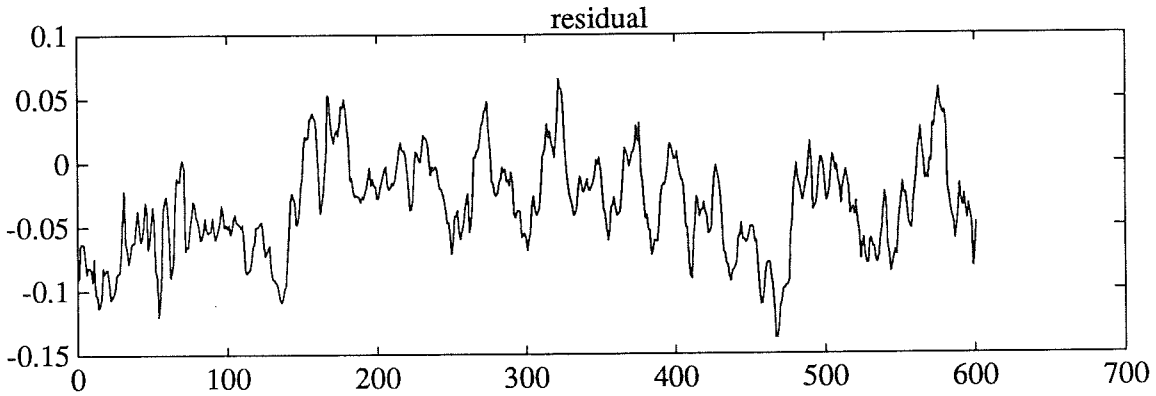

validiation test 704
1.5
1
0.5
0
-0.5
-1
0
100
200
300
400
500
600
700

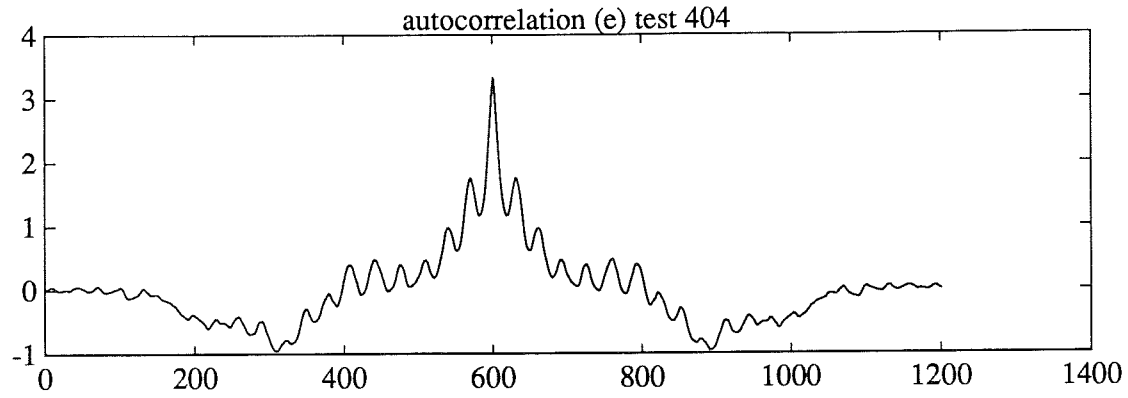

residual
0.05
0
-0.05
-0.1
-0.15
0
100
200
300
400
500
600
700

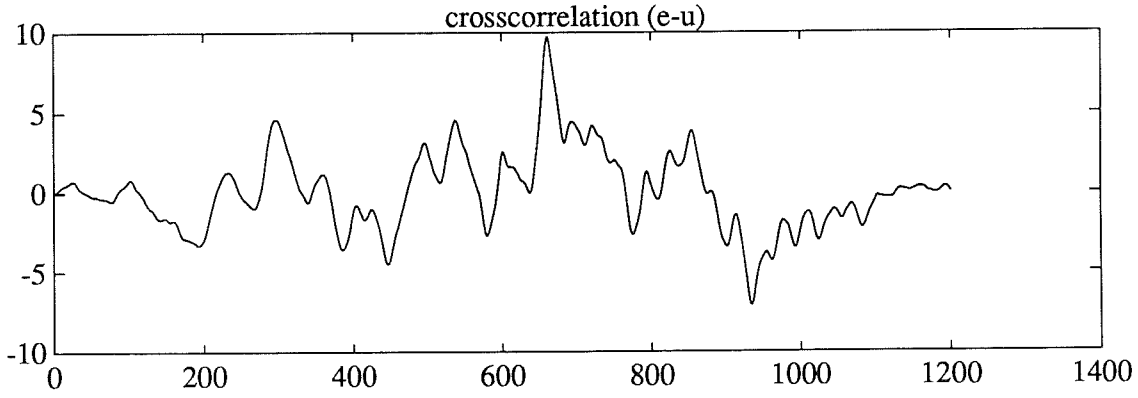

validiation test 904
1.5
1
0.5
0
-0.5
-1
0
100
200
300
400
500
600
700

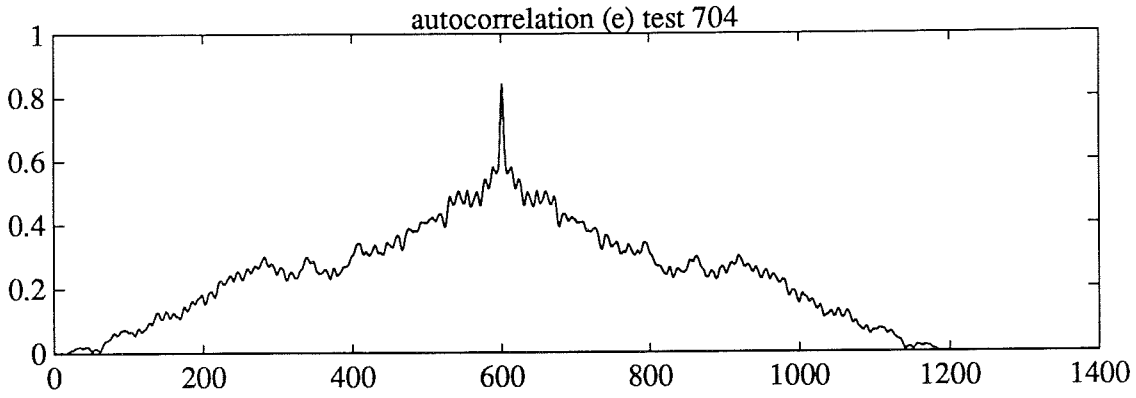

residual
0.1
0.05
0
-0.05
-0.1
-0.15
0
100
200
300
400
500
600
700

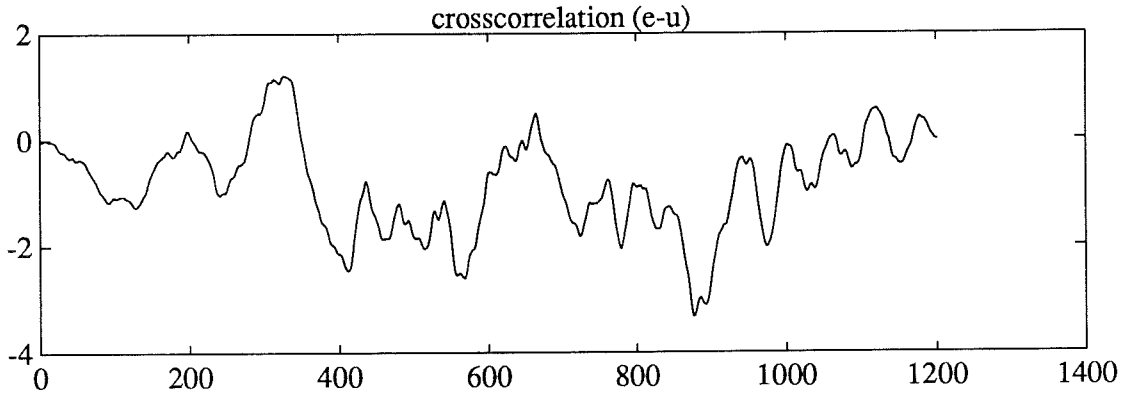

validiation test 1204
2
1
0
-1
0
100
200
300
400
500
600
700

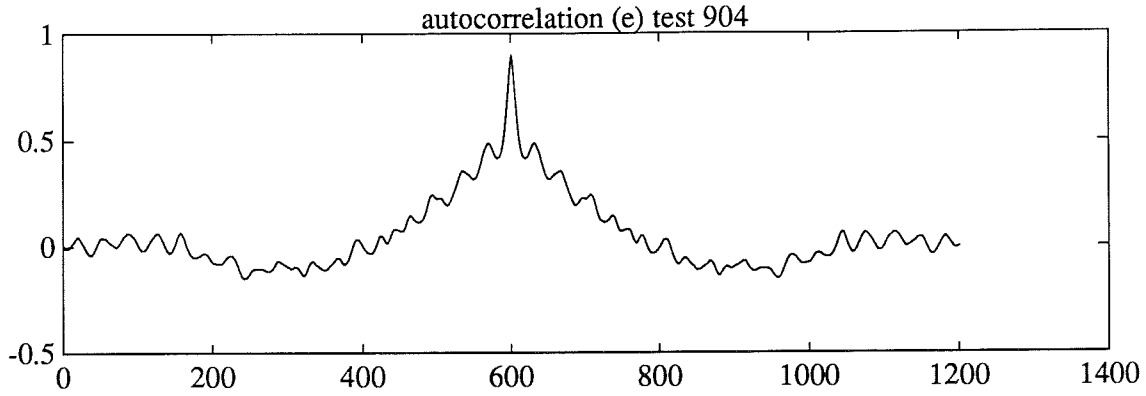

residual
0.1
0.05
0
-0.05
-0.1
-0.15
0
100
200
300
400
500
600
700

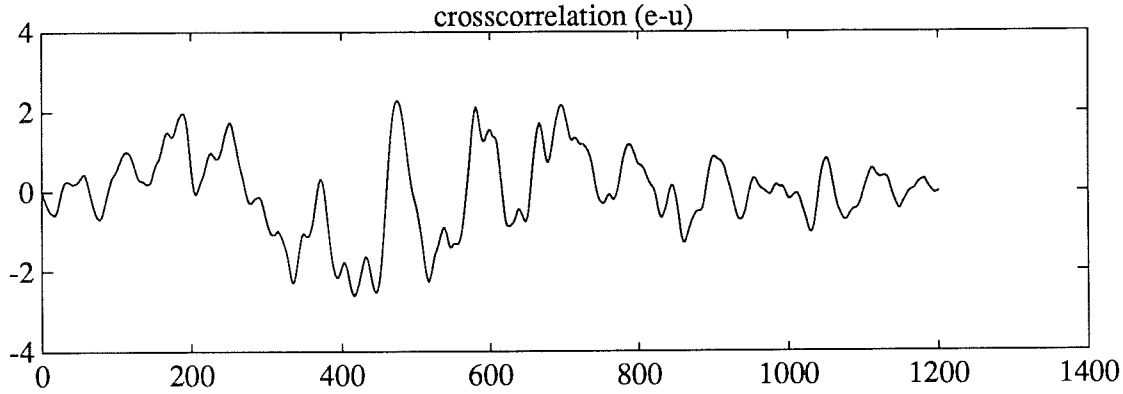

autocorrelation (e) test 404
4
3
2
1
0
-1
0
200
400
600
800
1000
1200
1400

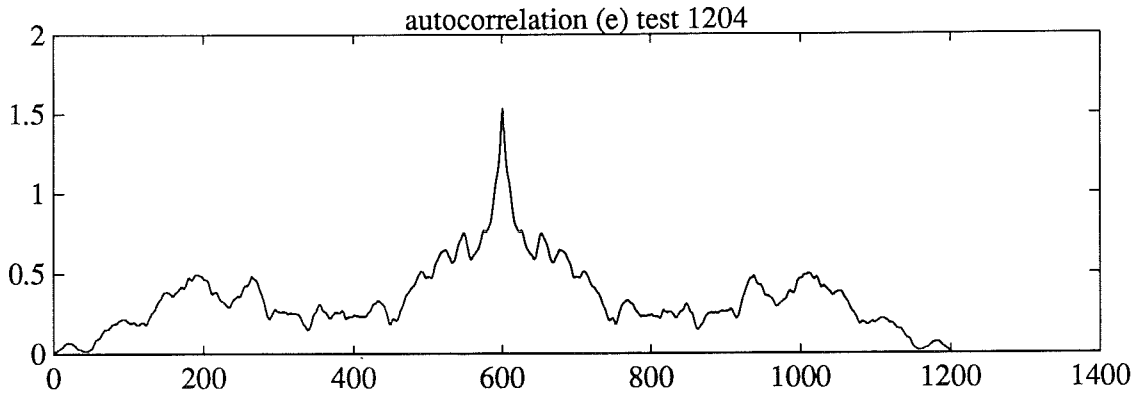

crosscorrelation (e-u)
10
5
0
-5
-10
0
200
400
600
800
1000
1200
1400

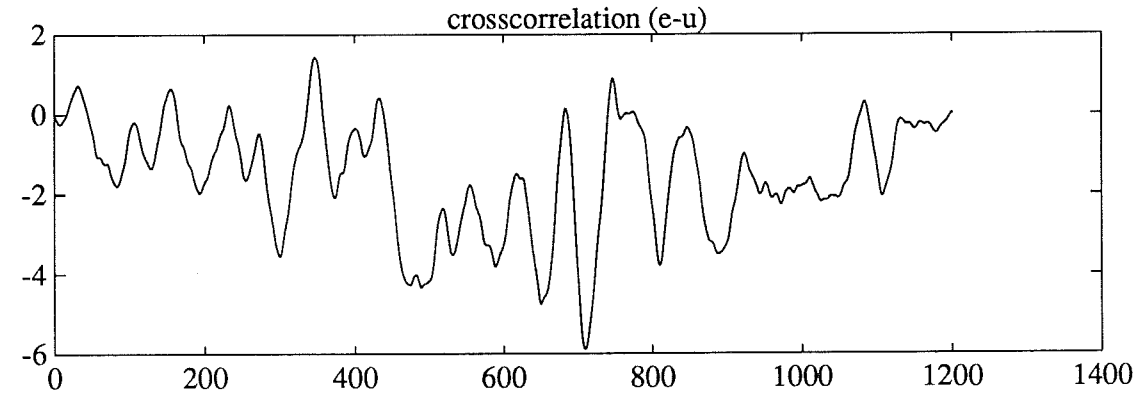

autocorrelation (e) test 704
1
0.8
0.6
0.4
0.2
0
0
200
400
600
800
1000
1200
1400

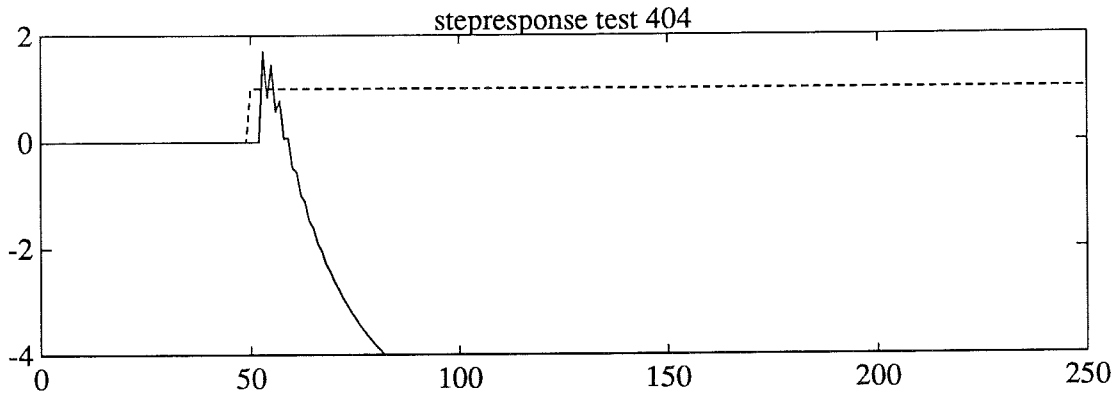

crosscorrelation (e-u)
2
0
-2
-4
0
200
400
600
800
1000
1200
1400

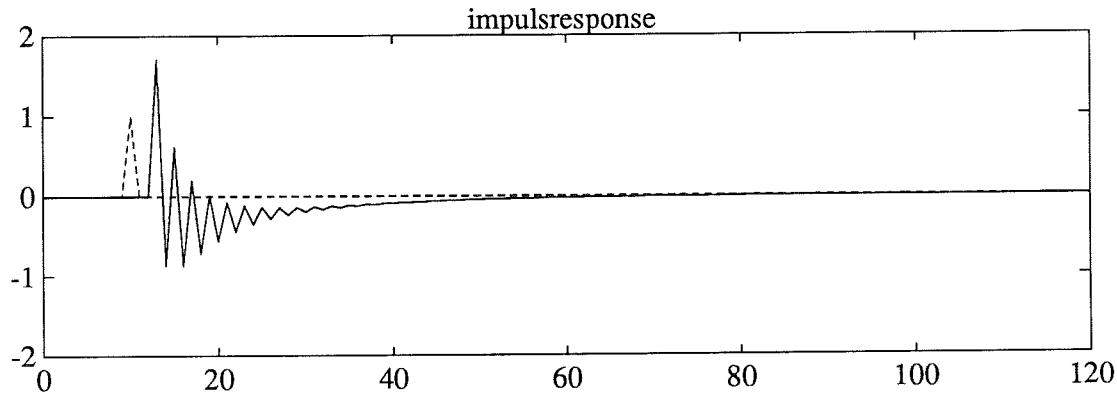

autocorrelation (e) test 904
1
0.5
0
-0.5
0
200
400
600
800
1000
1200
1400

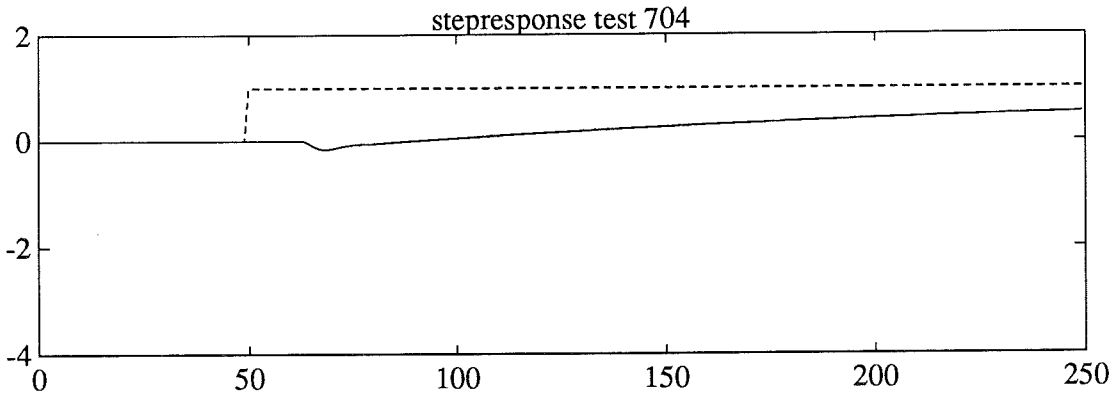

crosscorrelation (e-u)
4
2
0
-2
-4
0
200
400
600
800
1000
1200
1400

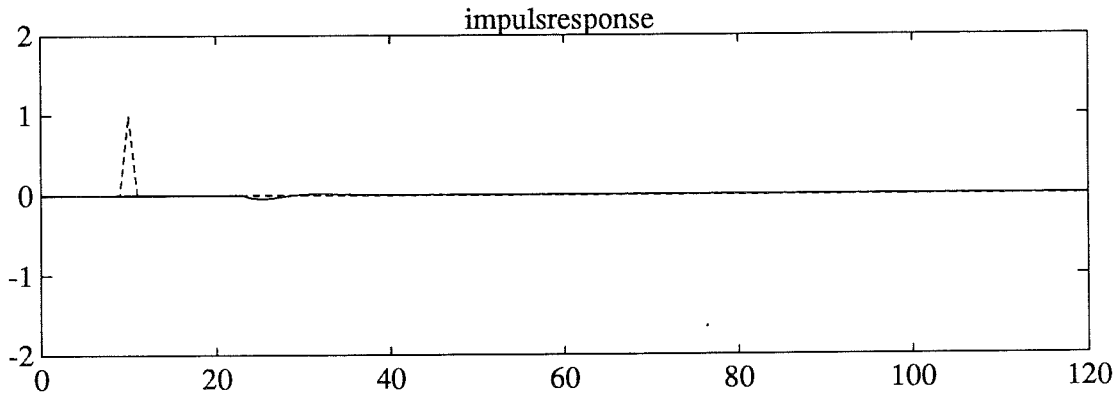

autocorrelation (e) test 1204
2
1.5
1
0.5
0
0
200
400
600
800
1000
1200
1400

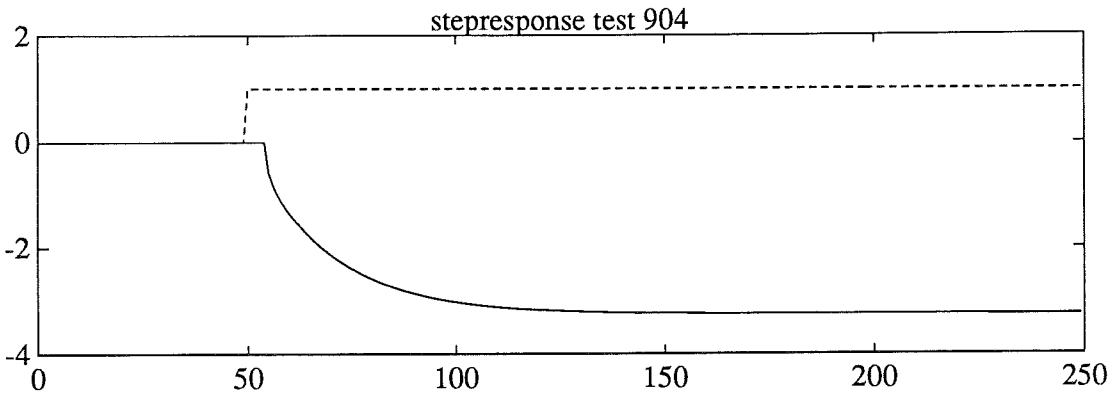

crosscorrelation (e-u)
2
0
-2
-4
-6
0
200
400
600
800
1000
1200
1400

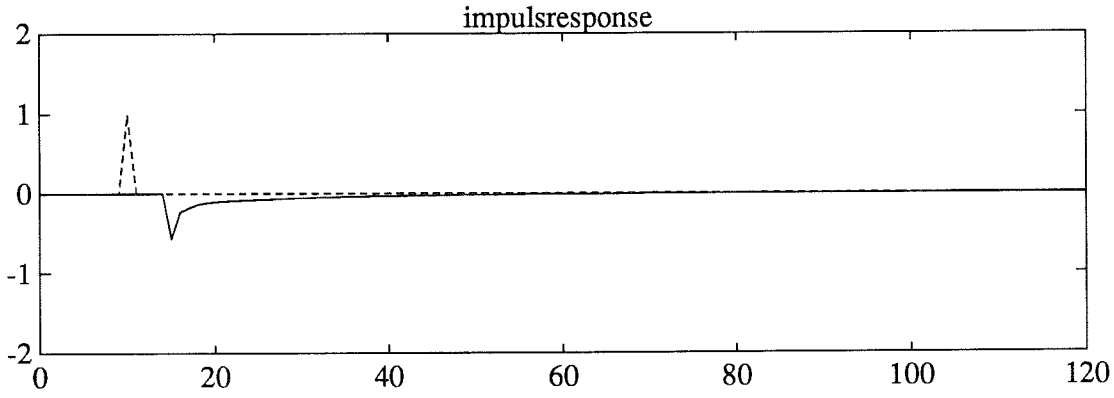

stepresponse test 404
2
0
-2
-4
0
50
100
150
200
250

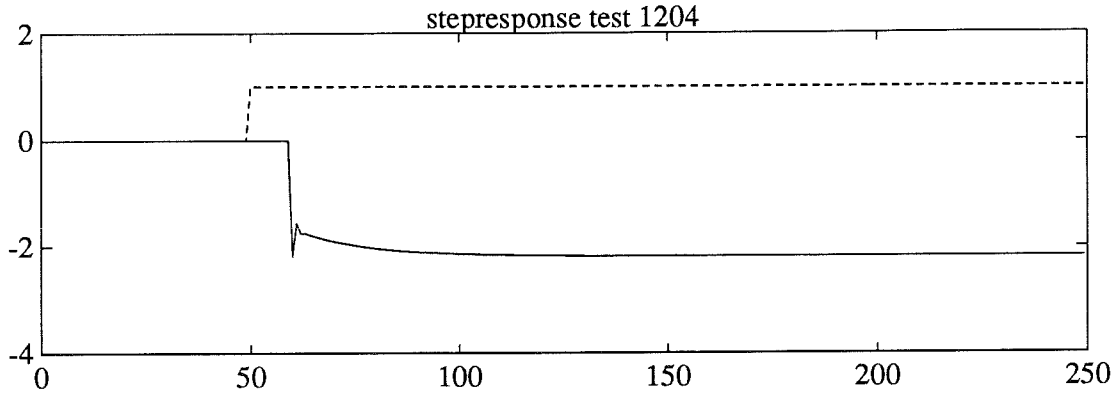

impulsresponse
2
1
0
-1
-2
0
20
40
60
80
100
120

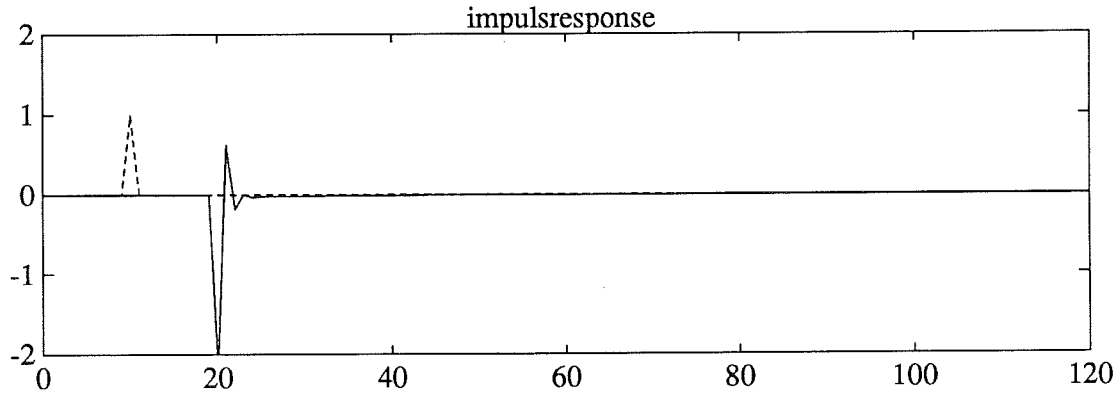

stepresponse test 704
2
0
-2
-4
0
50
100
150
200
250

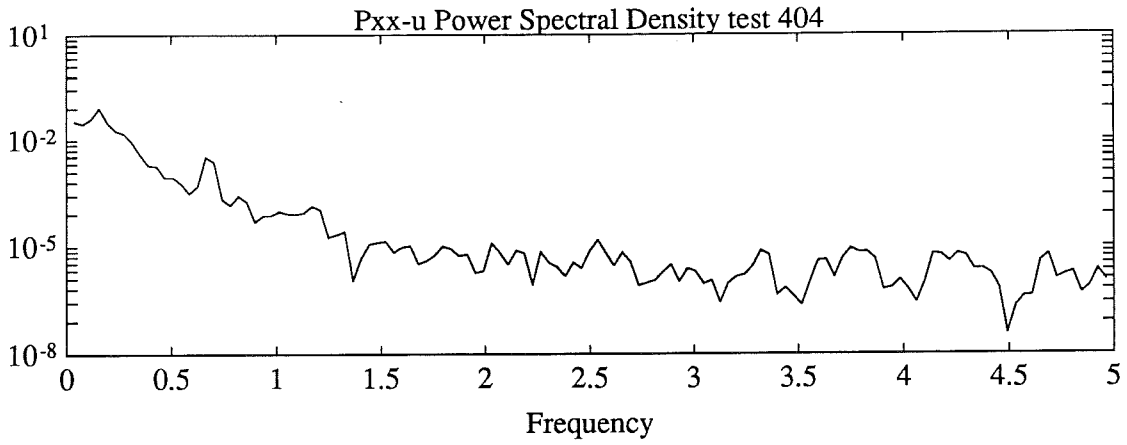

impulsresponse
2
1
0
-1
-2
0
20
40
60
80
100
120

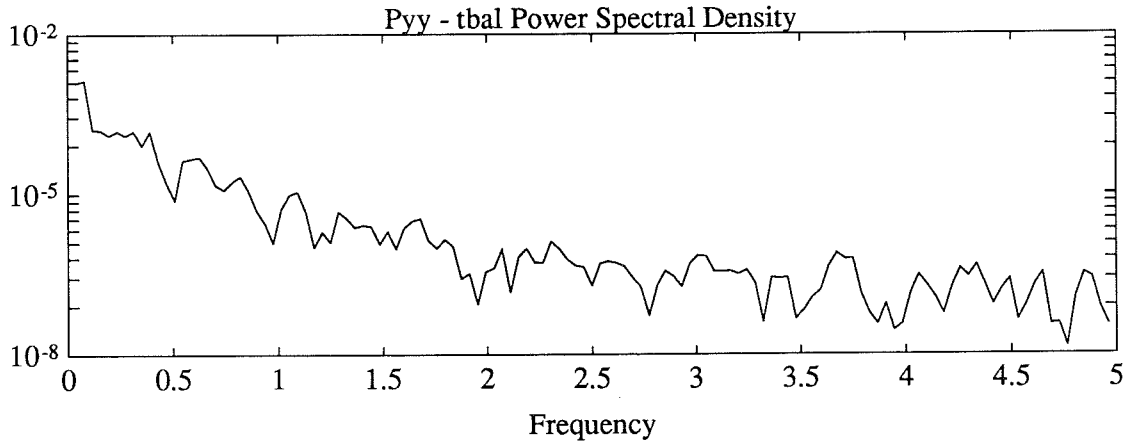

stepresponse test 904
2
0
-2
-4
0
50
100
150
200
250

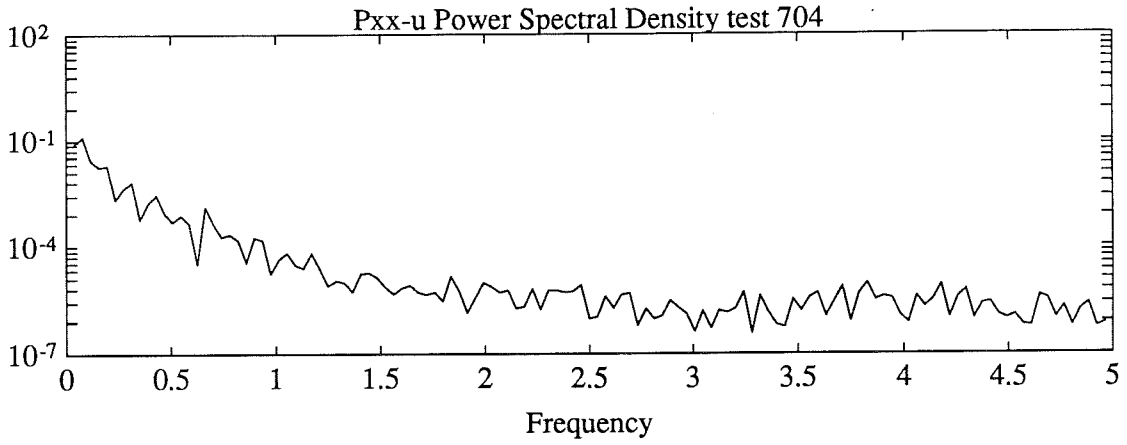

impulsresponse
2
1
0
-1
-2
0
20
40
60
80
100
120

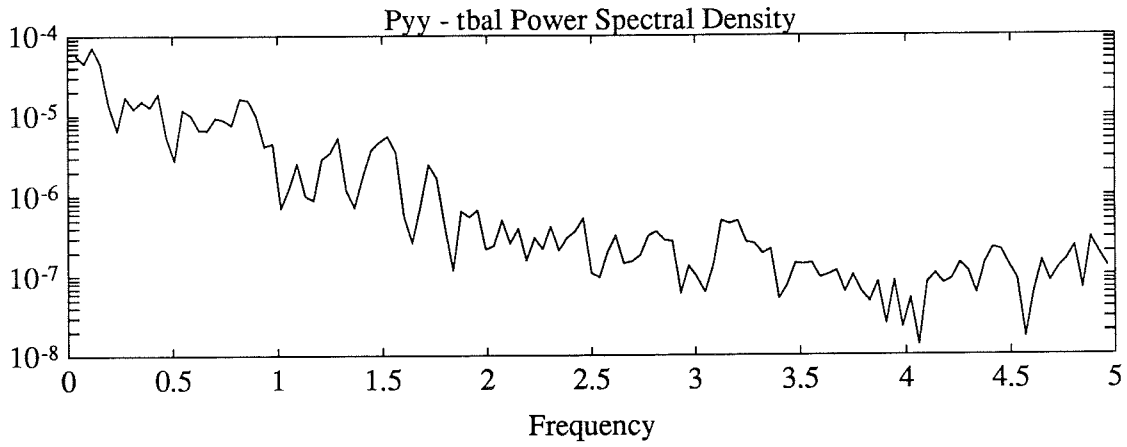

stepresponse test 1204
2
0
-2
-4
0
50
100
150
200
250

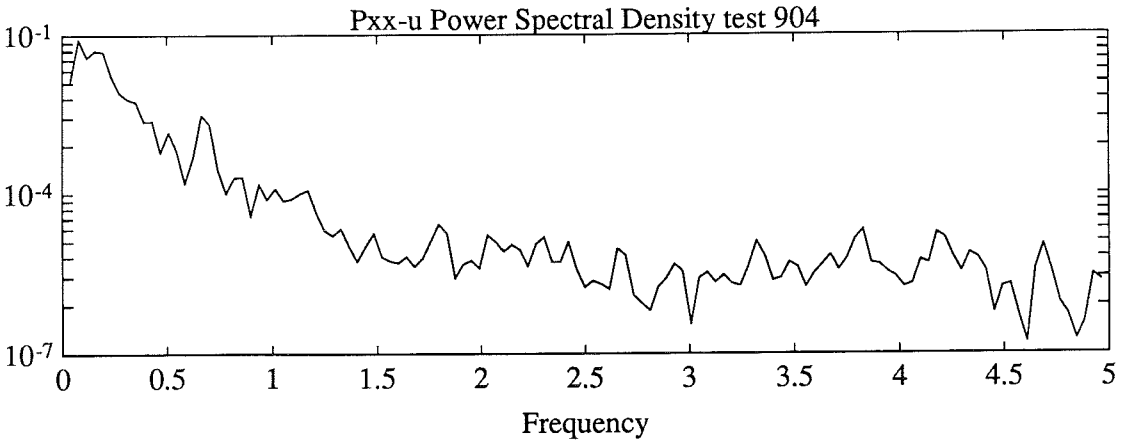

impulsresponse
2
1
0
-1
-2
0
20
40
60
80
100
120

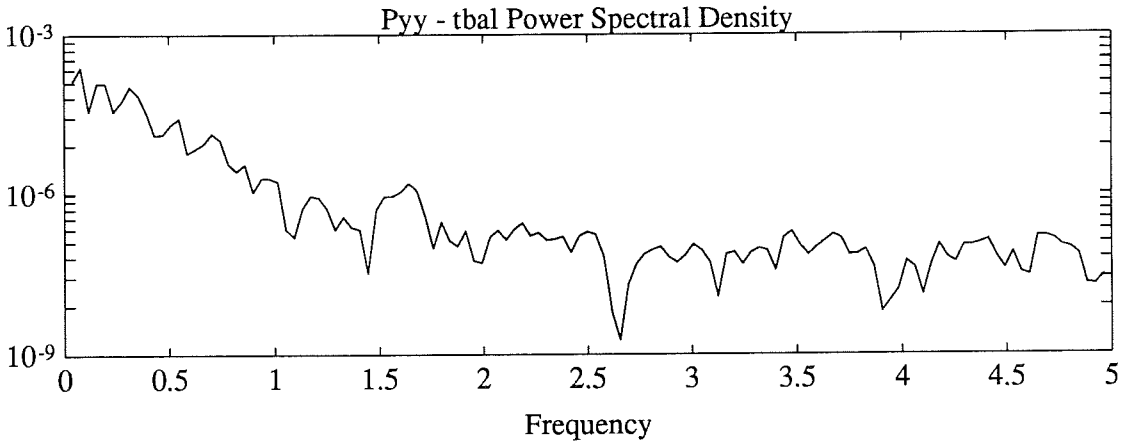

Pxx-u Power Spectral Density test 404
10^1
10^-2
10^-5
10^-8
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

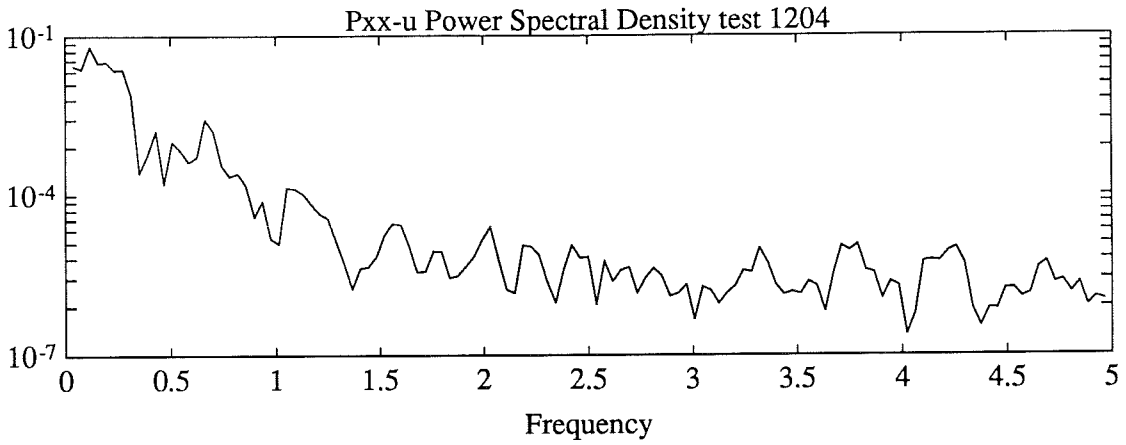

Pyy - tbal Power Spectral Density
10^-2
10^-5
10^-8
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

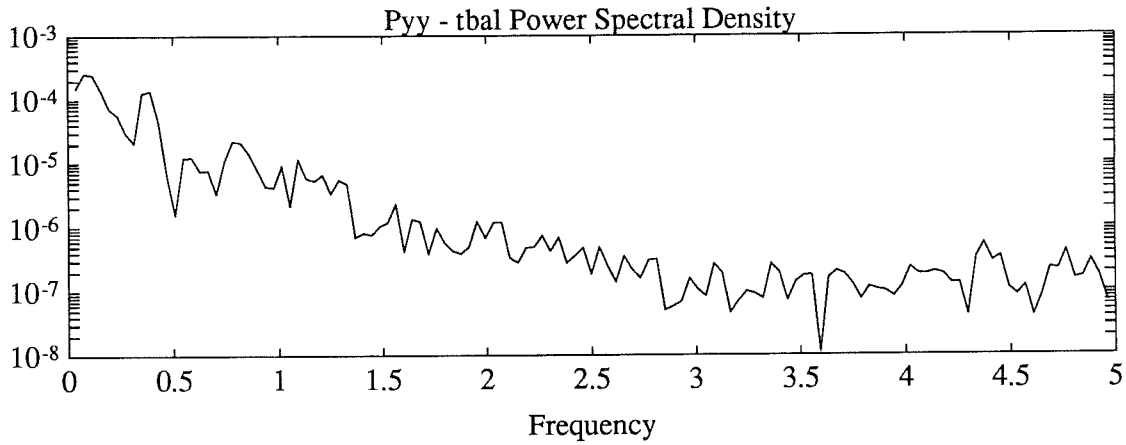

Pxx-u Power Spectral Density test 704
10^2
10^-1
10^-4
10^-7
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

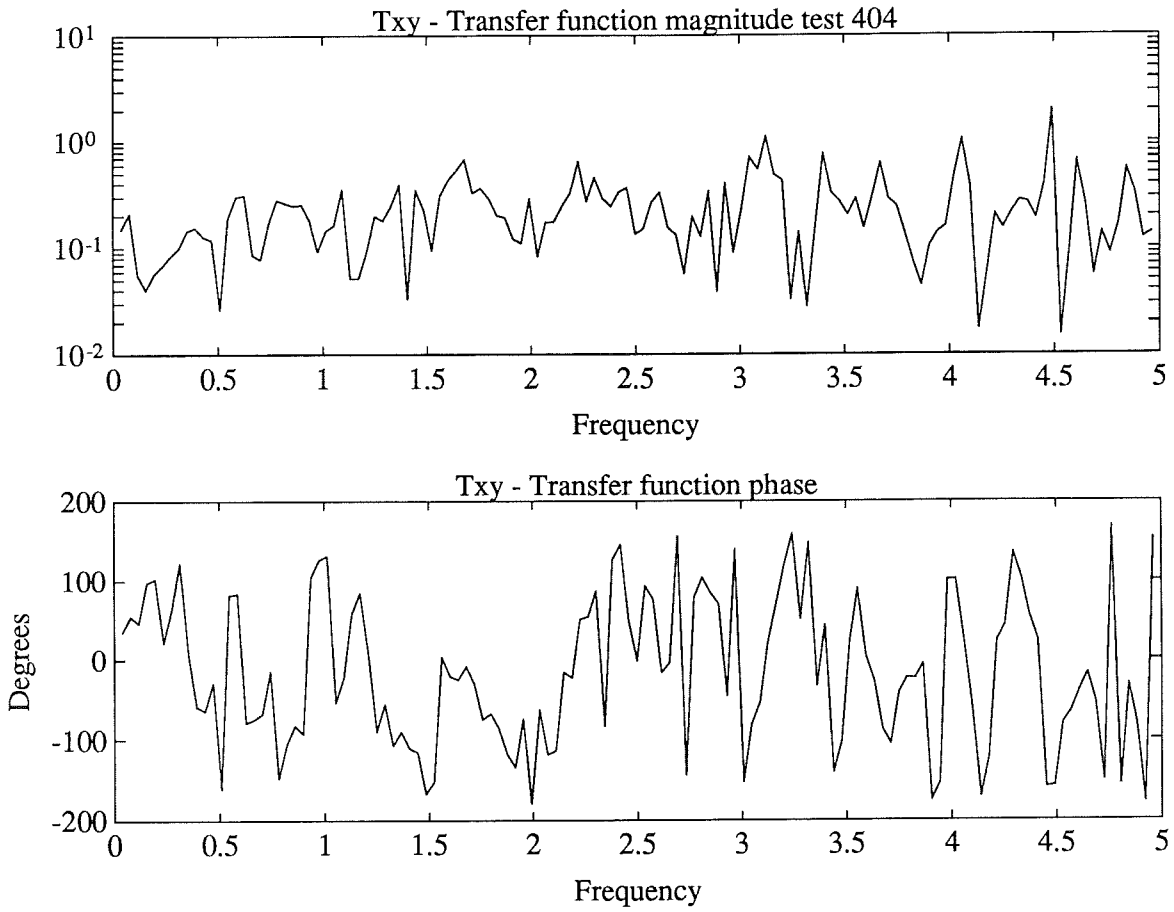

Pyy - tbal Power Spectral Density
10^-4
10^-5
10^-6
10^-7
10^-8
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

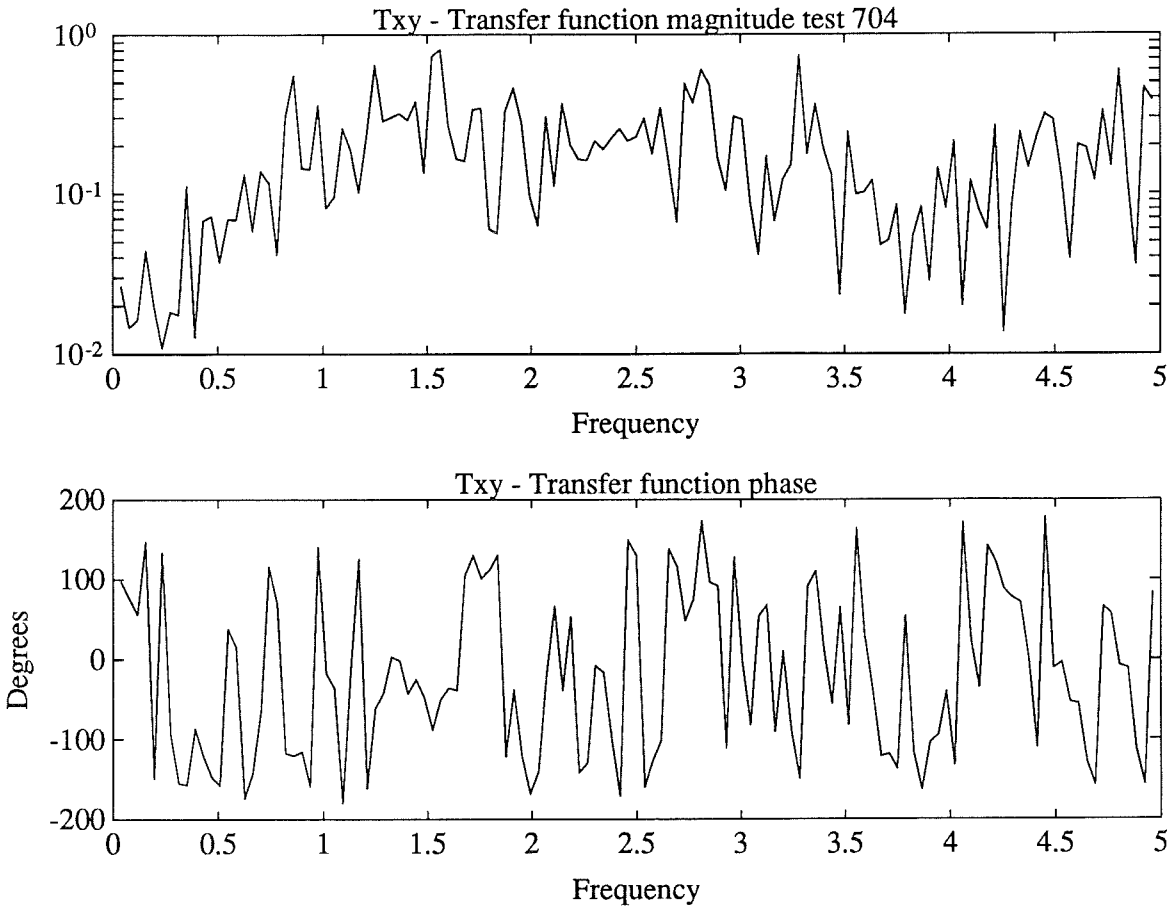

Pxx-u Power Spectral Density test 904
10-1
10-4
10-7
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

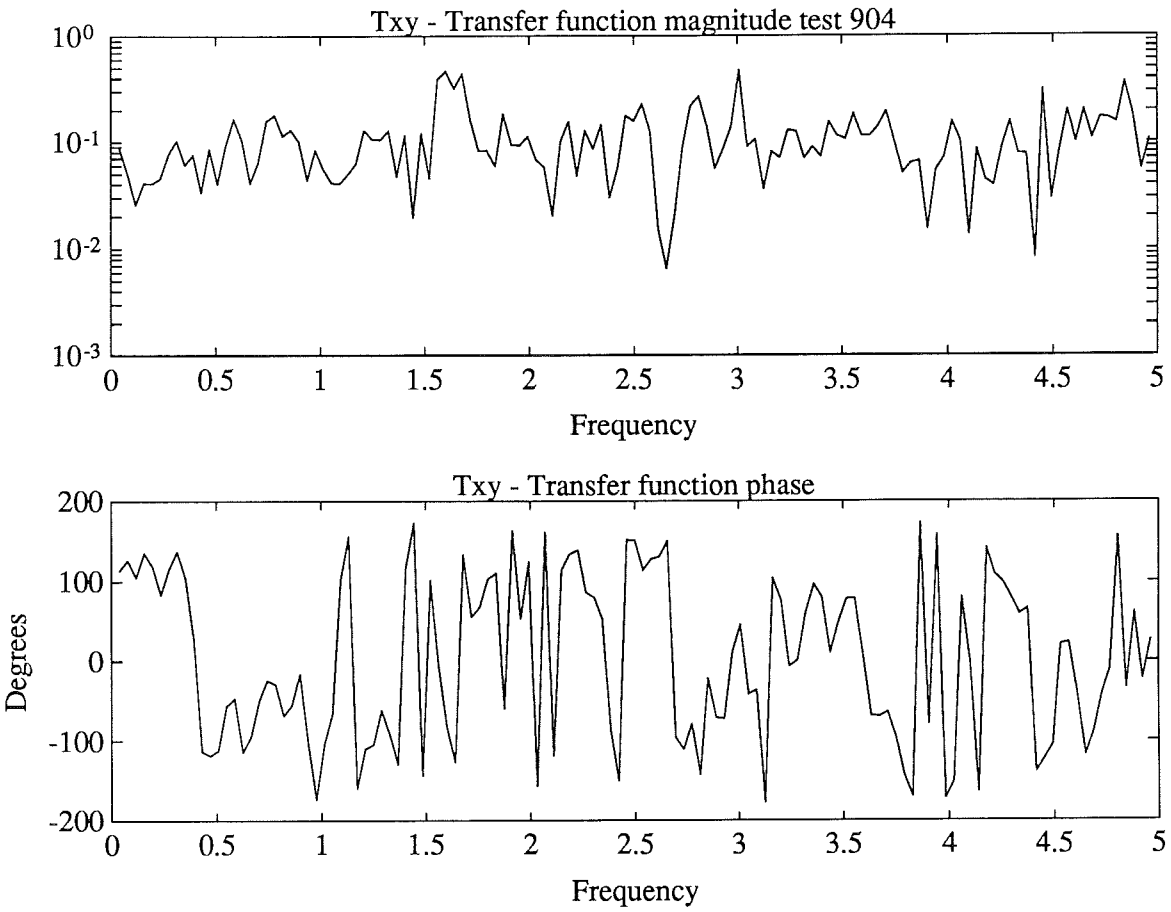

Pyy - tbal Power Spectral Density
10-3
10-6
10-9
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

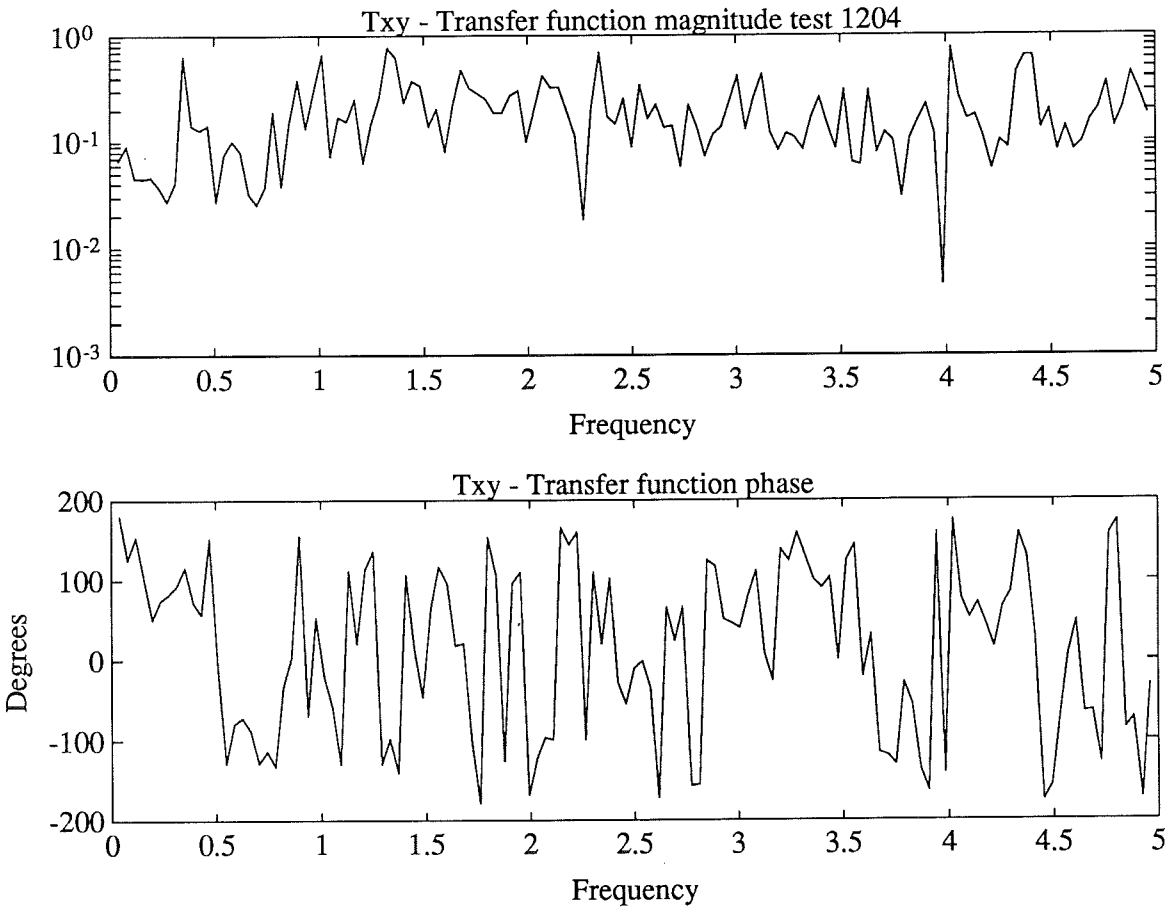

Pxx-u Power Spectral Density test 1204
10-1
10-4
10-7
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

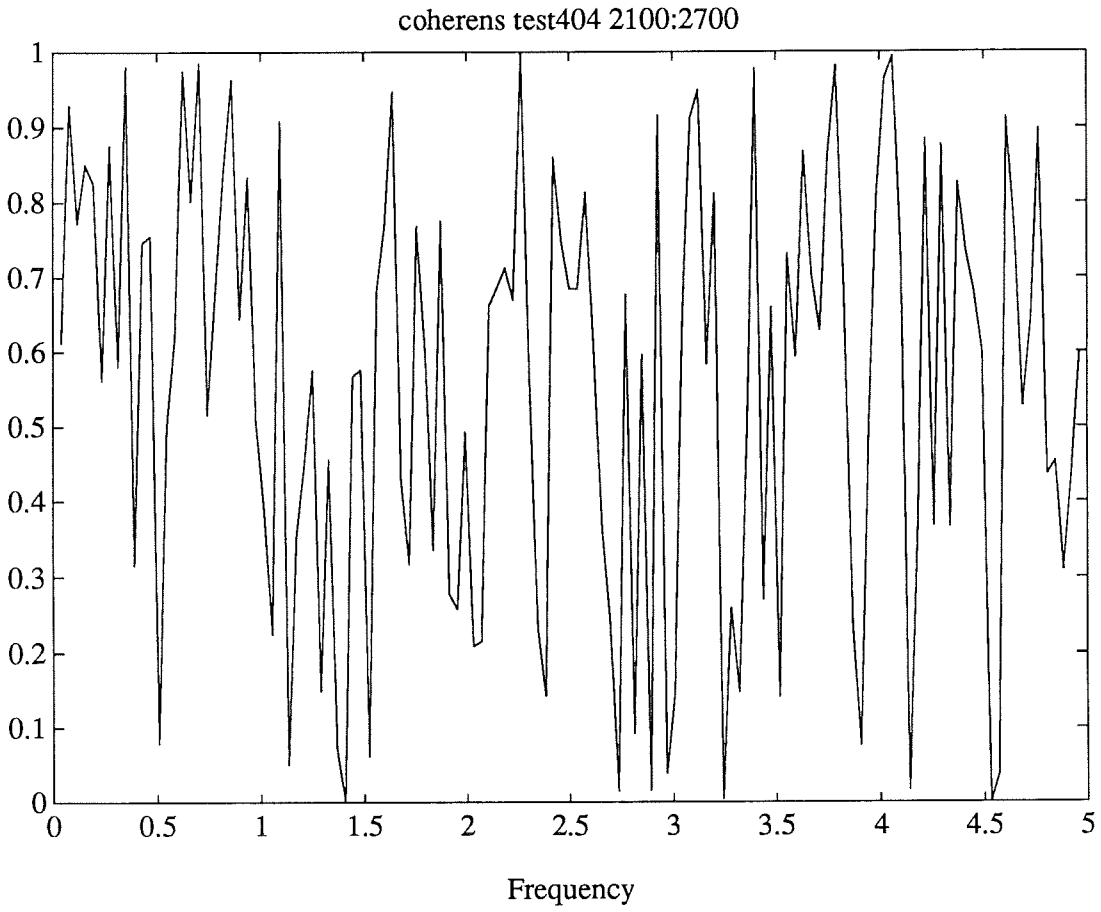

Pyy - tbal Power Spectral Density
10-3
10-4
10-5
10-6
10-7
10-8
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

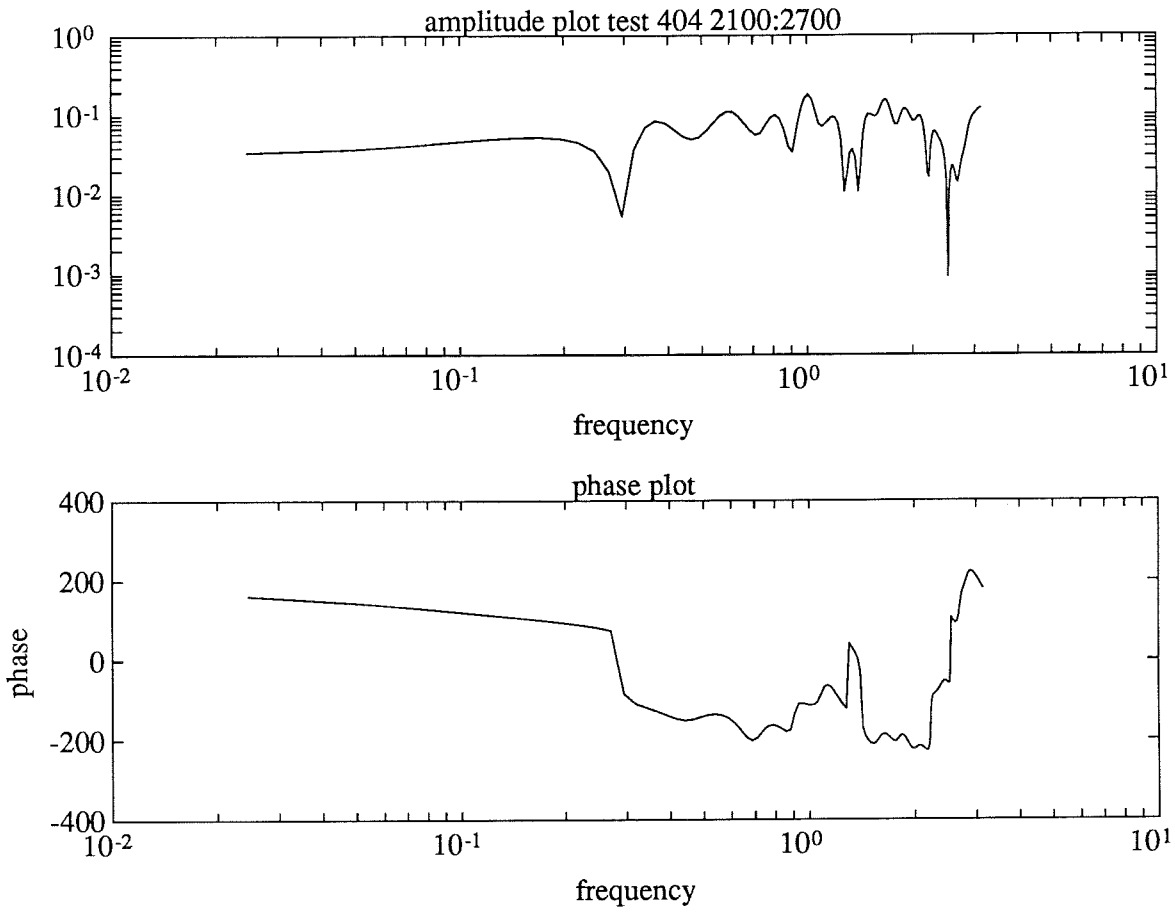

Txy - Transfer function magnitude test 404
10^1
10^0
10^{-1}
10^{-2}
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency
Txy - Transfer function phase
200
100
0
-100
-200
Degrees
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

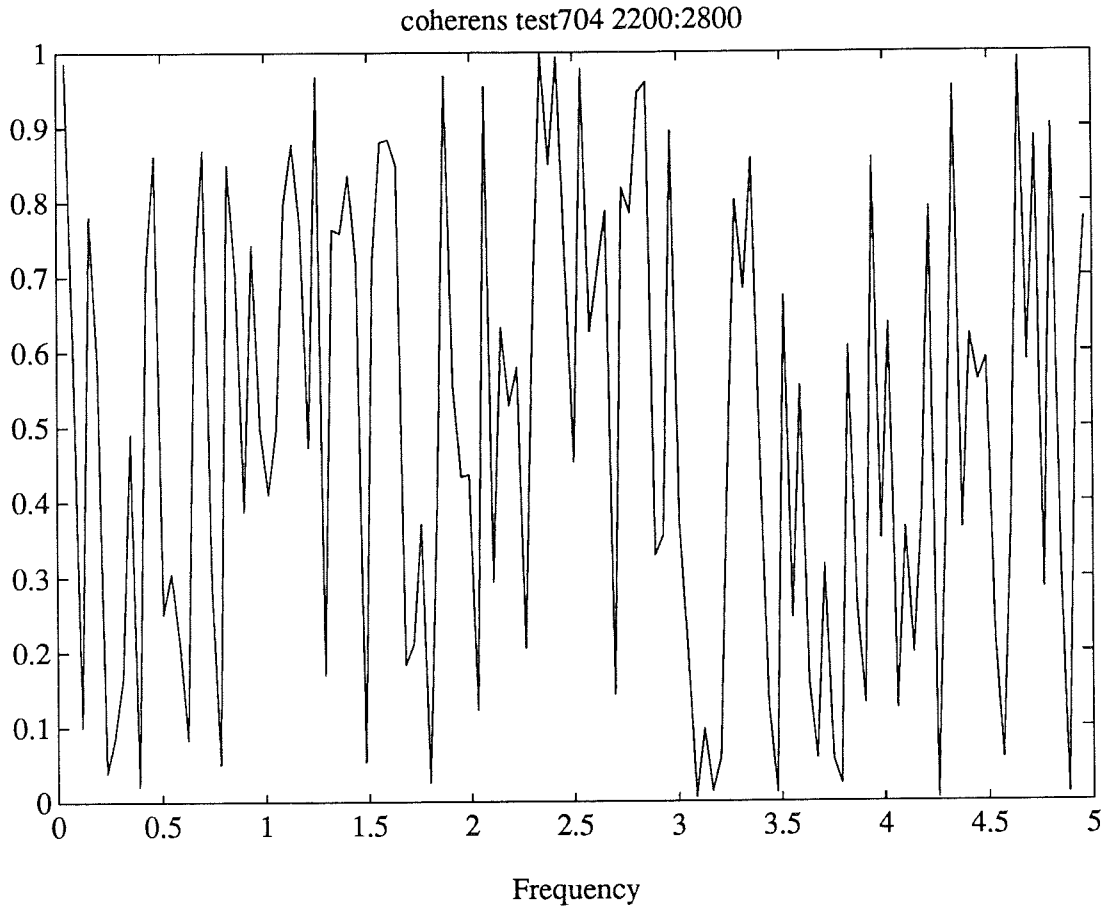

Txy - Transfer function magnitude test 704
10^0
10^{-1}
10^{-2}
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency
Txy - Transfer function phase
200
100
0
-100
-200
Degrees
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

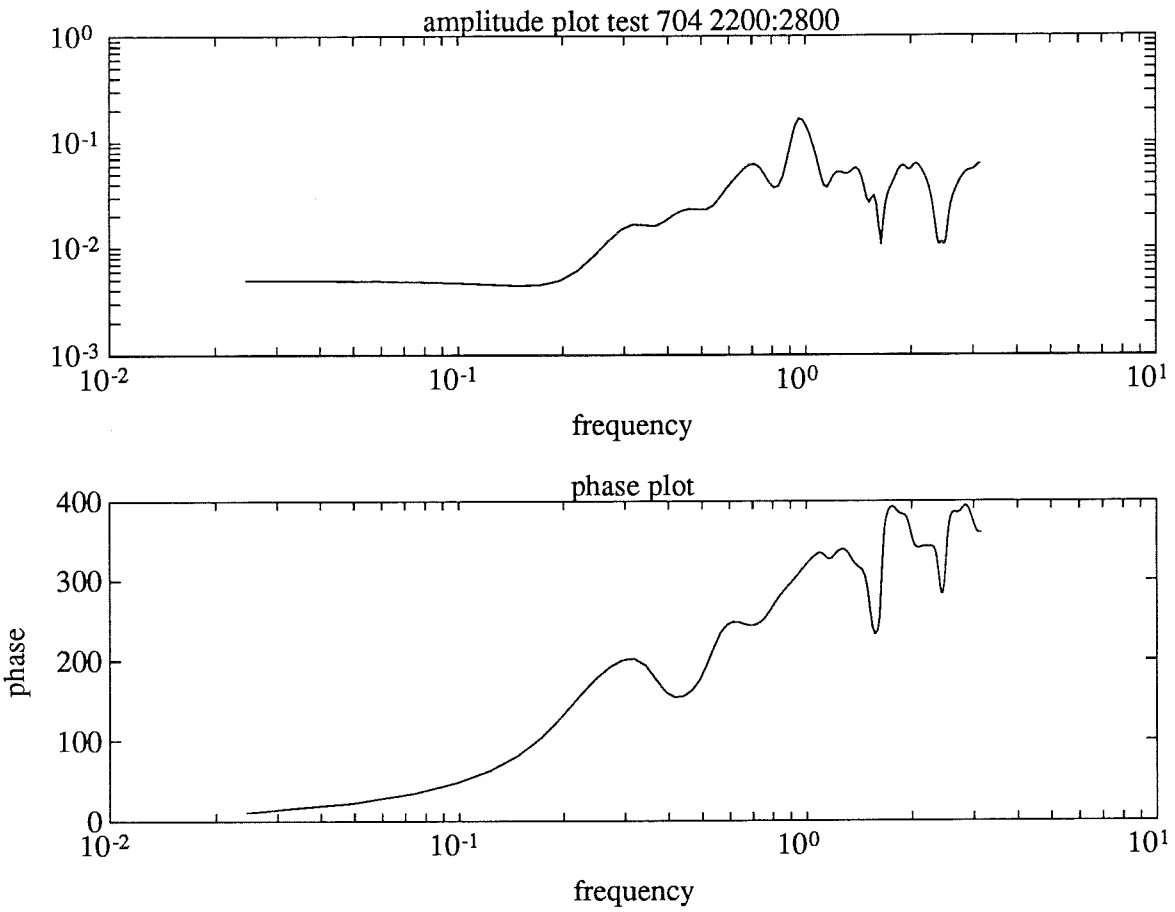

Txy - Transfer function magnitude test 904
10^0
10^-1
10^-2
10^-3
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency
Txy - Transfer function phase
Degrees
200
100
0
-100
-200
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

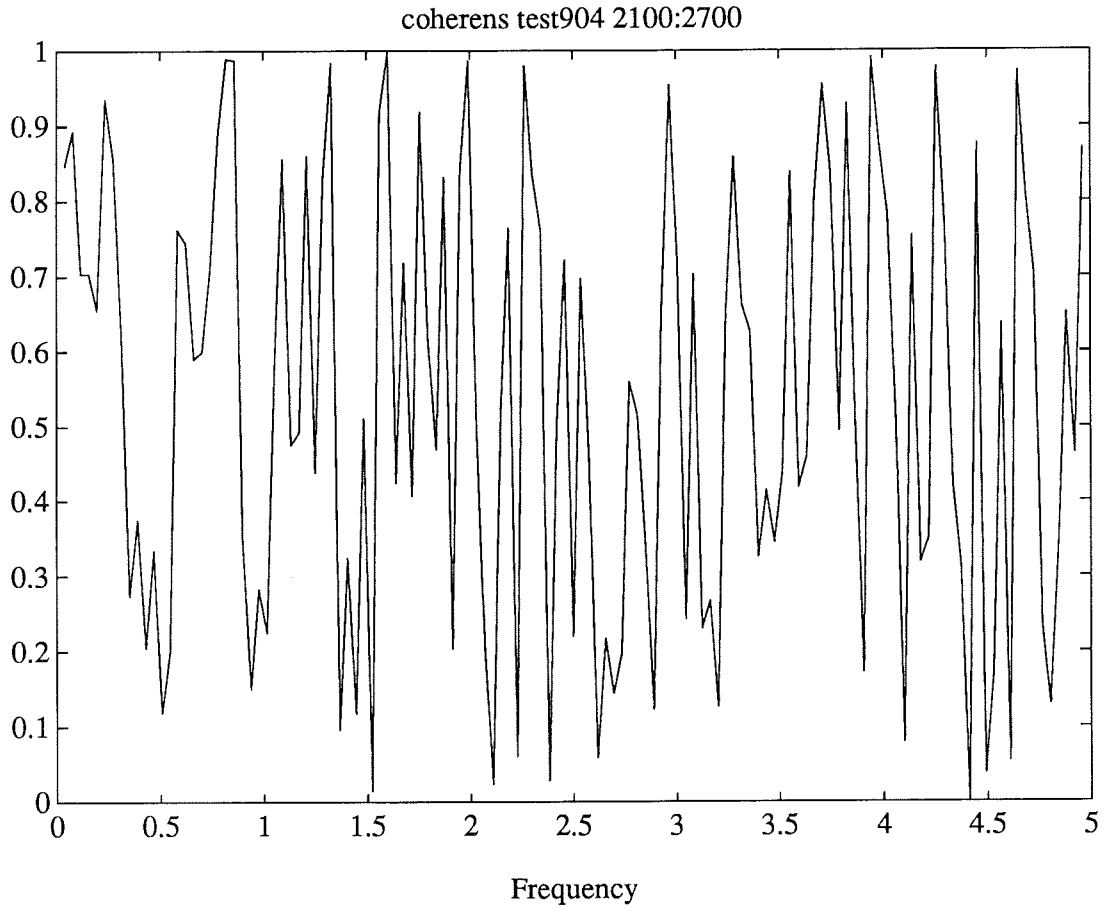

Txy - Transfer function magnitude test 1204
10^0
10^-1
10^-2
10^-3
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency
Txy - Transfer function phase
Degrees
200
100
0
-100
-200
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

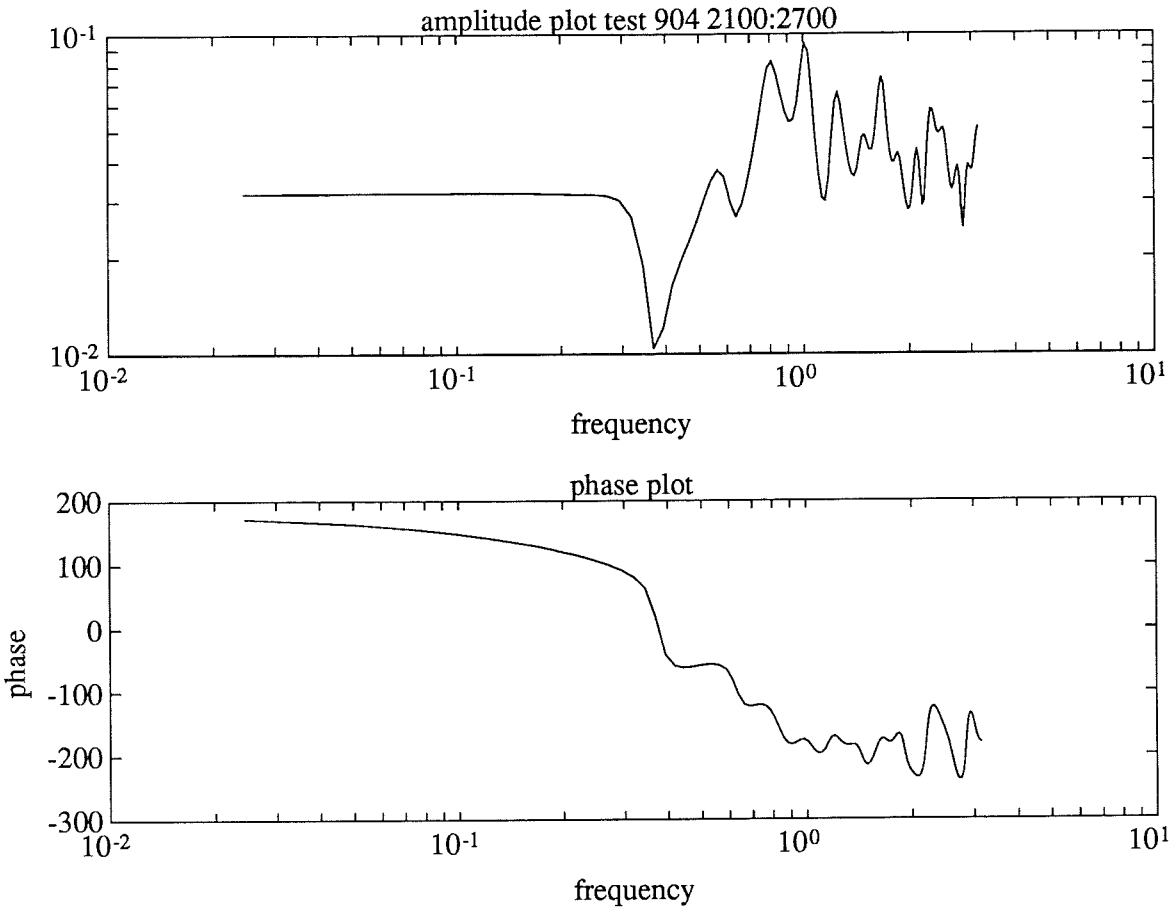

coherens test404 2100:2700
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

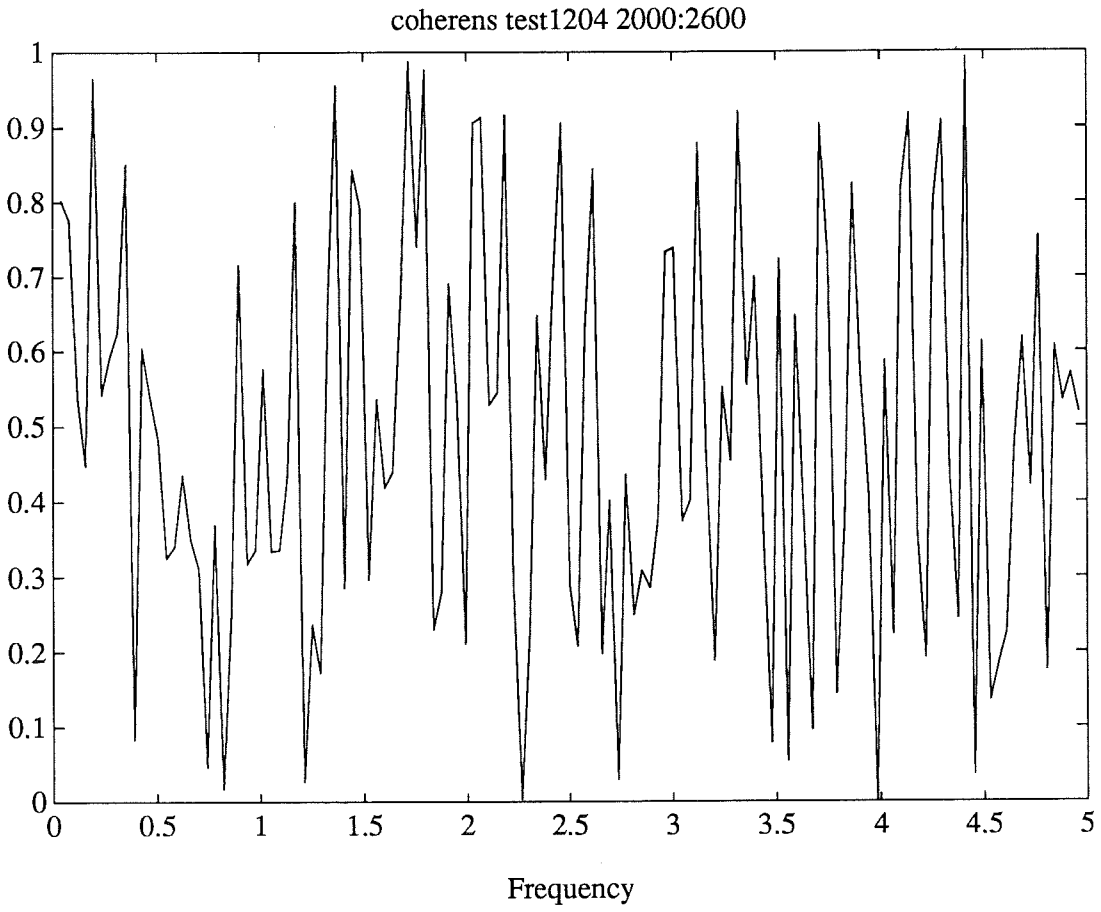

amplitude plot test 404 2100:2700
10^0
10^-1
10^-2
10^-3
10^-4
10^-2
10^-1
10^0
10^1
frequency
phase plot
400
200
0
-200
-400
phase
10^-2
10^-1
10^0
10^1
frequency

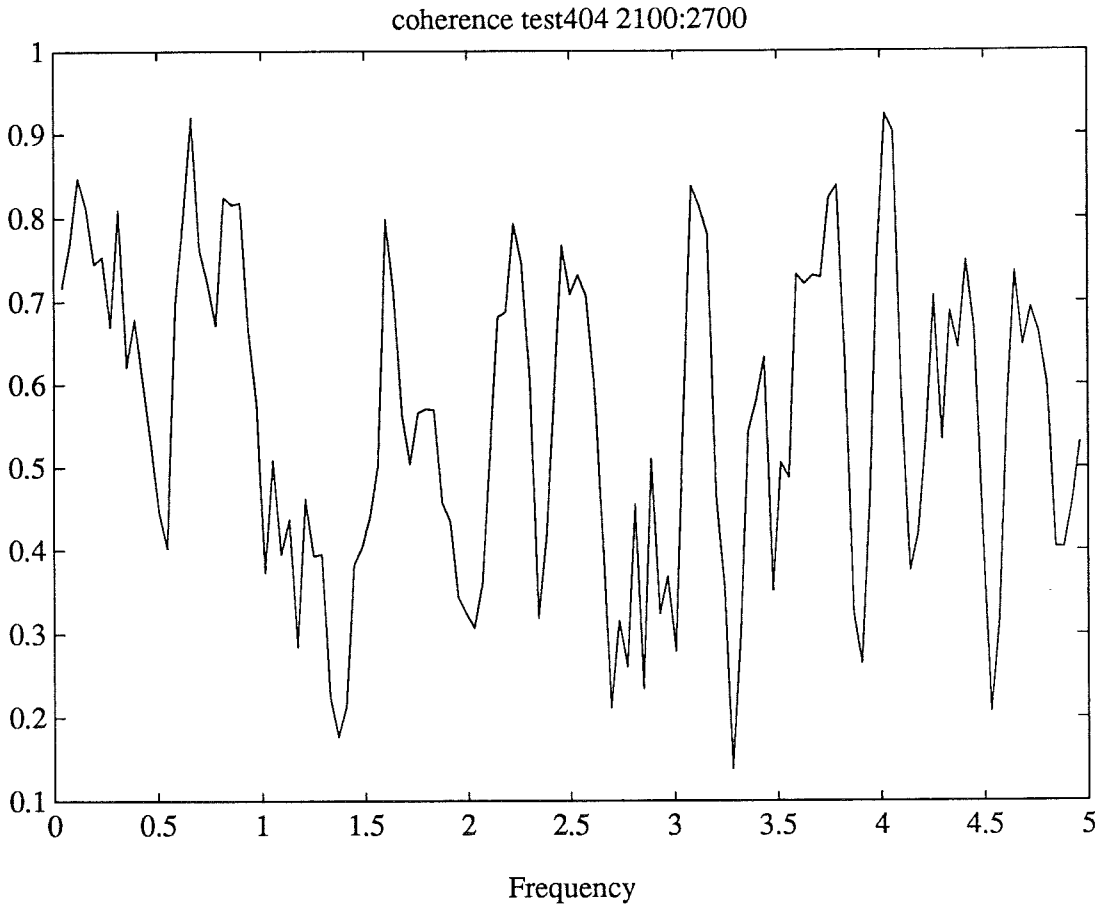

coherens test704 2200:2800
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

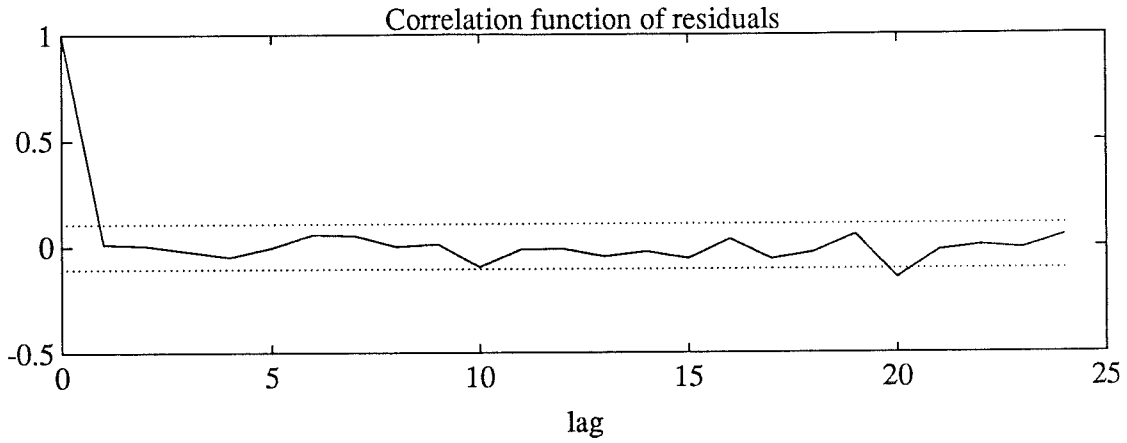

amplitude plot test 704 2200:2800
10^0
10^-1
10^-2
10^-3
10^-2
10^-1
10^0
10^1
frequency
phase plot
400
300
200
100
0
phase
10^-2
10^-1
10^0
10^1
frequency

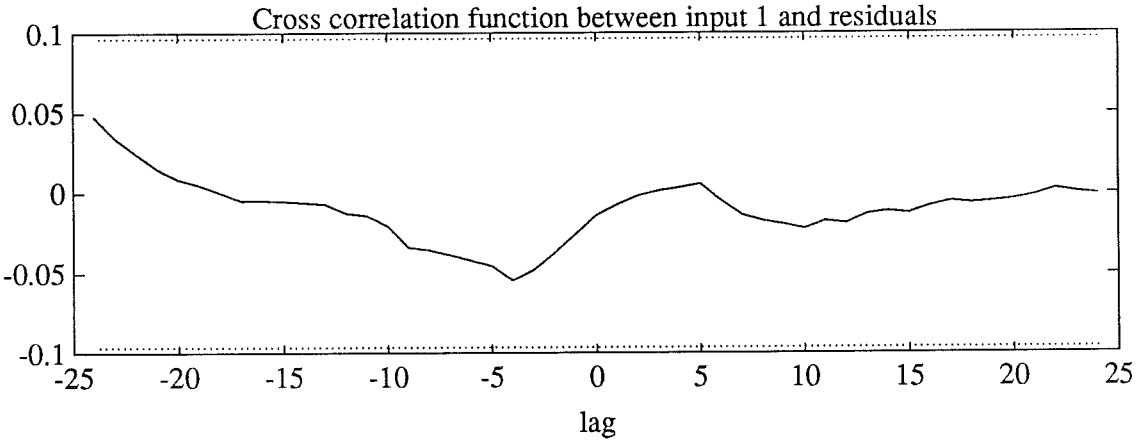

coherens test904 2100:2700
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

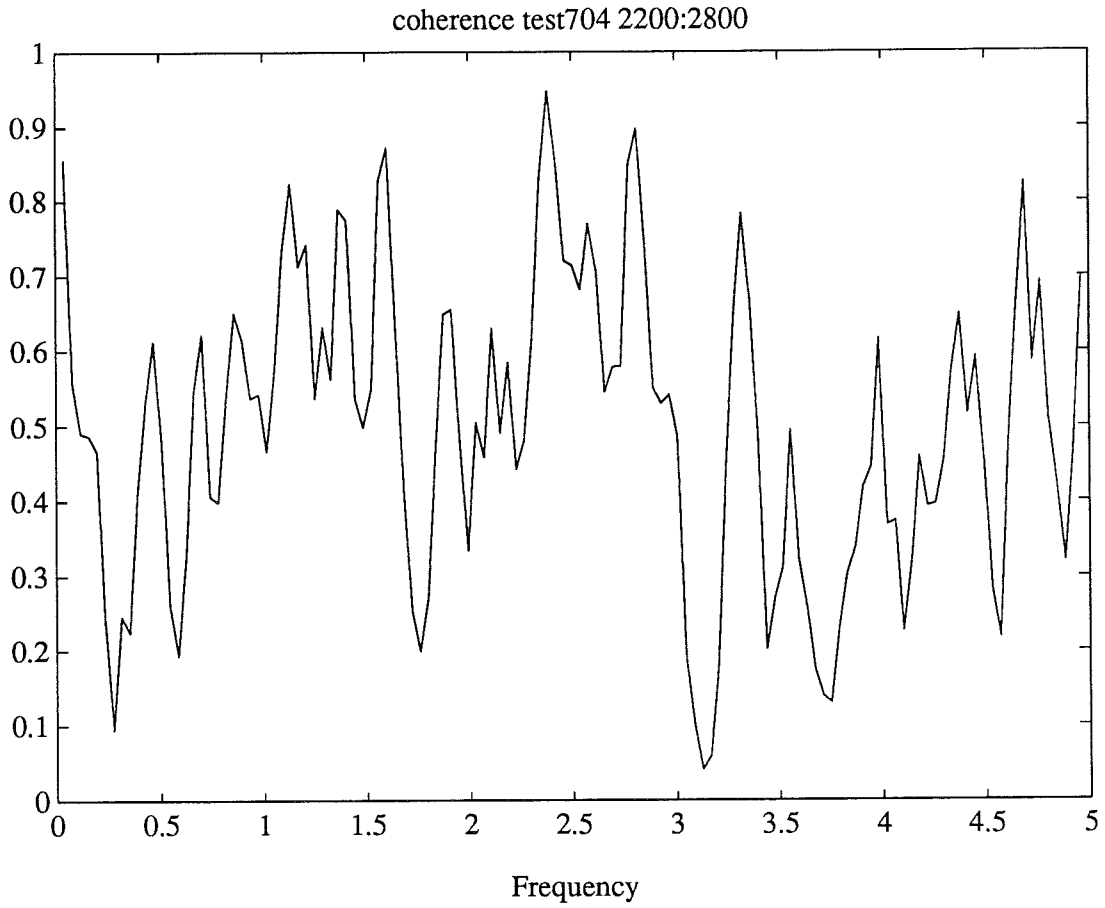

amplitude plot test 904 2100:2700
10-1
10-2
10-2
10-1
100
101
frequency
phase plot
200
100
0
-100
-200
-300
phase
10-2
10-1
100
101
frequency

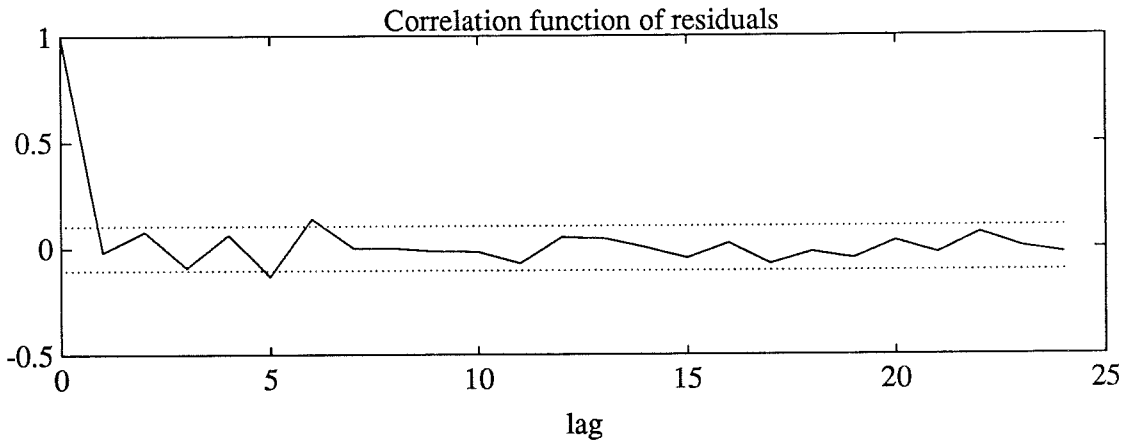

coherens test1204 2000:2600
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

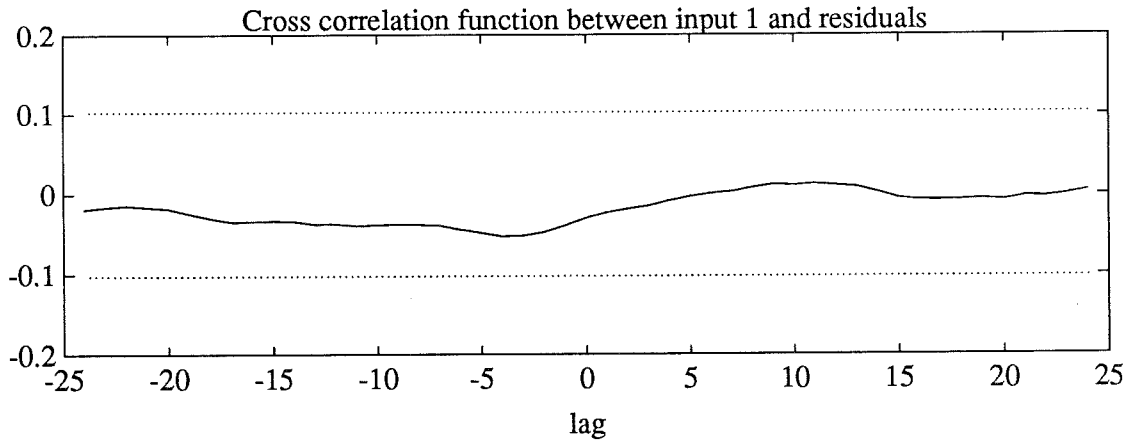

coherence test404 2100:2700
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

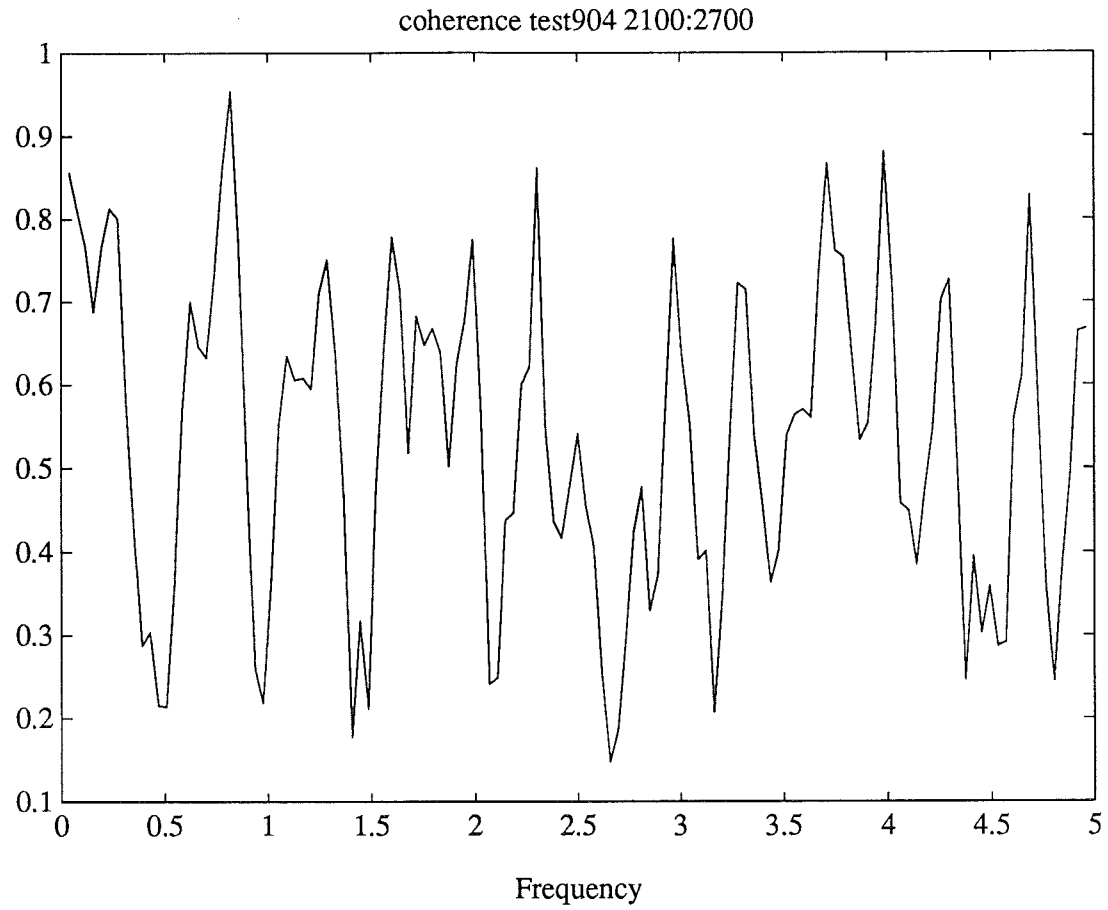

Correlation function of residuals
1
0.5
0
-0.5
0
5
10
15
20
25
lag

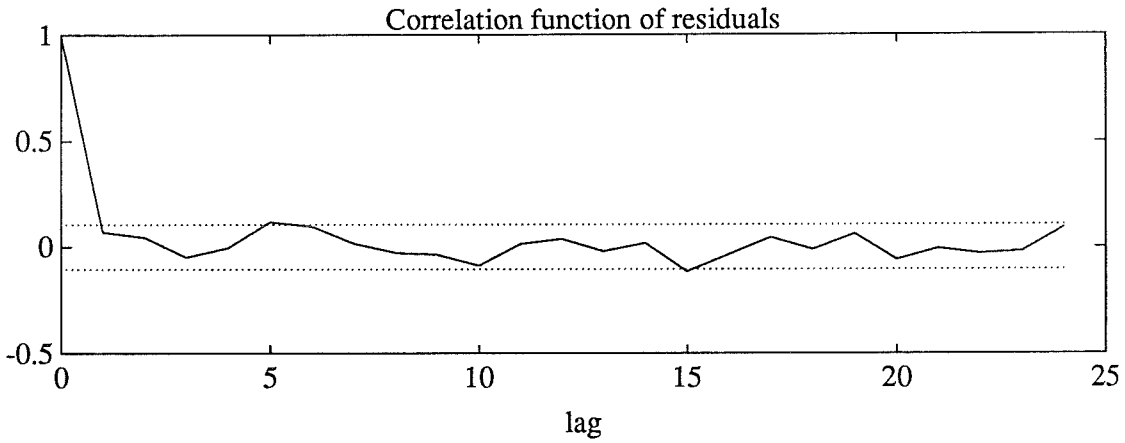

Cross correlation function between input 1 and residuals
0.1
0.05
0
-0.05
-0.1
-25
-20
-15
-10
-5
0
5
10
15
20
25
lag

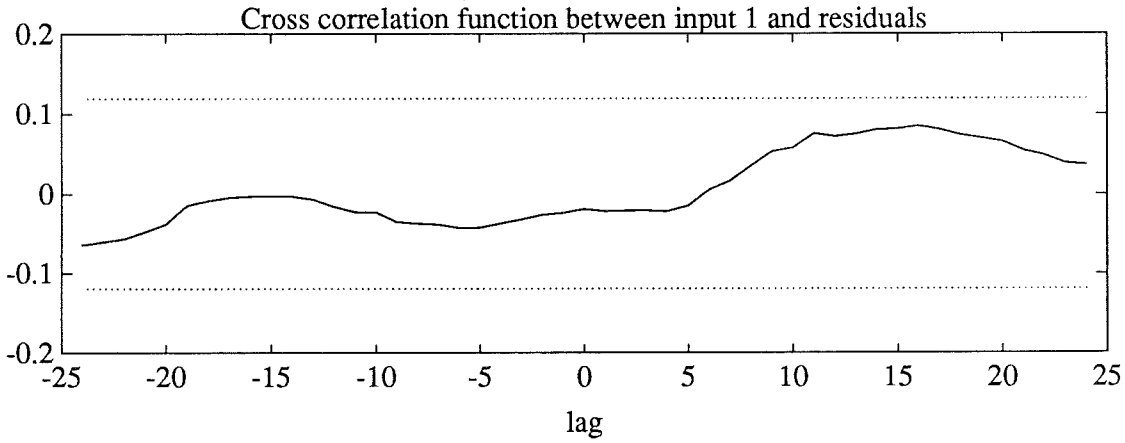

coherence test704 2200:2800
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

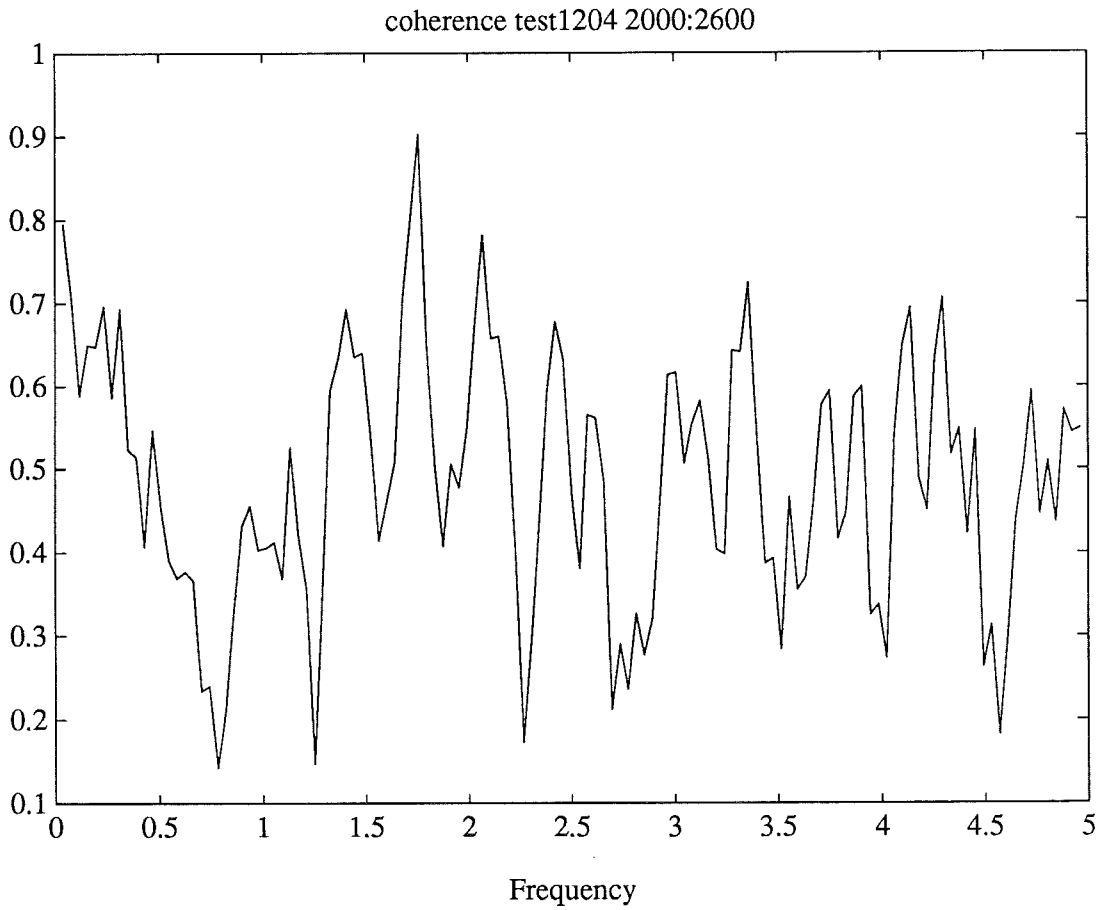

Correlation function of residuals
1
0.5
0
-0.5
0
5
10
15
20
25
lag

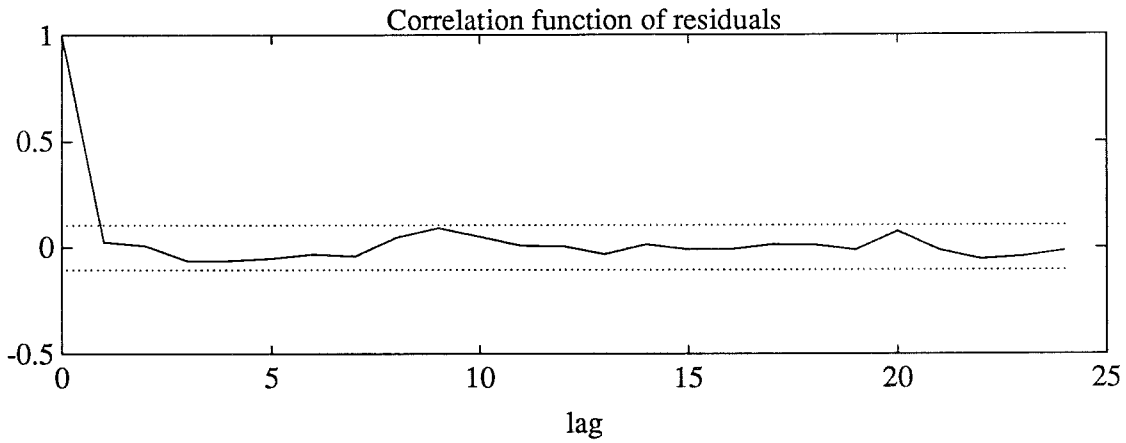

Cross correlation function between input 1 and residuals
0.2
0.1
0
-0.1
-0.2
-25
-20
-15
-10
-5
0
5
10
15
20
25
lag

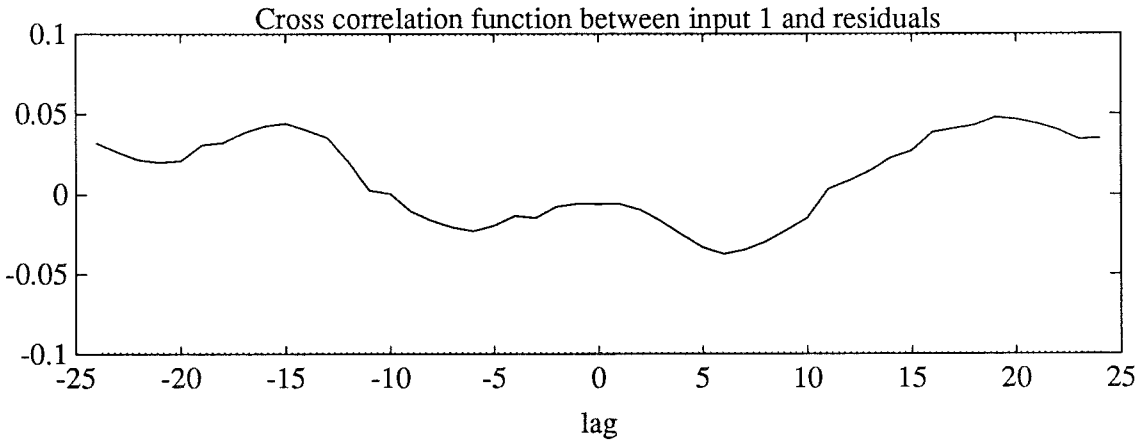

coherence test904 2100:2700
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

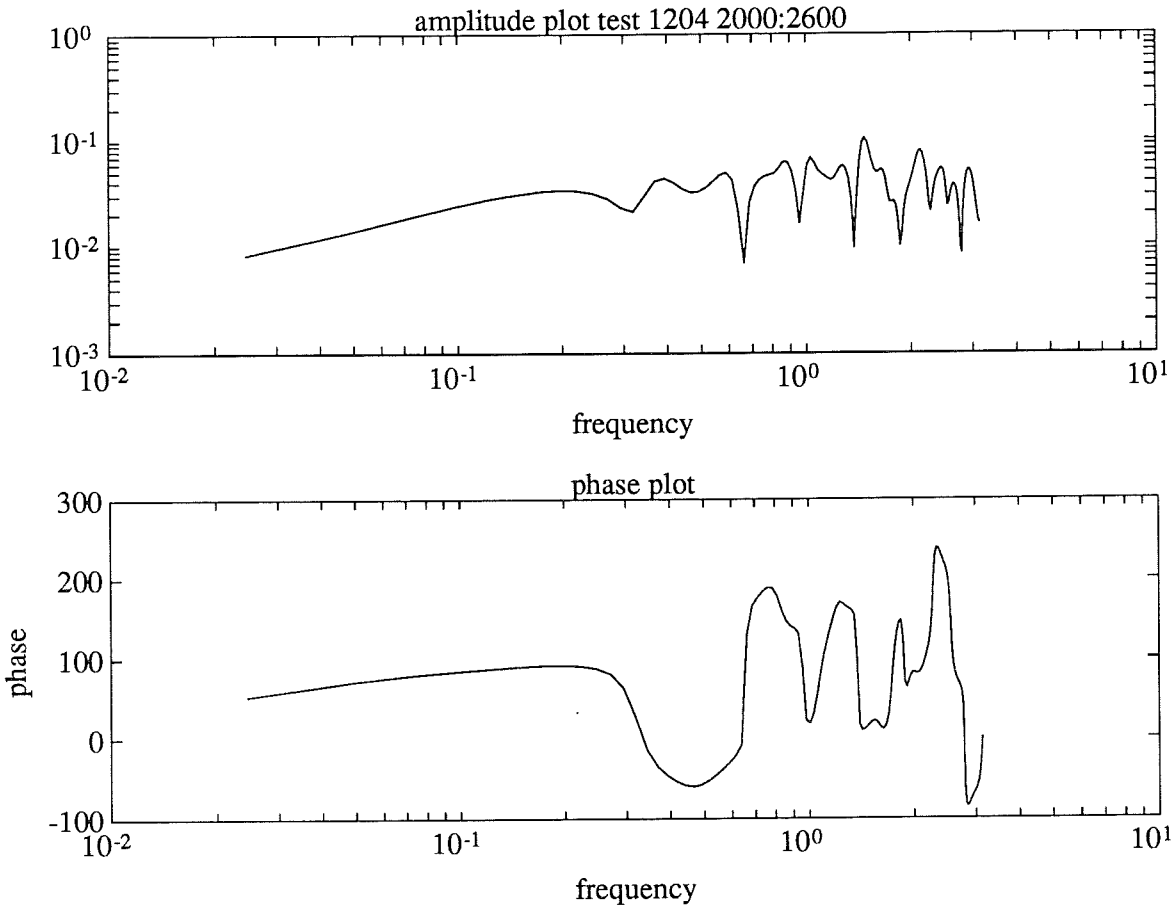

Correlation function of residuals
1
0.5
0
-0.5
0
5
10
15
20
25
lag

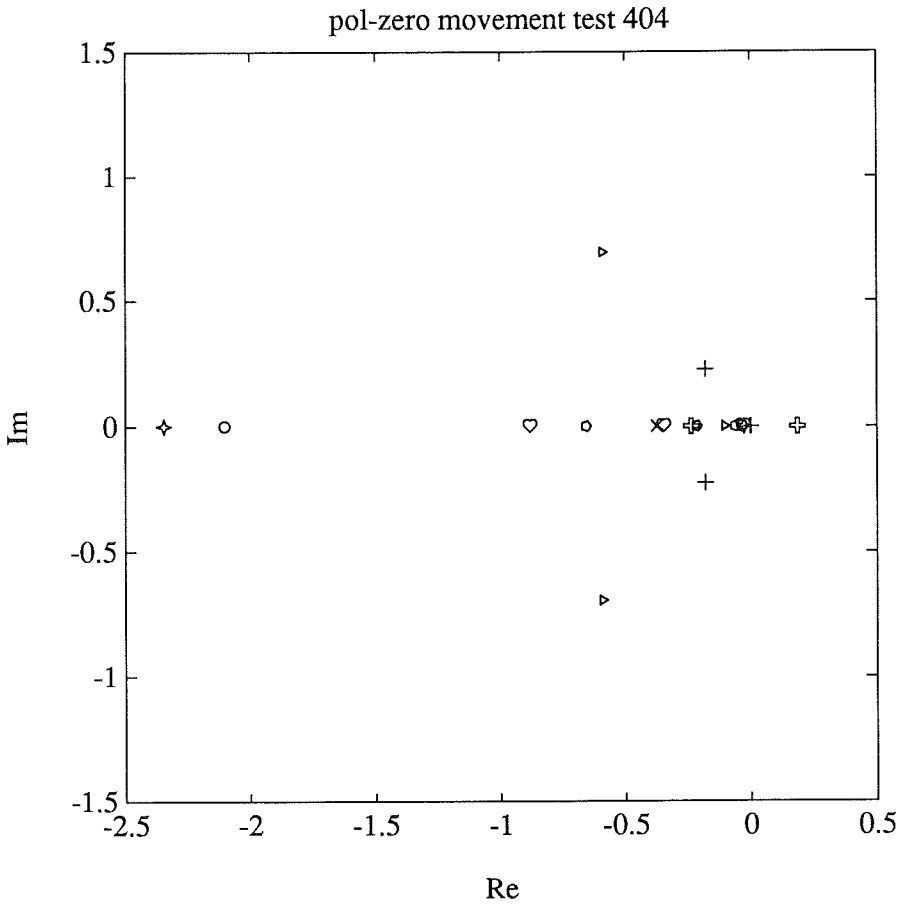

Cross correlation function between input 1 and residuals
0.2
0.1
0
-0.1
-0.2
-25
-20
-15
-10
-5
0
5
10
15
20
25
lag

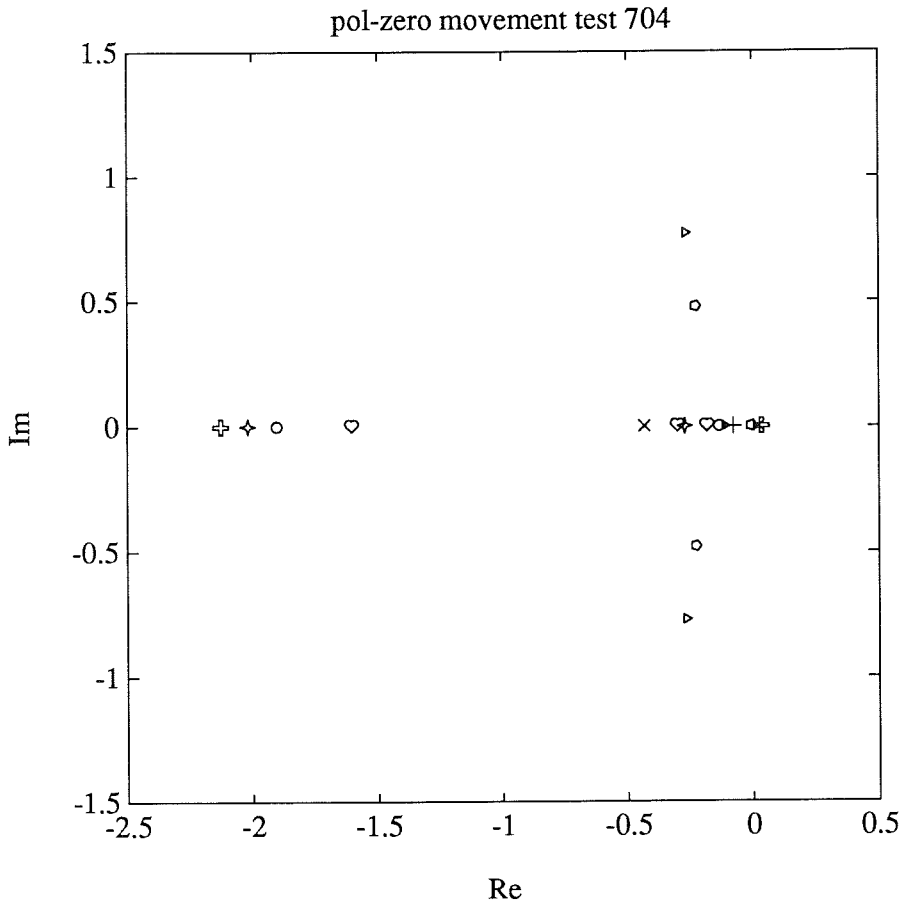

coherence test1204 2000:2600
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

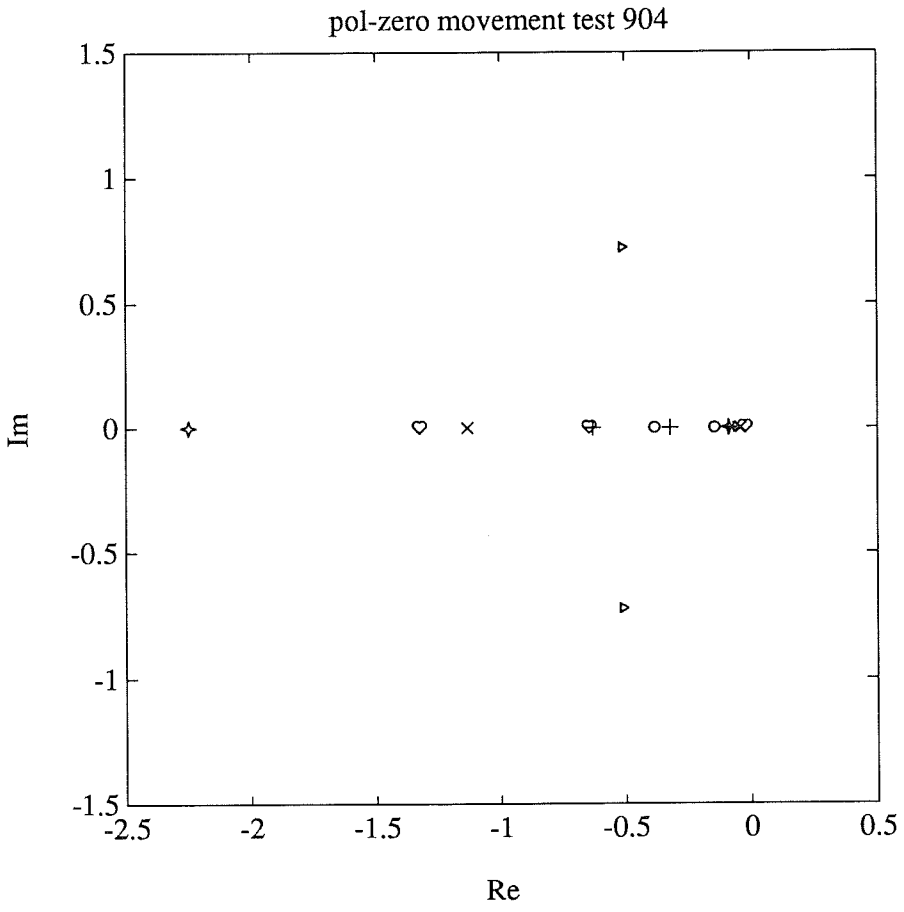

Correlation function of residuals
1
0.5
0
-0.5
0
5
10
15
20
25
lag

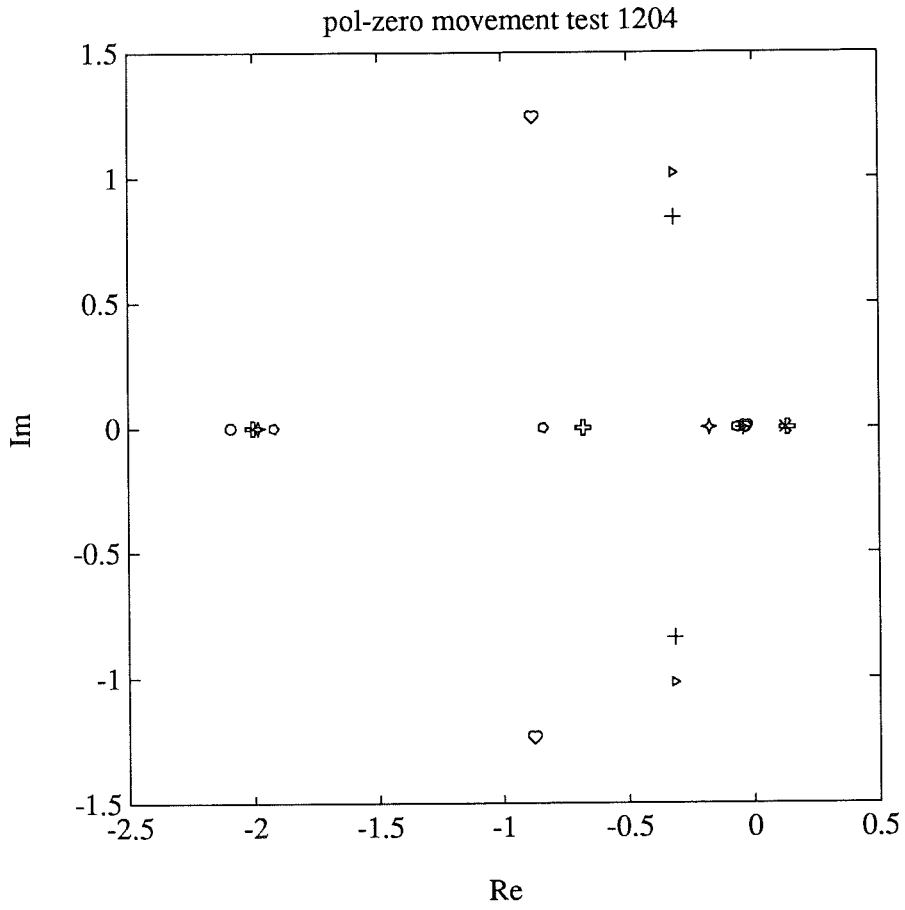

Cross correlation function between input 1 and residuals
0.1
0.05
0
-0.05
-0.1
-25
-20
-15
-10
-5
0
5
10
15
20
25
lag

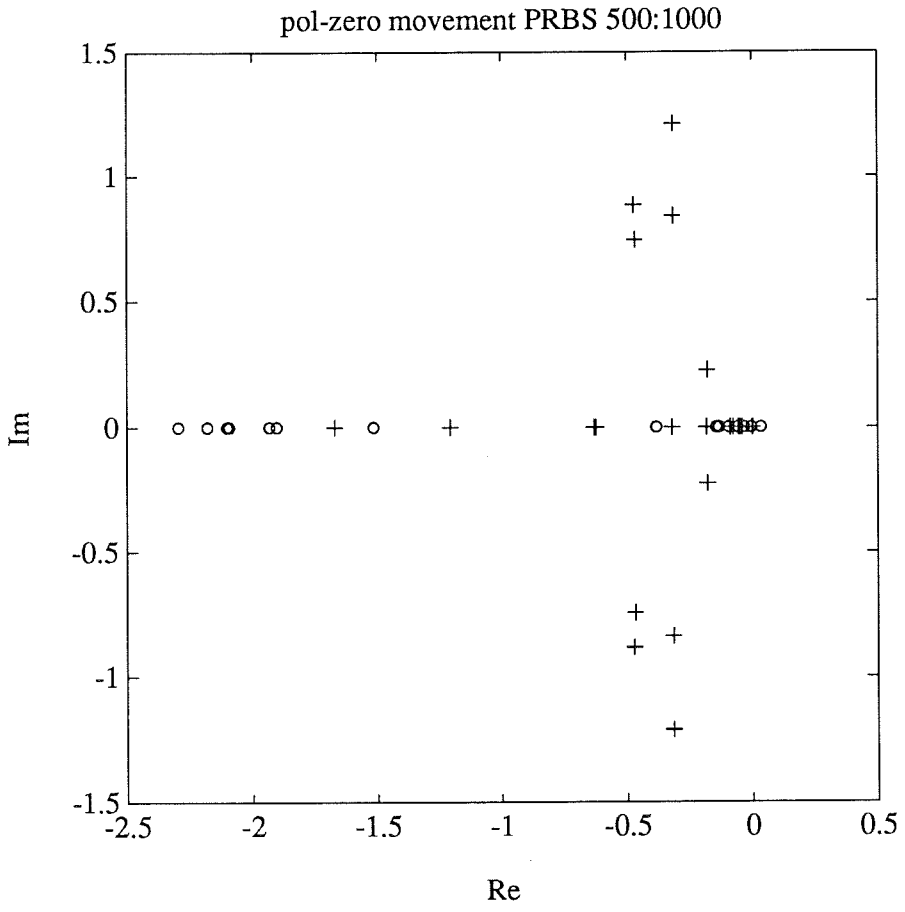

amplitude plot test 1204 2000:2600
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-2}
10^{-1}
10^0
10^1
frequency
phase plot
300
200
100
0
-100
phase
10^{-2}
10^{-1}
10^0
10^1
frequency

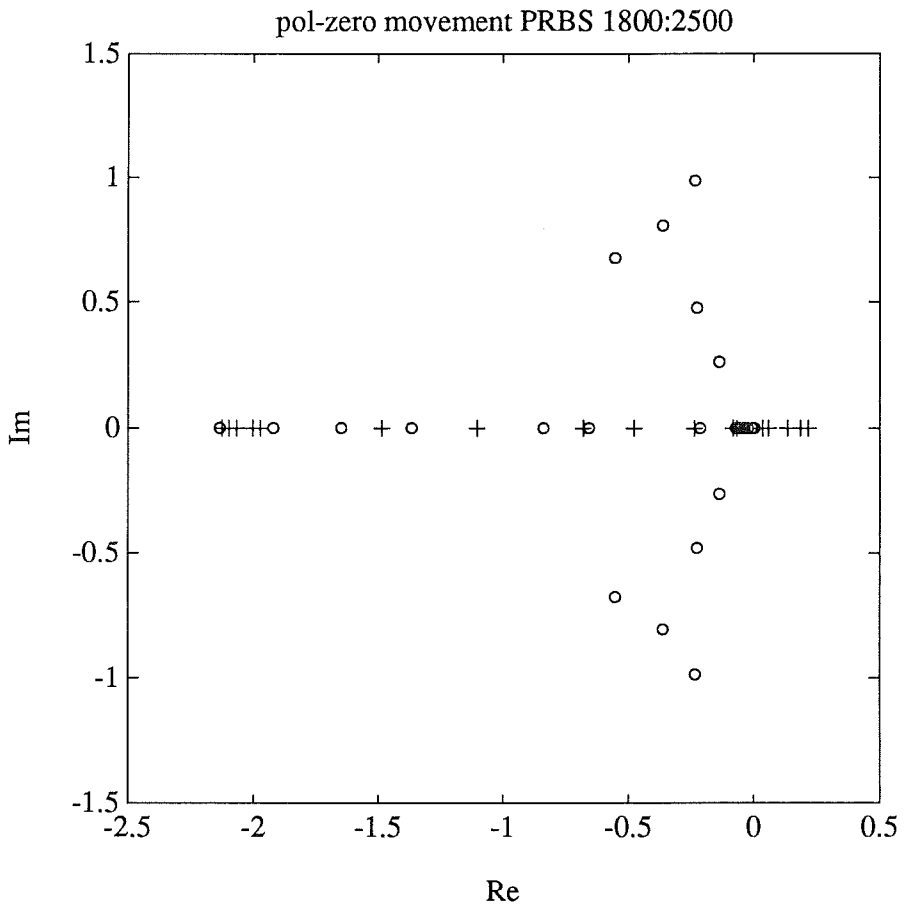

pol-zero movement test 404
1.5
1
0.5
0
-0.5
-1
-1.5
Im
-2.5
-2
-1.5
-1
-0.5
0
0.5
Re


pol-zero movement test 704
1.5
1
0.5
0
-0.5
-1
-1.5
Im
-2.5
-2
-1.5
-1
-0.5
0
0.5
Re


pol-zero movement test 904
1.5
1
0.5
0
-0.5
-1
-1.5
Im
-2.5
-2
-1.5
-1
-0.5
0
0.5
Re


pol-zero movement test 1204
1.5
1
0.5
0
-0.5
-1
-1.5
Im
-2.5
-2
-1.5
-1
-0.5
0
0.5
Re


pol-zero movement PRBS 500:1000
1.5
1
0.5
0
-0.5
-1
-1.5
Im
-2.5
-2
-1.5
-1
-0.5
0
0.5
Re


pol-zero movement PRBS 1800:2500
1.5
1
0.5
0
-0.5
-1
-1.5
Im
-2.5
-2
-1.5
-1
-0.5
0
0.5
Re

VII.4 Estimation and validation of PRBSsinus-stimulation.

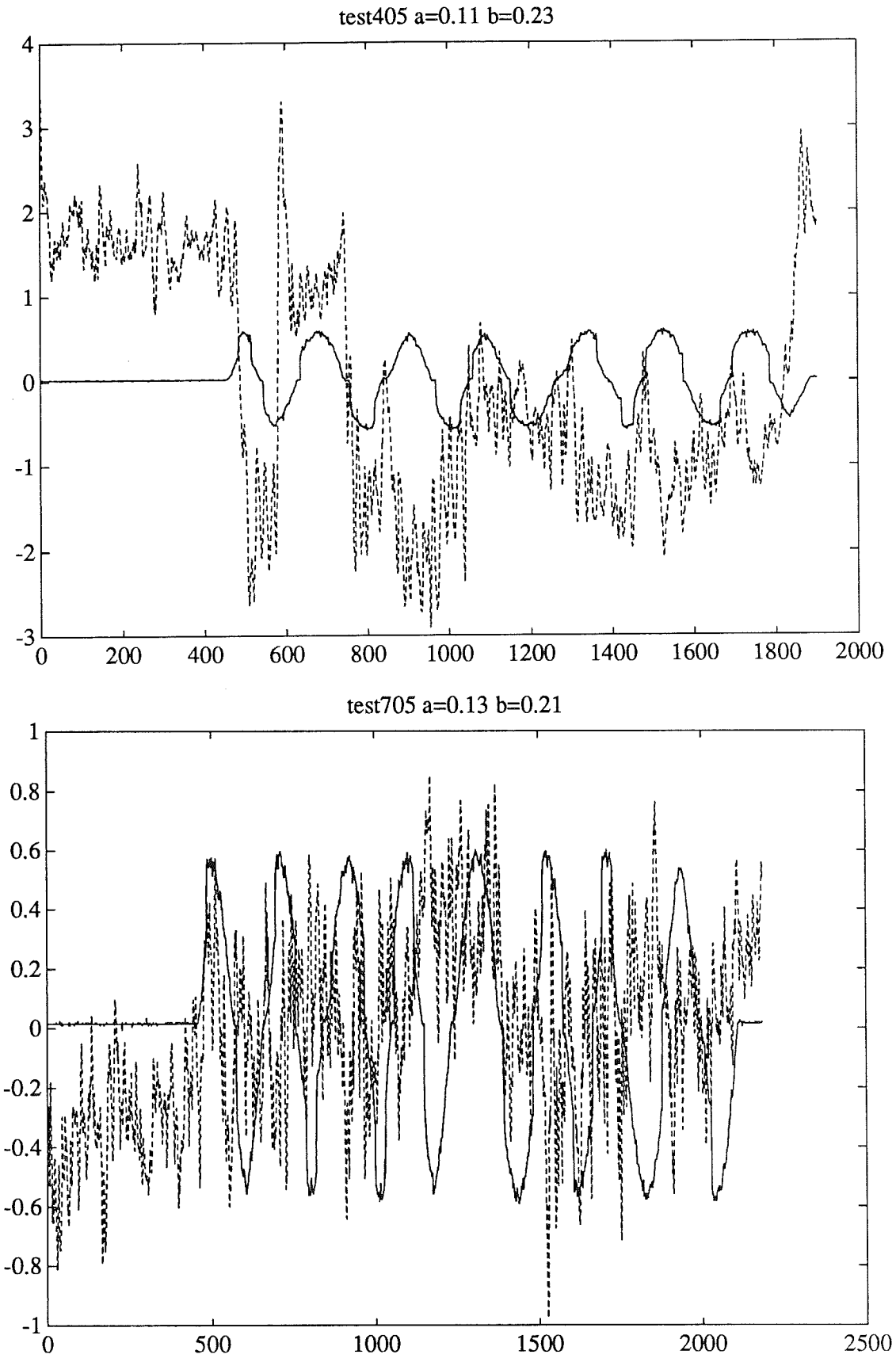
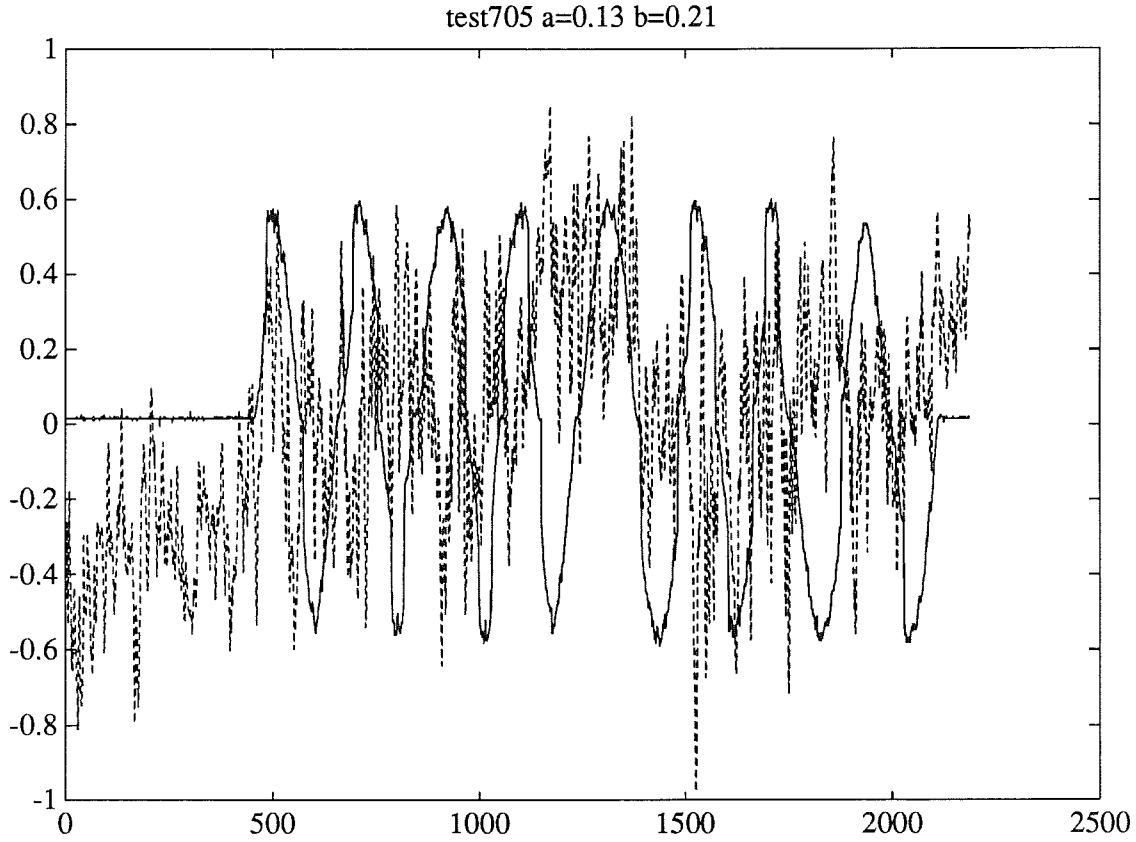
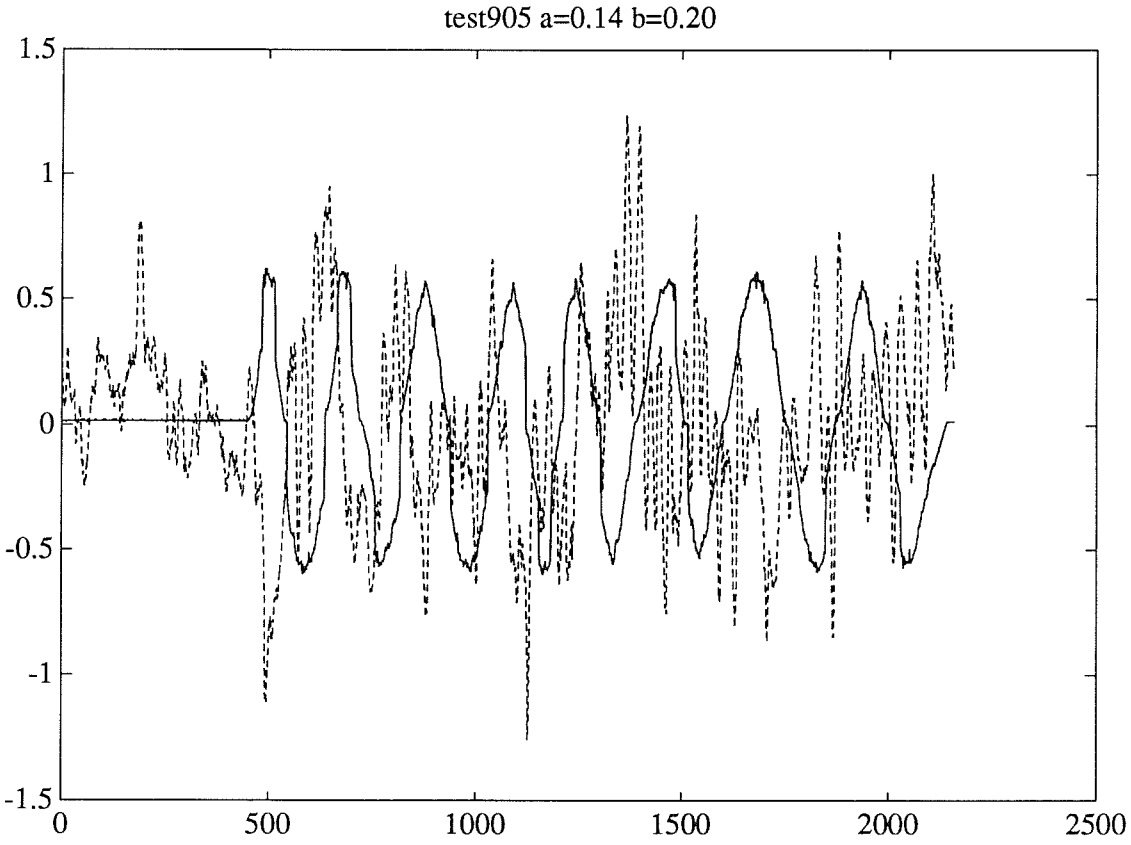
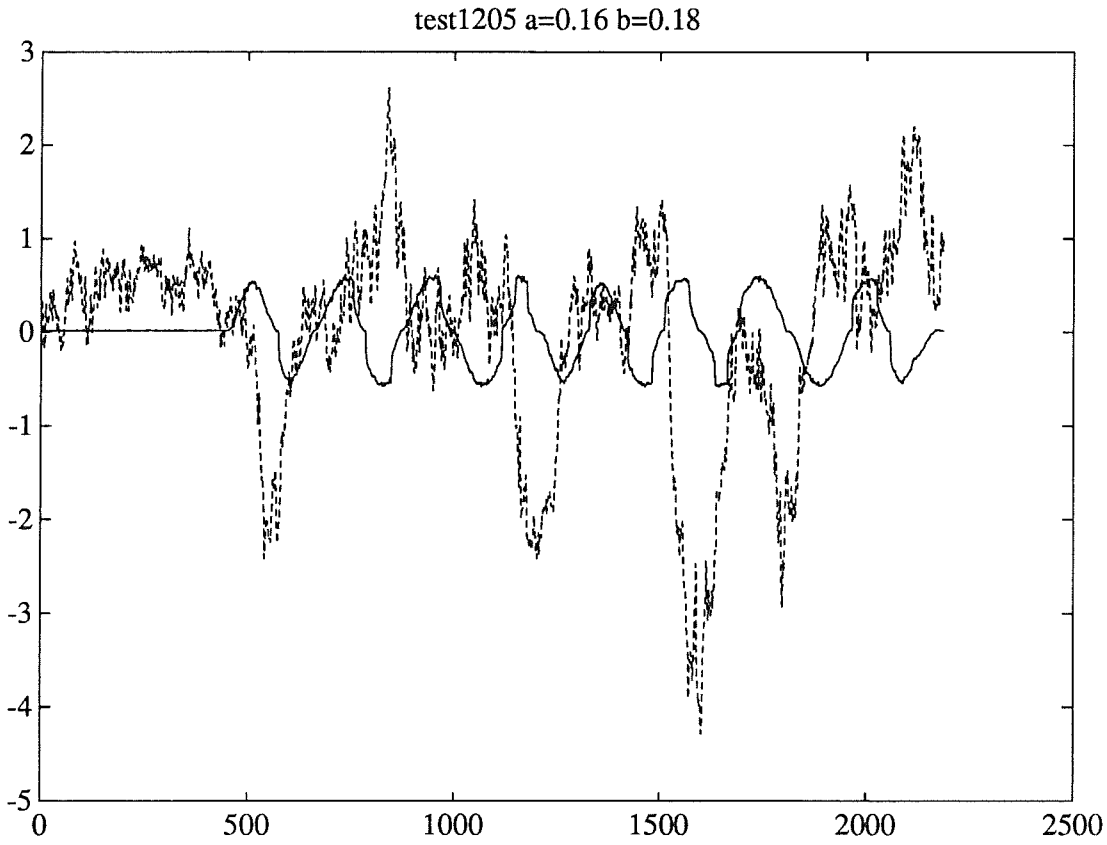
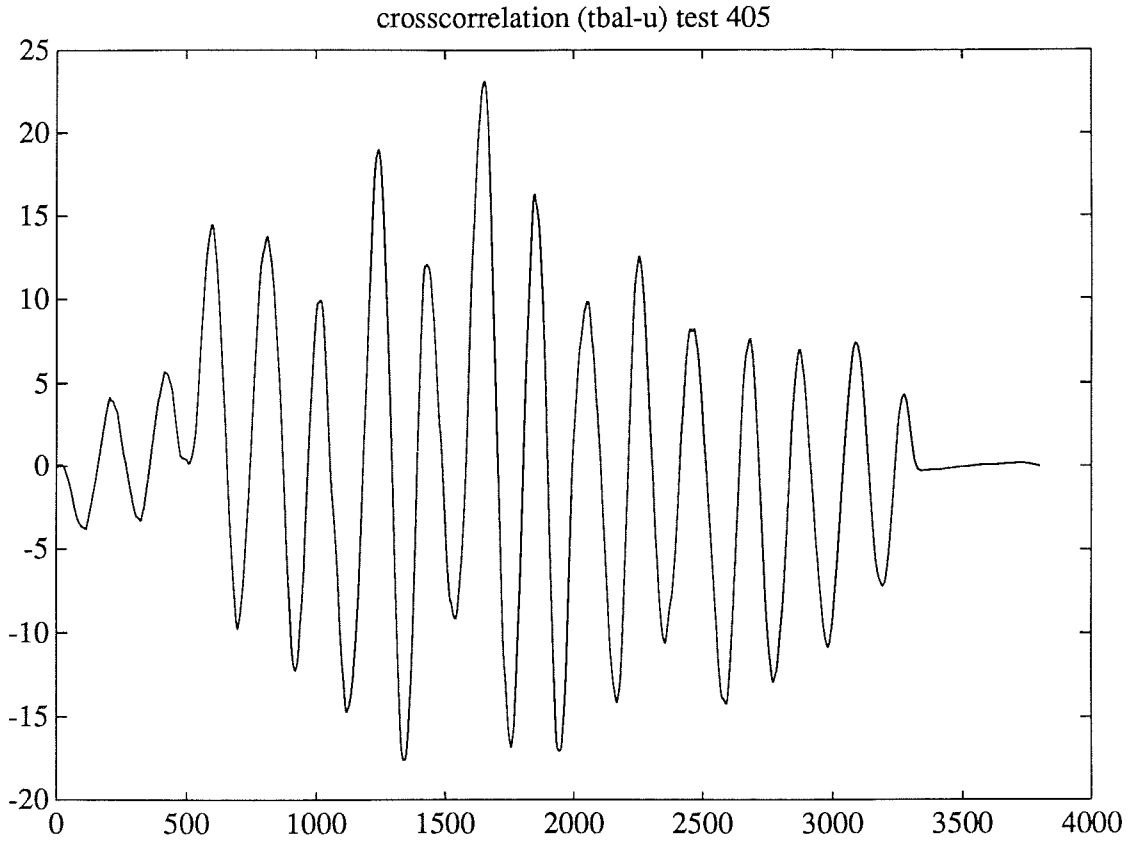


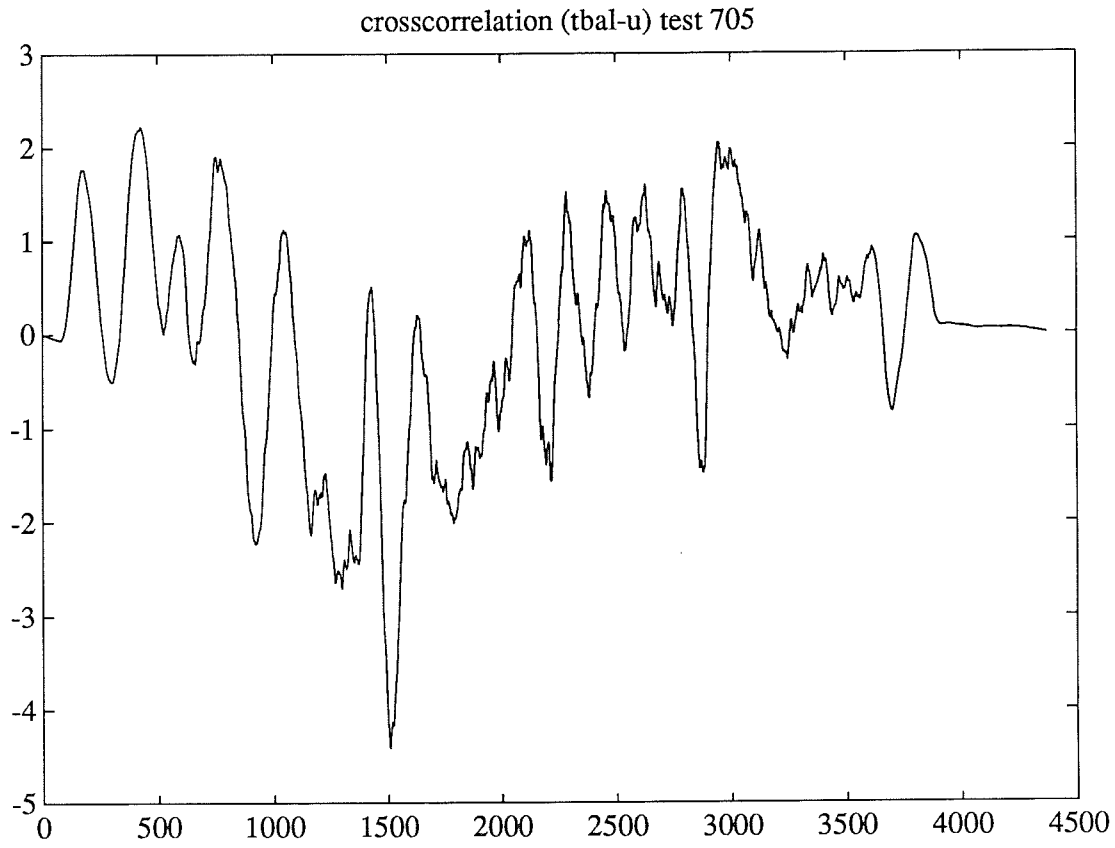
Figure 6.13 Measured results from tests with PRBSsinus as visual stimuli.

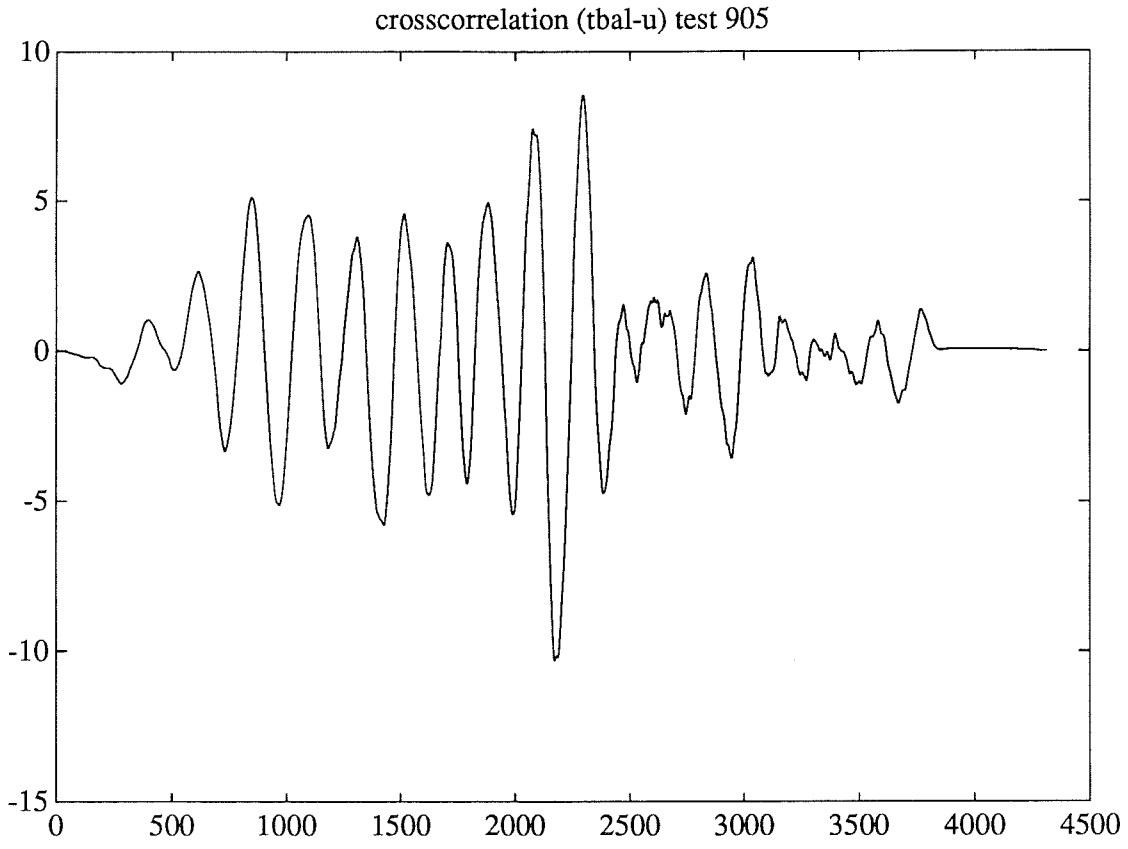

test705 a=0.13 b=0.21
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
500
1000
1500
2000
2500

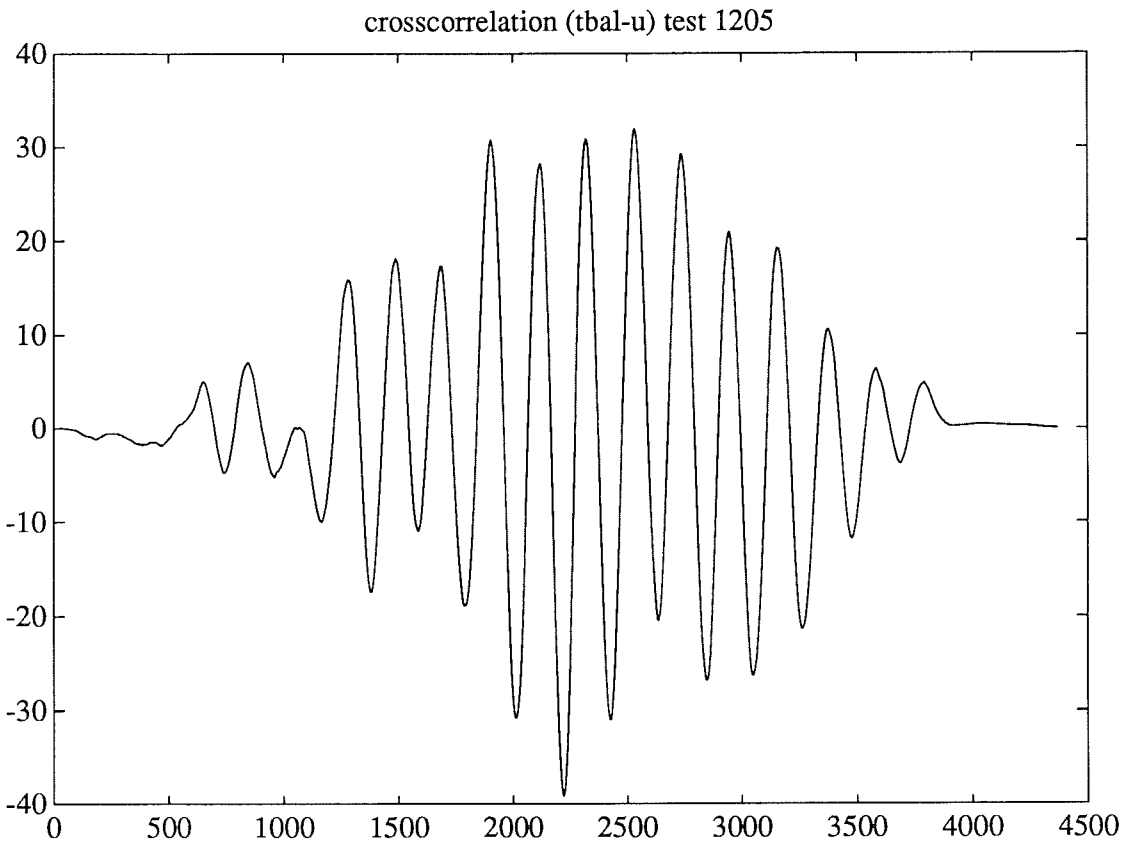

test905 a=0.14 b=0.20
1.5
1
0.5
0
-0.5
-1
-1.5
0
500
1000
1500
2000
2500

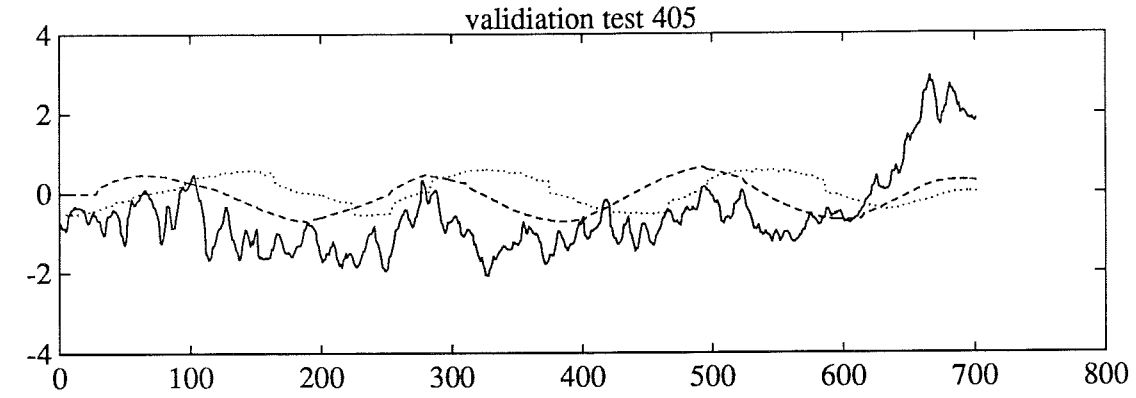

test1205 a=0.16 b=0.18
3
2
1
0
-1
-2
-3
-4
-5
0
500
1000
1500
2000
2500

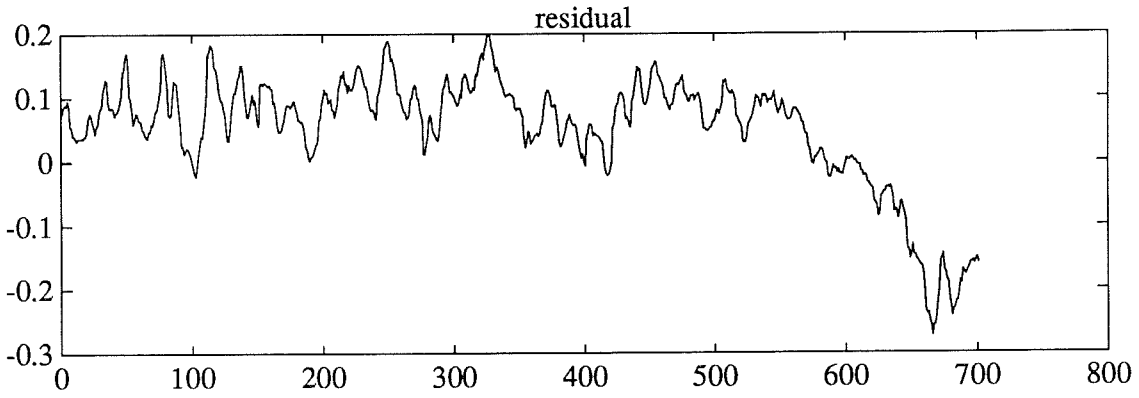

crosscorrelation (tbal-u) test 405
25
20
15
10
5
0
-5
-10
-15
-20
0
500
1000
1500
2000
2500
3000
3500
4000

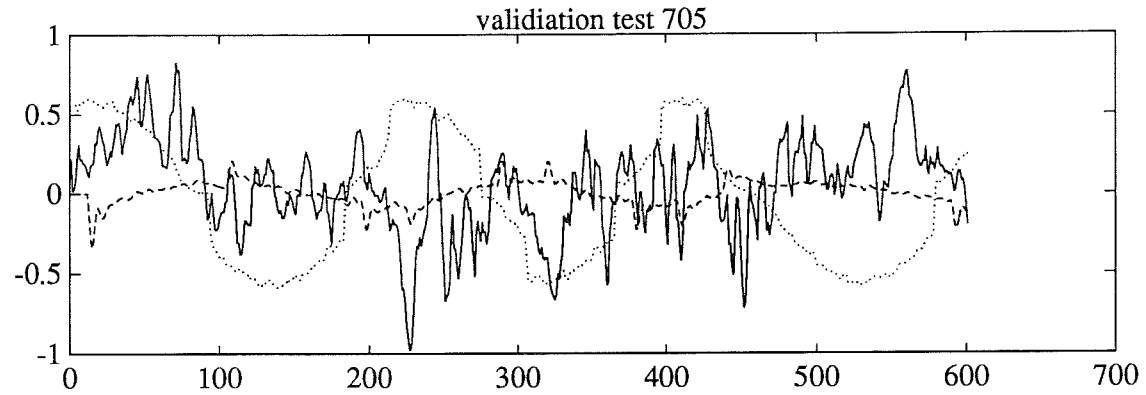

crosscorrelation (tbal-u) test 705
3
2
1
0
-1
-2
-3
-4
-5
0
500
1000
1500
2000
2500
3000
3500
4000
4500

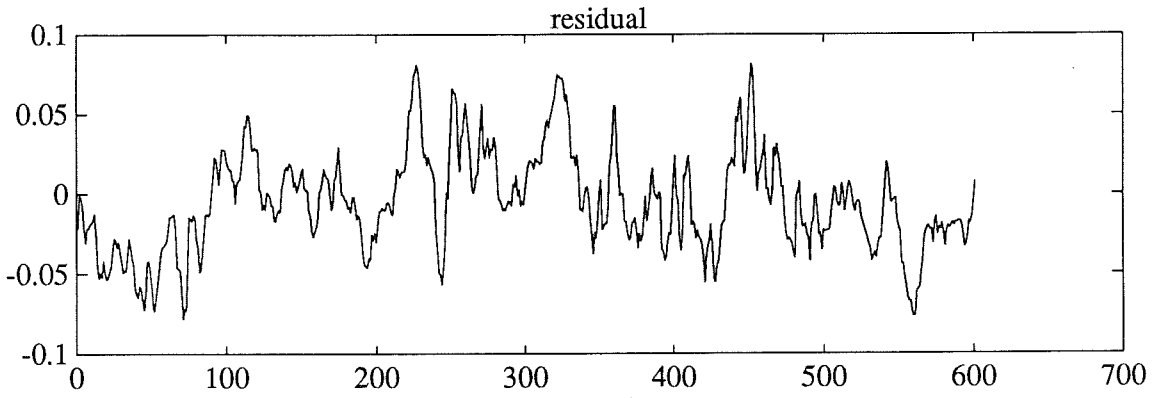

crosscorrelation (tbal-u) test 905
10
5
0
-5
-10
-15
0
500
1000
1500
2000
2500
3000
3500
4000
4500

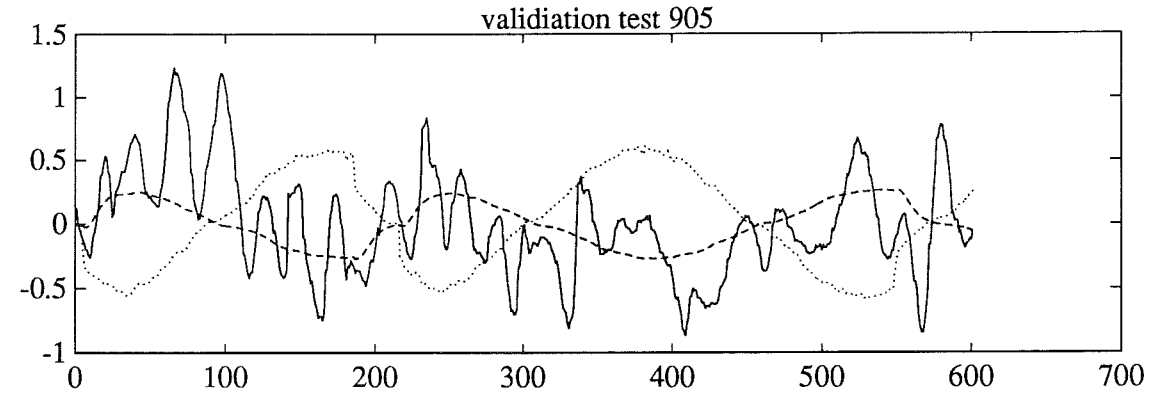

crosscorrelation (tbal-u) test 1205
40
30
20
10
0
-10
-20
-30
-40
0
500
1000
1500
2000
2500
3000
3500
4000
4500

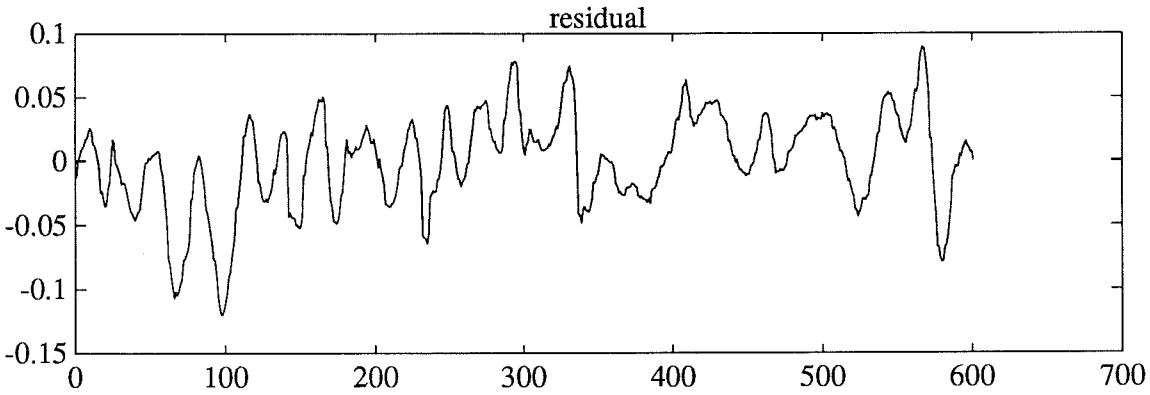

validiation test 405
4
2
0
-2
-4
0
100
200
300
400
500
600
700
800

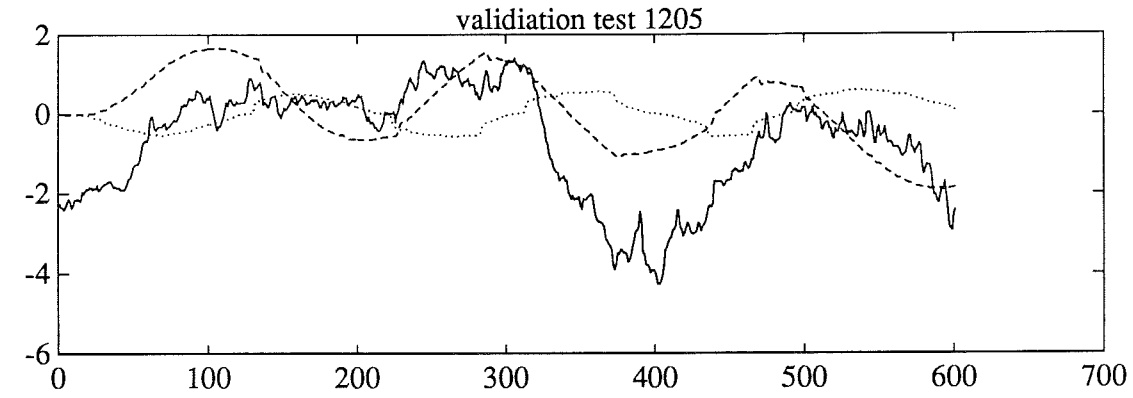

residual
0.2
0.1
0
-0.1
-0.2
-0.3
0
100
200
300
400
500
600
700
800

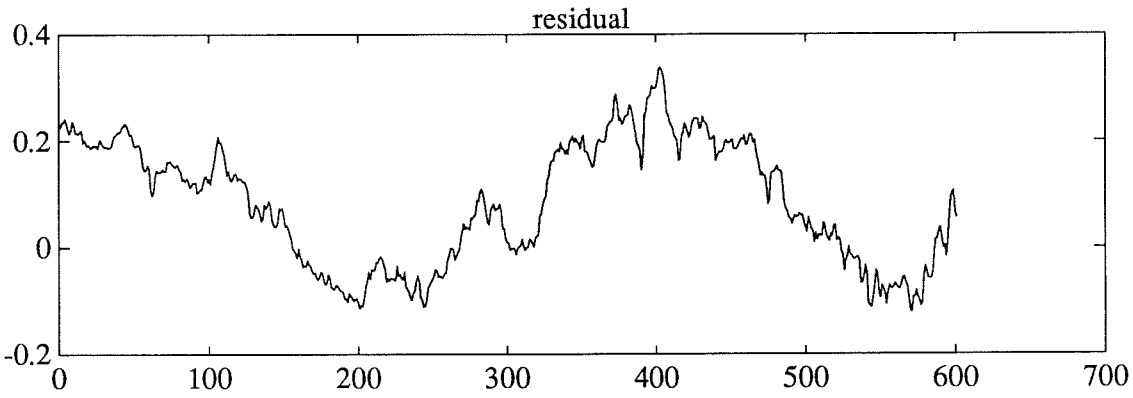

validiation test 705
1
0.5
0
-0.5
-1
0
100
200
300
400
500
600
700

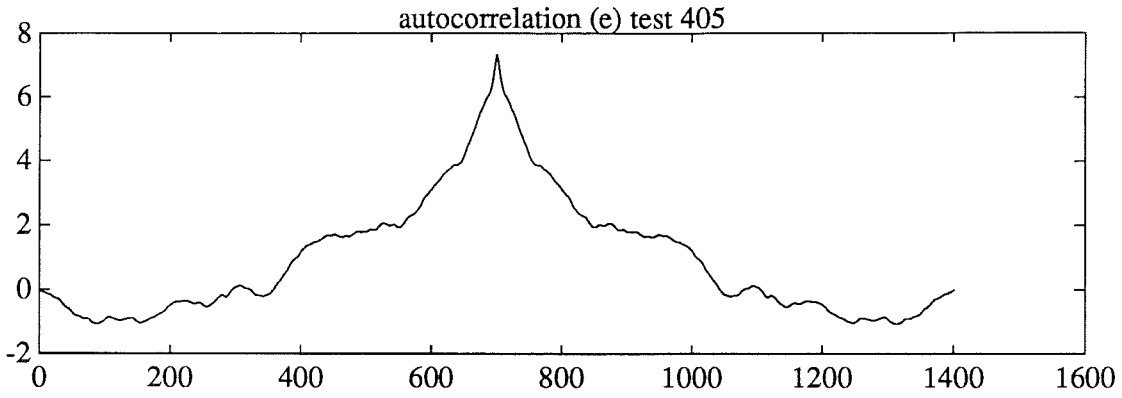

residual
0.1
0.05
0
-0.05
-0.1
0
100
200
300
400
500
600
700

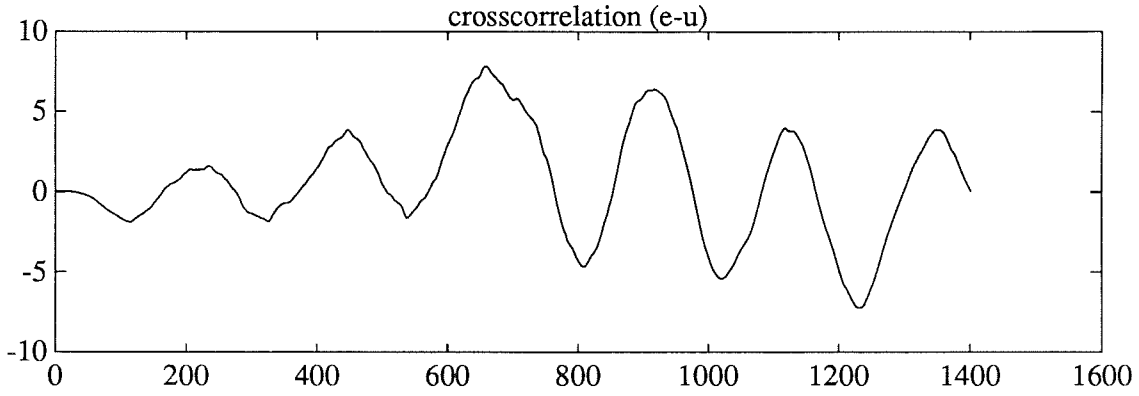

validiation test 905
1.5
1
0.5
0
-0.5
-1
0
100
200
300
400
500
600
700

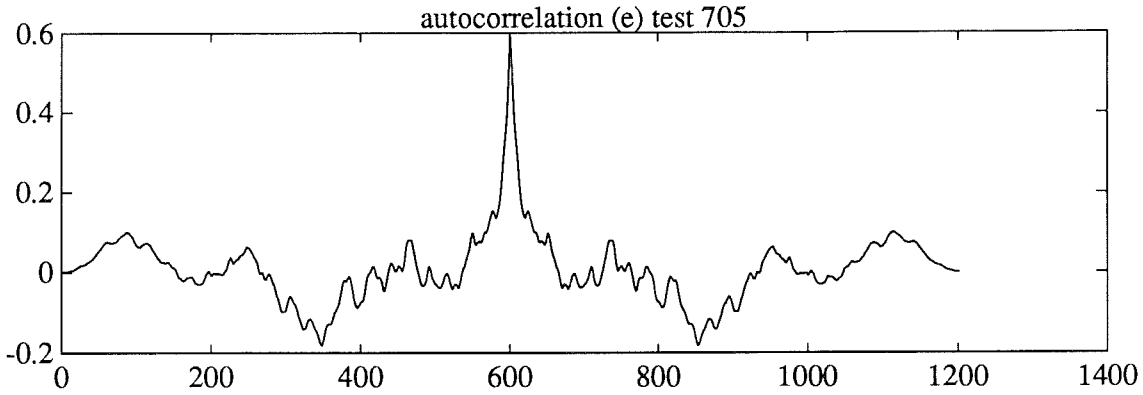

residual
0.1
0.05
0
-0.05
-0.1
-0.15
0
100
200
300
400
500
600
700

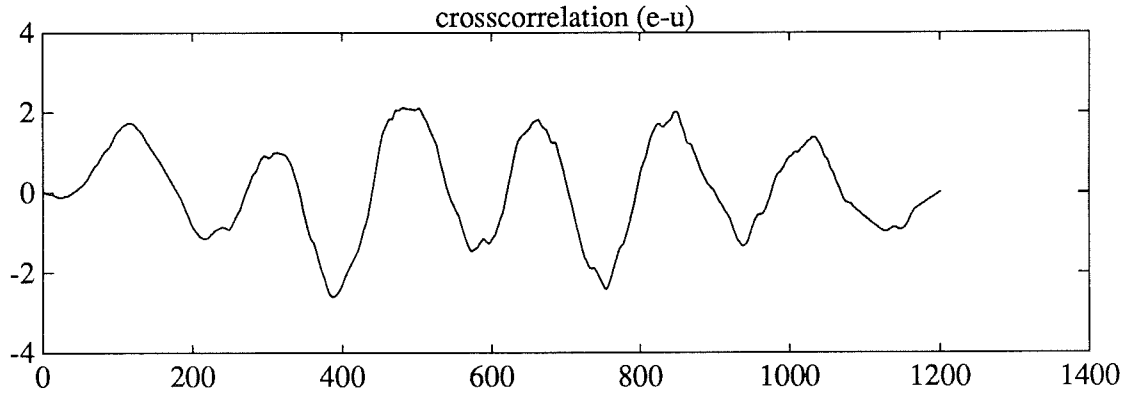

validiation test 1205
2
0
-2
-4
-6
0
100
200
300
400
500
600
700

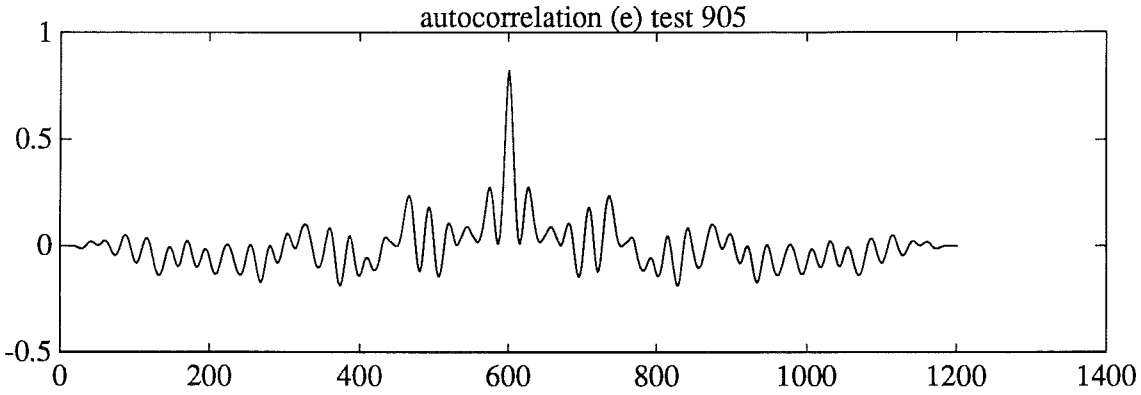

residual
0.4
0.2
0
-0.2
0
100
200
300
400
500
600
700

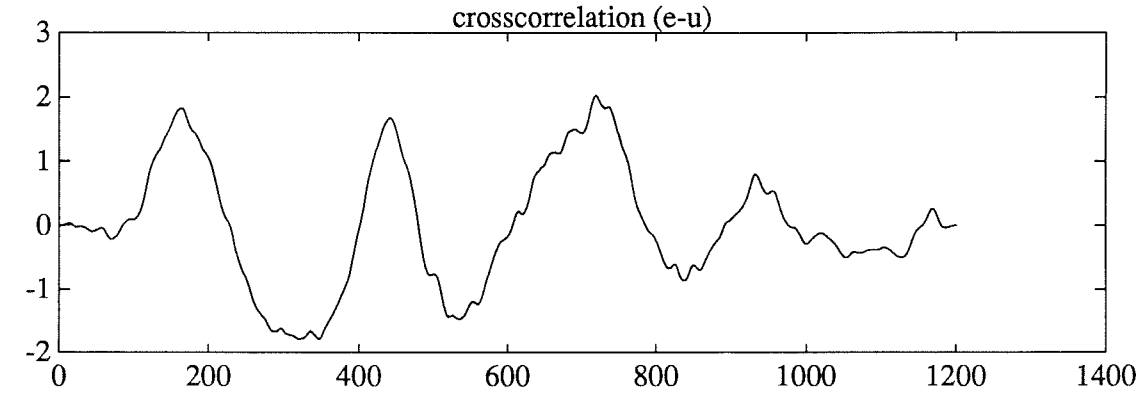

autocorrelation (e) test 405
8
6
4
2
0
-2
0
200
400
600
800
1000
1200
1400
1600

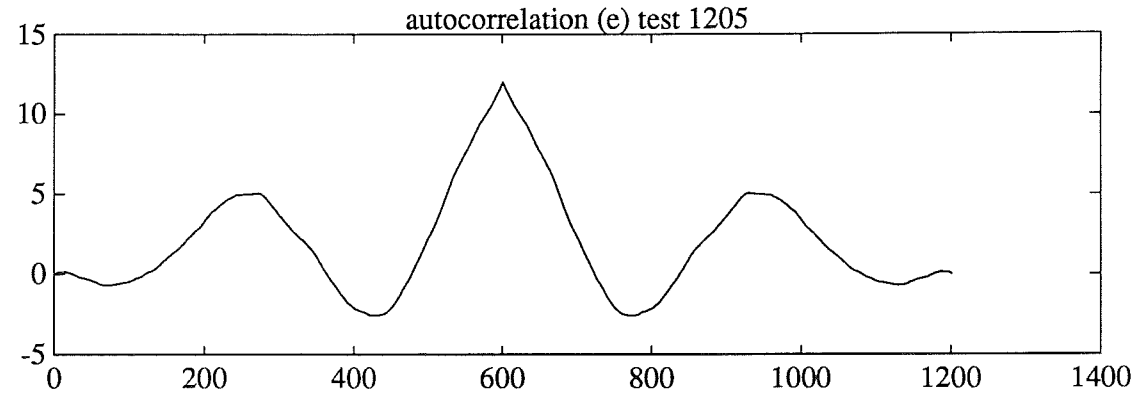

crosscorrelation (e-u)
10
5
0
-5
-10
0
200
400
600
800
1000
1200
1400
1600

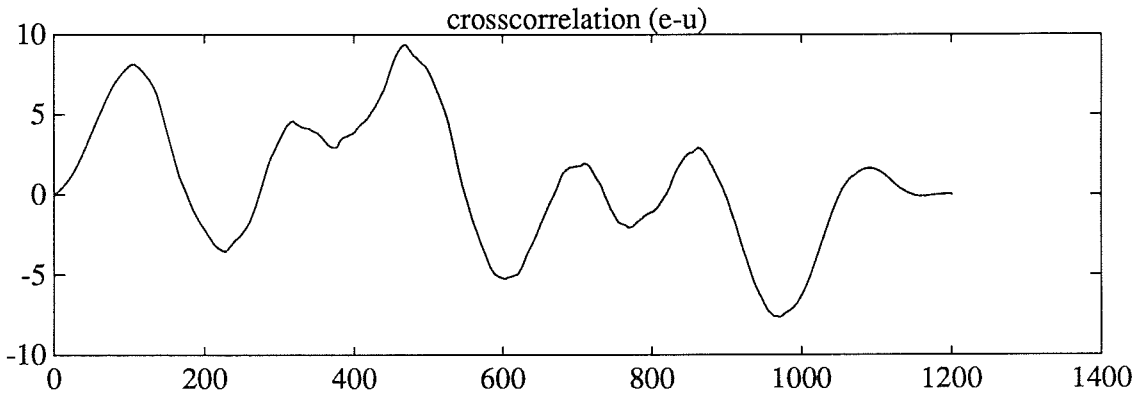

autocorrelation (e) test 705
0.6
0.4
0.2
0
-0.2
0
200
400
600
800
1000
1200
1400

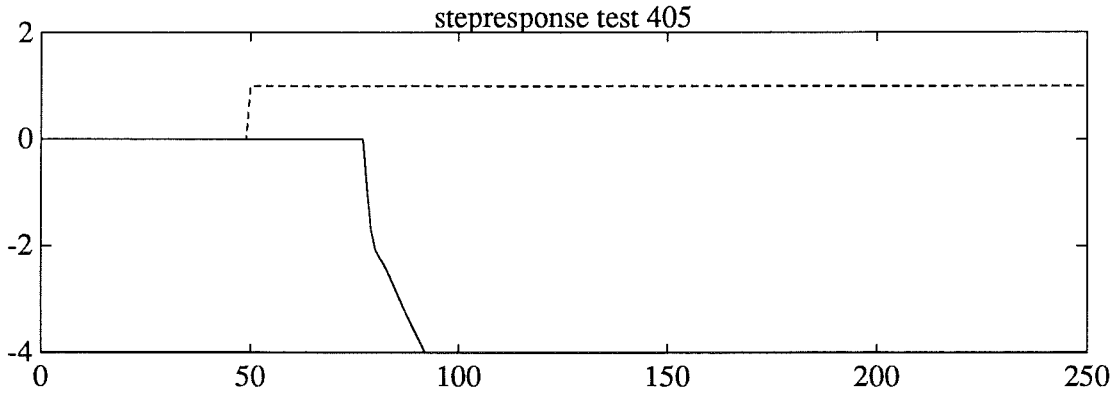

crosscorrelation (e-u)
4
2
0
-2
-4
0
200
400
600
800
1000
1200
1400

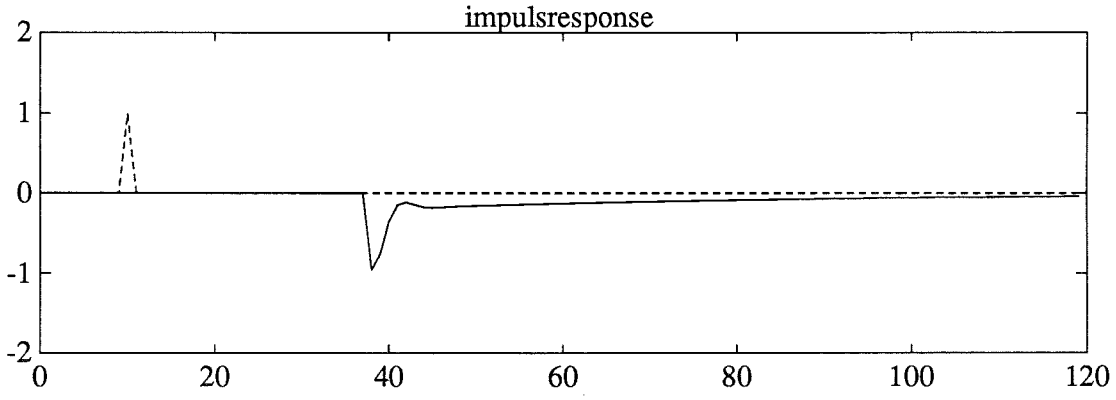

autocorrelation (e) test 905
1
0.5
0
-0.5
0
200
400
600
800
1000
1200
1400

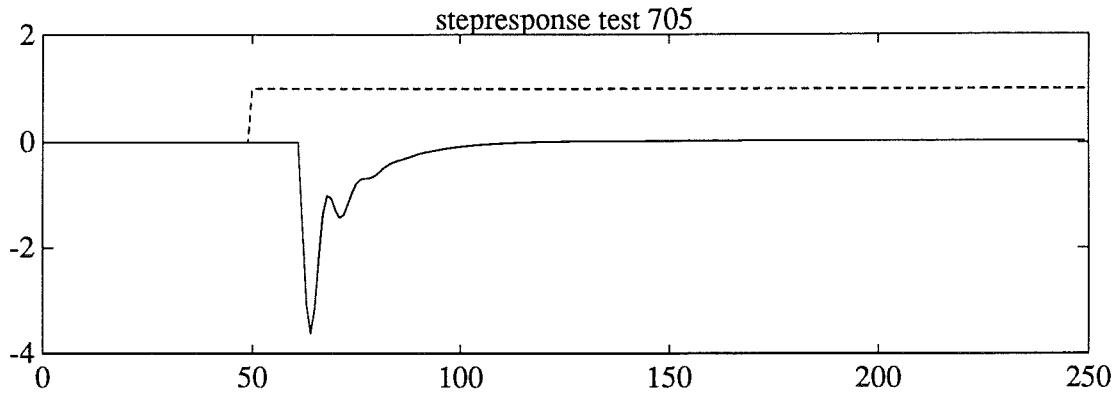

crosscorrelation (e-u)
3
2
1
0
-1
-2
0
200
400
600
800
1000
1200
1400

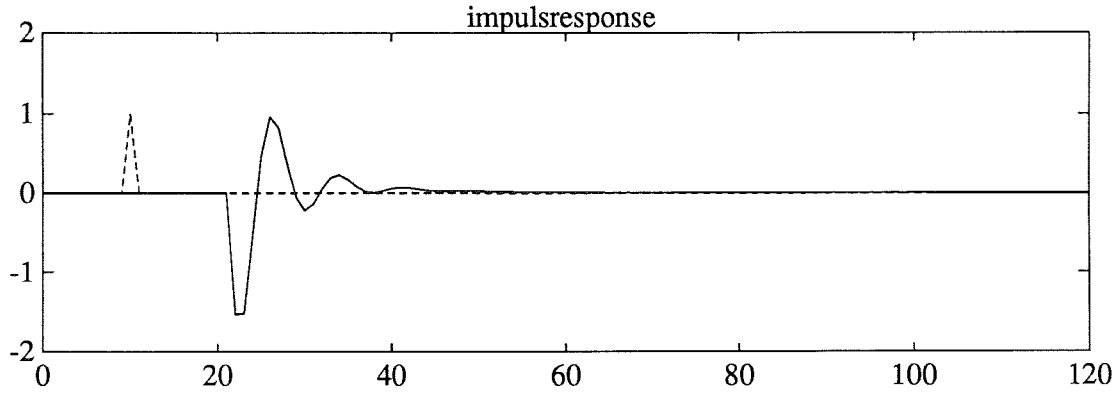

autocorrelation (e) test 1205
15
10
5
0
-5
0
200
400
600
800
1000
1200
1400

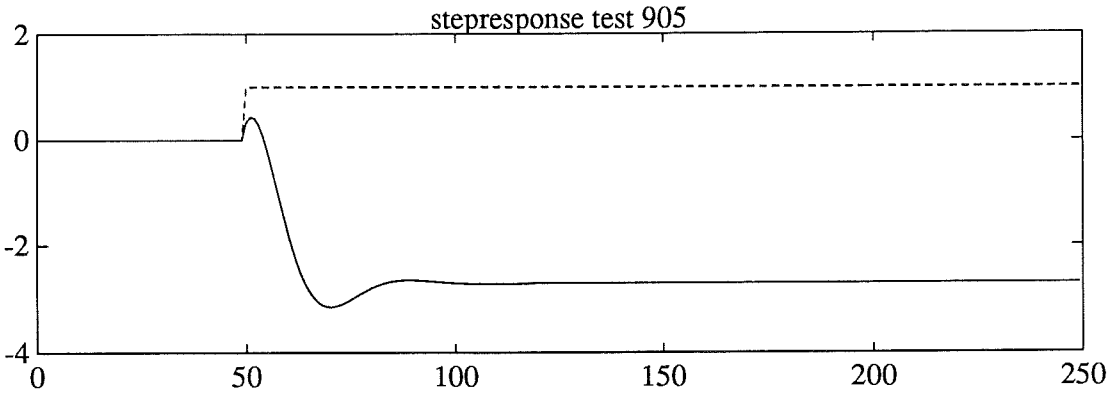

crosscorrelation (e-u)
10
5
0
-5
-10
0
200
400
600
800
1000
1200
1400

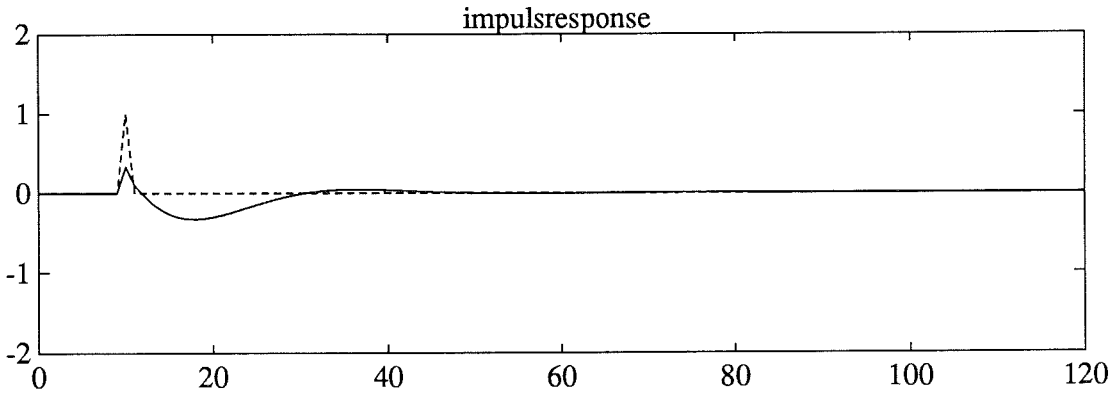

stepresponse test 405
2
0
-2
-4
0
50
100
150
200
250

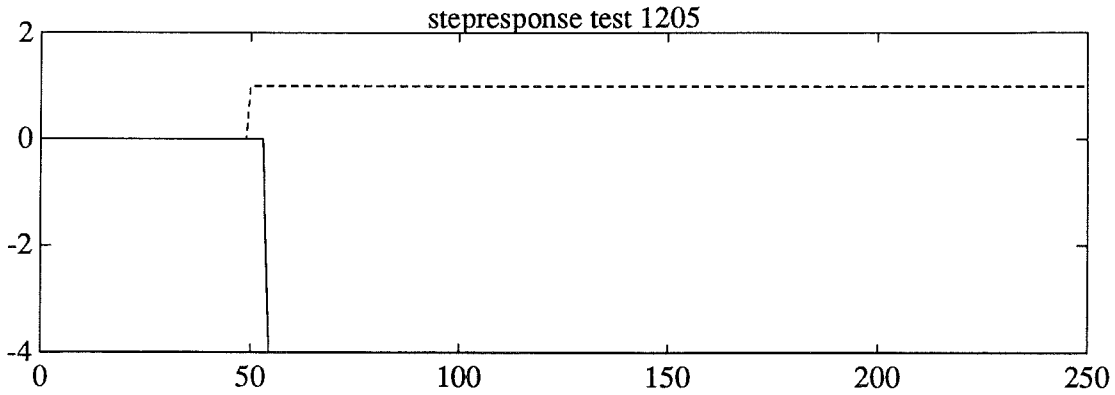

impulsresponse
2
1
0
-1
-2
0
20
40
60
80
100
120

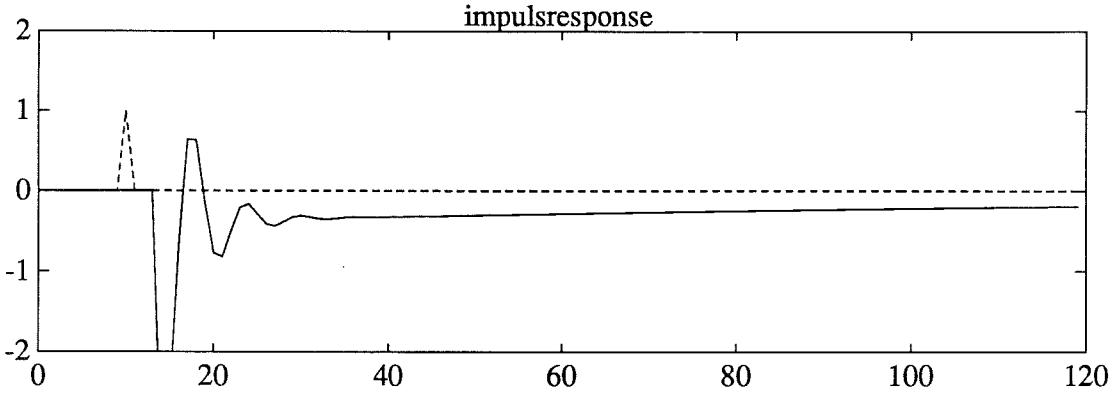

stepresponse test 705
2
0
-2
-4
0
50
100
150
200
250

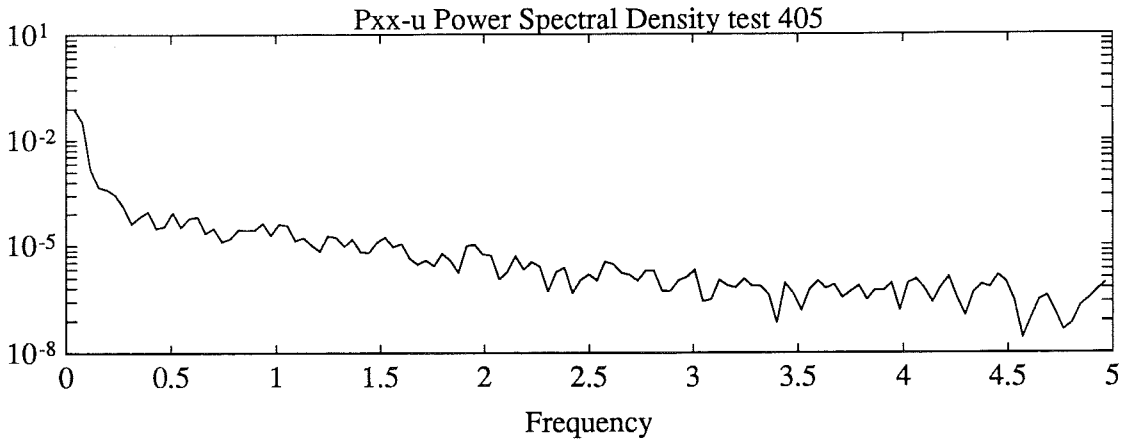

impulsresponse
2
1
0
-1
-2
0
20
40
60
80
100
120

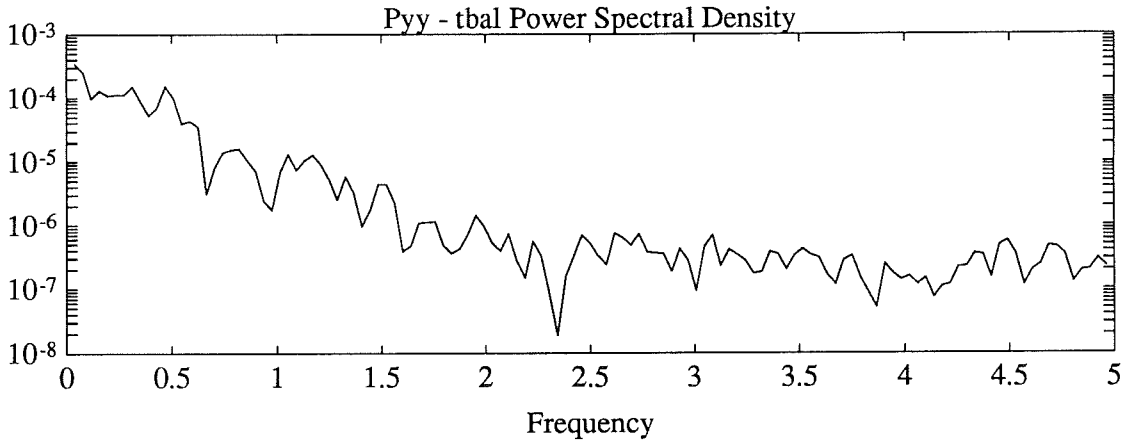

stepresponse test 905
2
0
-2
-4
0
50
100
150
200
250

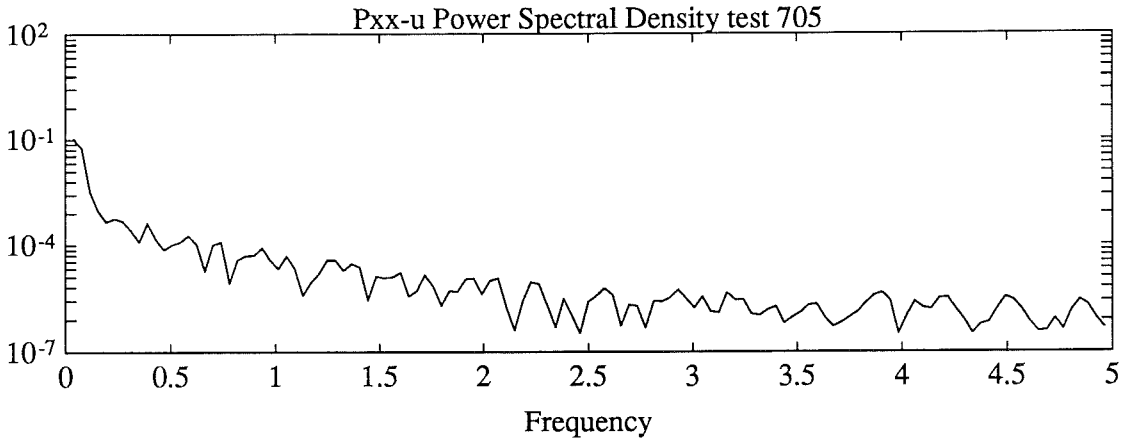

impulsresponse
2
1
0
-1
-2
0
20
40
60
80
100
120

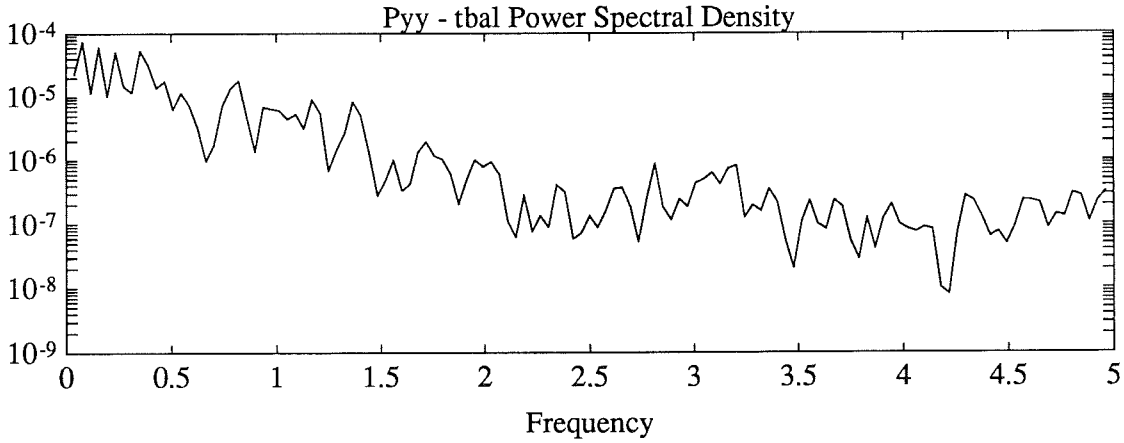

stepresponse test 1205
2
0
-2
-4
0
50
100
150
200
250

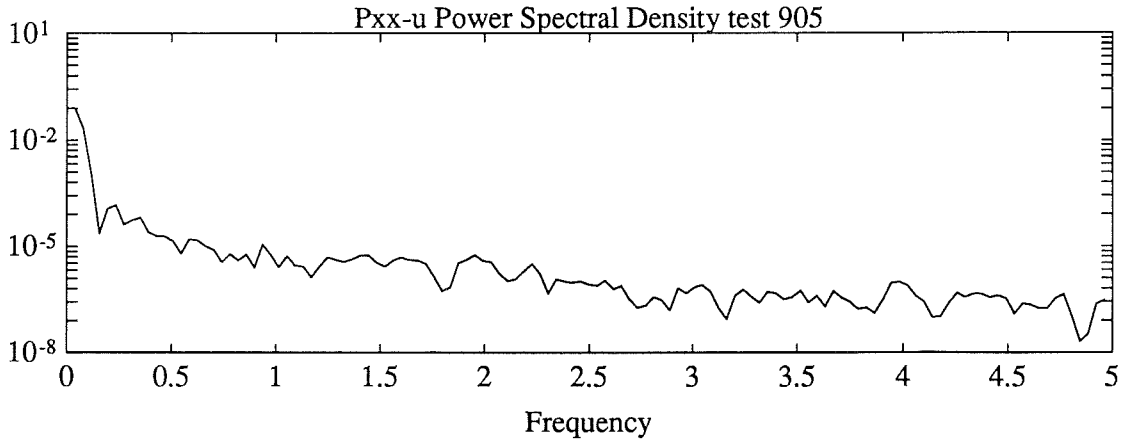

impulsresponse
2
1
0
-1
-2
0
20
40
60
80
100
120

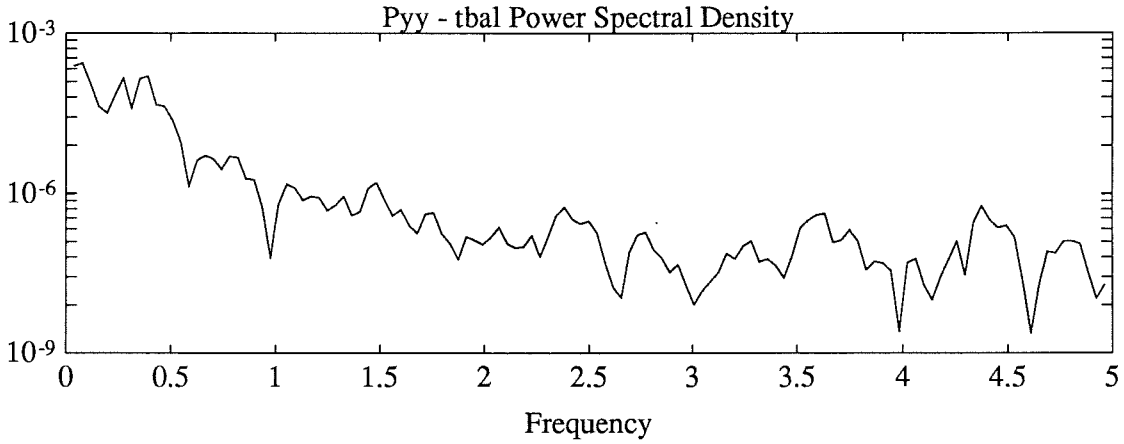

Pxx-u Power Spectral Density test 405
10^1
10^-2
10^-5
10^-8
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

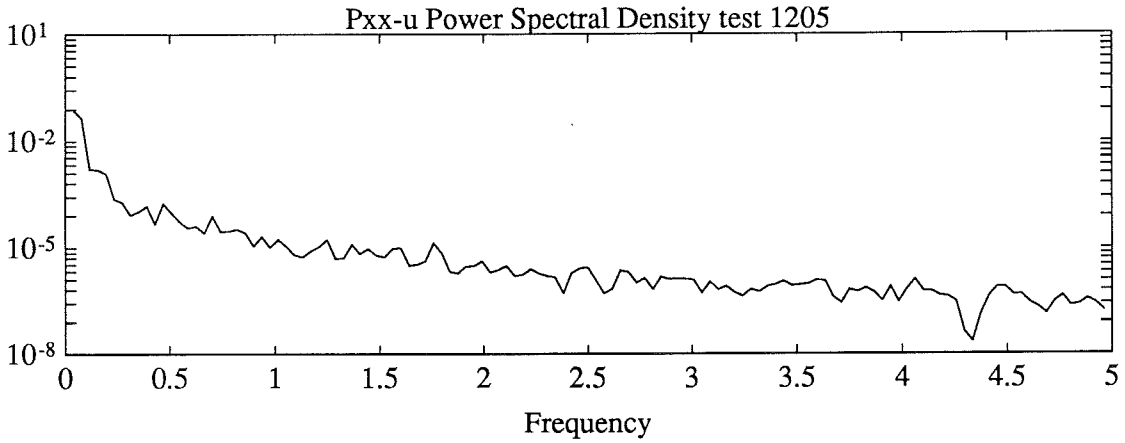

Pyy - tbal Power Spectral Density
10^-3
10^-4
10^-5
10^-6
10^-7
10^-8
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

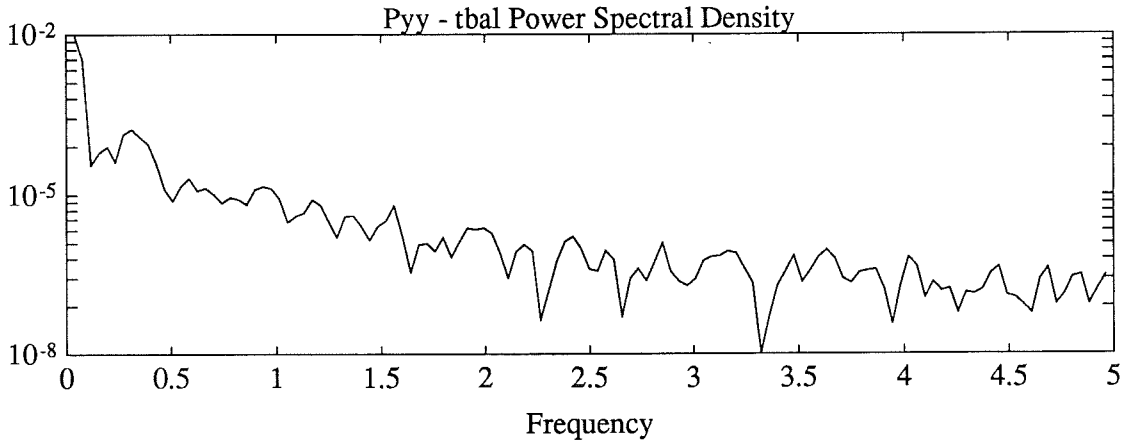

Pxx-u Power Spectral Density test 705
10^2
10^-1
10^-4
10^-7
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

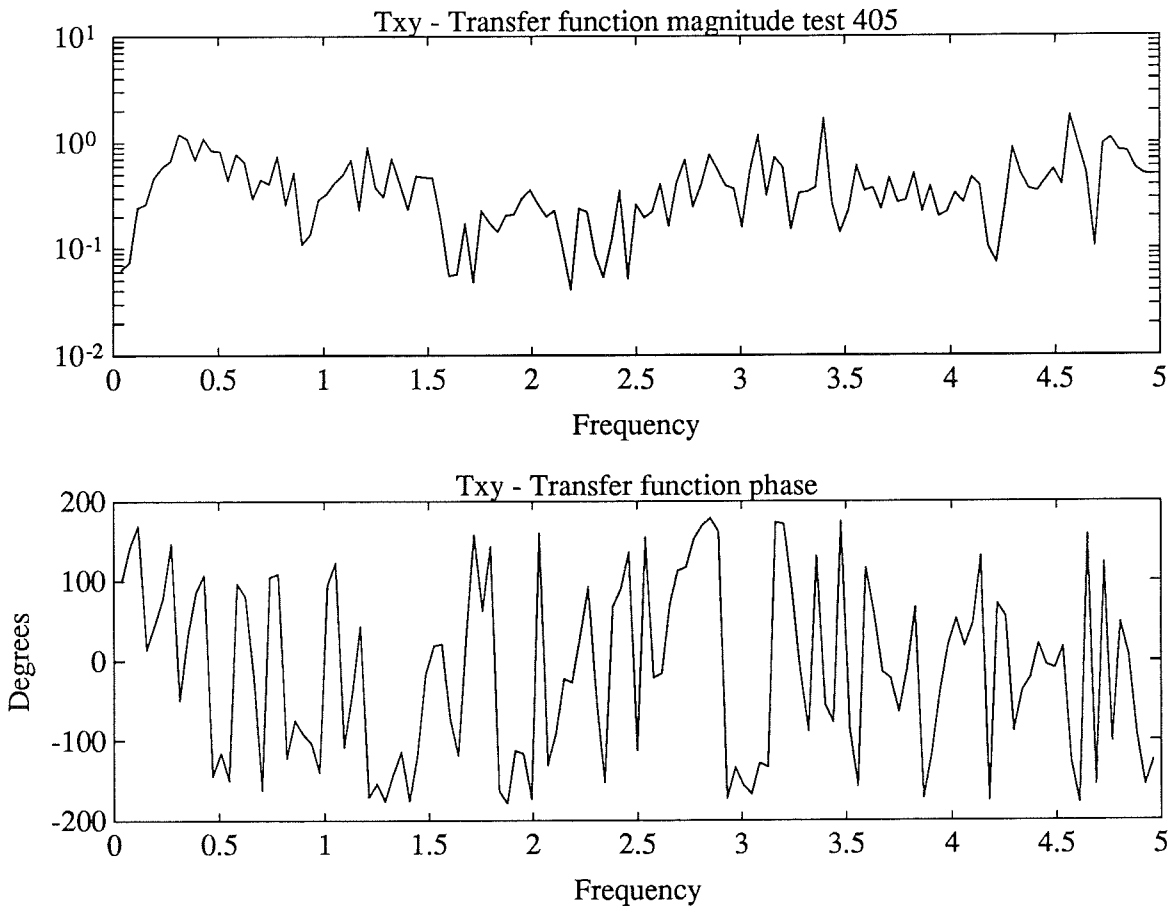

Pyy - tbal Power Spectral Density
10^-4
10^-5
10^-6
10^-7
10^-8
10^-9
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

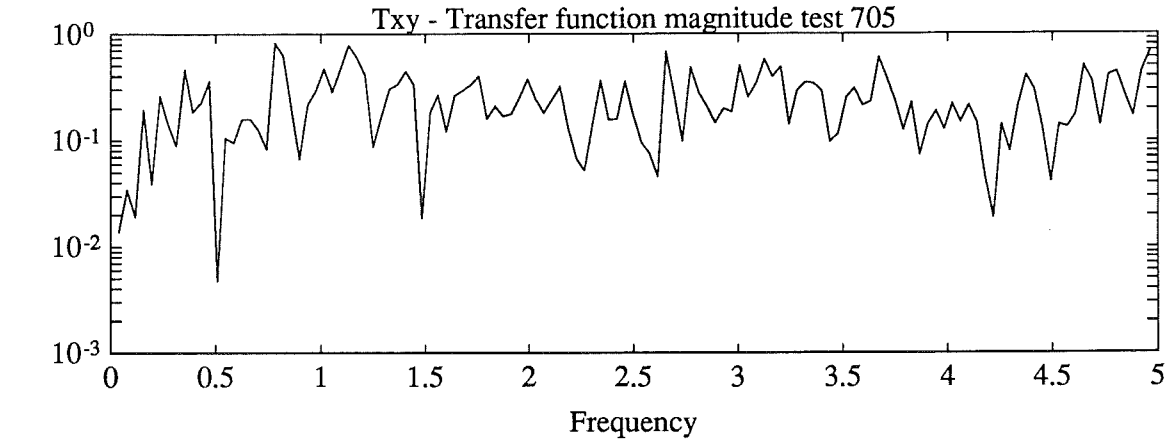

Pxx-u Power Spectral Density test 905
10^1
10^-2
10^-5
10^-8
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

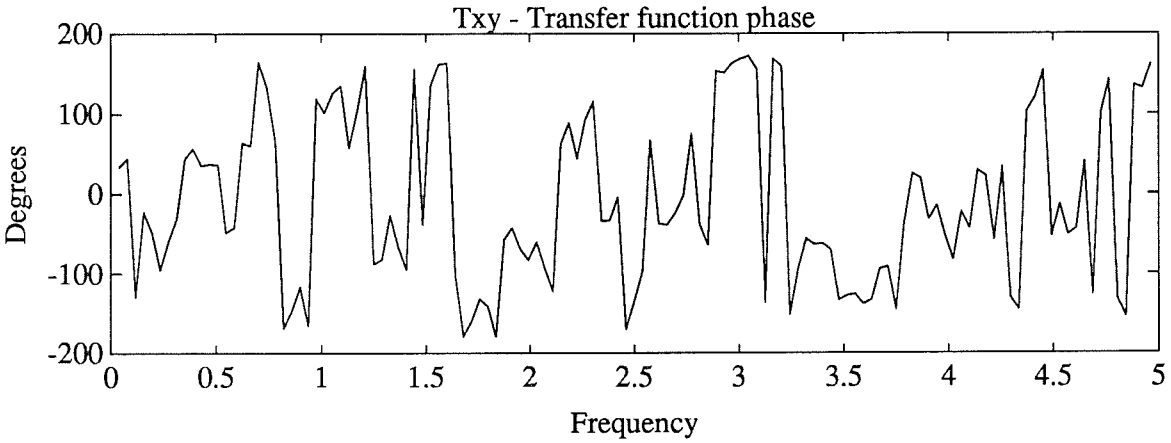

Pyy - tbal Power Spectral Density
10^-3
10^-6
10^-9
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

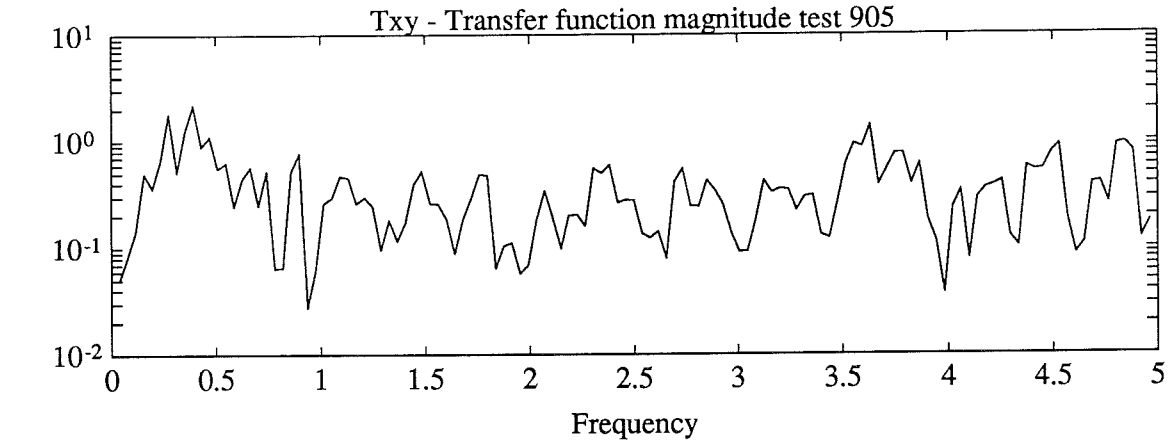

Pxx-u Power Spectral Density test 1205
10^1
10^-2
10^-5
10^-8
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

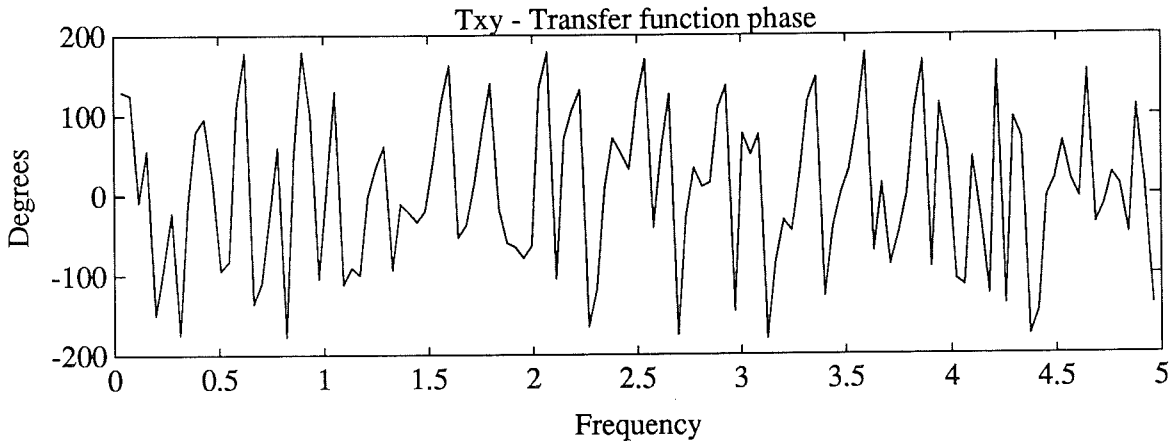

Pyy - tbal Power Spectral Density
10^-2
10^-5
10^-8
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

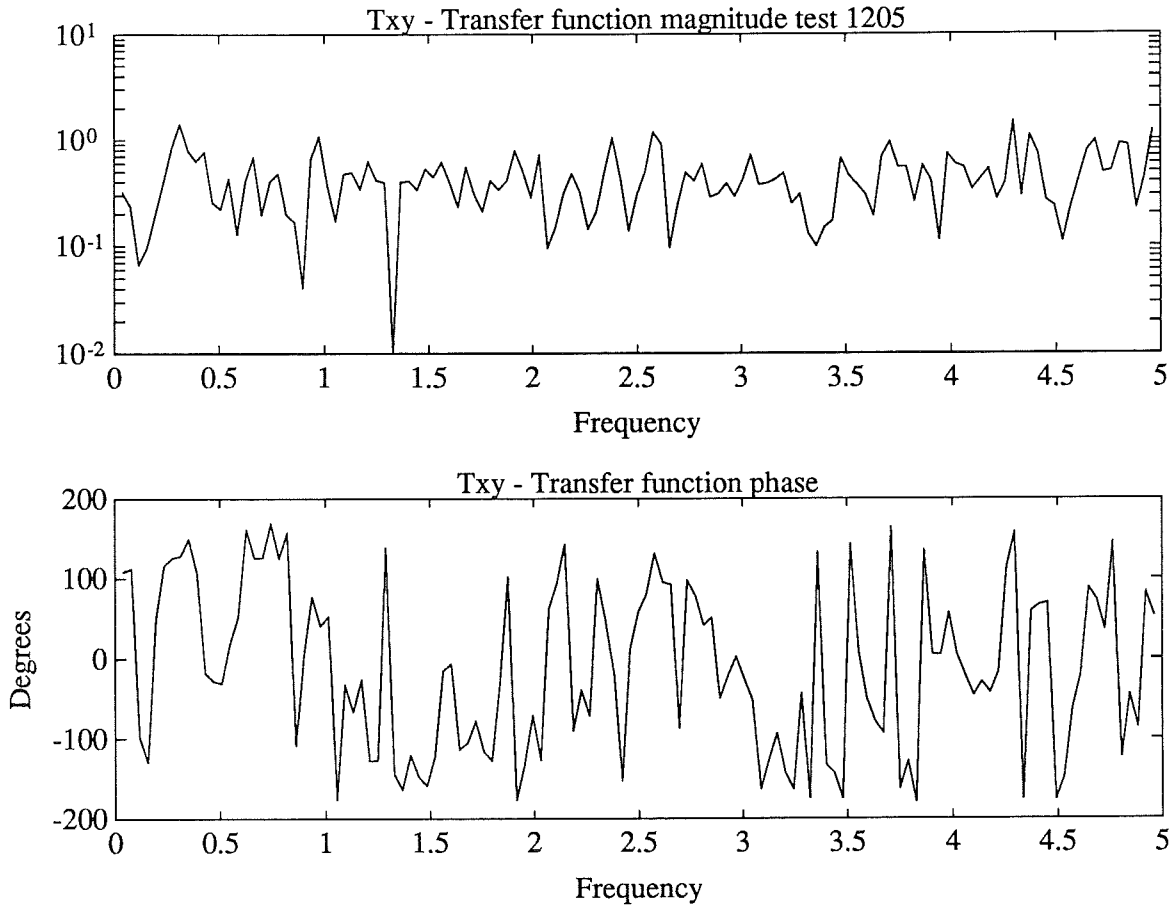

Txy - Transfer function magnitude test 405
10^1
10^0
10^-1
10^-2
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency
Txy - Transfer function phase
200
100
0
-100
-200
Degrees
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

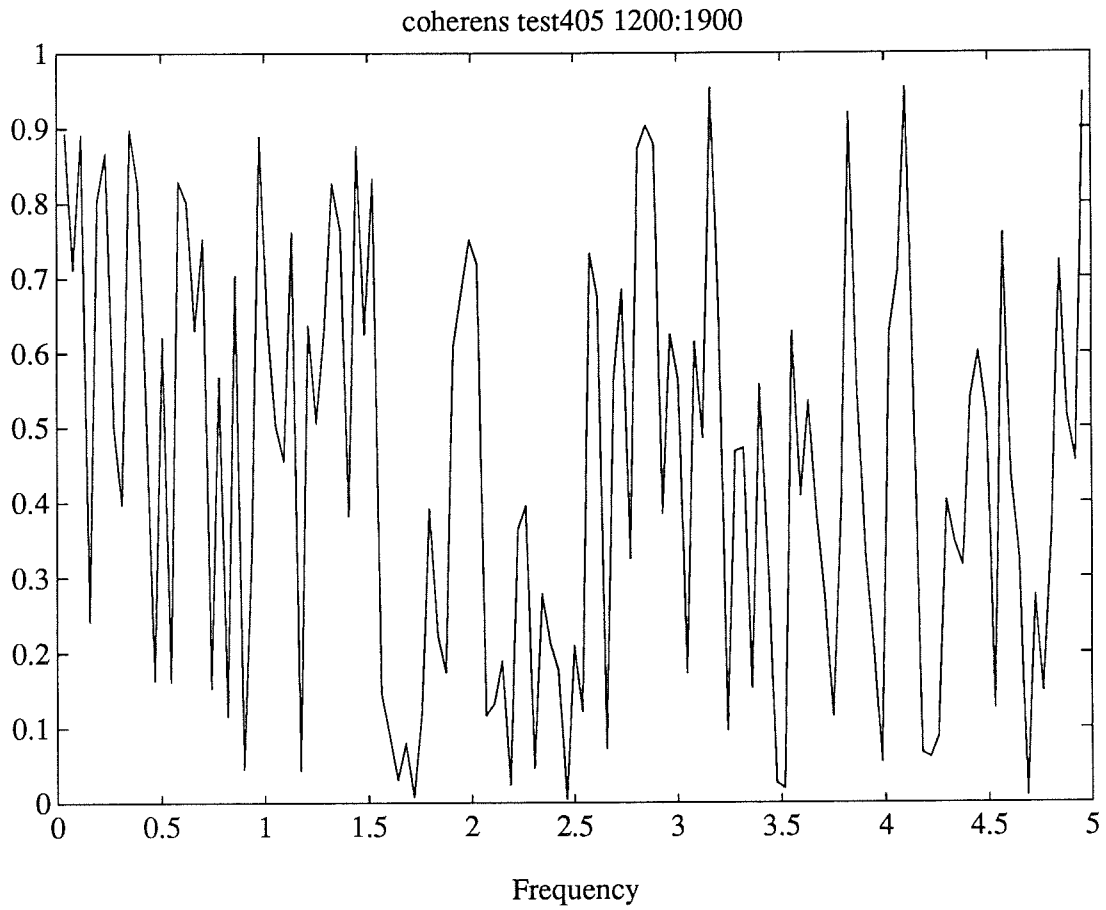

Txy - Transfer function magnitude test 705
10^0
10^{-1}
10^{-2}
10^{-3}
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

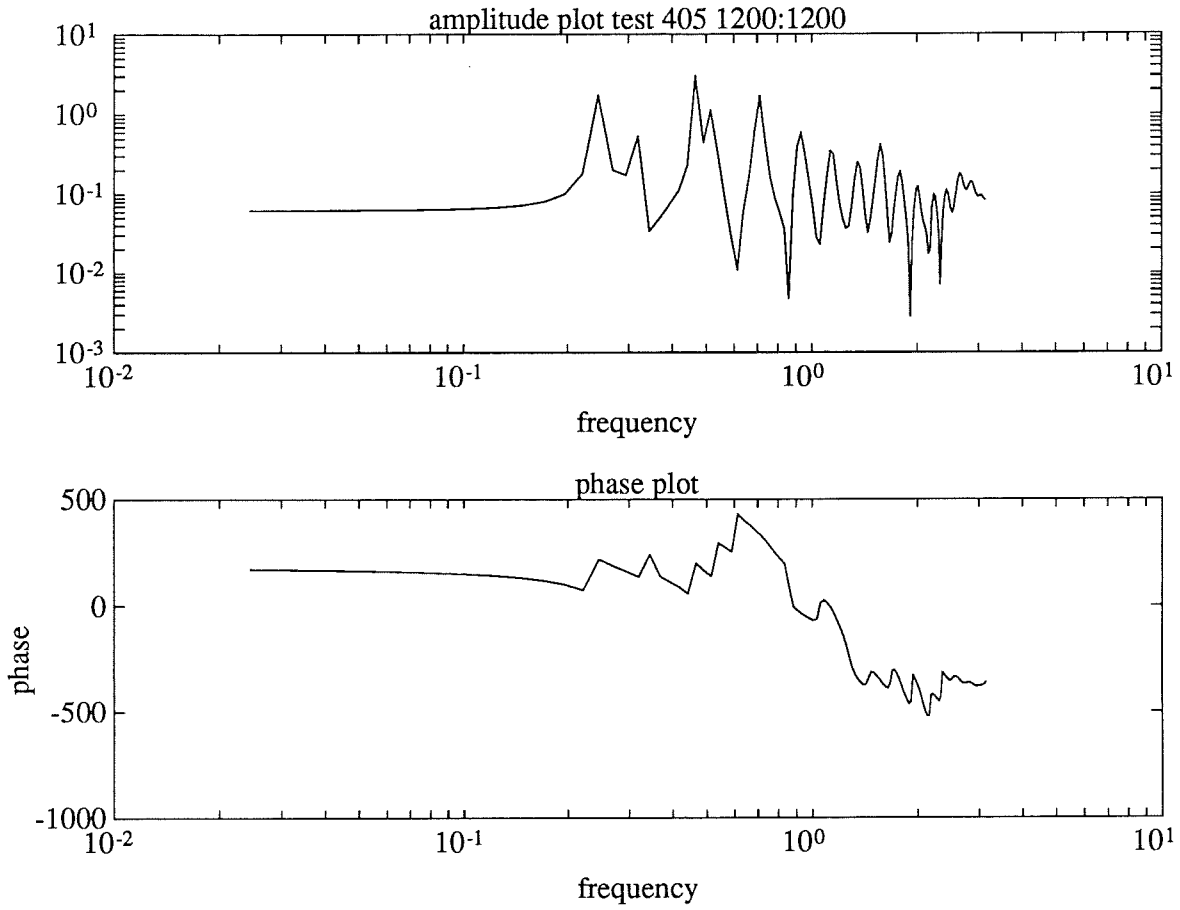

Txy - Transfer function phase
Degrees
200
100
0
-100
-200
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

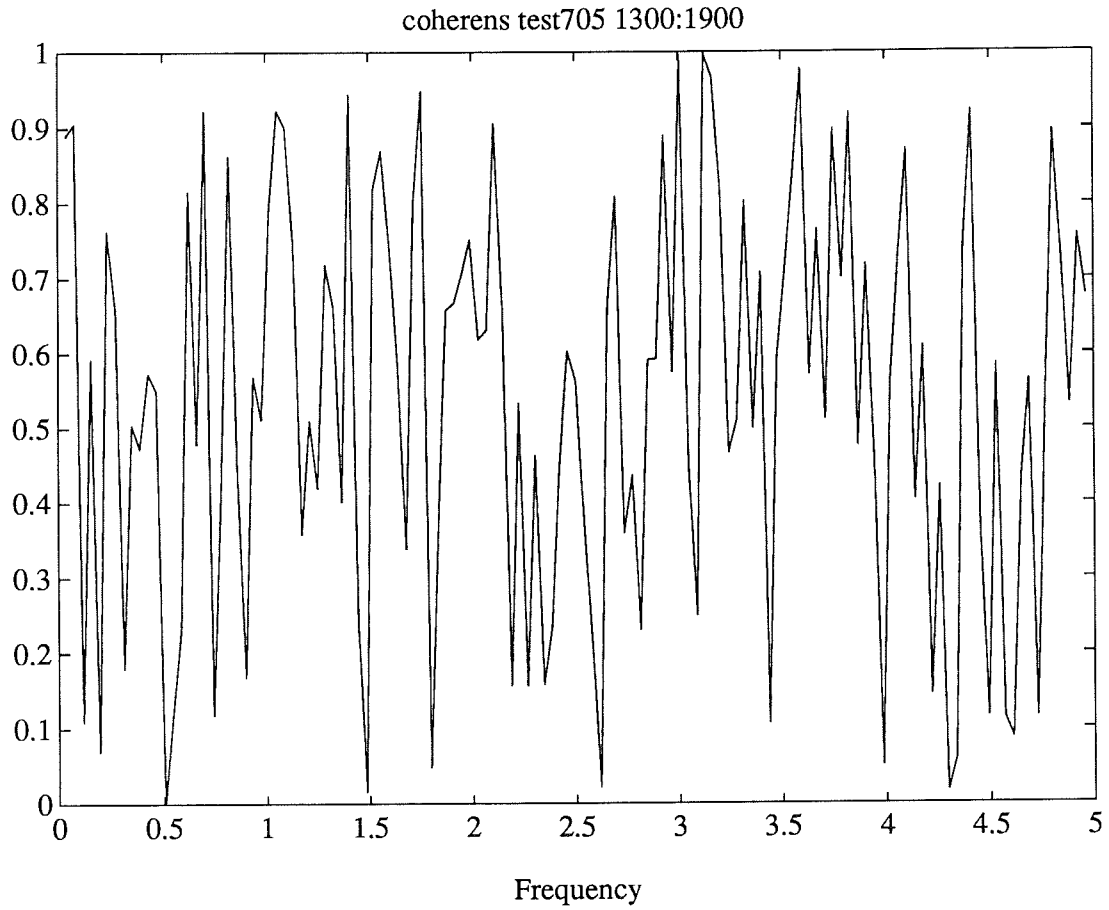

Txy - Transfer function magnitude test 905
10^1
10^0
10^-1
10^-2
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

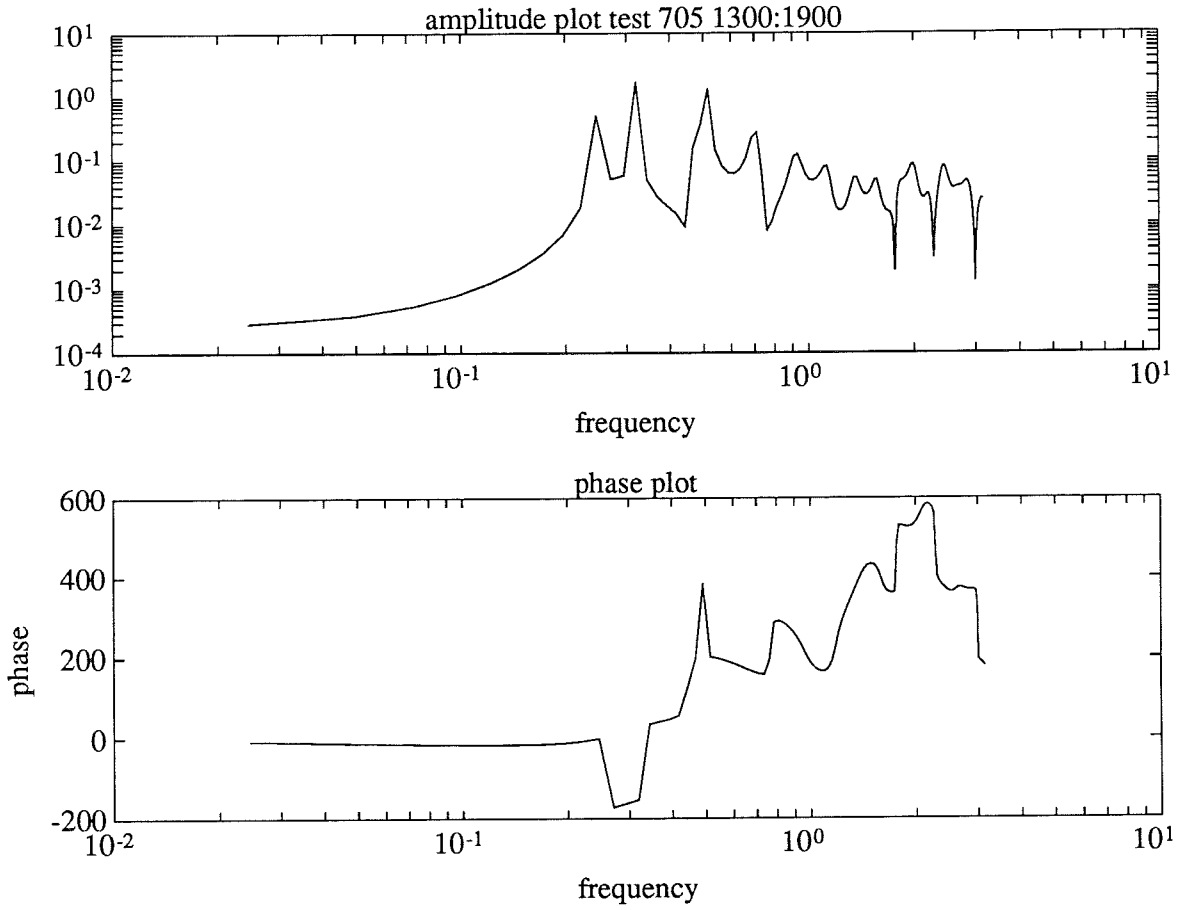

Txy - Transfer function phase
Degrees
200
100
0
-100
-200
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

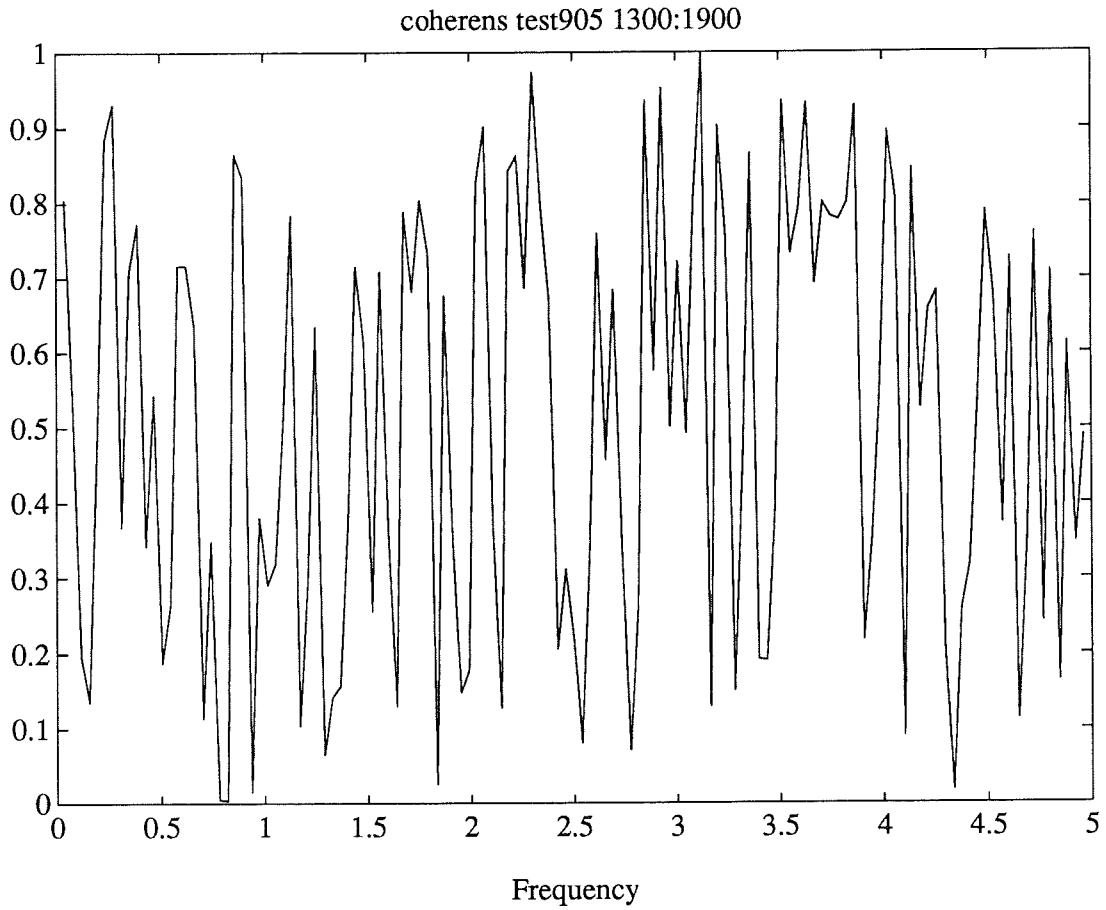

Txy - Transfer function magnitude test 1205
10^1
10^0
10^-1
10^-2
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency
Txy - Transfer function phase
200
100
0
-100
-200
Degrees
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

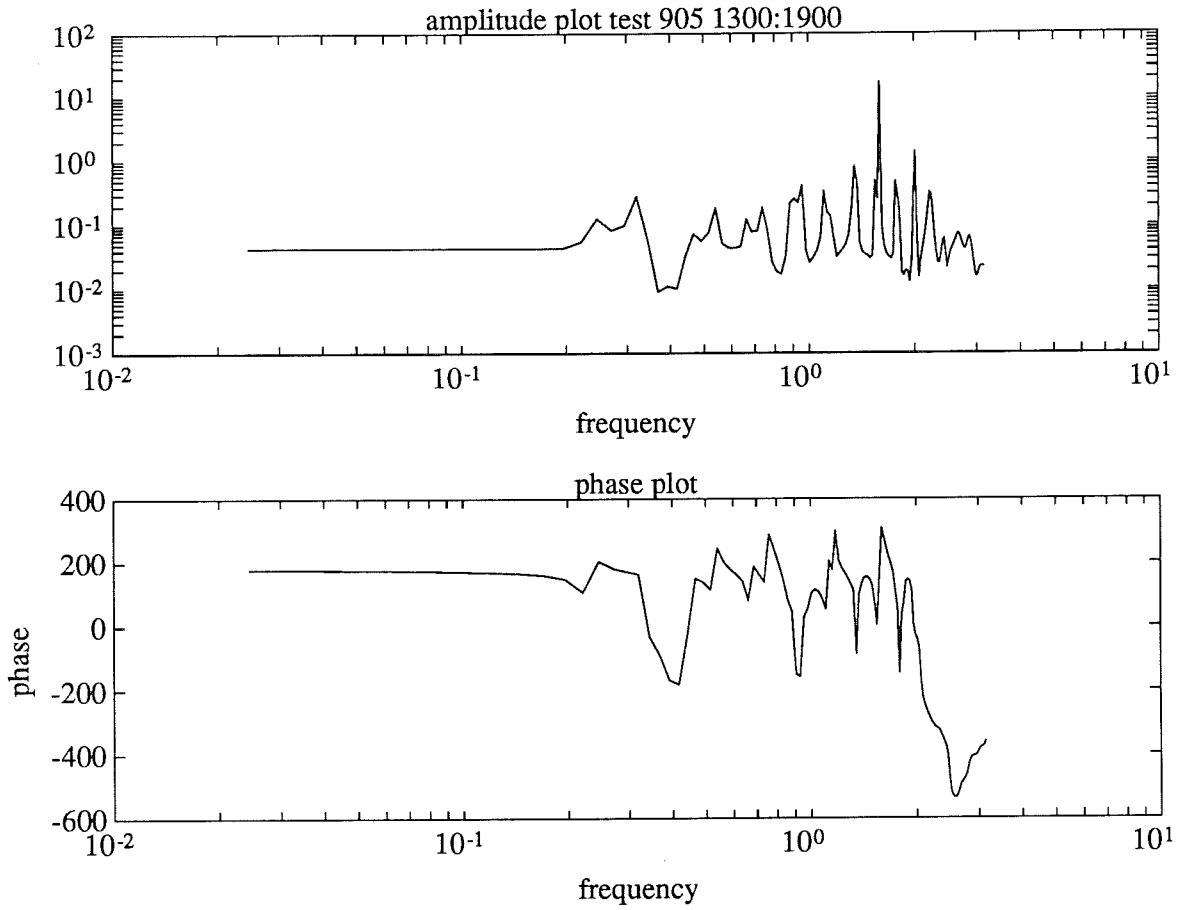

coherens test405 1200:1900
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

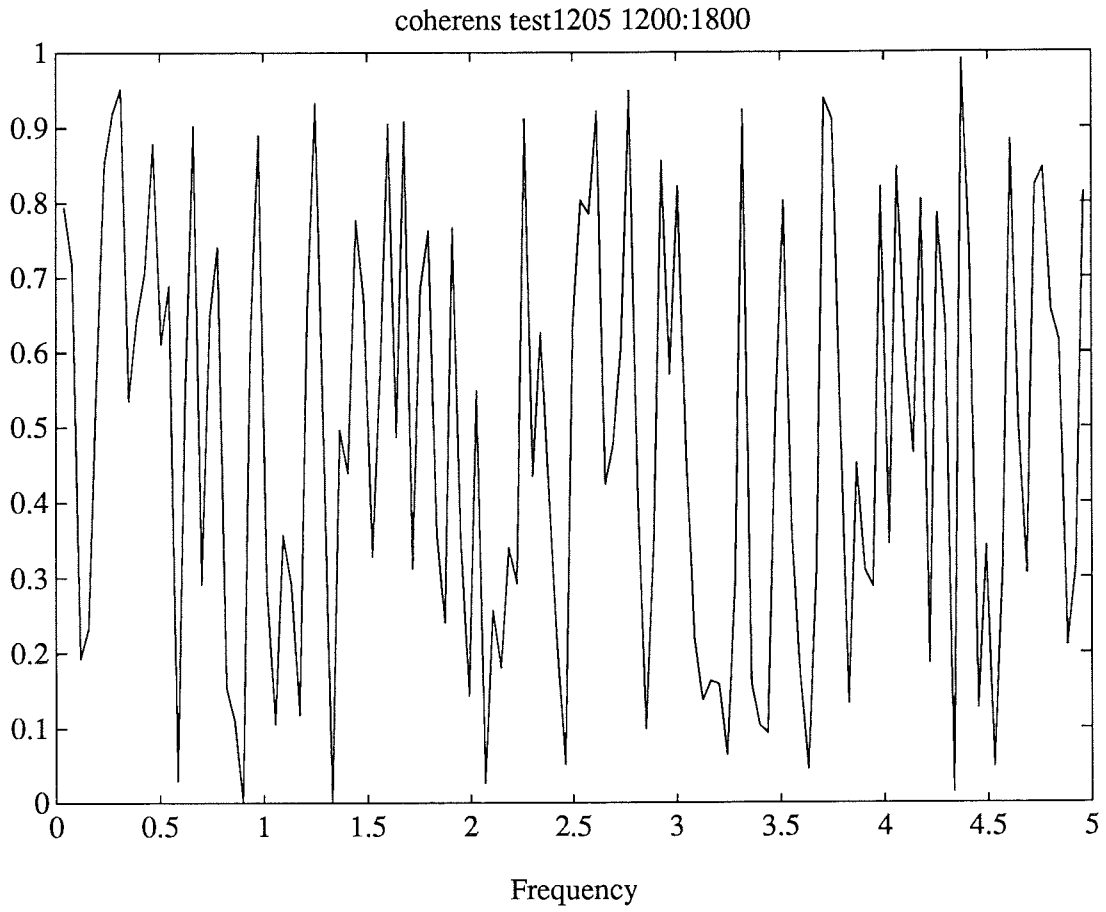

amplitude plot test 405 1200:1200
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
frequency
phase plot
500
0
-500
-1000
phase
10^{-2}
10^{-1}
10^{0}
10^{1}
frequency

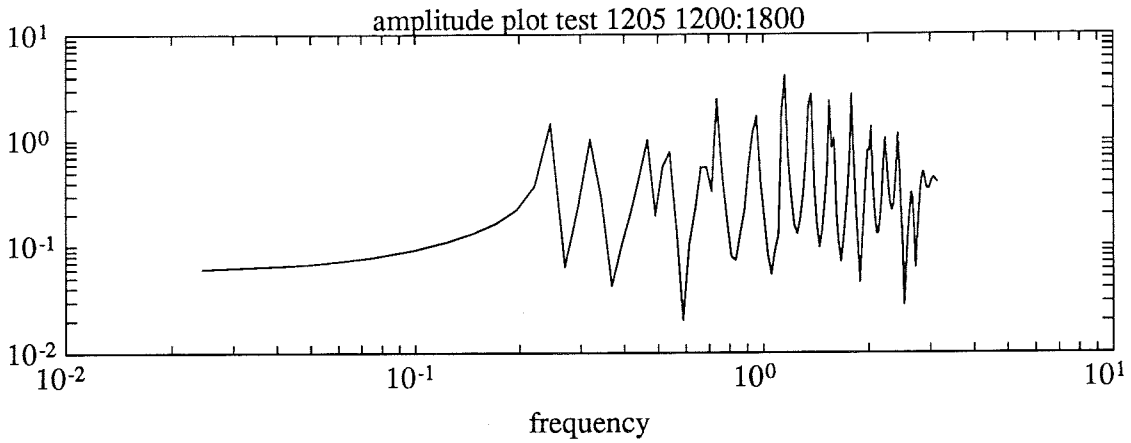

coherens test705 1300:1900
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

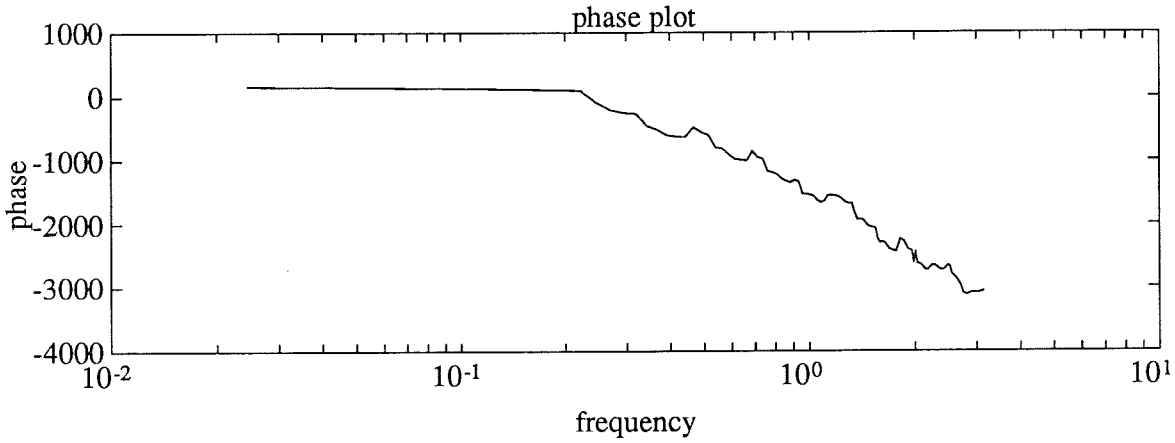

amplitude plot test 705 1300:1900
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-2}
10^{-1}
10^{0}
10^{1}
frequency
phase plot
600
400
200
0
-200
phase
10^{-2}
10^{-1}
10^{0}
10^{1}
frequency

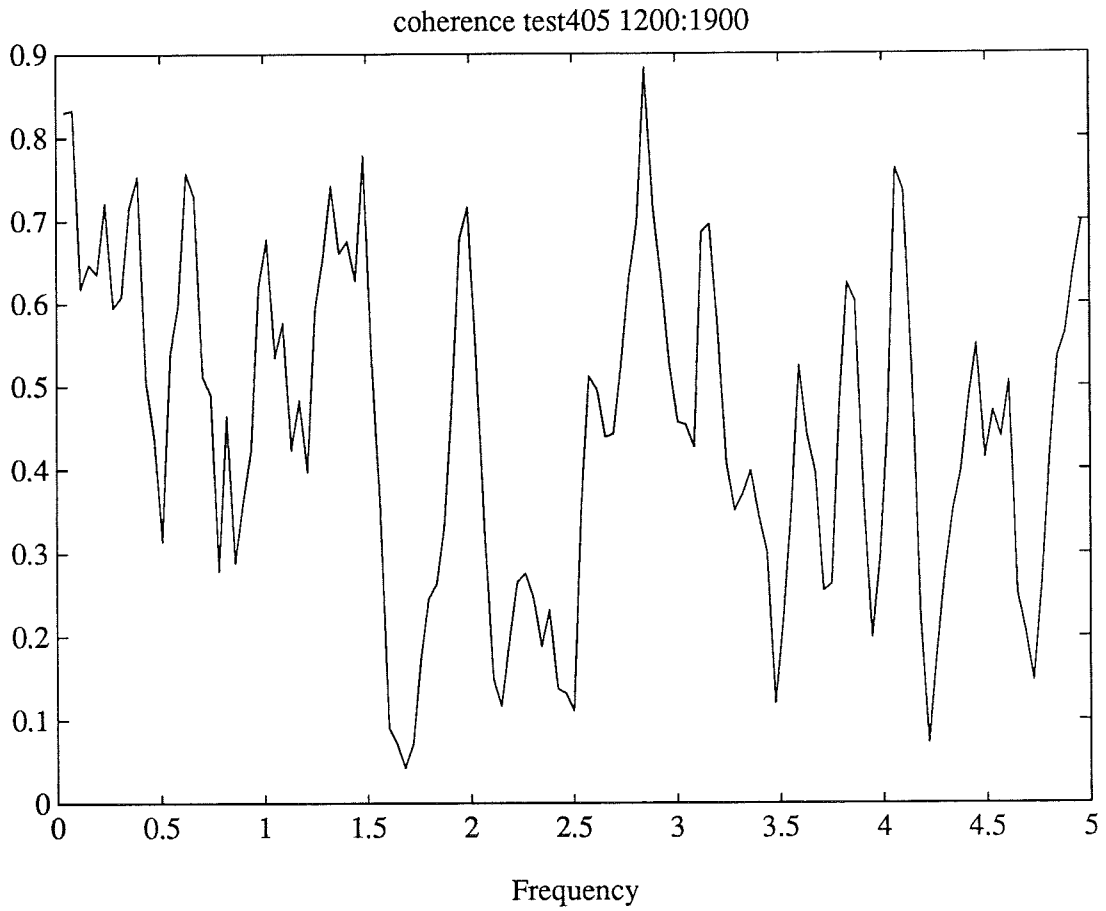

coherens test905 1300:1900
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

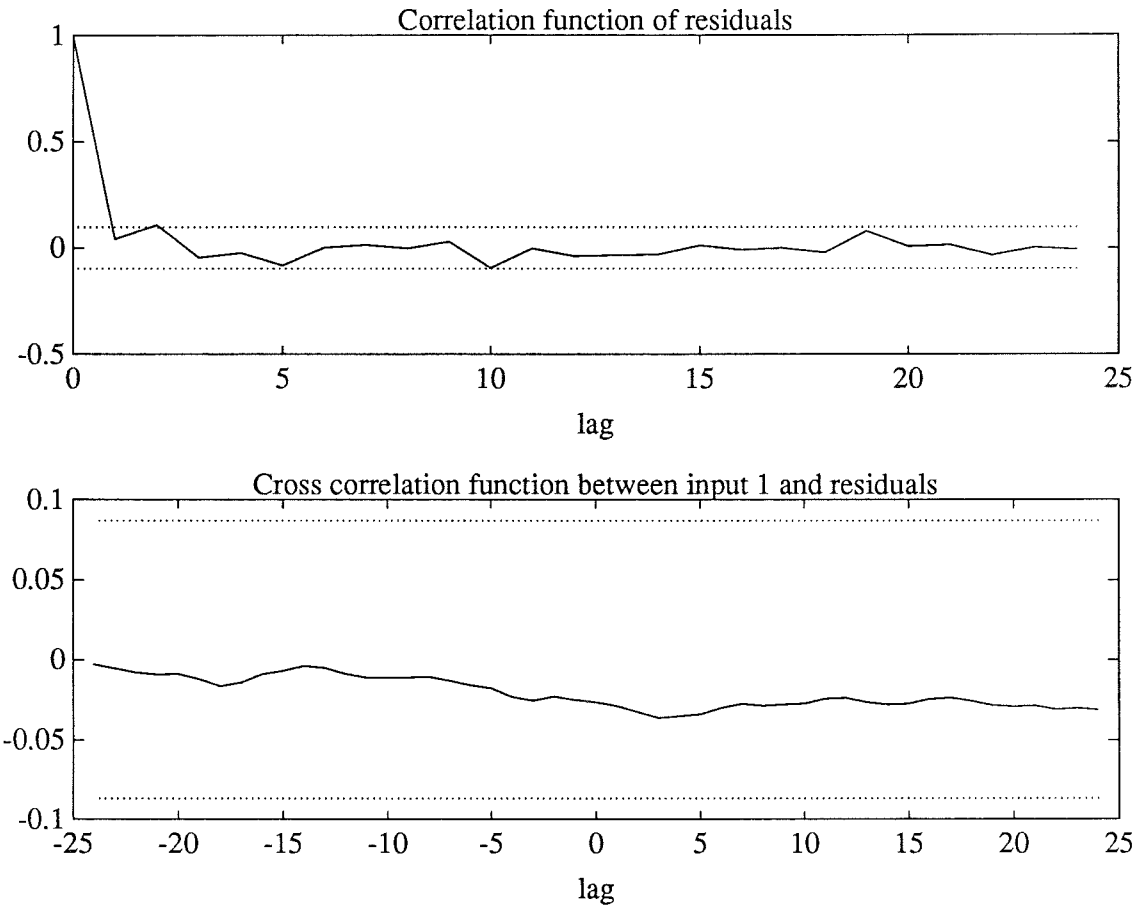

amplitude plot test 905 1300:1900
10^2
10^1
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-2}
10^{-1}
10^0
10^1
frequency
phase plot
400
200
0
-200
-400
-600
phase
10^{-2}
10^{-1}
10^0
10^1
frequency

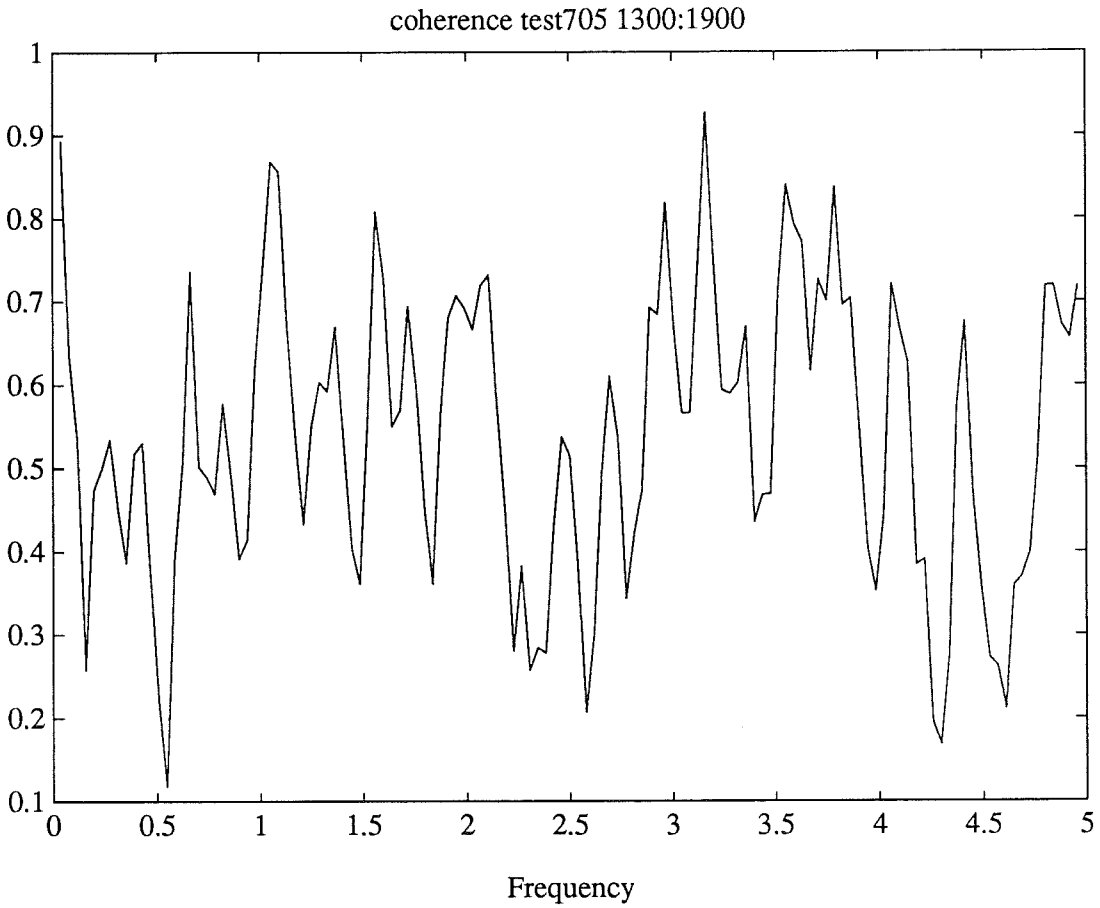

coherens test1205 1200:1800
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

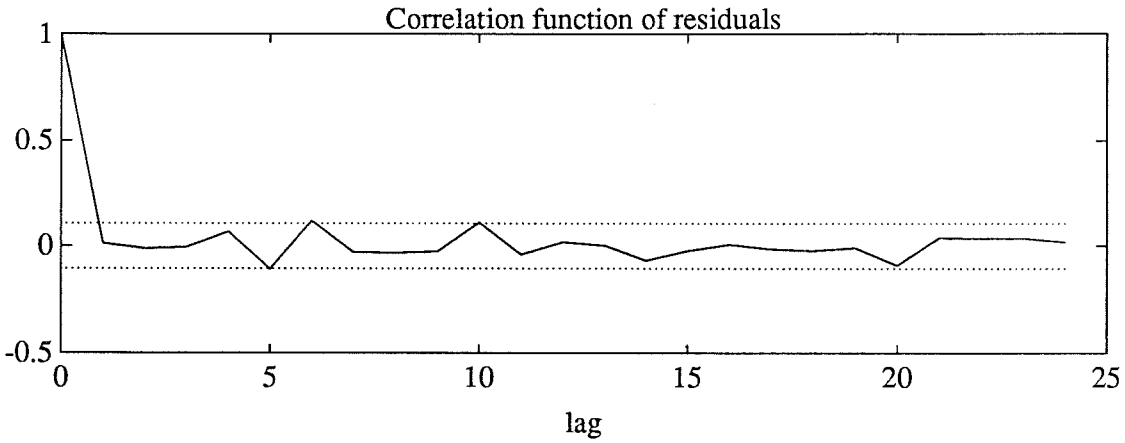

amplitude plot test 1205 1200:1800
10^1
10^0
10^{-1}
10^{-2}
10^{-2}
10^{-1}
10^0
10^1
frequency

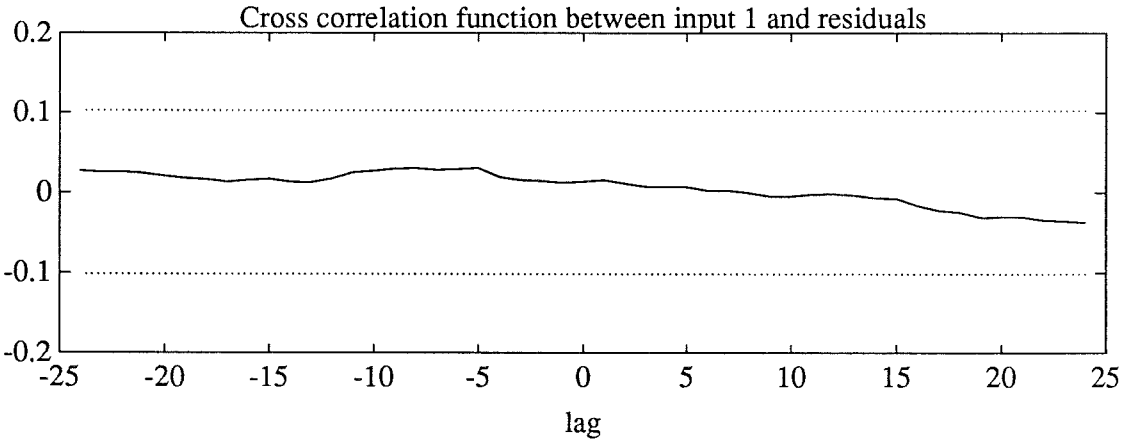

phase plot
1000
0
-1000
-2000
-3000
-4000
phase
10^{-2}
10^{-1}
10^0
10^1
frequency

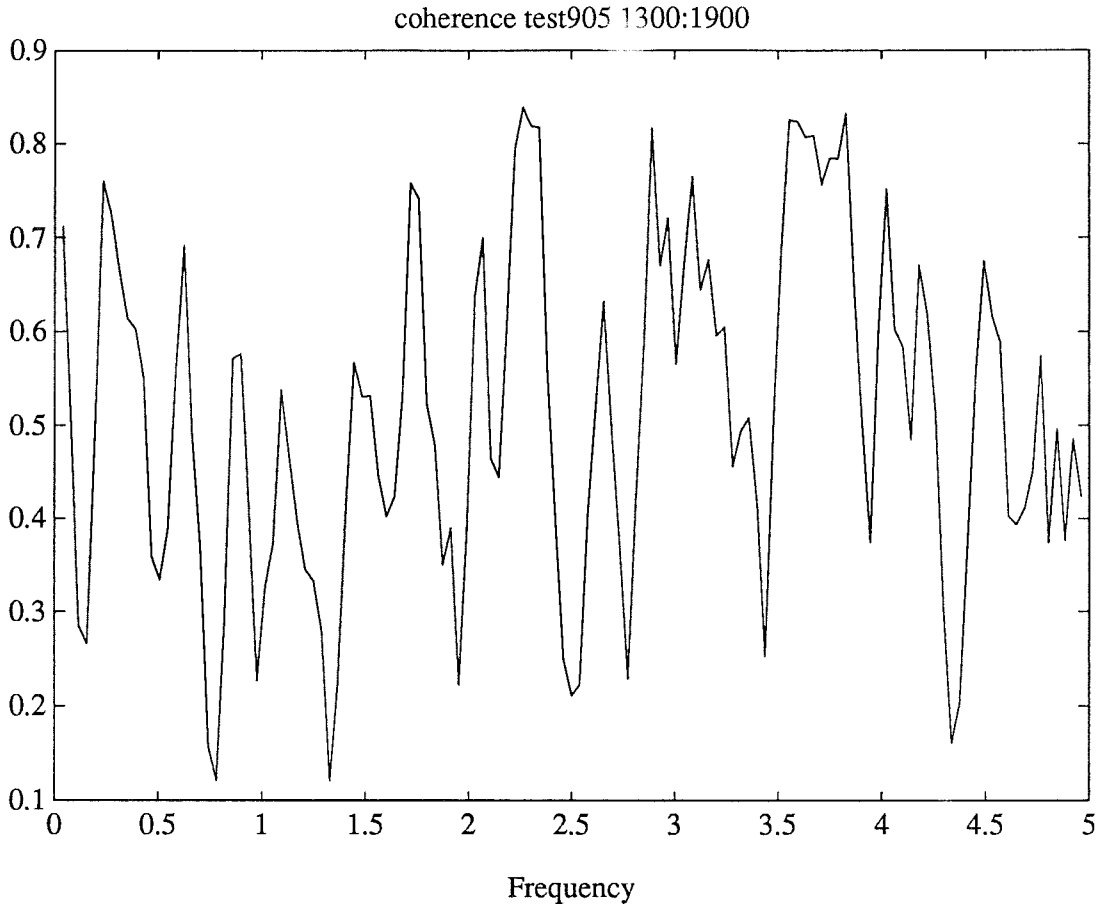

coherence test405 1200:1900
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

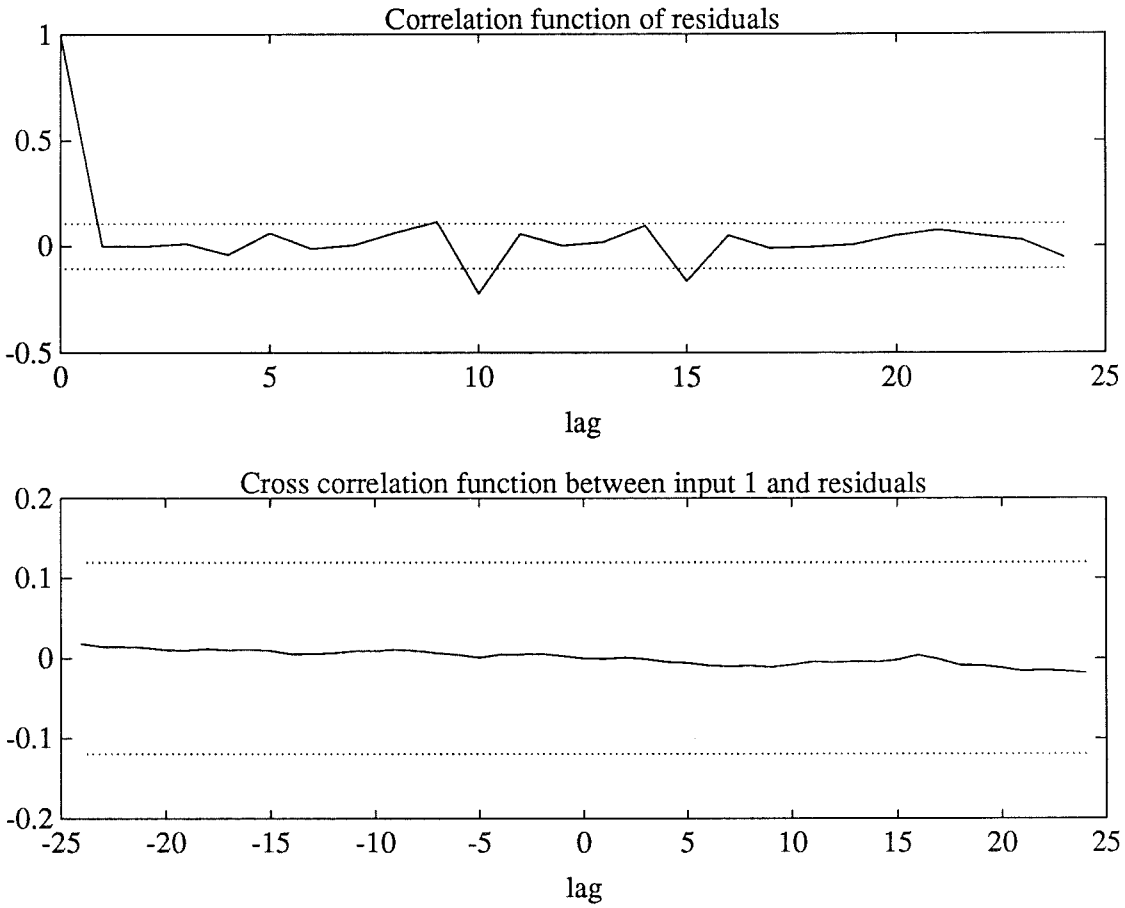

Correlation function of residuals
1
0.5
0
-0.5
0
5
10
15
20
25
lag
Cross correlation function between input 1 and residuals
0.1
0.05
0
-0.05
-0.1
-25
-20
-15
-10
-5
0
5
10
15
20
25
lag

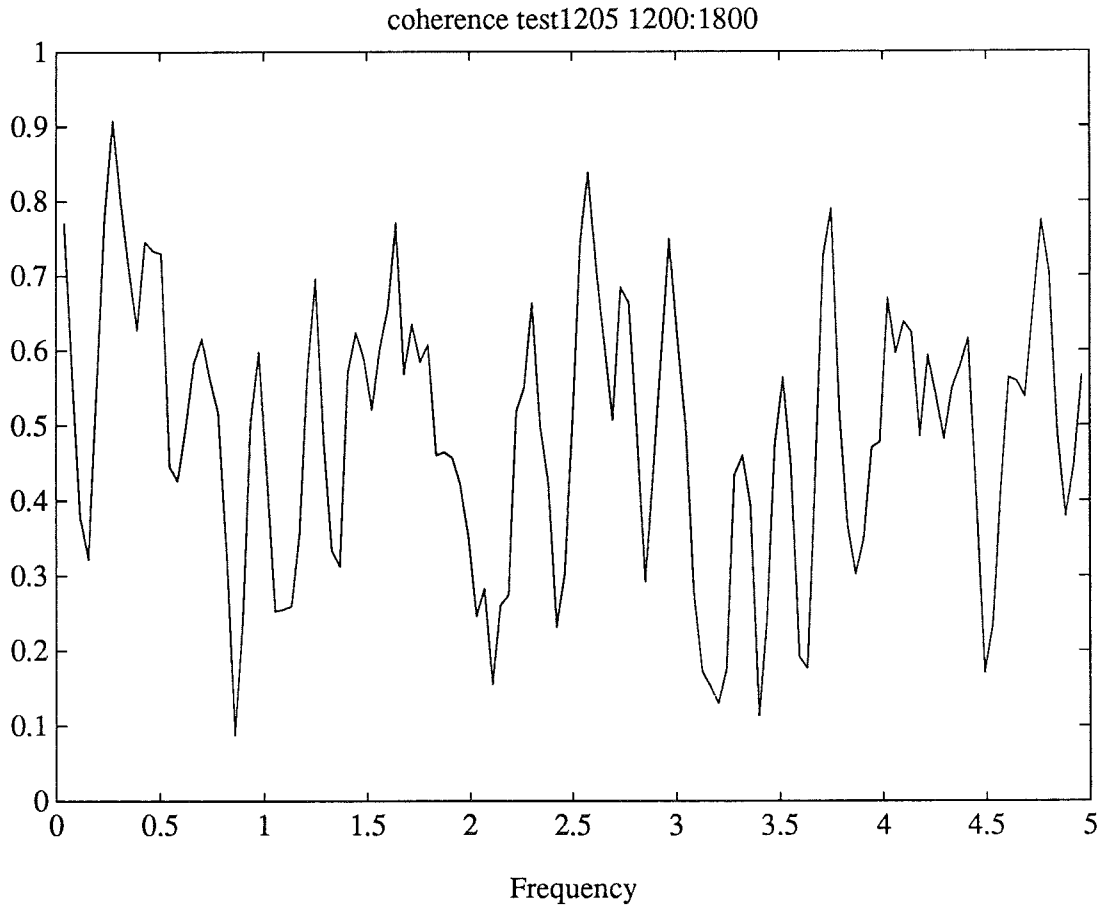

coherence test705 1300:1900
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

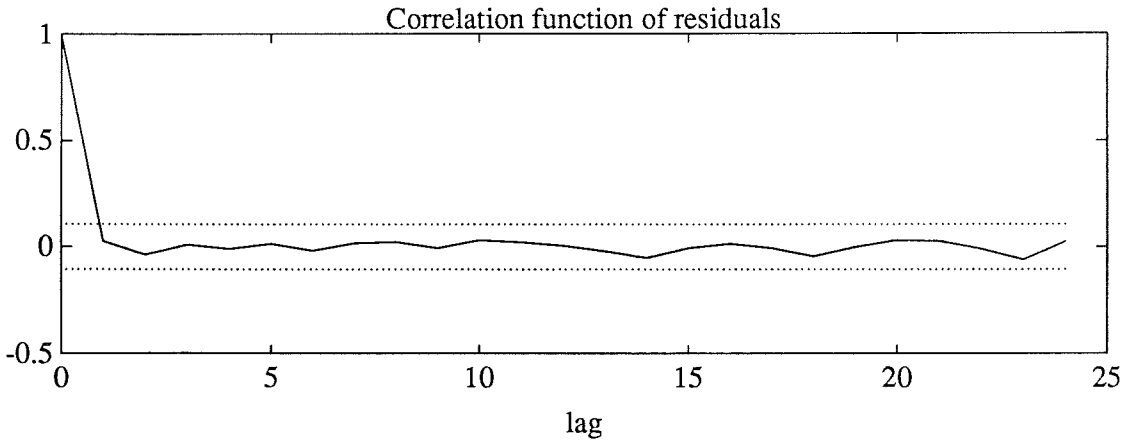

Correlation function of residuals
1
0.5
0
-0.5
0
5
10
15
20
25
lag

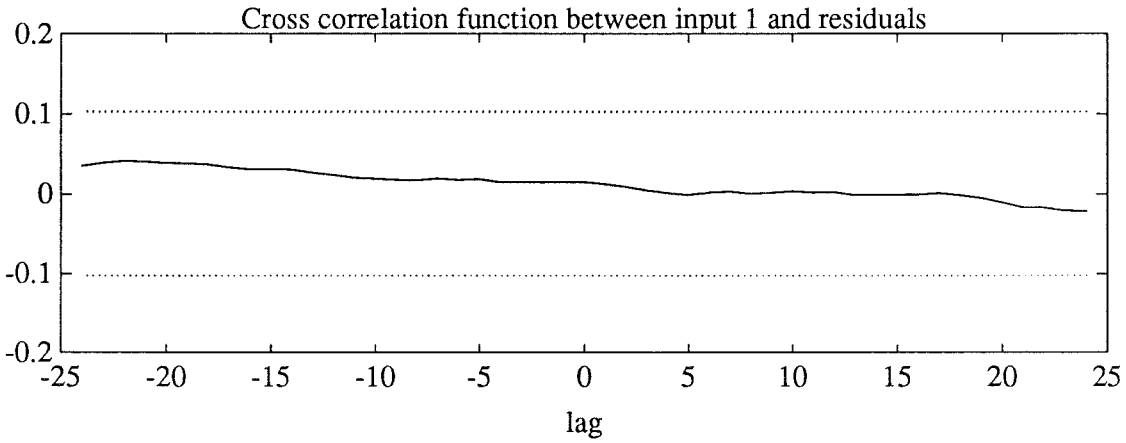

Cross correlation function between input 1 and residuals
0.2
0.1
0
-0.1
-0.2
-25
-20
-15
-10
-5
0
5
10
15
20
25
lag

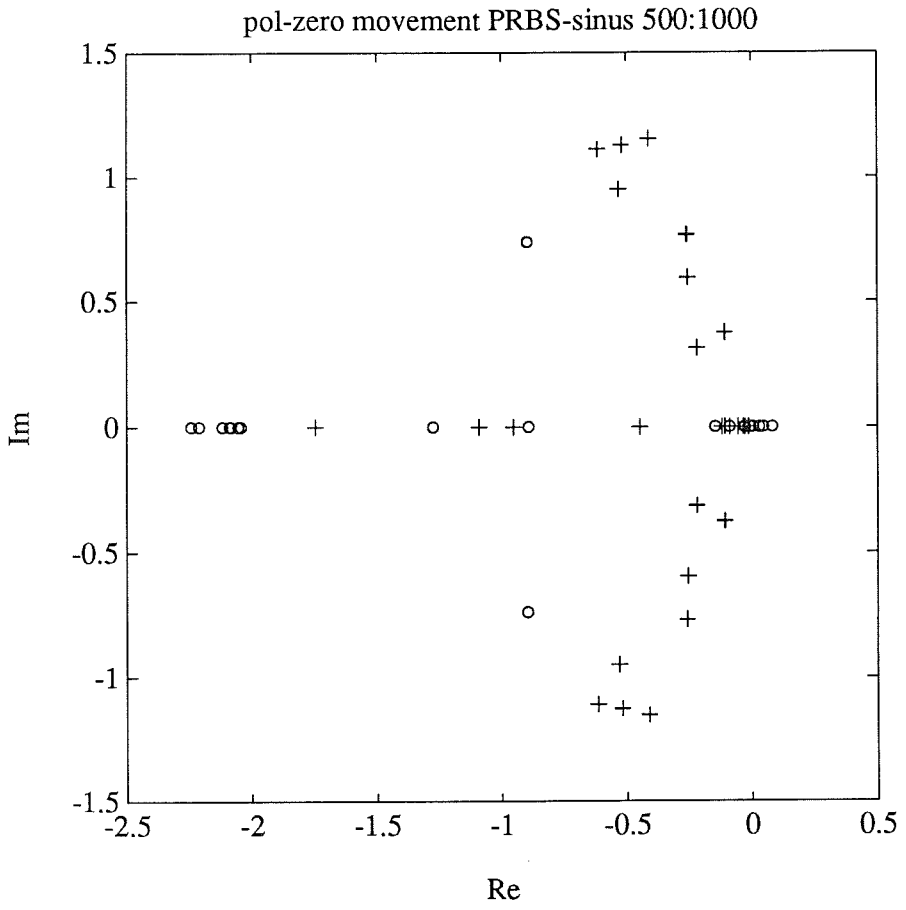

coherence test905 1300:1900
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

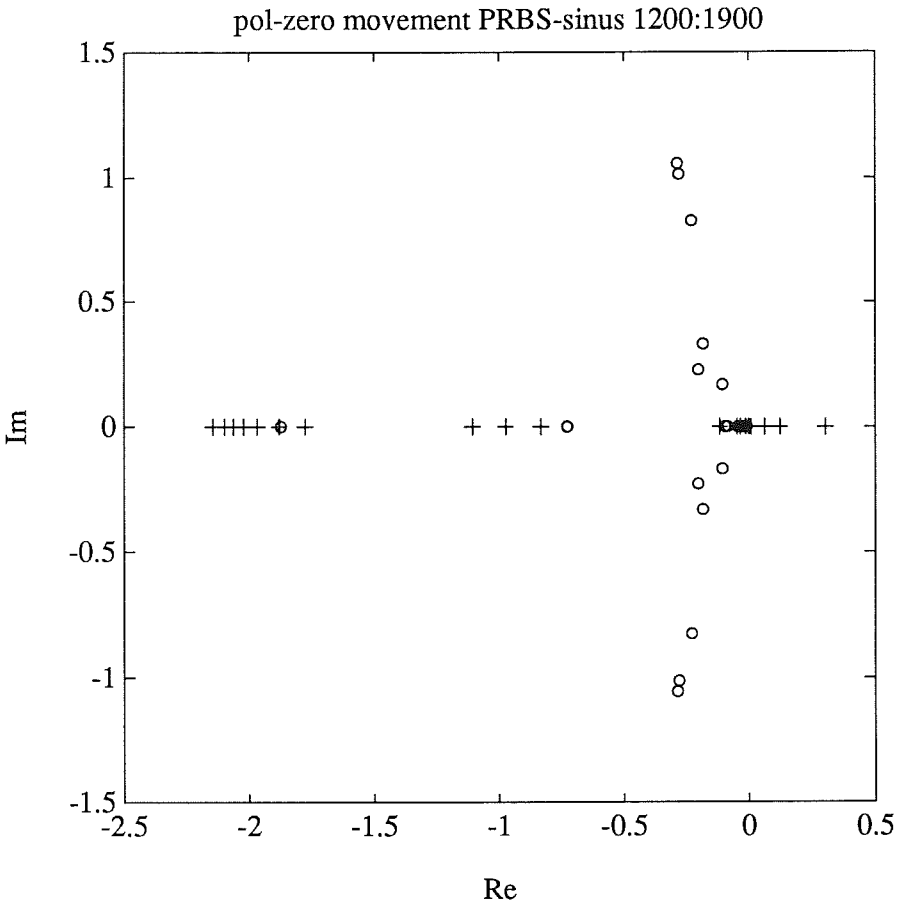

Correlation function of residuals
1
0.5
0
-0.5
0
5
10
15
20
25
lag
Cross correlation function between input 1 and residuals
0.2
0.1
0
-0.1
-0.2
-25
-20
-15
-10
-5
0
5
10
15
20
25
lag

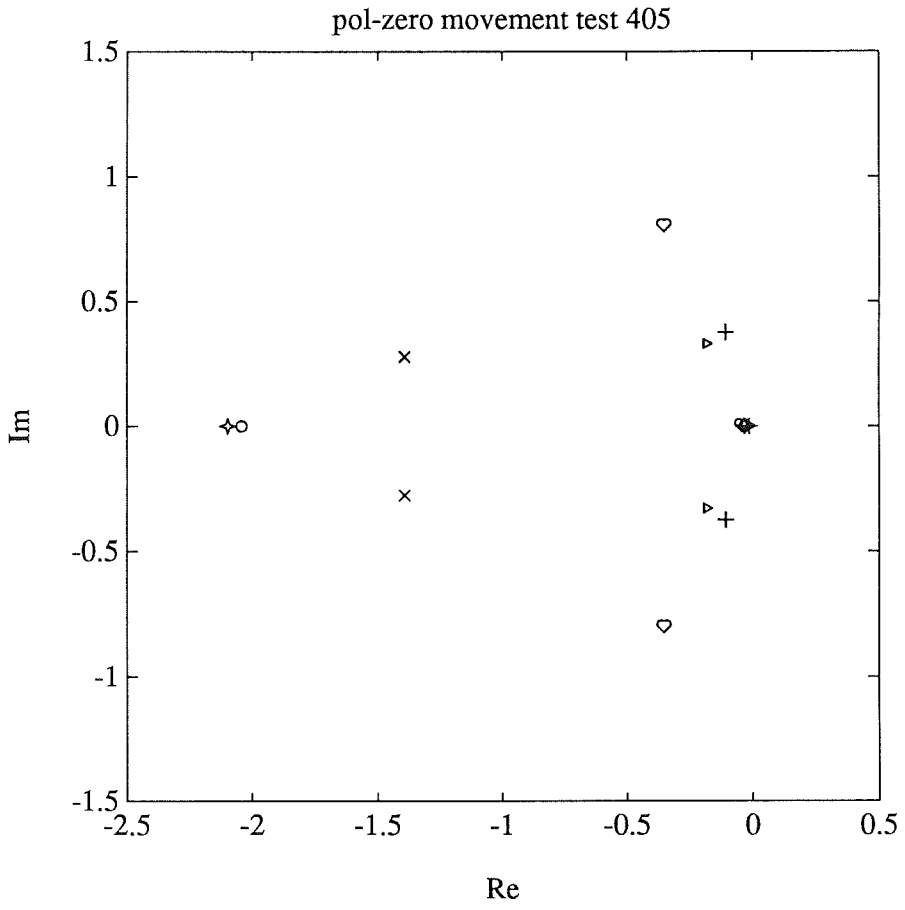

coherence test1205 1200:1800
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Frequency

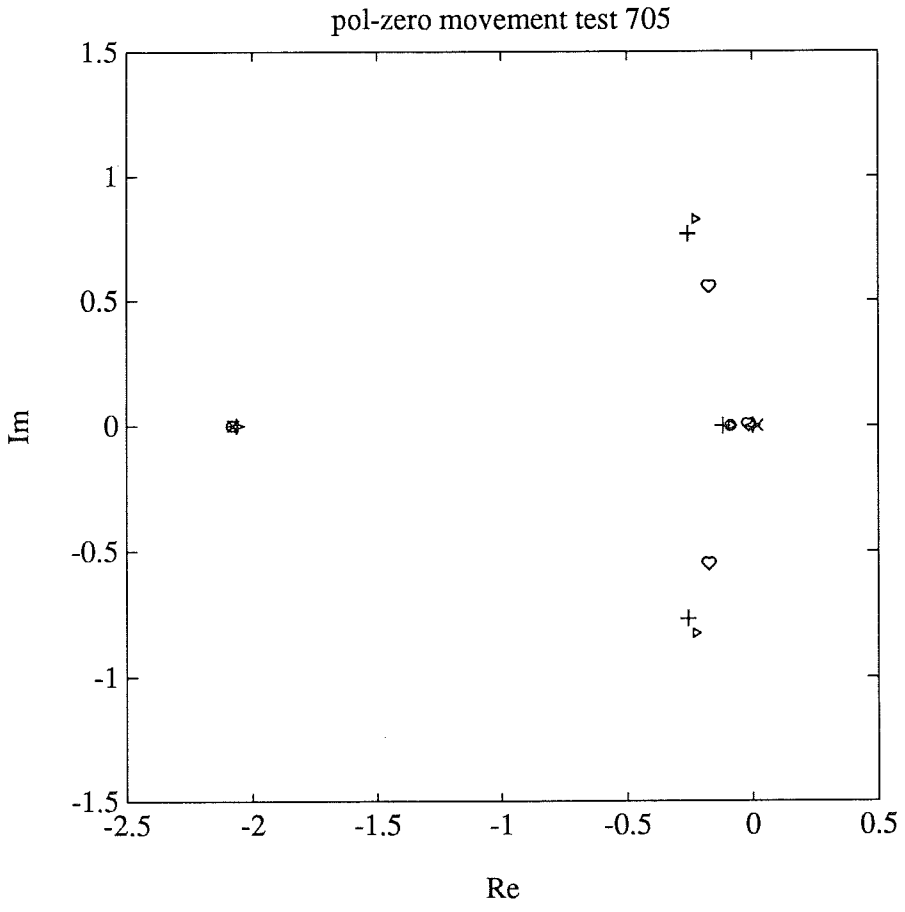

Correlation function of residuals
1
0.5
0
-0.5
0
5
10
15
20
25
lag

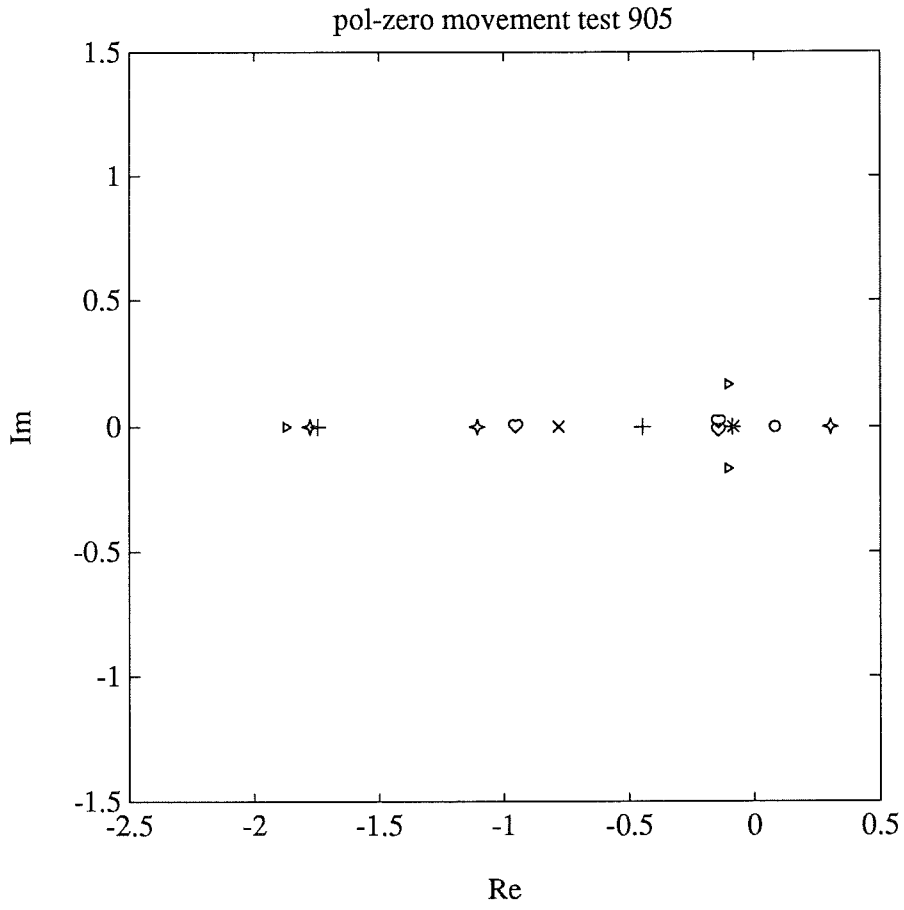

Cross correlation function between input 1 and residuals
0.2
0.1
0
-0.1
-0.2
-25
-20
-15
-10
-5
0
5
10
15
20
25
lag

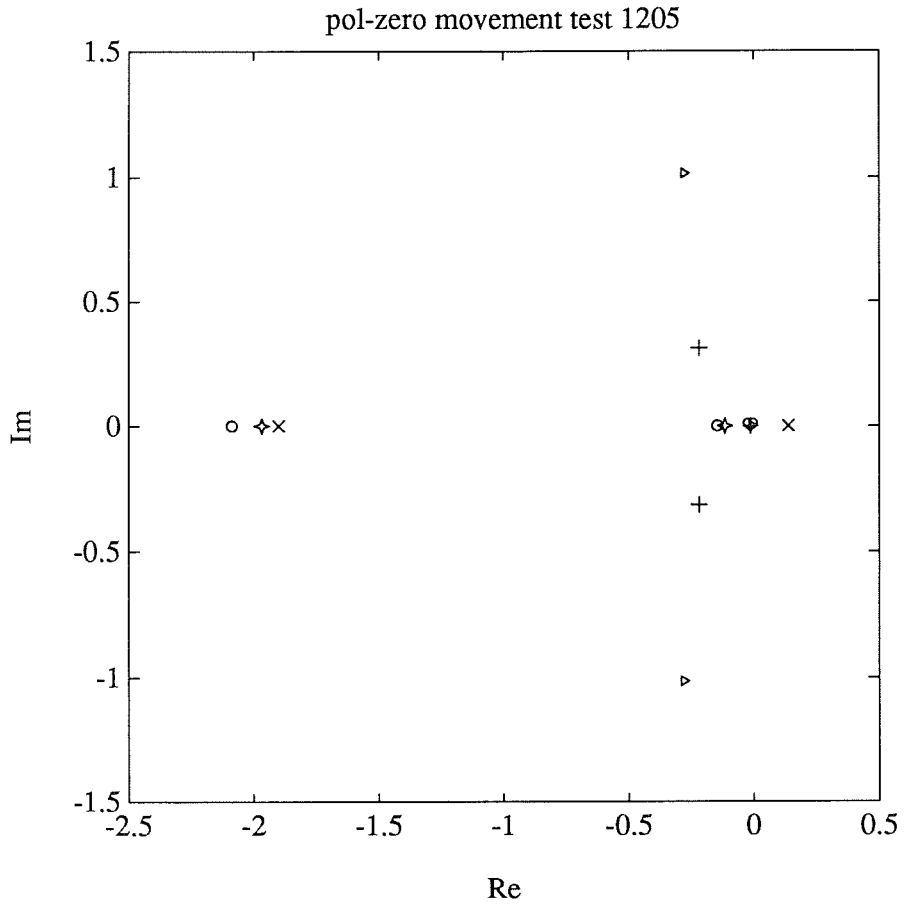

pol-zero movement PRBS-sinus 500:1000
1.5
1
0.5
0
-0.5
-1
-1.5
Im
-2.5
-2
-1.5
-1
-0.5
0
0.5
Re


pol-zero movement PRBS-sinus 1200:1900
1.5
1
0.5
0
-0.5
-1
-1.5
Im
-2.5
-2
-1.5
-1
-0.5
0
0.5
Re


pol-zero movement test 405
1.5
1
0.5
0
-0.5
-1
-1.5
Im
-2.5
-2
-1.5
-1
-0.5
0
0.5
Re


pol-zero movement test 705
1.5
1
0.5
0
-0.5
-1
-1.5
Im
-2.5
-2
-1.5
-1
-0.5
0
0.5
Re


pol-zero movement test 905
1.5
1
0.5
0
-0.5
-1
-1.5
Im
-2.5
-2
-1.5
-1
-0.5
0
0.5
Re


pol-zero movement test 1205
1.5
1
0.5
0
-0.5
-1
-1.5
Im
-2.5
-2
-1.5
-1
-0.5
0
0.5
Re

Appendix VIII: Program-listings

This appendix contains listings over the control- and measure program VISCON. The program consists of a total of 9 separate modules for different purposes.

* *Viscon* : Main module for starting and initiation of all processes. The program is also ended in this module.

* *Decl* : The different processes communicates through a number of monitors, whose purpose are to protect common data and syncronise the activity of different processes to each other. All monitor routines have been collected in this module.

* *Control* : Contains all routines necessary for the user interface. These routines control the activity of the other processes. There are also routines for the error- and help processes.

* *Measure* : Holds procedures for motor control and measuring. Here are also routines for handling data and external files.

* *Safe* : Contains a special emergency process for a quick stop of the motor.

* *Testgenerator* : Procedures for generating the test sequences.

* *Plotprocess* : Holds all procedures for plotting selected data to screen.

* *Estimprocess* : Contains algorithms for estimating a control model in real time. Data for the estimation are collected from the calculation procedure in Measure

```
MODULE Viscon;

 IMPORT RTMouse,DebugPMD;

 FROM Decl           IMPORT PBox,MBox,EBox,ErrorBox,PMailType,
                            TheEnd,InitMonitors,OutBox;

 FROM Safe           IMPORT SafeProc;

 FROM Measure        IMPORT DataMeasure;

 FROM Control        IMPORT InitControl;

 FROM EstimProcess   IMPORT Estimate;

 FROM HelpProcess    IMPORT HelpProc;

 FROM TestGenerator IMPORT TestManager;

 FROM PlotProcess    IMPORT Plot;

 FROM Graphics       IMPORT Shutdown;

 FROM Messages       IMPORT InitMailBox,SendMessage,AcceptMessage;

 FROM Kernel         IMPORT MaxPriority,CreateProcess,InitSem,Wait;

 FROM AnalogIO       IMPORT DAOut;

 FROM Storage        IMPORT ALLOCATE,DEALLOCATE;

 CONST  PoolSize=10;

 VAR              i : CARDINAL;
              PMail : PMailType;

 BEGIN
    RTMouse.Init;
    InitSem(TheEnd,0);
    InitMonitors;
    InitMailBox(PBox,2*PoolSize);
    InitMailBox(MBox,PoolSize);
    InitMailBox(EBox,PoolSize);
    InitMailBox(ErrorBox,10);
    InitMailBox(OutBox,10);
    FOR i:=1 TO PoolSize DO (* Create message pooles *)
      NEW(PMail);
      SendMessage(MBox,PMail);
      NEW(PMail);
      SendMessage(EBox,PMail);
    END;
    InitControl;
    CreateProcess(DataMeasure,5000);
    CreateProcess(Estimate,5000);
    CreateProcess(TestManager,3000);
    CreateProcess(SafeProc,1000);
    CreateProcess(Plot,5000);
    CreateProcess(HelpProc,1000);
    Wait(TheEnd);
    DAOut(0,0.0);
    DAOut(1,0.0);
    Shutdown;
 END Viscon.
```

```
DEFINITION MODULE Decl;

FROM Messages IMPORT MailBox;

FROM Kernel IMPORT Semaphore;

FROM Graphics IMPORT handle,rectangle;

EXPORT QUALIFIED     PlotKinds,WhatType,MeasType,SpeedType,TestText,
                     FileText,MeasureType,DataType,MotorType,
                     ErrorPtrType,ErrorType,ABType,PMailType,
                     PlotWindowType,GetNr,PutNr,
                     PBox,MBox,EBox,OutBox,ErrorBox,TheEnd,
                     InitMonitors,
                     PutCLSpec,GetCLSpec,GetEstim,GetEstPar,SetReady,
                     SetFilled,GetCLDeg,
                     Write,Error,
                     PutMV,GetMV,GetValue,GetDataPar,SetPreLength,
                     GetPreLength,SetDelay,ResetEstim,PutEmerg,
                     SetFileConst,GetFileConst,
                     CheckWhat,PlotData,PlotSekv,PlotEstim,ChangeWhat,
                     CheckPlotWhat,
                     PausTest,SetTest,GetTest,GetFileName,
                     SetMeasWork,GetMeasWork,HOldData,HSekv,HMeas,
                     StopMeasure,StopTest,GetMomentData,
                     PutMomentData,EndMeasure,GetSample,GetCounter,
                     SetBeep,SetEnabled,GetSpeed,PutSpeed,GetMotorPar,
                     GetMotorData,PutMotorData,GetTStart,SetTStart;

TYPE     PlotKinds    =(Re,Es);
         WhatType     =(Par,Test,Data,Oper);
         MeasType     =ARRAY[1..6] OF REAL;
         SpeedType    =ARRAY[1..2] OF REAL;
         TestText     =ARRAY[0..8] OF CHAR;
         FileText     =ARRAY[0..11] OF CHAR;
         MeasureType  =(OldData,Sekv,Meas,Paus);
         DataType     =RECORD
                          u : SpeedType;
                          y : MeasType;
                       END;
         MotorType    =RECORD
                          RampTime,WaitPaus,BreakTime,
                          BeepDelay,Period,MaxSpeed,RandomTime,
                          RandomSpeed,tmax,PRBSPeriod          : REAL;
                       END;
         ErrorType    =ARRAY[0..20] OF CHAR;
         ErrorPtrType=POINTER TO RECORD
                          ErrStr:ErrorType;
                        END;
         ABType       =ARRAY[1..4] OF REAL;
         PMailType    =POINTER TO RECORD
           CASE  Kind:PlotKinds OF
                    Re :        PForce,TForce : ARRAY[1..3] OF REAL;
                                        u,uref : REAL|
                    Es : Start,Ready,ExChange : BOOLEAN;
                                            A,B : ABType;
                                              V : REAL;
                  END;
           END;
         PlotWindowType = POINTER TO RECORD
                             XMin, YMin, XMax, YMax : REAL;
                                    WindowRectangle : rectangle;
                                                  H : handle;
                           END;
```

```
 VAR             PBox,MBox,EBox,ErrorBox,OutBox : MailBox;
                                        TheEnd : Semaphore;

(* Initiation *)
PROCEDURE InitMonitors;
(* CLSpecMonitor *)
PROCEDURE PutCLSpec(A1,B1:ABType; MinErr1,P01,Lambda1 : REAL;
                    CLDeg1 : INTEGER; Estim1,Reset1 : BOOLEAN);
PROCEDURE GetCLSpec(VAR A1,B1 : ABType; VAR MinErr1,P01,Lambda1: REAL ;
                    VAR CLDeg1 : INTEGER; VAR Reset1,Ready1,
                    PreFilled1,Estim1 : BOOLEAN);
PROCEDURE GetCLDeg(VAR CLDeg1 : INTEGER);
PROCEDURE GetEstim(VAR Estim: BOOLEAN);
PROCEDURE GetEstPar(VAR Ready1,PreFilled1: BOOLEAN);
PROCEDURE SetReady(Ready1 : BOOLEAN);
PROCEDURE SetFilled(PreFilled1 : BOOLEAN);
(* Errormonitor *)
PROCEDURE Error(ErrStr:ErrorType);
PROCEDURE Write(ErrStr:ErrorType);
(* DataMonitor *)
PROCEDURE PutMV(U,Y:REAL);
PROCEDURE GetMV(VAR U,Y:REAL);
PROCEDURE GetValue(VAR U,Y:REAL;k:CARDINAL);
PROCEDURE GetDataPar(VAR PreLength1 : CARDINAL;
                     VAR Emerg1 : BOOLEAN);
PROCEDURE SetPreLength(Length:CARDINAL);
PROCEDURE GetPreLength(VAR Length:CARDINAL);
PROCEDURE SetFileConst(Number : CARDINAL);
PROCEDURE GetFileConst(VAR Number : CARDINAL);
PROCEDURE SetDelay(Delay1:CARDINAL);
PROCEDURE ResetEstim;
PROCEDURE PutEmerg(e:BOOLEAN);
(* Plotmonitor *)
PROCEDURE CheckWhat(VAR WhatKind:WhatType);
PROCEDURE PlotData():BOOLEAN;
PROCEDURE PlotSekv():BOOLEAN;
PROCEDURE PlotEstim():BOOLEAN;
PROCEDURE PutNr( Nr : INTEGER );
PROCEDURE GetNr(VAR Nr : INTEGER );
PROCEDURE ChangeWhat(WhatKind:WhatType);
PROCEDURE CheckPlotWhat(VAR WhatKind:WhatType;VAR Change : BOOLEAN);
(* StatusMonitor *)
PROCEDURE PausTest;
PROCEDURE SetTest(Name : TestText;TestNr : CARDINAL);
PROCEDURE GetTest(VAR TestNr : CARDINAL);
PROCEDURE GetFileName(VAR Name : FileText);
PROCEDURE SetMeasWork(MWork : MeasureType);
PROCEDURE GetMeasWork(VAR MWork : MeasureType);
PROCEDURE StopMeasure;
PROCEDURE HOldData():BOOLEAN;
PROCEDURE HSekv():BOOLEAN;
PROCEDURE HMeas():BOOLEAN;
PROCEDURE EndMeasure():BOOLEAN;
PROCEDURE StopTest;
PROCEDURE GetSample(VAR h1 :REAL);
PROCEDURE GetCounter(VAR c:INTEGER);
PROCEDURE GetMomentData(VAR a1,b1,gamma1 : REAL);
PROCEDURE PutMomentData(c: INTEGER;h1,a1,b1,gamma1 : REAL);
(* MotorMonitor *)
PROCEDURE SetBeep(Beep1 : BOOLEAN);
PROCEDURE SetEnabled(Enabled1 : BOOLEAN);
PROCEDURE SetTStart(TStart1 : REAL);
PROCEDURE GetTStart(VAR TStart1 : REAL);
```

```
PROCEDURE GetSpeed(VAR CurrentSpeed:REAL);
PROCEDURE PutSpeed(CurrentSpeed,t : REAL;Test : INTEGER;
                   VAR TestEnd : BOOLEAN);
PROCEDURE GetMotorPar(VAR Enabled1,Beep1 : BOOLEAN;VAR Speed1:REAL);
PROCEDURE PutMotorData(MData : MotorType;Nr :INTEGER);
PROCEDURE GetMotorData(VAR MData : MotorType;Nr :INTEGER);

END Decl.
```

```
IMPLEMENTATION MODULE Decl;

FROM Messages IMPORT MailBox,SendMessage,ReceiveMessage,AcceptMessage,
                     InitMailBox;

FROM Kernel   IMPORT Wait,Signal,Semaphore,Event,InitEvent,Await,Cause,
                     InitSem;

FROM Storage  IMPORT ALLOCATE,DEALLOCATE;

FROM Graphics IMPORT rectangle,handle,WriteString,ReadString,HideCursor,
                     ShowCursor;


TYPE    CLSpecMonType =RECORD
                         CLSpecSem          : Semaphore;
                         E                  : Event;
                         A,B                : ABType;
                         MinErr,P0,Lambda   : REAL;
                         CLDeg              : INTEGER;
                         Estim,Reset,Ready,
                         PreFilled          : BOOLEAN;
                        END;
        PlotMonType   =RECORD
                         WhatSem : Semaphore;
                         Change  : BOOLEAN;
                         What    : WhatType;
                         HelpNr  : INTEGER;
                        END;
        StatMonType   =RECORD
                         StatSem      : Semaphore;
                         MeasureWork  : MeasureType;
                         Test         : CARDINAL;
                         Stop,NewName : BOOLEAN;
                         TestName     : FileText;
                         Counter      : INTEGER;
                         h,a,b,gamma  : REAL;
                         Change       : Event;
                        END;
        MotMonType    =RECORD
                         MotSem                 : Semaphore;
                         Enabled,Beep           : BOOLEAN;
                         Speed,TStart           : REAL;
                         MaxSpeed,RandomTime,
                         RampTime,WaitPaus,
                         BreakTime,tmax,Period,
                         PRBSPeriod,RandomSpeed,
                         BeepDelay              : ARRAY[1..5] OF REAL;
                         Change                 : Event;
                        END;
        DataMonType   =RECORD
                         DataSem                  : Semaphore;
                         Get,Put,Delay,PreLength,
                         FileConst                : CARDINAL;
                         Emerg                    : BOOLEAN;
                         UU,YY                    : ARRAY[1..1000]OF REAL;
                         Change                   : Event;
                        END;

VAR             CLSpecMon : CLSpecMonType;
                WhatMon   : PlotMonType;
                StatMon   : StatMonType;
                MotMon    : MotMonType;
```

```
                DataMon    : DataMonType;
                MVErr      : BOOLEAN;

(*********************************************************************)

(*****************    initial  procedure        **********************)


 PROCEDURE InitMonitors;

 BEGIN
    InitCLSpecMon;
    InitWhatMon;
    InitStatusMonitor;
    InitMotorMonitor;
    InitDataMonitor;
    MVErr:=FALSE;
 END InitMonitors;


(*********************************************************************)

(*****************    initial  procedure        **********************)


 PROCEDURE InitCLSpecMon;

 VAR   j : CARDINAL;

 BEGIN
    InitSem(CLSpecMon.CLSpecSem,1);
    InitEvent(CLSpecMon.E,CLSpecMon.CLSpecSem);
    WITH CLSpecMon DO
      FOR j:=1 TO 2 DO
        A[j]:=0.0;
        B[j]:=0.0;
      END;
    MinErr:=0.1;
    P0:=1000.0;
    Lambda:=0.95;
    CLDeg:=4;
    Estim:=FALSE;
    Reset:=TRUE;
    Ready:=FALSE;
    PreFilled:=FALSE;
   END;
 END InitCLSpecMon;


 PROCEDURE PutCLSpec(A1,B1:ABType; MinErr1,P01,Lambda1 : REAL;
                     CLDeg1 : INTEGER; Estim1,Reset1 : BOOLEAN);
 BEGIN
    WITH CLSpecMon DO
      Wait(CLSpecSem);
      A:=A1;
      B:=B1;
      MinErr:=MinErr1;
      P0:=P01;
      Lambda:=Lambda1;
      CLDeg:=CLDeg1;
      Estim:=Estim1;
      Reset:=Reset1;
      Cause(E);
```

```
      Signal(CLSpecSem);
    END;
END PutCLSpec;


PROCEDURE GetCLSpec(VAR A1,B1:ABType; VAR MinErr1,P01,Lambda1 : REAL;
                    VAR CLDeg1 : INTEGER; VAR Reset1,Ready1,
                    PreFilled1,Estim1 : BOOLEAN);
BEGIN
    WITH CLSpecMon DO
      Wait(CLSpecSem);
      WHILE NOT (PreFilled AND Estim) AND NOT Reset DO
        Await(E);
      END;
      A1:=A;
      B1:=B;
      MinErr1:=MinErr;
      P01:=P0;
      Lambda1:=Lambda;
      CLDeg1:=CLDeg;
      Reset1:=Reset;
      Ready1:=Ready;
      Estim1:=Estim;
      PreFilled1:=PreFilled;
      Reset:=FALSE;
      Signal(CLSpecSem);
    END;
END GetCLSpec;


(* monitor*)PROCEDURE GetCLDeg(VAR CLDeg1 : INTEGER);

BEGIN
  Wait(CLSpecMon.CLSpecSem);
  CLDeg1:=CLSpecMon.CLDeg;
  Signal(CLSpecMon.CLSpecSem);
END GetCLDeg;


(* monitor*)PROCEDURE GetEstim(VAR Estim1: BOOLEAN);

BEGIN
  Wait(CLSpecMon.CLSpecSem);
  Estim1:=CLSpecMon.Estim;
  Signal(CLSpecMon.CLSpecSem);
END GetEstim;


(* monitor*)PROCEDURE GetEstPar(VAR Ready1,PreFilled1: BOOLEAN);

BEGIN
  Wait(CLSpecMon.CLSpecSem);
  Ready1:=CLSpecMon.Ready;
  PreFilled1:=CLSpecMon.PreFilled;
  Signal(CLSpecMon.CLSpecSem);
END GetEstPar;


(*monitor*)PROCEDURE SetReady(Ready1 : BOOLEAN);

BEGIN
  WITH CLSpecMon DO
    Wait(CLSpecSem);
    Ready:=Ready1;
```

```
    Cause(E);
    Signal(CLSpecSem);
  END;
END SetReady;


(*monitor*)PROCEDURE SetFilled(PreFilled1 : BOOLEAN);

BEGIN
  WITH CLSpecMon DO
    Wait(CLSpecSem);
    PreFilled:=PreFilled1;
    Cause(E);
    Signal(CLSpecSem);
  END;
END SetFilled;


(*********************************************************************)

(*****************    datamonitor  procedure    **********************)



(*monitor*)PROCEDURE InitDataMonitor;

BEGIN
  WITH DataMon DO
    InitSem(DataSem,1);
    InitEvent(Change,DataSem);
    Get:=0;
    Put:=0;
    Delay:=0;
    PreLength:=300;
    FileConst:=5;
    Emerg:=FALSE;
  END;
END InitDataMonitor;


(*monitor*)PROCEDURE GetDataPar(VAR PreLength1 : CARDINAL;
                               VAR Emerg1 : BOOLEAN);

BEGIN
  WITH DataMon DO
    Wait(DataSem);
    PreLength1:=PreLength;
    Emerg1:=Emerg;
    Signal(DataSem);
  END;
END GetDataPar;


(*monitor*)PROCEDURE SetPreLength(Length : CARDINAL);

BEGIN
  WITH DataMon DO
    Wait(DataSem);
    PreLength:=Length;
    Cause(Change);
    Signal(DataSem);
  END;
END SetPreLength;
```

```
(*monitor*)PROCEDURE GetPreLength(VAR Length : CARDINAL);

BEGIN
  WITH DataMon DO
    Wait(DataSem);
    Length:=PreLength;
    Cause(Change);
    Signal(DataSem);
  END;
END GetPreLength;

 (*monitor*)PROCEDURE SetFileConst(Number : CARDINAL);

BEGIN
  WITH DataMon DO
    Wait(DataSem);
    FileConst:=Number;
    Cause(Change);
    Signal(DataSem);
  END;
END SetFileConst;


(*monitor*)PROCEDURE GetFileConst(VAR Number : CARDINAL);

BEGIN
  WITH DataMon DO
    Wait(DataSem);
    Number:=FileConst;
    Signal(DataSem);
  END;
END GetFileConst;


(*monitor*)PROCEDURE SetDelay(Delay1 : CARDINAL);

BEGIN
  WITH DataMon DO
    Wait(DataSem);
    Delay:=Delay1;
    Cause(Change);
    Signal(DataSem);
  END;
END SetDelay;


(*monitor*)PROCEDURE ResetEstim;

BEGIN
  WITH DataMon DO
    Wait(DataSem);
    Get:=0;
    Put:=0;
    Signal(DataSem);
  END;
END ResetEstim;


(*Monitor*)PROCEDURE PutEmerg(e :BOOLEAN);

BEGIN
  WITH DataMon DO
     Wait (DataSem);
```

```
      Emerg:= e;
      Signal(DataSem);
  END (*WITH*);
END PutEmerg;


(*monitor*)PROCEDURE GetValue(VAR U,Y:REAL;Delay:CARDINAL);

VAR       up,yp : CARDINAL;

BEGIN
  WITH DataMon DO
      Wait (DataSem);
      up:=(Get)MOD(PreLength)+1;
      yp:=(Get+Delay)MOD(PreLength)+1;
      Get:=(Get)MOD(PreLength)+1;
      U:=UU[up];
      Y:=YY[yp];
      IF yp=1000 THEN
        Get:=0;
      END;
      Signal(DataSem);
  END;
END GetValue;


(*monitor*)PROCEDURE PutMV(U,Y:REAL);

VAR       up,yp : CARDINAL;

BEGIN
  WITH DataMon DO
      Wait (DataSem);
      Put:=(Put)MOD(1000)+1;
      UU[Put]:=U;
      YY[Put]:=Y;
      Cause(Change);
      Signal(DataSem);
  END;
END PutMV;


(*monitor*)PROCEDURE GetMV(VAR U,Y:REAL);

VAR       up,yp,Value : CARDINAL;

BEGIN
  WITH DataMon DO
      Wait (DataSem);
      IF Put<Get THEN
        WHILE (Put+1000-Get)<=Delay DO
          Await(Change);
        END;
      ELSE
        Value:=ABS(Put-Get);
        WHILE Value<=Delay DO
          Await(Change);
          IF Put<Get THEN
            Value:=ABS(Put+1000-Get);
          ELSE
            Value:=ABS(Put-Get);
          END;
        END;
      END;
```

```
      up:=(Get)MOD(1000)+1;
      yp:=(Get+Delay)MOD(1000)+1;
      Get:=(Get)MOD(1000)+1;
      U:=UU[up];
      Y:=YY[yp];
      Signal(DataSem);
   END;
 END GetMV;


(*********************************************************************)

(*****************    plotmonitor  procedure    **********************)


(*monitor*)PROCEDURE InitWhatMon;

 BEGIN
    InitSem(WhatMon.WhatSem,1);
    WhatMon.What:=Oper;
    WhatMon.Change:=FALSE;
    WhatMon.HelpNr:=0;
 END InitWhatMon;


(*Monitor*) PROCEDURE PlotData():BOOLEAN;

 VAR  ret:BOOLEAN;

 BEGIN
    Wait(WhatMon.WhatSem);
    IF WhatMon.What=Data THEN
     ret:=TRUE;
    ELSE
     ret:=FALSE;
    END;
    Signal(WhatMon.WhatSem);
    RETURN ret;
 END PlotData;


(*Monitor*) PROCEDURE PlotEstim():BOOLEAN;

 VAR  ret:BOOLEAN;

 BEGIN
    Wait(WhatMon.WhatSem);
    IF WhatMon.What=Par THEN
     ret:=TRUE;
    ELSE
     ret:=FALSE;
    END;
    Signal(WhatMon.WhatSem);
    RETURN ret;
 END PlotEstim;


(*Monitor*) PROCEDURE PlotSekv():BOOLEAN;

 VAR  ret:BOOLEAN;

 BEGIN
    Wait(WhatMon.WhatSem);
    IF WhatMon.What=Test THEN
```

```
     ret:=TRUE;
    ELSE
     ret:=FALSE;
    END;
    Signal(WhatMon.WhatSem);
    RETURN ret;
 END PlotSekv;


(*Monitor*) PROCEDURE ChangeWhat(WhatKind:WhatType);

 BEGIN
    Wait(WhatMon.WhatSem);
    WhatMon.What:=WhatKind;
    WhatMon.Change:=TRUE;
    Signal(WhatMon.WhatSem);
 END ChangeWhat;


(*Monitor*) PROCEDURE PutNr( Nr : INTEGER );

 BEGIN
    Wait(WhatMon.WhatSem);
    WhatMon.HelpNr:=Nr;
    Signal(WhatMon.WhatSem);
 END PutNr;


(*Monitor*) PROCEDURE GetNr(VAR Nr : INTEGER );

 BEGIN
    Wait(WhatMon.WhatSem);
    Nr:=WhatMon.HelpNr;
    Signal(WhatMon.WhatSem);
 END GetNr;


(*Monitor*) PROCEDURE CheckWhat(VAR WhatKind:WhatType);

 BEGIN
    Wait(WhatMon.WhatSem);
    WhatKind:=WhatMon.What;
    Signal(WhatMon.WhatSem);
 END CheckWhat;


(*Monitor*) PROCEDURE CheckPlotWhat(VAR WhatKind:WhatType;VAR Change : BOOLEA

 BEGIN
    Wait(WhatMon.WhatSem);
    WhatKind:=WhatMon.What;
    Change:=WhatMon.Change;
    WhatMon.Change:=FALSE;
    Signal(WhatMon.WhatSem);
 END CheckPlotWhat;


(*****************************************************************)

(*****************    errormonitor  procedure   *********************)
```

```
PROCEDURE Error(ErrStr:ErrorType);

VAR      ErrorPtr : ErrorPtrType;

BEGIN
   NEW(ErrorPtr);
   ErrorPtr^.ErrStr:=ErrStr;
   SendMessage(ErrorBox,ErrorPtr);
END Error;


PROCEDURE Write(ErrStr:ErrorType);

VAR      ErrorPtr : ErrorPtrType;

BEGIN
   NEW(ErrorPtr);
   ErrorPtr^.ErrStr:=ErrStr;
   SendMessage(OutBox,ErrorPtr);
END Write;


(*************************************************************************)

(*****************   statusmonitor  procedure   *********************)


 (*monitor*)PROCEDURE InitStatusMonitor;

BEGIN
  WITH StatMon DO
    InitSem(StatSem,1);
    InitEvent(Change,StatSem);
    MeasureWork:=Paus;
    Stop:=TRUE;
    Test:=0;
    NewName:=FALSE;
    Counter:=10;
    h:=0.1;
    a:=1.0;
    b:=1.0;
    gamma:=1.0;
  END;
END InitStatusMonitor;


 (*monitor*)PROCEDURE PausTest;

BEGIN
  WITH StatMon DO
    Wait(StatSem);
    WHILE Test=0 DO
      Await(Change);
    END;
    Signal(StatSem);
  END;
END PausTest;


 (*monitor*)PROCEDURE SetTest(Name : TestText;TestNr : CARDINAL);

VAR          K : INTEGER;

BEGIN
```

```
  WITH StatMon DO
    Wait(StatSem);
    FOR K:=0 TO 7 DO
      TestName[K]:=Name[K];
    END;
    TestName[8]:='.';
    TestName[9]:='m';
    TestName[10]:='a';
    TestName[11]:='t';
    Test:=TestNr;
    Stop:=FALSE;
    NewName:=TRUE;
    Cause(Change);
    Signal(StatSem);
  END;
END SetTest;


(*monitor*)PROCEDURE GetTest(VAR TestNr : CARDINAL);

BEGIN
  WITH StatMon DO
    Wait(StatSem);
    IF MeasureWork=OldData THEN
      Await(Change);
    END;
    TestNr:=Test;
    Signal(StatSem);
  END;
END GetTest;


(*monitor*)PROCEDURE GetFileName(VAR Name : FileText);

BEGIN
  WITH StatMon DO
    Wait(StatSem);
    WHILE NOT NewName AND (MeasureWork=OldData) DO
      Await(Change);
    END;
    Name:=TestName;
    NewName:=FALSE;
    Signal(StatSem);
  END;
END GetFileName;


(*monitor*)PROCEDURE SetMeasWork(MWork : MeasureType);

BEGIN
  WITH StatMon DO
    Wait(StatSem);
    MeasureWork:=MWork;
    Cause(Change);
    Signal(StatSem);
  END;
END SetMeasWork;


(*monitor*)PROCEDURE GetMeasWork( VAR MWork : MeasureType);

BEGIN
  WITH StatMon DO
    Wait(StatSem);
```

```
    MWork:=MeasureWork;
    Signal(StatSem);
  END;
END GetMeasWork;


(*monitor*)PROCEDURE StopMeasure;

BEGIN
  WITH StatMon DO
    Wait(StatSem);
    Test:=0;
    Stop:=TRUE;
    MeasureWork:=Paus;
    Signal(StatSem);
  END;
END StopMeasure;


(*monitor*)PROCEDURE StopTest;

BEGIN
  Wait(StatMon.StatSem);
  StatMon.Stop:=TRUE;
  Signal(StatMon.StatSem);
END StopTest;


(*monitor*)PROCEDURE EndMeasure():BOOLEAN;

VAR       Running : BOOLEAN;
BEGIN
  Wait(StatMon.StatSem);
  Running:= StatMon.Stop;
  Signal(StatMon.StatSem);
  RETURN Running;
END EndMeasure;


(*monitor*)PROCEDURE HOldData():BOOLEAN;

VAR     Running : BOOLEAN;

BEGIN
   WITH StatMon DO
     Wait(StatMon.StatSem);
     WHILE Stop DO
       Await(Change);
     END;
     IF StatMon.MeasureWork=OldData THEN
       Running:=TRUE
     ELSE
       Running:=FALSE;
     END;
     Signal(StatMon.StatSem);
   END;
   RETURN Running;
END HOldData;


(*monitor*)PROCEDURE HSekv():BOOLEAN;

VAR     Running : BOOLEAN;
```

```
BEGIN
   WITH StatMon DO
     Wait(StatMon.StatSem);
     IF Stop THEN
       Await(Change);
     END;
     IF StatMon.MeasureWork=Sekv THEN
       Running:=TRUE
     ELSE
       Running:=FALSE;
     END;
     Signal(StatMon.StatSem);
   END;
   RETURN Running;
END HSekv;


(*monitor*)PROCEDURE HMeas():BOOLEAN;

VAR     Running : BOOLEAN;

BEGIN
   WITH StatMon DO
     Wait(StatMon.StatSem);
     IF StatMon.MeasureWork=Meas THEN
       Running:=TRUE
     ELSE
       Running:=FALSE;
     END;
     Signal(StatMon.StatSem);
   END;
   RETURN Running;
END HMeas;


(*Monitor*)PROCEDURE GetSample(VAR h1 :REAL);

BEGIN
  WITH StatMon DO
     Wait (StatSem);
     h1:=h;
     Signal(StatSem);
  END (*WITH*);
END GetSample;


(*Monitor*)PROCEDURE GetCounter(VAR c :INTEGER);

BEGIN
  WITH StatMon DO
     Wait (StatSem);
     c:=Counter;
     Signal(StatSem);
  END (*WITH*);
END GetCounter;


(*Monitor*)PROCEDURE GetMomentData(VAR a1,b1,gamma1 : REAL);

BEGIN
  WITH StatMon DO
     Wait (StatSem);
     a1:=a;
     b1:=b;
```

```
      gamma1:=gamma;
      Signal(StatSem);
  END (*WITH*);
END GetMomentData;


(*Monitor*)PROCEDURE PutMomentData(c : INTEGER; h1,a1,b1,gamma1 : REAL);

BEGIN
  WITH StatMon DO
      Wait (StatSem);
      Counter:=c;
      h:=h1;
      a:=a1;
      b:=b1;
      gamma:=gamma1;
      Signal(StatSem);
  END (*WITH*);
END PutMomentData;


(*********************************************************************)

(*****************    motormonitor  procedure   *********************)


 (*monitor*)PROCEDURE InitMotorMonitor;

 VAR         K : INTEGER;

 BEGIN
   WITH MotMon DO
     InitSem(MotSem,1);
     InitEvent(Change,MotSem);
     Enabled:=FALSE;
     Beep:=FALSE;
     Speed:=0.0;
     TStart:=30.0;
     FOR K:=1 TO 4 DO
       BeepDelay[K]:=1.0;
       PRBSPeriod[K]:=2.0;
       RandomSpeed[K]:=2.0;
       MaxSpeed[K]:=5.0;
       RampTime[K]:=5.0;
       WaitPaus[K]:=10.0;
       RandomTime[K]:=10.0;
       BreakTime[K]:=10.0;
       Period[K]:=10.0;
       tmax[K]:=50.0;
     END;
   END;
 END InitMotorMonitor;


 (*monitor*)PROCEDURE SetBeep(Beep1 : BOOLEAN);

 BEGIN
   WITH MotMon DO
     Wait(MotSem);
     Beep:=Beep1;
     Cause(Change);
     Signal(MotSem);
   END;
 END SetBeep;
```

```
(*monitor*)PROCEDURE SetTStart(TStart1 : REAL);

BEGIN
  WITH MotMon DO
    Wait(MotSem);
    TStart:=TStart1;
    Signal(MotSem);
  END;
END SetTStart;


(*monitor*)PROCEDURE GetTStart(VAR TStart1 : REAL);

BEGIN
  WITH MotMon DO
    Wait(MotSem);
    TStart1:=TStart;
    Signal(MotSem);
  END;
END GetTStart;



(*monitor*)PROCEDURE SetEnabled(Enabled1 : BOOLEAN);

BEGIN
  WITH MotMon DO
    Wait(MotSem);
    Enabled:=Enabled1;
    Cause(Change);
    Signal(MotSem);
  END;
END SetEnabled;


(*monitor*)PROCEDURE GetSpeed(VAR CurrentSpeed:REAL);

BEGIN
   Wait(MotMon.MotSem);
   CurrentSpeed:=MotMon.Speed;
   Signal(MotMon.MotSem);
END GetSpeed;


(*monitor*)PROCEDURE PutSpeed(CurrentSpeed,t:REAL;Test : INTEGER;
                              VAR TestEnd : BOOLEAN);

BEGIN
   TestEnd:=FALSE;
   Wait(MotMon.MotSem);
   MotMon.Speed:=CurrentSpeed;
   IF t>MotMon.tmax[Test] THEN
      TestEnd:=TRUE;
   END;
   Signal(MotMon.MotSem);
END PutSpeed;


(*monitor*)PROCEDURE GetMotorPar(VAR Enabled1,Beep1 : BOOLEAN;
                                 VAR Speed1:REAL);

BEGIN
```

```
    WITH MotMon DO
      Wait(MotSem);
      Enabled1:=Enabled;
      Beep1:=Beep;
      Speed1:=Speed;
      Signal(MotSem);
    END;
 END GetMotorPar;


 (*monitor*)PROCEDURE PutMotorData(MData : MotorType; Nr : INTEGER);

 BEGIN
    WITH MotMon DO
      Wait(MotSem);
      RampTime[Nr]:=MData.RampTime;
      WaitPaus[Nr]:=MData.WaitPaus;
      RandomTime[Nr]:=MData.RandomTime;
      BreakTime[Nr]:=MData.BreakTime;
      BeepDelay[Nr]:=MData.BeepDelay;
      Period[Nr]:=MData.Period;
      PRBSPeriod[Nr]:=MData.PRBSPeriod;
      MaxSpeed[Nr]:=MData.MaxSpeed;
      RandomSpeed[Nr]:=MData.RandomSpeed;
      tmax[Nr]:=MData.tmax;
      Signal(MotSem);
    END;
 END PutMotorData;


 (*monitor*)PROCEDURE GetMotorData(VAR MData : MotorType; Nr : INTEGER);

 BEGIN
    WITH MotMon DO
      Wait(MotSem);
      MData.RampTime:=RampTime[Nr];
      MData.WaitPaus:=WaitPaus[Nr];
      MData.RandomTime:=RandomTime[Nr];
      MData.BreakTime:=BreakTime[Nr];
      MData.MaxSpeed:=MaxSpeed[Nr];
      MData.BeepDelay:=BeepDelay[Nr];
      MData.Period:=Period[Nr];
      MData.PRBSPeriod:=PRBSPeriod[Nr];
      MData.RandomSpeed:=RandomSpeed[Nr];
      MData.tmax:=tmax[Nr];
      Signal(MotSem);
    END;
 END GetMotorData;

END Decl.
```

```
DEFINITION MODULE Safe;


EXPORT QUALIFIED SafeProc;


PROCEDURE SafeProc;
(* Creates the emergencywindow and stops the regulator when the
   emergencybutton is pressed.*)


END Safe.
```

```
IMPLEMENTATION MODULE Safe;

 IMPORT RTMouse;

 FROM Kernel   IMPORT CreateProcess, SetPriority,Signal;

 FROM Graphics IMPORT handle,rectangle,color,point,SetWindow,
                      SetViewPort,VirtualScreen,SetFillColor,
                      FillRectangle,DrawRectangle,WaitMouseRectangle,
                      HideCursor,SetMouseRectangle,SetLineColor,
                      SetTextColor,WriteString,ShowCursor;

 FROM Decl     IMPORT PutEmerg,TheEnd;

 CONST     EmergencyColor     = red;
           EmergencyTextColor = yellow;

 PROCEDURE SetRectangle(VAR R :rectangle; Xlo,Ylo,Xhi,Yhi :REAL);

 BEGIN
    R.xlo:= Xlo;
    R.ylo:= Ylo;
    R.xhi:= Xhi;
    R.yhi:= Yhi;
 END SetRectangle;


 PROCEDURE InitEmergencyWindow(VAR H : handle);

 VAR ViewPortRectangle,WindowRectangle : rectangle;
     TextPoint                         : point;

 BEGIN                    (* initiate emergensy window *)
    VirtualScreen(H);
    SetRectangle(ViewPortRectangle, 0.05, 0.05, 0.45, 0.15);
    SetViewPort(H,ViewPortRectangle);
    SetRectangle(WindowRectangle, 0.0, 0.0, 1.0, 1.0);
    SetWindow(H,WindowRectangle);
    SetTextColor(H, EmergencyTextColor);
    SetLineColor(H, EmergencyTextColor);
    SetFillColor(H, EmergencyColor);
    HideCursor;
    FillRectangle(H,WindowRectangle);
    TextPoint.h:= 0.3;
    TextPoint.v:= 0.3;
    DrawRectangle(H,WindowRectangle);
    WriteString(H, TextPoint, "Emergency");
    SetMouseRectangle(H,WindowRectangle,1);
    ShowCursor;
 END InitEmergencyWindow;

(*Process*)PROCEDURE SafeProc;

 VAR       SelectedBox : CARDINAL;
           H           : handle;

 BEGIN
    SetPriority(89);
    InitEmergencyWindow(H);
    LOOP
      SelectedBox:=WaitMouseRectangle(H);
      PutEmerg(TRUE);
    END;
 END SafeProc;
```

```
END Safe.
```

```
DEFINITION MODULE Measure;

EXPORT QUALIFIED DataMeasure;

PROCEDURE DataMeasure;

END Measure.
```

```
IMPLEMENTATION MODULE Measure;

 FROM AnalogIO    IMPORT DAOut,ADIn;

 FROM Kernel      IMPORT WaitUntil,GetTime,IncTime,SetPriority,Time;

 FROM Decl        IMPORT PMailType,MBox,PBox,WhatType,GetCounter,PutMV,
                         PlotData,PlotSekv,PlotKinds,Error,ErrorType,
                         SpeedType,MeasType,MeasureType,FileText,DataType,
                         GetSpeed,EndMeasure,GetDataPar,GetMeasWork,
                         SetFilled,GetMotorPar,GetFileName,HOldData,
                         HMeas,HSekv,GetMomentData,GetSample,GetEstim,
                         StopMeasure,StopTest,GetEstPar,PutSpeed,
                         GetFileConst,PausTest;

 FROM Messages    IMPORT AcceptMessage,SendMessage;

 FROM Strings     IMPORT Assign;

 FROM ConvReal    IMPORT RealToString,StringToReal;

 FROM MathLib     IMPORT round,float;

 FROM FileSystem IMPORT File,Lookup,Delete,WriteChar,ReadChar,Close,
                         Again,done,notdone;


 CONST              EOL       = 36C;
                    SPACE     = ' ';
                    BeepSound = 0.8;

 TYPE               DataVect = ARRAY[1..50] OF DataType;

 VAR                PlotCount              : INTEGER;
                    PreLength,ArrayNumber,
                    FileConst,Nr           : CARDINAL;
                    t                      : Time;
                    Allowed,Ready,
                    Emerg,PreFilled,Estim  : BOOLEAN;
                    Out                    : ErrorType;
                    Work                   : MeasureType;
                    a,b,gamma,Index        : REAL;
                    dataFile               : File;
                    Name                   : FileText;


(*****************************************************************)

(*****************        file procedure        *********************)


 PROCEDURE CharOk(Ch : CHAR) : BOOLEAN;

 VAR       Number : CARDINAL;

 BEGIN
    Number:= ORD(Ch);
    RETURN (Number >= 33) AND (Number <= 126);
 END CharOk;


 PROCEDURE SkipAsciiBlanks(VAR f : File;VAR FirstNonBlank : CHAR);

 VAR         Ch : CHAR;
```

```
            ok : BOOLEAN;

BEGIN
   REPEAT
     ReadChar(f,Ch);              (* remove blanks in line *)
     ok:= f.res = done;
   UNTIL ((Ch <> ' ') AND CharOk(Ch)) OR f.eof;
   FirstNonBlank:= Ch;
END SkipAsciiBlanks;


PROCEDURE ReadString(VAR f:File; VAR s : ARRAY OF CHAR);

CONST       maxi = 40;

VAR         Ch      : CHAR;
            Temps   : ARRAY [0..maxi] OF CHAR;
            i       : CARDINAL;
            Stop,Ok : BOOLEAN;

BEGIN
   SkipAsciiBlanks(f,Temps[0]);    (* reads string from file *)
   i:= 1;
   Stop:= (NOT CharOk(Temps[0])) OR f.eof;
   Ok:= f.res = done;
   WHILE Ok AND (NOT Stop) DO
     ReadChar(f,Ch);
     Ok:= (f.res = done) AND (i <= maxi);
     Stop:= (NOT CharOk(Ch)) OR (Ch = ' ') OR f.eof;
     IF NOT Stop THEN
       Temps[i]:= Ch;
       INC(i);
     END;
   END;
   Again(f);
   IF (NOT Ok) AND (NOT f.eof) THEN
     Error('error reading');
   END;
   Temps[i]:= 0C;
   Assign(Temps,s);
END ReadString;


PROCEDURE ReadReal(VAR f:File; VAR Value : REAL;VAR Last : BOOLEAN);

VAR        Text : ARRAY [0..20] OF CHAR;
           Pos  : CARDINAL;

BEGIN
   ReadString(f,Text);     (* reads real from file *)
   IF f.eof THEN
     Last:=TRUE;
   ELSE
     Last:=FALSE;
   END;
   Pos := 0;
   StringToReal(Text,Pos,Value);
   IF (Pos = 0) AND (NOT f.eof) THEN
     Error('error reading');
   END;
END ReadReal;


PROCEDURE CloseWrite(VAR f : File);
```

```
BEGIN
   WriteString(f,0C);
   Close(f);
END CloseWrite;


PROCEDURE CloseRead(VAR f : File);

BEGIN
   Close(f);
END CloseRead;

PROCEDURE WriteString(VAR f : File; s : ARRAY OF CHAR);

VAR
   i, maxi : CARDINAL;
        ok : BOOLEAN;
BEGIN
   i := 0;
   maxi := HIGH(s);       (* writes real on string *)
   ok := TRUE;
   WHILE (i <= maxi) AND (s[i] <> 0C) AND ok DO
     WriteChar(f,s[i]);
     INC(i);
     ok := f.res = done;
   END;
   IF NOT ok THEN
     Error('error writing');
   END;
END WriteString;


PROCEDURE WriteReal(VAR f : File; Value : REAL);

VAR Text : ARRAY [0..20] OF CHAR;

BEGIN
   RealToString(Value,Text,8);  (* writes real on file *)
   WriteString(f,Text);
END WriteReal;


(*****************************************************************)

(*****************      in/out procedure        ******************)


PROCEDURE GetDataMeasure(VAR u : SpeedType;VAR y : MeasType);

BEGIN
   y[1]:= 10.0*ADIn(15);              (* gets data from I/O *)
   y[2]:= 10.0*ADIn(16);
   y[3]:= 10.0*ADIn(17);
   y[4]:= 10.0*ADIn(12);
   y[5]:= 10.0*ADIn(13);
   y[6]:= 10.0*ADIn(14);
   GetSpeed(u[1]);
   u[2]:= -10.0*ADIn(18);
END GetDataMeasure;


PROCEDURE MotorControl;
```

```
VAR             Enabled,Beep : BOOLEAN;
                Speed        : REAL;

BEGIN
   GetMotorPar(Enabled,Beep,Speed);
   IF Enabled THEN
     IF Speed>10.0 THEN
       Speed:=10.0
     END;
     IF Speed<-10.0 THEN
       Speed:=-10.0
     END;                              (* puts control signals *)
     DAOut(0,Speed*0.1);               (* to I/O               *)
   ELSE
     DAOut(0,0.0);
   END;
   IF Beep THEN
     DAOut(1,BeepSound);
   ELSE
     DAOut(1,0.0);
   END;
END MotorControl;


PROCEDURE EmergencyLoop;

VAR             h1,Speed1,Speed      : REAL;
                h,K                  : INTEGER;
                Enabled,Beep,TestEnd : BOOLEAN;
                y                    : MeasType;
                u                    : SpeedType;

BEGIN
   DAOut(1,0.0);
   GetMotorPar(Enabled,Beep,Speed); (* controls output after *)
   FOR K:=1 TO 10 DO                (* emergency signal      *)
     WaitUntil(t);
     IncTime(t,100);
     Speed1:=Speed*(1.0-(float(K)/10.0));
     DAOut(0,Speed1*0.1);
     PutSpeed(Speed1,Speed1,1,TestEnd);
     GetDataMeasure(u,y);
     PlotRoutine(u,y);
   END;
   StopMeasure;
   GetSample(h1);
   h:=round(h1*1000.0);
   GetTime(t);
   WHILE Emerg DO      (* emergency rest *)
     GetDataPar(PreLength,Emerg);
     WaitUntil(t);
     IncTime(t,h);
     GetSample(h1);
     h:=round(h1*1000.0);
   END;
END EmergencyLoop;


(*****************************************************************)

(*****************    calculation procedure     *********************)


PROCEDURE PreCalculate(VAR y : REAL; y1 : MeasType);
```

```
VAR       Tbal : REAL;

BEGIN
   Tbal:=gamma*(b*(y1[1]+y1[2])-a*y1[3]);
   y:=Tbal;           (* calculate measured powers *)
END PreCalculate;     (* to moment                  *)


PROCEDURE EstimMeasure(u2 : SpeedType; y2 : MeasType);

VAR        y1          : INTEGER;
           u,y         : REAL;

BEGIN
   GetEstPar(Ready,PreFilled);
   IF NOT Ready THEN
     IF NOT PreFilled THEN    (* measures data for estimation *)
       PreCalculate(y,y2);
       PutMV(u2[2],y);
       IF ArrayNumber>=PreLength THEN
         ArrayNumber:=1;
         SetFilled(TRUE);
       ELSE
         ArrayNumber:=ArrayNumber+1;
       END;
     END;
   ELSE
     PreCalculate(y,y2);
     PutMV(u2[2],y);
   END;
END EstimMeasure;

                                  E
(*********************************************************************)

(*****************   plotmessage  procedure     *********************)


PROCEDURE PlotRoutine(u1 : SpeedType; y : MeasType);


VAR           what   : WhatType;
              PMail  : PMailType;
              c      : INTEGER;

BEGIN
   GetCounter(c);
   IF c<1 THEN
       c:=1;
   END;
   PlotCount:=(PlotCount+1) MOD c;
   IF (PlotData() OR PlotSekv()) AND (PlotCount=0) THEN
     AcceptMessage(MBox,PMail);
     IF PMail<> NIL THEN
       WITH PMail^ DO
         IF PlotSekv() THEN      (* sends data to plotprocess *)
           Kind:= Re;
           uref:=u1[1];
         ELSIF PlotData() THEN
           Kind:= Re;
           PForce[1]:=gamma*y[1];
           PForce[2]:=gamma*y[2];
           PForce[3]:=gamma*y[3];
```

```
            TForce[1]:=gamma*(b*(y[1]+y[2])-a*y[3]);
            TForce[2]:=gamma*y[4];
            TForce[3]:=gamma*(y[6]+y[5]);
            uref:=u1[1];
            u:=u1[2];
          END;
        END;
        SendMessage(PBox,PMail);
        Allowed:= TRUE;
      ELSIF Allowed THEN
        Error('Plot data ovfl (Meas)');
        Allowed:= FALSE;
      END;
    END;
  END PlotRoutine;


(*********************************************************************)

(*****************   filehandle procedure       *********************)


  PROCEDURE GetData(VAR Data : DataType;VAR Ok :BOOLEAN);

  VAR           i,k       : CARDINAL;
                FileEnd   : BOOLEAN;
                WaitFile  : DataVect;

  BEGIN
     Ok:=TRUE;
     IF Nr=1 THEN
       k:=1;
       FileEnd:=FALSE;
       WHILE (k<=FileConst) AND NOT FileEnd DO
         ReadReal(dataFile,Index,FileEnd);
         IF NOT FileEnd THEN
           FOR i:=1 TO 6 DO
             ReadReal(dataFile,WaitFile[k].y[i],FileEnd);
           END;
           FOR i:=1 TO 2 DO
             ReadReal(dataFile,WaitFile[k].u[i],FileEnd);
           END;
           k:=k+1;          (* fetchs data from external *)
         END;               (* memory                    *)
       END;
       IF FileEnd THEN
         Ok:=FALSE;
       ELSE
         Data:=WaitFile[1];
         Nr:=2;
       END;
     ELSE
       Data:=WaitFile[Nr];
       Nr:=(Nr) MOD FileConst+1;;
     END;
  END GetData;


  PROCEDURE StoreData(Data : DataType);

  VAR          i,K      : CARDINAL;
               WaitFile : DataVect;

  BEGIN
```

```
   WaitFile[Nr]:=Data;          (* stores measured data on *)
   IF (Nr=FileConst) THEN       (* external memory         *)
     FOR K:=1 TO FileConst DO
         WriteReal(dataFile,Index);
         WriteChar(dataFile, SPACE);
       FOR i := 1 TO 6 DO
         WriteReal(dataFile,WaitFile[K].y[i]);
         WriteChar(dataFile, SPACE);
       END;
       WriteReal(dataFile,WaitFile[K].u[1]);
       WriteChar(dataFile, SPACE);
       WriteReal(dataFile,WaitFile[K].u[2]);
       WriteChar(dataFile, EOL);
       Index:=Index+1.0;
     END;
   END;
   Nr:=(Nr)MOD FileConst+1;
END StoreData;


(*********************************************************************)

(*****************    datahandle  procedure     *********************)


PROCEDURE HandleOldData;

VAR                Ok   : BOOLEAN;
                   data : DataType;
                   u1   : SpeedType;
                   y1   : MeasType;
                   h1   : REAL;
                   h    : CARDINAL;

BEGIN
   WHILE HOldData() DO              (* procedure for fetching old *)
     GetFileConst(FileConst);       (* data stored on external    *)
     GetMomentData(a,b,gamma);      (* memory *)
     GetFileName(Name);
     IF (Name[1]<>' ') THEN
       ArrayNumber:=1;
       Nr:=1;
       Ok:=TRUE;
       Lookup(dataFile,Name,FALSE);
       IF dataFile.res=notdone THEN
         Error('no such file');
         Ok:=FALSE;
       END;
       GetTime(t);
       WHILE NOT EndMeasure() AND Ok DO
         GetData(data,Ok);
         IF Ok THEN
           GetDataPar(PreLength,Emerg);
           GetEstim(Estim);
           GetSample(h1);
           h:=round(h1*1000.0);
           WaitUntil(t);
           IncTime(t,h);
           u1:=data.u;
           y1:=data.y;
           IF Estim THEN
             EstimMeasure(u1,y1);
           END;
           PlotRoutine(u1,y1);
```

```
          ELSE
            StopTest;
          END;
        END;
        CloseRead(dataFile);
      END;
    END;
END HandleOldData;


PROCEDURE HandleSekv;

VAR                 y  : MeasType;
                    u  : SpeedType;
                    h1 : REAL;
                    h  : CARDINAL;

BEGIN
    GetTime(t);
    WHILE HSekv() DO
      GetSample(h1);
      h:= round(h1*1000.0);
      WaitUntil(t);                 (* sends control signal for *)
      IncTime(t,h);                 (* "visual speed" to plot   *)
      GetDataMeasure(u,y);
      PlotRoutine(u,y);
    END;
END HandleSekv;


PROCEDURE NewMeasure;

VAR               h    : INTEGER;
                  Data : DataType;
                  y    : MeasType;
                  u    : SpeedType;
                  h1   : REAL;
                  Save : BOOLEAN;

BEGIN
    Save:=TRUE;
    GetFileName(Name);              (* makes a new measure and     *)
    Delete(Name,dataFile);          (* store it on external memory *)
    IF (Name[1]=' ') AND (Name[2]=' ') THEN
      Save:=FALSE;
    ELSE
      Lookup(dataFile,Name,TRUE);
    END;
    GetFileConst(FileConst);
    Index:=1.0;
    Nr:=1;
    GetMomentData(a,b,gamma);
    GetTime(t);
    ArrayNumber:=1;
    WHILE HMeas() DO
      GetDataPar(PreLength,Emerg);
      IF Emerg THEN
         EmergencyLoop;
      END;
      GetSample(h1);
      h:=round(h1*1000.0);
      WaitUntil(t);
      IncTime(t,h);
      GetDataMeasure(u,y);
```

```
      MotorControl;
      GetEstim(Estim);
      IF Estim THEN
        EstimMeasure(u,y);
      ELSIF Save THEN
        Data.u:=u;
        Data.y:=y;
        StoreData(Data);
      END;
      PlotRoutine(u,y);
    END;
    IF Save THEN
      CloseWrite(dataFile);
    END;
 END NewMeasure;


 PROCEDURE HandlePaus;

 BEGIN
    DAOut(0,0.0);
    DAOut(1,0.0);
    PausTest;
 END HandlePaus;


(*PROCESS*)PROCEDURE DataMeasure;

 BEGIN
   SetPriority(100);
   DAOut(0,0.0);
   DAOut(1,0.0);
   PlotCount:=0;          (* selects wanted operation *)
   Allowed:=TRUE;         (* to measured data         *)
   LOOP
      GetMeasWork(Work);
      CASE Work OF
        OldData : HandleOldData|
        Sekv    : HandleSekv|
        Meas    : NewMeasure|
        Paus    : HandlePaus;
      END;
   END;
 END DataMeasure;

END Measure.
```

```
DEFINITION MODULE Control;

EXPORT QUALIFIED ControlProc,InitControl;

PROCEDURE ControlProc;

PROCEDURE InitControl;

END Control.
```

```
IMPLEMENTATION MODULE Control;

IMPORT RTMouse;

FROM Kernel    IMPORT CreateProcess,WaitUntil,WaitTime,IncTime,GetTime,
                      Time,SetPriority,Wait,Signal,Semaphore,InitSem;

FROM Graphics  IMPORT handle,rectangle,color,point,SetWindow,SetViewPort,
                      VirtualScreen,SetFillColor,FillRectangle,
                      DrawRectangle,WaitMouseRectangle,HideCursor,
                      SetMouseRectangle,SetLineColor,SetTextColor,
                      WriteString,ReadString,ShowCursor,buttonset,
                      GetMouseRectangle,WaitForMouse;

FROM Storage   IMPORT ALLOCATE,DEALLOCATE;

FROM MathLib   IMPORT float,round;

FROM Messages  IMPORT ReceiveMessage,SendMessage;

FROM NumMenu   IMPORT MakeNumMenu,SetNumMenuEntry,ShowNumMenu,InitNumMenu,
                      GetNumMenuState,HideNumMenu,SetNumMenuColors,
                      GetNumMenuStateWait,NumMenuType;

FROM LogMenu   IMPORT MakeLogMenu,SetLogMenuEntry,ShowLogMenu,InitLogMenu,
                      GetLogMenuState,HideLogMenu,SetLogMenuColors,
                      GetLogMenuStateWait,LogMenuType;

FROM Decl      IMPORT TheEnd,ErrorBox,ErrorPtrType,ErrorType,Error,
                      OutBox,PutCLSpec,ABType,SetDelay,MotorType,
                      PutMotorData,ChangeWhat,WhatType,SetTest,SetEnabled,
                      SetMeasWork,MeasureType,PutEmerg,SetPreLength,
                      TestText,StopTest,PutMomentData,SetFileConst,SetReady,
                      SetFilled,CheckPlotWhat,SetTStart,PutNr;

FROM ConvReal IMPORT RealToString;

CONST HMColor = blue;          HMTextColor = intensewhite;
      TMColor = green;         TMTextColor = black;
      TMAlertColor = red;
      PMColor = cyan;          PMTextColor = black;
      EMColor = magenta;       EMTextColor = black;
      EMTrue = lightgreen;     EMAlertColor = blue;
      QMColor = lightred;      QMTextColor = blue;
      SPMColor = white;        SPMTextColor = black;
      SPMAlertColor = red;
      HideColor = black;
      ErrorColor = lightred; ErrorTextColor = yellow;


TYPE  TextStr         = ARRAY[0..16] OF CHAR;
      EStrType        = ARRAY [1..7] OF TextStr;
      RealVectorType  = ARRAY[0..10] OF REAL;
      WindowType      = (Ord,Num,Log);

      CommandWindowType = POINTER TO RECORD
                             H : handle;
                             CASE Kind : WindowType OF
                                Ord  : WindowRectangle :rectangle|
                                Num  : NM : NumMenuType|
                                Log  : LM : LogMenuType|
                             END;
                          END;
```

```
      EstParType        =  RECORD
                             EA,EB                 : ABType;
                             EMinErr,EP0,ELambda   : REAL;
                             ECLDeg                : INTEGER;
                             EEstim,EReset         : BOOLEAN;
                             EDelay,EPreLength     : INTEGER;
                             EPreEstim             : BOOLEAN;
                           END;

      SysParType        =  RECORD
                             SampleTime            : REAL;
                             PlotConst             : INTEGER;
                             A,B,Gamma,TStart      : REAL;
                             FileConst             : INTEGER;
                           END;


VAR           OnOffStr : TextStr;
              EstPar   : EstParType;
              MotPar   : ARRAY[1..5] OF MotorType;
              SysPar   : SysParType;
              Tabbe    : BOOLEAN;


PROCEDURE InitParameters;
BEGIN
  WITH EstPar DO
    EA[1]:=1.0;
    EA[2]:=1.0;
    EA[3]:=1.0;
    EA[4]:=1.0;
    EB[1]:=-0.05;
    EB[2]:=-0.05;
    EB[3]:=-0.05;
    EB[4]:=-0.05;
    EMinErr:=0.05;
    EP0:=1000.0;
    ELambda:=0.95;
    ECLDeg:=3;
    EEstim:=FALSE;
    EReset:=TRUE;
    EDelay:=10;
    EPreLength:=200;
    EPreEstim:=TRUE;
  END;
  WITH EstPar DO
    PutCLSpec(EA,EB,EMinErr,EP0,ELambda,ECLDeg,EEstim,EReset);
    SetPreLength(EPreLength);
  END;
  WITH MotPar[1] DO
    RampTime:=5.0;
    WaitPaus:=30.0;
    BreakTime:=5.0;
    RandomTime:=10.0;
    BeepDelay:=1.0;
    MaxSpeed:=7.0;
    tmax:=100.0;
  END;
  PutMotorData(MotPar[1],1);
  WITH MotPar[2] DO
    RampTime:=5.0;
    WaitPaus:=10.0;
    BreakTime:=5.0;
```

```
    MaxSpeed:=3.0;
    RandomTime:=10.0;
    tmax:=240.0;
    RandomSpeed:=4.0;
  END;
  PutMotorData(MotPar[2],2);
  WITH MotPar[3] DO
    RampTime:=5.0;
    WaitPaus:=10.0;
    BreakTime:=5.0;
    MaxSpeed:=0.0;
    RandomSpeed:=6.0;
    Period:=30.0;
    tmax:=240.0;
  END;
  PutMotorData(MotPar[3],3);
  WITH MotPar[4] DO
    RampTime:=5.0;
    WaitPaus:=10.0;
    BreakTime:=5.0;
    MaxSpeed:=0.0;
    RandomSpeed:=2.0;
    Period:=30.0;
    PRBSPeriod:=3.0;
    tmax:=240.0;
  END;
  PutMotorData(MotPar[4],4);
  WITH MotPar[5] DO
    RampTime:=5.0;
    WaitPaus:=10.0;
    BreakTime:=5.0;
    MaxSpeed:=0.0;
    RandomSpeed:=2.0;
    PRBSPeriod:=1.5;
    tmax:=240.0;
  END;
  PutMotorData(MotPar[5],5);
  WITH SysPar DO
    SampleTime:=0.1;
    PlotConst:=5;
    A:=0.170;
    B:=0.170;
    Gamma:=200.0;
    FileConst:=5;
    TStart:=30.0;
    PutMomentData(PlotConst,SampleTime,A,B,Gamma);
    SetFileConst(FileConst);
    SetTStart(TStart);
  END;
END InitParameters;


PROCEDURE SetRectangle(VAR R : rectangle; Xlo,Ylo,Xhi,Yhi : REAL);

BEGIN
   R.xlo:=Xlo;
   R.ylo:=Ylo;
   R.xhi:=Xhi;
   R.yhi:=Yhi;
END SetRectangle;


PROCEDURE DefineMouseWindow(H : handle;y : INTEGER);
```

```
VAR     R : rectangle;

BEGIN
   SetRectangle(R, 0.0, float(y-1), 1.0, float(y));
   SetMouseRectangle(H,R,y);
END DefineMouseWindow;


PROCEDURE ShowText(H :handle; y :REAL; str :TextStr);

VAR     Textpoint : point;
        R         : rectangle;

BEGIN
   Textpoint.h:=0.05;
   Textpoint.v:=y+0.15;
   SetRectangle(R,0.0,y,1.0,y+1.0);
   HideCursor;
   DrawRectangle(H,R);
   WriteString(H,Textpoint,str);
   ShowCursor;
END ShowText;


PROCEDURE HideMenu(VAR Window : CommandWindowType);

BEGIN
   WITH Window^ DO
      SetFillColor(H, HideColor);
      HideCursor;
      FillRectangle(H,WindowRectangle);
      ShowCursor;
   END;
END HideMenu;


PROCEDURE NiceStartUpMessage;

VAR         Handtag     : handle;
            Fonster     : rectangle;
            Punkt       : point;
            Klar        : ARRAY[0..1]OF CHAR;
CONST       Fonsterfarg = lightmagenta;
            Textfarg    = lightcyan;

BEGIN
  VirtualScreen(Handtag);
  SetRectangle(Fonster,0.05,0.05,1.45,0.95);
  SetFillColor(Handtag,Fonsterfarg);
  SetLineColor(Handtag,Textfarg);
  SetTextColor(Handtag,Textfarg);
  Punkt.h:=0.31;
  Punkt.v:=0.85;
  FillRectangle(Handtag,Fonster);
  DrawRectangle(Handtag,Fonster);
  WriteString(Handtag,Punkt,'                    VISCON v1.0');
  Punkt.v:=0.75;
  WriteString(Handtag,Punkt,'Control program for a Visual stimulation equipme
  Punkt.v:=0.55;
  WriteString(Handtag,Punkt,'                    written by  ');
  Punkt.v:=0.50;
  WriteString(Handtag,Punkt,'    PER-ANDERS FRANSSON  &  PATRIK SUNDSTROM ');
  Punkt.v:=0.45;
  WriteString(Handtag,Punkt,'                        for ');
```

```
  Punkt.h:=0.09;
  Punkt.v:=0.40;
  WriteString(Handtag,Punkt,'  the Department of Automatic Control, Lund Inst
  Punkt.h:=0.31;
  Punkt.v:=0.35;
  WriteString(Handtag,Punkt,'                          and          ');
  Punkt.h:=0.09;
  Punkt.v:=0.30;
  WriteString(Handtag,Punkt,'   the Department of OtoRhinoLaryngology, Lund U
  WaitTime(5000);
  SetFillColor(Handtag,black);
  SetLineColor(Handtag,black);
  DrawRectangle(Handtag,Fonster);
  FillRectangle(Handtag,Fonster);
END NiceStartUpMessage;


(*************************************************************************)

(******************** Procedure for errorwindow ***************************)


PROCEDURE InitErrorWindow(VAR errwindow : CommandWindowType);

VAR      errbox : rectangle;

BEGIN
   NEW(errwindow);
   WITH errwindow^ DO
      VirtualScreen(H);
      Kind:= Ord;
      SetRectangle(errbox,0.05,0.27,0.45,0.38);
      SetViewPort(H,errbox);
      SetRectangle(WindowRectangle,0.0,0.0,4.0,1.0);
      SetWindow(H,WindowRectangle);
      SetTextColor(H,ErrorTextColor);
      SetLineColor(H,ErrorTextColor);
      SetFillColor(H,ErrorColor);
   END;
END InitErrorWindow;


(*************************************************************************)

(******************** Procedures for HeadMenu ****************************


PROCEDURE InitHeadMenu(VAR HMwindow : CommandWindowType);

VAR       ViewPortRectangle : rectangle;
          i                 : INTEGER;

BEGIN
   NEW(HMwindow);
   WITH HMwindow^ DO
      VirtualScreen(H);
      Kind:= Ord;
      SetRectangle(ViewPortRectangle, 0.05, 0.40, 0.45, 1.0);
      SetViewPort(H,ViewPortRectangle);
      SetRectangle(WindowRectangle, 0.0, 0.0, 1.0, 6.0);
      SetWindow(H,WindowRectangle);
      SetTextColor(H, HMTextColor);
      SetLineColor(H, HMTextColor);
      SetFillColor(H, HMColor);
```

```
      FOR i:= 1 TO 6 DO
         DefineMouseWindow(H,i);
      END;
   END;
END InitHeadMenu;


PROCEDURE ShowHeadMenu(VAR HMwindow : CommandWindowType);
(*Shows headmenu after return from submenus*)

VAR       String : ARRAY [0..5] OF TextStr;
          i      : INTEGER;

BEGIN
   WITH HMwindow^ DO
      PutNr(1);
      SetFillColor(H,HMColor);
      HideCursor;
      FillRectangle(H,WindowRectangle);
      ShowCursor;
      String[0]:= "Quit program";
      String[1]:= "Stop test";
      String[2]:= "Plot";
      String[3]:= "Estimator";
      String[4]:= "Test";
      String[5]:= "System parameters";
      FOR i:= 0 TO 5 DO
         ShowText(H,float(i),String[i]);
      END;
   END;
END ShowHeadMenu;


(*************************************************************************

(********************** Procedures for TestMenu ****************************


PROCEDURE InitTestMenu(VAR TMwindow : CommandWindowType);

VAR     ViewPortRectangle : rectangle;
        i                 : INTEGER;

BEGIN
  NEW(TMwindow);
  WITH TMwindow^ DO
    VirtualScreen(H);
    Kind:=Ord;
    SetRectangle(ViewPortRectangle,0.05,0.40,0.45,0.80);
    SetViewPort(H,ViewPortRectangle);
    SetRectangle(WindowRectangle,0.0,0.0,1.0,8.0);
    SetWindow(H,WindowRectangle);
    SetTextColor(H,TMTextColor);
    SetLineColor(H,TMTextColor);
    SetFillColor(H,TMColor);
    FOR i:= 1 TO 8 DO
      DefineMouseWindow(H,i);
    END;
  END;
END InitTestMenu;


PROCEDURE InitTestParametersMenu(VAR TMPwindow : CommandWindowType);
```

```
VAR     ViewPortRectangle : rectangle;
        i                 : INTEGER;

BEGIN
  NEW(TMPwindow);
  WITH TMPwindow^ DO
    VirtualScreen(H);
    Kind:=Ord;
    SetRectangle(ViewPortRectangle,0.05,0.40,0.45,0.80);
    SetViewPort(H,ViewPortRectangle);
    SetRectangle(WindowRectangle,0.0,0.0,1.0,6.0);
    SetWindow(H,WindowRectangle);
    SetTextColor(H,TMTextColor);
    SetLineColor(H,TMTextColor);
    SetFillColor(H,TMColor);
    FOR i:= 1 TO 6 DO
      DefineMouseWindow(H,i);
    END;
  END;
END InitTestParametersMenu;


PROCEDURE ShowTestMenu(VAR TMwindow : CommandWindowType);

VAR      String : ARRAY[0..7] OF TextStr;
         i      : INTEGER;

BEGIN
  PutNr(2);
  WITH TMwindow^ DO
    SetFillColor(H,TMColor);
    HideCursor;
    FillRectangle(H,WindowRectangle);
    ShowCursor;
    String[7]:=' BeepTest';
    String[6]:=' RampRandom';
    String[5]:=' Sinus';
    String[4]:=' PRBSSinus';
    String[3]:=' PRBS';
    String[2]:=' Parameters';
    String[1]:=' Run from file';
    String[0]:=' Done';
    FOR i:=0 TO 7 DO
      ShowText(H,float(i),String[i]);
    END;
  END;
END ShowTestMenu;


PROCEDURE ShowTestParametersMenu(VAR TMPwindow : CommandWindowType);

VAR     String : ARRAY[0..5] OF TextStr;
        i      : INTEGER;

BEGIN
  PutNr(3);
  WITH TMPwindow^ DO
    SetFillColor(H,TMColor);
    HideCursor;
    FillRectangle(H,WindowRectangle);
    ShowCursor;
    String[5]:=' BeepTest';
    String[4]:=' RampRandom';
    String[3]:=' Sinus';
```

```
    String[2]:=' PRBSSinus';
    String[1]:=' PRBS';
    String[0]:=' DONE';
    FOR i:=0 TO 5 DO
      ShowText(H,float(i),String[i]);
    END;
  END;
END ShowTestParametersMenu;


PROCEDURE InitQuestionWindow(VAR QMwindow : CommandWindowType);

VAR       ViewPortRectangle : rectangle;

BEGIN
  NEW(QMwindow);
  WITH QMwindow^ DO
    VirtualScreen(H);
    Kind:=Ord;
    SetRectangle(ViewPortRectangle,0.05,0.40,0.45,0.60);
    SetViewPort(H,ViewPortRectangle);
    SetRectangle(WindowRectangle,0.0,0.0,1.0,2.0);
    SetWindow(H,WindowRectangle);
    SetTextColor(H,QMTextColor);
    SetLineColor(H,QMTextColor);
  END;
END InitQuestionWindow;


PROCEDURE ShowQuestionWindow(VAR HMwindow,QMwindow : CommandWindowType;
                                 Str : TextStr;VAR Name : TestText);

VAR       Pos       : point;
          Number,k  : CARDINAL;
          Ch        : CHAR;

BEGIN
  HideMenu(HMwindow);
  PutNr(7);
  HideCursor;
  WITH QMwindow^ DO
    SetFillColor(H,QMColor);
    FillRectangle(H,WindowRectangle);
    Pos.h:=0.05;
    Pos.v:=1.10;
    WriteString(H,Pos,Str);
    IF Str[0]='S' THEN
      Pos.h:=0.05;
      Pos.v:=0.55;
      WriteString(H,Pos,'No name = No save');
    END;
    Pos.h:=0.05;
    Pos.v:=0.10;
    FOR k:=0 TO 7 DO
      Name[k]:=' ';
    END;
    ReadString(H,Pos,Name);
    FOR k:=0 TO 7 DO
      Ch:=Name[k];
      Number:= ORD(Ch);
      IF ((Number >=48) AND (Number<=57)) OR
         ((Number >=65) AND (Number<=90)) OR
         ((Number >=97) AND (Number<=122)) THEN
        Name[k]:=Ch;
```

```
      ELSE
        Name[k]:=' ';
      END;
    END;
    HideMenu(QMwindow);
  END;
  ShowCursor;
END ShowQuestionWindow;


PROCEDURE TMRoutine(VAR HMwindow,TMwindow,QMwindow : CommandWindowType;
                        TMPwindow,PMW1,PMW2,PMW3,PMW4,PMW5 : CommandWindowTyp

VAR       SelectedBox : CARDINAL;
          Name        : TestText;

BEGIN
  HideMenu(HMwindow);
  ShowTestMenu(TMwindow);
  SelectedBox:=WaitMouseRectangle(TMwindow^.H);
  PutEmerg(FALSE);
  CASE SelectedBox OF
    5 :StopTest;
       ShowQuestionWindow(TMwindow,QMwindow,'Save test as?',Name);
       SetEnabled(TRUE);SetMeasWork(Meas);SetTest(Name,4)|
    6 :StopTest;
       ShowQuestionWindow(TMwindow,QMwindow,'Save test as?',Name);
       SetEnabled(TRUE);SetMeasWork(Meas);SetTest(Name,3)|
    7 :StopTest;
       ShowQuestionWindow(TMwindow,QMwindow,'Save test as?',Name);
       SetEnabled(TRUE);SetMeasWork(Meas);SetTest(Name,2)|
    8 :StopTest;
       ShowQuestionWindow(TMwindow,QMwindow,'Save test as?',Name);
       SetEnabled(TRUE);SetMeasWork(Meas);SetTest(Name,1)|
    3 :TParMRoutine(TMwindow,QMwindow,TMPwindow,PMW1,PMW2,PMW3,PMW4,PMW5)|
    2 :StopTest;
       ShowQuestionWindow(TMwindow,QMwindow,'File to load?',Name);
       SetEnabled(TRUE);SetMeasWork(OldData);SetTest(Name,1)|
    4 :StopTest;
       ShowQuestionWindow(TMwindow,QMwindow,'Save test as?',Name);
       SetEnabled(TRUE);SetMeasWork(Meas);SetTest(Name,5)|
    1 :;
    ELSE;
  END;
  ShowHeadMenu(HMwindow);
END TMRoutine;


PROCEDURE InitParSubMenus(VAR PMW1,PMW2,PMW3,PMW4,PMW5 : CommandWindowType);

VAR       Pos    : point;
          i      : CARDINAL;
          ParStr : ARRAY[1..8] OF TextStr;

BEGIN
  NEW(PMW1);
  WITH PMW1^ DO
    VirtualScreen(H);
    Kind:=Num;
    Pos.h:=0.00;
    Pos.v:=0.55;
    MakeNumMenu(NM,Pos,7);
    SetNumMenuColors(NM,TMColor,TMTextColor,TMAlertColor,HideColor);
    SetNumMenuEntry(NM,'Ramptime',MotPar[1].RampTime);
```

```
  SetNumMenuEntry(NM,'Breaktime',MotPar[1].BreakTime);
  SetNumMenuEntry(NM,'RandomTime',MotPar[1].RandomTime);
  SetNumMenuEntry(NM,'MinTime',MotPar[1].WaitPaus);
  SetNumMenuEntry(NM,'MaxSpeed',MotPar[1].MaxSpeed);
  SetNumMenuEntry(NM,'BeepDelay',MotPar[1].BeepDelay);
  SetNumMenuEntry(NM,'Max testtime',MotPar[1].tmax);
END;
NEW(PMW2);
WITH PMW2^ DO
  VirtualScreen(H);
  Kind:=Num;
  Pos.h:=0.00;
  Pos.v:=0.55;
  MakeNumMenu(NM,Pos,7);
  SetNumMenuColors(NM,TMColor,TMTextColor,TMAlertColor,HideColor);
  SetNumMenuEntry(NM,'Ramptime',MotPar[2].RampTime);
  SetNumMenuEntry(NM,'Breaktime',MotPar[2].BreakTime);
  SetNumMenuEntry(NM,'MinTime',MotPar[2].WaitPaus);
  SetNumMenuEntry(NM,'Center speed',MotPar[2].MaxSpeed);
  SetNumMenuEntry(NM,'RandomTime',MotPar[2].RandomTime);
  SetNumMenuEntry(NM,'RandomSpeed',MotPar[2].RandomSpeed);
  SetNumMenuEntry(NM,'Max testtime',MotPar[2].tmax);
END;
NEW(PMW3);
WITH PMW3^ DO
  VirtualScreen(H);
  Kind:=Num;
  Pos.h:=0.00;
  Pos.v:=0.55;
  MakeNumMenu(NM,Pos,6);
  SetNumMenuColors(NM,TMColor,TMTextColor,TMAlertColor,HideColor);
  SetNumMenuEntry(NM,'Ramptime',MotPar[3].RampTime);
  SetNumMenuEntry(NM,'WaitPause',MotPar[3].WaitPaus);
  SetNumMenuEntry(NM,'Center speed',MotPar[3].MaxSpeed);
  SetNumMenuEntry(NM,'Period time',MotPar[3].Period);
  SetNumMenuEntry(NM,'Amplitude',MotPar[3].RandomSpeed);
  SetNumMenuEntry(NM,'Max testtime',MotPar[3].tmax);
END;
NEW(PMW4);
WITH PMW4^ DO
  VirtualScreen(H);
  Kind:=Num;
  Pos.h:=0.00;
  Pos.v:=0.55;
  MakeNumMenu(NM,Pos,7);
  SetNumMenuColors(NM,TMColor,TMTextColor,TMAlertColor,HideColor);
  SetNumMenuEntry(NM,'RampTime',MotPar[4].RampTime);
  SetNumMenuEntry(NM,'WaitPause',MotPar[4].WaitPaus);
  SetNumMenuEntry(NM,'Center speed',MotPar[4].MaxSpeed);
  SetNumMenuEntry(NM,'Amplitude',MotPar[4].RandomSpeed);
  SetNumMenuEntry(NM,'Period',MotPar[4].Period);
  SetNumMenuEntry(NM,'PRBSPeriod',MotPar[4].PRBSPeriod);
  SetNumMenuEntry(NM,'Max testtime',MotPar[4].tmax);
END;
NEW(PMW5);
WITH PMW5^ DO
  VirtualScreen(H);
  Kind:=Num;
  Pos.h:=0.00;
  Pos.v:=0.55;
  MakeNumMenu(NM,Pos,6);
  SetNumMenuColors(NM,TMColor,TMTextColor,TMAlertColor,HideColor);
  SetNumMenuEntry(NM,'Ramptime',MotPar[5].RampTime);
  SetNumMenuEntry(NM,'WaitPause',MotPar[5].WaitPaus);
```

```
    SetNumMenuEntry(NM,'Center speed',MotPar[5].MaxSpeed);
    SetNumMenuEntry(NM,'Amplitude',MotPar[5].RandomSpeed);
    SetNumMenuEntry(NM,'PRBSPeriod',MotPar[5].PRBSPeriod);
    SetNumMenuEntry(NM,'Max testtime',MotPar[5].tmax);
   END;
END InitParSubMenus;


PROCEDURE CopyPar1(a : ARRAY OF REAL);

BEGIN
  WITH MotPar[1] DO
    RampTime:=a[0];
    BreakTime:=a[1];
    RandomTime:=a[2];
    WaitPaus:=a[3];
    MaxSpeed:=a[4];
    BeepDelay:=a[5];
    tmax:=a[6];
  END;
  PutMotorData(MotPar[1],1);
END CopyPar1;

PROCEDURE CopyPar2(a : ARRAY OF REAL);

BEGIN
  WITH MotPar[2] DO
    RampTime:=a[0];
    BreakTime:=a[1];
    WaitPaus:=a[2];
    MaxSpeed:=a[3];
    RandomTime:=a[4];
    RandomSpeed:=a[5];
    tmax:=a[6];
  END;
  PutMotorData(MotPar[2],2);
END CopyPar2;


PROCEDURE CopyPar3(a : ARRAY OF REAL);

BEGIN
    WITH MotPar[3] DO
      RampTime:=a[0];
      WaitPaus:=a[1];
      MaxSpeed:=a[2];
      Period:=a[3];
      RandomSpeed:=a[4];
      tmax:=a[5];
    END;
    PutMotorData(MotPar[3],3);
    Tabbe := a[3] < 5.0;
END CopyPar3;

PROCEDURE CopyPar4(a : ARRAY OF REAL);

BEGIN
  WITH MotPar[4] DO
    RampTime:=a[0];
    WaitPaus:=a[1];
    MaxSpeed:=a[2];
    RandomSpeed:=a[3];
    Period:=a[4];
    PRBSPeriod:=a[5];
```

```
    tmax:=a[6];
  END;
  PutMotorData(MotPar[4],4);
  Tabbe := a[4] < 5.0;
END CopyPar4;

PROCEDURE CopyPar5(a : ARRAY OF REAL);

BEGIN
    WITH MotPar[5] DO
      RampTime:=a[0];
      WaitPaus:=a[1];
      MaxSpeed:=a[2];
      RandomSpeed:=a[3];
      PRBSPeriod:=a[4];
      tmax:=a[5];
    END;
    PutMotorData(MotPar[5],5);
    Tabbe := a[3] > 3.0;
END CopyPar5;


PROCEDURE TPSubRoutine(VAR TMwindow,QMwindow,SubMenu : CommandWindowType;
                       nr : INTEGER);

VAR       Pos  : point;
          Name : ARRAY[0..1] OF CHAR;

BEGIN
  HideMenu(TMwindow);
  WITH SubMenu^ DO
    ShowNumMenu(NM);
    CASE nr OF

      1:PutNr(8);GetNumMenuStateWait(NM,CopyPar1)|
      2:PutNr(9);GetNumMenuStateWait(NM,CopyPar2)|
      3:PutNr(10);
        REPEAT
          GetNumMenuStateWait(NM,CopyPar3);
          IF Tabbe THEN
              HideNumMenu(NM);
              HideCursor;
              WITH QMwindow^ DO
                SetFillColor(H,QMColor);
                FillRectangle(H,WindowRectangle);
                Pos.h:=0.05;
                Pos.v:=1.10;
                WriteString(H,Pos,'Period > 5 please');
                Pos.h:=0.05;
                Pos.v:=0.10;
                WriteString(H,Pos,'hit return');
                Pos.h:=1.55;
                ReadString(H,Pos,Name);
              END;
              HideMenu(QMwindow);
              ShowCursor;
              ShowNumMenu(NM);
           END;
        UNTIL NOT Tabbe|
      4:PutNr(11);
        REPEAT
          GetNumMenuStateWait(NM,CopyPar4);
          IF Tabbe THEN
              HideNumMenu(NM);
```

```
              HideCursor;
              WITH QMwindow^ DO
                SetFillColor(H,QMColor);
                FillRectangle(H,WindowRectangle);
                Pos.h:=0.05;
                Pos.v:=1.10;
                WriteString(H,Pos,'Period > 5 please');
                Pos.h:=0.05;
                Pos.v:=0.10;
                WriteString(H,Pos,'hit return');
                Pos.h:=1.55;
                ReadString(H,Pos,Name);
              END;
              HideMenu(QMwindow);
              ShowCursor;
              ShowNumMenu(NM);
           END;
        UNTIL NOT Tabbe|
      5:PutNr(12);
        REPEAT
          GetNumMenuStateWait(NM,CopyPar5);
          IF Tabbe THEN
              HideNumMenu(NM);
              HideCursor;
              WITH QMwindow^ DO
                SetFillColor(H,QMColor);
                FillRectangle(H,WindowRectangle);
                Pos.h:=0.05;
                Pos.v:=1.10;
                WriteString(H,Pos,'MaxSpeed <= 3 please');
                Pos.h:=0.05;
                Pos.v:=0.10;
                WriteString(H,Pos,'hit return');
                Pos.h:=1.55;
                ReadString(H,Pos,Name);
              END;
              HideMenu(QMwindow);
              ShowCursor;
              ShowNumMenu(NM);
           END;
        UNTIL NOT Tabbe;
    ELSE;
    END;
    HideNumMenu(NM);
  END;
END TPSubRoutine;


PROCEDURE TParMRoutine(VAR TMwindow,QMwindow,TMPwindow,PMW1,PMW2,
                       PMW3,PMW4,PMW5 : CommandWindowType);

VAR     SelectedBox : CARDINAL;
        Done        : BOOLEAN;
BEGIN
  Done:=FALSE;
  HideMenu(TMwindow);
  REPEAT
    ShowTestParametersMenu(TMPwindow);
    SelectedBox:=WaitMouseRectangle(TMPwindow^.H);
    CASE SelectedBox OF
      1 :Done:=TRUE|
      2 :TPSubRoutine(TMPwindow,QMwindow,PMW5,5)|
      3 :TPSubRoutine(TMPwindow,QMwindow,PMW4,4)|
      4 :TPSubRoutine(TMPwindow,QMwindow,PMW3,3)|
```

```
      5 :TPSubRoutine(TMPwindow,QMwindow,PMW2,2)|
      6 :TPSubRoutine(TMPwindow,QMwindow,PMW1,1);
    ELSE;
    END;
  UNTIL Done=TRUE;
END TParMRoutine;


(*************************************************************************

(******************** Procedures for ParmeterMenu ************************


PROCEDURE InitSystemParametersMenu(VAR SPwindow : CommandWindowType);

VAR       Pos    : point;
          i      : CARDINAL;
          SPStr  : ARRAY[1..7] OF TextStr;

BEGIN
  SPStr[1]:='Sampletime [s]';   SPStr[2]:='Plot const';
  SPStr[3]:='File const';       SPStr[4]:='foot-back  [m]';
  SPStr[5]:='foot-front [m]';   SPStr[6]:='load koef [N/V]';
  SPStr[7]:='StabTime   [s]';
  NEW(SPwindow);
  WITH SPwindow^ DO
    VirtualScreen(H);
    Kind:=Num;
    Pos.h:=0.0;
    Pos.v:=0.55;
    MakeNumMenu(NM,Pos,7);
    SetNumMenuColors(NM,SPMColor,SPMTextColor,SPMAlertColor,HideColor);
    WITH SysPar DO
      SetNumMenuEntry(NM,SPStr[1],SampleTime);
      SetNumMenuEntry(NM,SPStr[2],float(PlotConst));
      SetNumMenuEntry(NM,SPStr[3],float(FileConst));
      SetNumMenuEntry(NM,SPStr[4],A);
      SetNumMenuEntry(NM,SPStr[5],B);
      SetNumMenuEntry(NM,SPStr[6],Gamma);
      SetNumMenuEntry(NM,SPStr[7],TStart);
    END;
  END;
END InitSystemParametersMenu;


PROCEDURE CopySysPar(a : ARRAY OF REAL);

BEGIN
  WITH SysPar DO
    SampleTime:=a[0];
    PlotConst:=round(a[1]);
    A:=a[3];
    B:=a[4];
    Gamma:=a[5];
    FileConst:=round(a[2]);
    TStart:=a[6];
    PutMomentData(PlotConst,SampleTime,A,B,Gamma);
    SetFileConst(FileConst);
    SetTStart(TStart);
  END;
END CopySysPar;


PROCEDURE SystemParametersRoutine(VAR HMwindow,SPwindow : CommandWindowType);
```

```
BEGIN
  HideMenu(HMwindow);
  PutNr(13);
  WITH SPwindow^ DO
    ShowNumMenu(NM);
    GetNumMenuStateWait(NM,CopySysPar);
    HideNumMenu(NM);
  END;
  ShowHeadMenu(HMwindow);
END SystemParametersRoutine;


(*************************************************************************

(******************** Procedures for EstimatorMenu ************************


PROCEDURE InitEstMenu(VAR EMwindow : CommandWindowType);

VAR       ViewPortRectangle : rectangle;
          i                 : INTEGER;

BEGIN
   NEW(EMwindow);
   WITH EMwindow^ DO
      VirtualScreen(H);
      Kind:= Ord;
      SetRectangle(ViewPortRectangle, 0.05, 0.40, 0.45, 0.85);
      SetViewPort(H,ViewPortRectangle);
      SetRectangle(WindowRectangle, 0.0, 0.0, 1.0, 5.0);
      SetWindow(H,WindowRectangle);
      SetTextColor(H, EMTextColor);
      SetLineColor(H, EMTextColor);
      SetFillColor(H, EMColor);
      FOR i:= 1 TO 5 DO
         DefineMouseWindow(H,i);
      END;
   END;
END InitEstMenu;


PROCEDURE ShowEstMenu(VAR window : CommandWindowType);

VAR       String : ARRAY [0..5] OF TextStr;
          i      : INTEGER;

BEGIN
   PutNr(4);
   WITH window^ DO
      SetFillColor(H,EMColor);
      HideCursor;
      FillRectangle(H,WindowRectangle);
      ShowCursor;
      String[0]:= "Done";
      String[1]:= "Reset";
      String[2]:= "EstimatorStatus";
      String[3]:= "EstimatorPar.";
      String[4]:= "A/B-Polynom";
      FOR i:= 0 TO 4 DO
         ShowText(H,float(i),String[i]);
      END;
   END;
END ShowEstMenu;
```

```
PROCEDURE InitEstSubMenus(VAR ABW,EPW,OOW : CommandWindowType);

VAR     Pos                          : point;
        i                            : CARDINAL;
        ABStr1,ABStr2,EPStr,OOStr : ARRAY[1..8] OF TextStr;

BEGIN
  ABStr1[1]:='A 1';       ABStr1[2]:='A 2';
  ABStr1[3]:='A 3';       ABStr1[4]:='A 4';
  ABStr1[5]:='B 1';       ABStr1[6]:='B 2';
  ABStr1[7]:='B 3';       ABStr1[8]:='B 4';
  EPStr[1]:='Proc deg';   EPStr[2]:='P0';
  EPStr[3]:='Lambda';     EPStr[4]:='MinErr';
  EPStr[5]:='Delay';      EPStr[6]:='Nr in preestim';
  OOStr[1]:='Estimator'; OOStr[2]:='PreEstim';
  WITH EstPar DO
    NEW(ABW);
    WITH ABW^ DO
      VirtualScreen(H);
      Kind:=Num;
      Pos.h:=0.00;
      Pos.v:=0.55;
      MakeNumMenu(NM,Pos,8);
      SetNumMenuColors(NM,EMColor,EMTextColor,EMAlertColor,HideColor);
      FOR i:=1 TO 4 DO
        SetNumMenuEntry(NM,ABStr1[i],EA[i]);
      END;
      FOR i:=1 TO 4 DO
        SetNumMenuEntry(NM,ABStr1[i+4],EB[i]);
      END;
    END;
    NEW(EPW);
    WITH EPW^ DO
      VirtualScreen(H);
      Kind:=Num;
      Pos.h:=0.00;
      Pos.v:=0.55;
      MakeNumMenu(NM,Pos,6);
      SetNumMenuColors(NM,EMColor,EMTextColor,EMAlertColor,HideColor);
      SetNumMenuEntry(NM,EPStr[1],float(ECLDeg));
      SetNumMenuEntry(NM,EPStr[2],EP0);
      SetNumMenuEntry(NM,EPStr[3],ELambda);
      SetNumMenuEntry(NM,EPStr[4],EMinErr);
      SetNumMenuEntry(NM,EPStr[5],float(EDelay));
      SetNumMenuEntry(NM,EPStr[6],float(EPreLength));
    END;
    NEW(OOW);
    WITH OOW^ DO
      VirtualScreen(H);
      Kind:=Log;
      Pos.h:=0.00;
      Pos.v:=0.55;
      MakeLogMenu(LM,Pos,2);
      SetLogMenuColors(LM,EMTrue,EMColor,EMTextColor,EMAlertColor,HideColor);
      SetLogMenuEntry(LM,OOStr[1],EstPar.EEstim);
      SetLogMenuEntry(LM,OOStr[2],EstPar.EPreEstim);
    END;
  END;
END InitEstSubMenus;


PROCEDURE CopyEstParameters(a : ARRAY OF REAL);
```

```
BEGIN
  WITH EstPar DO
    IF a[0]<5.0 THEN
      ECLDeg:=round(a[0]);
    END;
    EP0:=a[1];
    ELambda:=a[2];
    EMinErr:=a[3];
    EDelay:=round(a[4]);
    EPreLength:=round(a[5]);
    PutCLSpec(EA,EB,EMinErr,EP0,ELambda,ECLDeg,EEstim,EReset);
    SetPreLength(EPreLength);
  END;
  Tabbe:= a[0]>4.0;
END CopyEstParameters;


PROCEDURE CopyAB(a : ARRAY OF REAL);

BEGIN
  WITH EstPar DO
    EA[1]:=a[0];
    EA[2]:=a[1];
    EA[3]:=a[2];
    EA[4]:=a[3];
    EB[1]:=a[4];
    EB[2]:=a[5];
    EB[3]:=a[6];
    EB[4]:=a[7];
    PutCLSpec(EA,EB,EMinErr,EP0,ELambda,ECLDeg,EEstim,EReset);
  END;
END CopyAB;


PROCEDURE CopyOO(a : ARRAY OF BOOLEAN);

BEGIN
  WITH EstPar DO
    EEstim:=a[0];
    EPreEstim:=a[1];
    IF NOT EPreEstim THEN
      SetDelay(EDelay);
    END;
    SetFilled(NOT a[1]);
    SetReady(NOT a[1]);
    PutCLSpec(EA,EB,EMinErr,EP0,ELambda,ECLDeg,EEstim,EReset);
  END;
END CopyOO;


PROCEDURE SubMenuRoutine(I : INTEGER; EMWindow,SubMenu,
                         QMwindow : CommandWindowType);

VAR       Pos  : point;
          Name : ARRAY[0..1] OF CHAR;

BEGIN
  HideMenu(EMWindow);
  WITH SubMenu^ DO
    IF I = 3 THEN
      ShowLogMenu(LM);
    ELSE
      ShowNumMenu(NM);
```

```
    END;
    CASE I OF
      3 :PutNr(14);GetLogMenuStateWait(LM,CopyOO)|
      4 :PutNr(15);
         REPEAT
          GetNumMenuStateWait(NM,CopyEstParameters);
          IF Tabbe THEN
              HideNumMenu(NM);
              HideCursor;
              WITH QMwindow^ DO
                SetFillColor(H,QMColor);
                FillRectangle(H,WindowRectangle);
                Pos.h:=0.05;
                Pos.v:=1.10;
                WriteString(H,Pos,'Degree < 5 please');
                Pos.h:=0.05;
                Pos.v:=0.10;
                WriteString(H,Pos,'hit return');
                Pos.h:=1.55;
                ReadString(H,Pos,Name);
              END;
              HideMenu(QMwindow);
              ShowCursor;
              ShowNumMenu(NM);
           END;
          UNTIL NOT Tabbe|

      5 : PutNr(16);GetNumMenuStateWait(NM,CopyAB);
    ELSE;
    END;
    IF I = 3 THEN
      HideLogMenu(LM);
    ELSE
      HideNumMenu(NM);
      END;
  END;
END SubMenuRoutine;


PROCEDURE EMRoutine(HMWindow,EMWindow,QMwindow,Sub3W,Sub4W,
                    Sub5W : CommandWindowType);

VAR           SelectedBox : CARDINAL;
              Ready,Ok    : BOOLEAN;

BEGIN
  Ready:=FALSE;
  HideMenu(HMWindow);
  REPEAT
    WITH EstPar DO
      ShowEstMenu(EMWindow);
      SelectedBox:=WaitMouseRectangle(EMWindow^.H);
      CASE SelectedBox OF
        1: Ready:=TRUE|
        2: PutCLSpec(EA,EB,EMinErr,EP0,ELambda,ECLDeg,EEstim,TRUE)|
        3: SubMenuRoutine(3,EMWindow,Sub3W,QMwindow)|
        4: SubMenuRoutine(4,EMWindow,Sub4W,QMwindow)|
        5: SubMenuRoutine(5,EMWindow,Sub5W,QMwindow);
      END;
    END;
  UNTIL Ready;
  HideMenu(EMWindow);
  ShowHeadMenu(HMWindow);
END EMRoutine;
```

```
(*************************************************************************

(********************** Procedures for PlotMenu ***************************


PROCEDURE InitPlotMenu(VAR PMwindow : CommandWindowType);

VAR      ViewPortRectangle : rectangle;
         i                 : INTEGER;

BEGIN
  NEW(PMwindow);
  WITH PMwindow^ DO
    VirtualScreen(H);
    Kind:=Ord;
    SetRectangle(ViewPortRectangle,0.05,0.40,0.45,0.80);
    SetViewPort(H,ViewPortRectangle);
    SetRectangle(WindowRectangle,0.0,0.0,1.0,3.0);
    SetWindow(H,WindowRectangle);
    SetTextColor(H,PMTextColor);
    SetLineColor(H,PMTextColor);
    SetFillColor(H,PMColor);
    FOR i:= 1 TO 3 DO
      DefineMouseWindow(H,i);
    END;
  END;
END InitPlotMenu;


PROCEDURE InitPlotTestMenu(VAR PTMwindow : CommandWindowType);

VAR      ViewPortRectangle : rectangle;
         i                 : INTEGER;

BEGIN
  NEW(PTMwindow);
  WITH PTMwindow^ DO
    VirtualScreen(H);
    Kind:=Ord;
    SetRectangle(ViewPortRectangle,0.05,0.40,0.45,0.80);
    SetViewPort(H,ViewPortRectangle);
    SetRectangle(WindowRectangle,0.0,0.0,1.0,6.0);
    SetWindow(H,WindowRectangle);
    SetTextColor(H,PMTextColor);
    SetLineColor(H,PMTextColor);
    SetFillColor(H,PMColor);
    FOR i:= 1 TO 6 DO
      DefineMouseWindow(H,i);
    END;
  END;
END InitPlotTestMenu;



PROCEDURE ShowPlotMenu(VAR PMwindow : CommandWindowType);

VAR      String : ARRAY[0..3] OF TextStr;
         i      : INTEGER;

BEGIN
  PutNr(5);
  WITH PMwindow^ DO
```

```
    SetFillColor(H,PMColor);
    HideCursor;
    FillRectangle(H,WindowRectangle);
    ShowCursor;
    String[2]:=' Measure ';
    String[1]:=' Estimator';
    String[0]:=' TestSeq';
    FOR i:=0 TO 2 DO
      ShowText(H,float(i),String[i]);
    END;
  END;
END ShowPlotMenu;


PROCEDURE ShowPlotTestMenu(VAR PTMwindow : CommandWindowType);

VAR      String : ARRAY[0..5] OF TextStr;
         i      : INTEGER;

BEGIN
  WITH PTMwindow^ DO
    PutNr(6);
    SetFillColor(H,PMColor);
    HideCursor;
    FillRectangle(H,WindowRectangle);
    ShowCursor;
    String[5]:=' BeepTest';
    String[4]:=' RampRandom';
    String[3]:=' Sinus';
    String[2]:=' PRBSSinus';
    String[1]:=' PRBS';
    String[0]:=' DONE';
    FOR i:=0 TO 5 DO
      ShowText(H,float(i),String[i]);
    END;
  END;
END ShowPlotTestMenu;


PROCEDURE PTMRoutine(VAR HMwindow,PTMwindow : CommandWindowType);

VAR          SelectedBox : CARDINAL;
             Name        : TestText;
             Done        : BOOLEAN;

BEGIN
  Done:=FALSE;
  HideMenu(HMwindow);
  REPEAT
    ShowPlotTestMenu(PTMwindow);
    SelectedBox:=WaitMouseRectangle(PTMwindow^.H);
    PutEmerg(FALSE);
    CASE SelectedBox OF
      2 :ChangeWhat(Test);SetMeasWork(Sekv);SetTest(Name,5)|
      3 :ChangeWhat(Test);SetMeasWork(Sekv);SetTest(Name,4)|
      4 :ChangeWhat(Test);SetMeasWork(Sekv);SetTest(Name,3)|
      5 :ChangeWhat(Test);SetMeasWork(Sekv);SetTest(Name,2)|
      6 :ChangeWhat(Test);SetMeasWork(Sekv);SetTest(Name,1)|
      1 :Done:=TRUE;
    ELSE;
    END;
  UNTIL Done;
  HideMenu(PTMwindow);
  ShowHeadMenu(HMwindow);
```

```
END PTMRoutine;


PROCEDURE PMRoutine(VAR HMwindow,PMwindow,PTMwindow,
                    QMwindow : CommandWindowType);

VAR  SelectedBox    : CARDINAL;

BEGIN
  HideMenu(HMwindow);
  ShowPlotMenu(PMwindow);
  SelectedBox:=WaitMouseRectangle(PMwindow^.H);
  CASE SelectedBox OF
    1 : ChangeWhat(Test);PTMRoutine(PMwindow,PTMwindow)|
    2 : ChangeWhat(Par)|
    3 : ChangeWhat(Data);
  ELSE;
  END;
  HideMenu(PMwindow);
  ShowHeadMenu(HMwindow);
END PMRoutine;


(*************************************************************************

(********************* Procedure for Quit ***********************************


PROCEDURE Quit;

BEGIN
   Signal(TheEnd);
END Quit;


(*************************************************************************

(********************* Procedure for ErrorProc ******************************


(*Process*)PROCEDURE ErrorProc;

VAR       ErrWindow : CommandWindowType;
          ErrorMess : ErrorPtrType;
          Textpoint : point;
          t         : Time;

BEGIN
   SetPriority(89);
   InitErrorWindow(ErrWindow);
   Textpoint.h:= 0.1;
   Textpoint.v:= 0.3;
   LOOP
      ReceiveMessage(ErrorBox,ErrorMess);
      HideCursor;
      WITH ErrWindow^ DO
         SetFillColor(H,ErrorColor);
         FillRectangle(H,WindowRectangle);
         DrawRectangle(H,WindowRectangle);
         WriteString(H,Textpoint,ErrorMess^.ErrStr);
      END;
      ShowCursor;
      DISPOSE(ErrorMess);
      GetTime(t);
```

```
      IncTime(t,5000);
      WaitUntil(t);
      HideMenu(ErrWindow);
   END;
END ErrorProc;


(*************************************************************************

(******************** Procedure for ControlProc ***************************


(*Process*)PROCEDURE ControlProc;

VAR SelectedBox                                    : CARDINAL;
    HeadMenuWindow,TestMenuWindow,PlotMenuWindow : CommandWindowType;
    EstMenuWindow,ABWindow,EstParWindow          : CommandWindowType;
    P1window,P2window,P3window,P4window,P5window : CommandWindowType;
    OnOffWindow,QuestWindow,TestParMenuWindow    : CommandWindowType;
    PlotTestMenuWindow,SysParMenuWindow          : CommandWindowType;


BEGIN
  SetPriority(90);
  InitParameters;
  InitLogMenu(90);
  InitNumMenu(90);
  InitHeadMenu(HeadMenuWindow);
  InitTestMenu(TestMenuWindow);
  InitPlotTestMenu(PlotTestMenuWindow);
  InitTestParametersMenu(TestParMenuWindow);
  InitEstMenu(EstMenuWindow);
  InitEstSubMenus(ABWindow,EstParWindow,OnOffWindow);
  InitParSubMenus(P1window,P2window,P3window,P4window,P5window);
  InitPlotMenu(PlotMenuWindow);
  InitQuestionWindow(QuestWindow);
  InitSystemParametersMenu(SysParMenuWindow);
  ShowHeadMenu(HeadMenuWindow);
  ShowCursor;
  LOOP
    SelectedBox:=WaitMouseRectangle(HeadMenuWindow^.H);
    CASE SelectedBox OF
      6 :SystemParametersRoutine(HeadMenuWindow,SysParMenuWindow);|
      5 :TMRoutine(HeadMenuWindow,TestMenuWindow,QuestWindow,TestParMenuWindo
                   P1window,P2window,P3window,P4window,P5window);|
      4 :EMRoutine(HeadMenuWindow,EstMenuWindow,QuestWindow,OnOffWindow,EstPa
      3 :PMRoutine(HeadMenuWindow,PlotMenuWindow,PlotTestMenuWindow,QuestWind
      2 :StopTest|
      1 :Quit;
    ELSE;
    END;
  END;
END ControlProc;


(*************************************************************************

(******************** Procedure for Initation *****************************


PROCEDURE InitControl;

BEGIN
  NiceStartUpMessage;
```

```
  CreateProcess(ControlProc,10000);
  CreateProcess(ErrorProc,1000);
END InitControl;

END Control.
```

```
DEFINITION MODULE EstimProcess;

EXPORT QUALIFIED Estimate;

PROCEDURE Estimate;

END EstimProcess.
```

```
IMPLEMENTATION MODULE EstimProcess;

FROM Kernel      IMPORT SetPriority;

FROM Messages    IMPORT ReceiveMessage,AcceptMessage,SendMessage;

FROM Decl        IMPORT EBox,PBox,GetCLSpec,PlotEstim,Error,GetMV,ABType,
                        GetCounter,PMailType,PlotKinds,ErrorType,
                        SetReady,GetValue,GetPreLength,GetEstim,SetDelay,
                        GetSample,ResetEstim;

FROM ConvReal    IMPORT RealToString;

FROM MathLib     IMPORT float,round;

CONST    EstPri = 100;
         MaxDeg = 4;

TYPE     MType = ARRAY[1..MaxDeg*2],[1..MaxDeg*2] OF REAL;
         VType = ARRAY[1..MaxDeg*2] OF REAL;

VAR      MinErr,P0,Lambda,sum,eps,V : REAL;
         Start,Ready,Reset,
         PreFilled,NoSend,Run       : BOOLEAN;
         PlotCount,CLDeg,D0,Nr,k    : INTEGER;
         K,th,th1,fi                : VType;
         A,B,A0,B0                  : ABType;
         P                          : MType;

(*****************************************************************)

(*****************   initiation  procedure      *******************)


 PROCEDURE InitEstim;

 VAR        N,M : INTEGER;

 BEGIN
    FOR N:=1 TO 2*CLDeg DO      (* initiate startvalues *)
      FOR M:=1 TO 2*CLDeg DO
        IF N=M THEN
          P[N,M]:=P0;
         ELSE
          P[N,M]:=0.0;
        END;
      END;
      fi[N]:=0.0;
    END;
    A:=A0;
    B:=B0;
    FOR N:=1 TO CLDeg DO
      th[N]:=B0[N];
      th[N+CLDeg]:=A0[N];
    END;
 END InitEstim;


(*****************************************************************)

(*****************   plotmessage  procedure     *******************)


 PROCEDURE CheckPlotEstim(VAR Change : BOOLEAN);
```

```
(* Checks if estimator parameters should be sent to plot *)

VAR                 PlotMess : PMailType;
                          c : INTEGER;
BEGIN
   GetCounter(c);
   IF c<1 THEN
      c:=1;
   END;
   PlotCount:=(PlotCount+1) MOD c;
   IF PlotEstim() AND ((PlotCount=0) OR NOT Ready OR Change) THEN
     AcceptMessage(EBox,PlotMess);
     IF Ready THEN
       PlotMess^.Ready:=TRUE;       (* sends data to plotprocedure *)
       PlotMess^.Start:=Start;
       PlotMess^.ExChange:=Change;
       PlotMess^.A:=A;
       PlotMess^.B:=B;
     ELSE
       PlotMess^.Ready:=FALSE;
       PlotMess^.Start:=Start;
       PlotMess^.ExChange:=Change;
       PlotMess^.V:=V;
     END;
     SendMessage(PBox,PlotMess);
     IF Start THEN
       Start:=FALSE;
     END;
     IF Change THEN
       Change:=FALSE;
     END;
   ELSIF NoSend THEN
     Error('Plot data ovfl (Est)'); (* errormessage to control *)
     NoSend:=FALSE;
   END;
END CheckPlotEstim;


(*****************************************************************)

(*****************   lossfunction  procedure    ******************)


PROCEDURE ErrorValue( D : CARDINAL; VAR V : REAL; PreLength : CARDINAL);

 VAR           N,Nr      : INTEGER;
               fi        : VType;
               eps1      : REAL;
               y         : REAL;

BEGIN
   eps1:=0.0;
   V:=0.0;
   FOR Nr:=1 TO 2*CLDeg DO
     fi[Nr]:=0.0;
   END;
   FOR Nr:=D TO (PreLength-1) DO    (* calculate error between *)
     FOR N:=2 TO CLDeg DO           (* modell and referens     *)
       fi[N]:=fi[N-1];
       fi[N+CLDeg]:=fi[N-1+CLDeg];
     END;
     sum:=0.0;
     GetValue(fi[1],fi[CLDeg+1],D);
     fi[CLDeg+1]:=-fi[CLDeg+1];
```

```
      y:=fi[CLDeg+1];
      FOR N:=1 TO 2*CLDeg DO
        sum:=sum+fi[N]*th[N];
      END;
      eps1:=y-sum;
      V:=V+eps1*eps1;
    END;
    eps:=float(PreLength-D);
    V:=V/eps;
END ErrorValue;


(*****************************************************************)

(*****************    estimation   procedure    *********************)


PROCEDURE   PreEstimAB(VAR D0 : INTEGER ; PreLength : INTEGER;
                       VAR th : VType; VAR Run : BOOLEAN);
(* Estimator according to formulas 11.5-11.7 in Adaptive Control book *)

VAR                          N,M,Nr : INTEGER;
                             Pfi    : VType;
                             y      : REAL;

BEGIN
   Nr:=0;
   eps:=0.0;
   WHILE (Nr<=PreLength) AND Run DO
     GetEstim(Run);
     sum:=0.0;                    (* algoritm for preestimation *)
     FOR N:=2 TO CLDeg DO         (* of timedelay               *)
       fi[N]:=fi[N-1];
       fi[N+CLDeg]:=fi[N-1+CLDeg];      (*  fi(t)   *)
     END;
     GetValue(fi[1],fi[CLDeg+1],D0);
     fi[CLDeg+1]:=-fi[CLDeg+1];
     y:=fi[CLDeg+1];
     FOR N:=1 TO 2*CLDeg DO
       sum:=sum+fi[N]*th[N];
     END;
     eps:=y-sum;
     FOR N:=1 TO 2*CLDeg DO
       sum:=0.0;
       FOR M:= 1 TO 2*CLDeg DO
         sum:=sum+P[N,M]*fi[N];
       END;
       Pfi[N]:=sum;
     END;
     sum:=0.0;
     FOR N:=1 TO 2*CLDeg DO
       sum:=sum+fi[N]*Pfi[N];
     END;
     sum:=sum+Lambda;
     FOR N:=1 TO 2*CLDeg DO
       K[N]:=Pfi[N]/sum;                  (*  K(t)    *)
     END;
     FOR N:=1 TO 2*CLDeg DO
       FOR M:= 1 TO 2*CLDeg DO
         IF N=M THEN
           P[N,M]:=(1.0-K[N]*fi[M])*P[N,M]/Lambda; (*  P(t)    *)
         ELSE
           P[N,M]:=-K[N]*Pfi[M]/Lambda;
         END;
```

```
      END;
      th[N]:=th[N]+K[N]*eps;              (*  th(t)    *)
    END;
    Nr:=Nr+1;
  END;
END PreEstimAB;


PROCEDURE   EstimAB(VAR  Aber,Bber : ABType;VAR th1 : VType);

(* Estimator according to formulas 11.5-11.7 in Adaptive Control book *)

VAR                          N,M : INTEGER;
                             Pfi : VType;
                             y   : REAL;
                             test,nr : CARDINAL;
                             vektor :ARRAY[1..4],[1..50] OF REAL;
BEGIN
  eps:=0.0;
  sum:=0.0;                       (* algoritm for parametric  *)
  FOR N:=2 TO CLDeg DO            (* estimation of the modell *)
    fi[N]:=fi[N-1];
    fi[N+CLDeg]:=fi[N-1+CLDeg];   (*  fi(t)    *)
  END;
  GetMV(fi[1],fi[CLDeg+1]);
  fi[CLDeg+1]:=-fi[CLDeg+1];
  y:=fi[CLDeg+1];
  FOR N:=1 TO 2*CLDeg DO
    sum:=sum+fi[N]*th1[N];
  END;
  eps:=y-sum;
  vektor[1,k]:=sum;
  vektor[4,k]:=y;
  IF ABS(eps)>MinErr THEN
    FOR N:=1 TO 2*CLDeg DO
      sum:=0.0;
      FOR M:= 1 TO 2*CLDeg DO
        sum:=sum+P[N,M]*fi[N];
      END;
      Pfi[N]:=sum;
    END;
    sum:=0.0;
    FOR N:=1 TO 2*CLDeg DO
      sum:=sum+fi[N]*Pfi[N];
    END;
    sum:=sum+Lambda;
    FOR N:=1 TO 2*CLDeg DO
      K[N]:=Pfi[N]/sum;            (*  K(t)    *)
    END;
    FOR N:=1 TO 2*CLDeg DO
      FOR M:= 1 TO 2*CLDeg DO
        IF N=M THEN
          P[N,M]:=(1.0-K[N]*fi[M])*P[N,M]/Lambda (*  P(t)    *)
        ELSE
          P[N,M]:=-K[N]*Pfi[M]/Lambda
        END;
        IF P[N,M]>100000.0 THEN
          P[N,M]:=100000.0;
        END;
        IF P[N,M]<-100000.0 THEN
          P[N,M]:=-100000.0;
        END;
      END;
      th1[N]:=th1[N]+K[N]*eps;              (*  th(t)    *)
```

```
      END;
      FOR N:= 1 TO CLDeg DO
        Bber[N]:=th1[N];
        Aber[N]:=th1[N+CLDeg];                    (* new modell *)
        test:=ABS(round(Aber[N]));(**)
      END;
      vektor[2,k]:=Aber[1];
      vektor[3,k]:=Aber[2];
      k:=k MOD 50+1;
      Nr:=Nr+1;
    END;
END EstimAB;


PROCEDURE PreEstim;

  VAR             Vmin,h     : REAL;
                  X,D,Limit  : INTEGER;
                  PreLength  : CARDINAL;
                  Change     : BOOLEAN;
BEGIN
    GetPreLength(PreLength);
    Start:=TRUE;
    Change:=FALSE;
    Vmin:=10000.0;
    D0:=0;
    D:=0;
    X:=0;
    Nr:=1;
    Run:=TRUE;
    GetSample(h);
    Limit:=round(10.0/h);
    IF NOT Ready THEN
      Error('estimate delay');
      REPEAT
        InitEstim;
        PreEstimAB(D,PreLength,th,Run);    (* makes delay estimation *)
        ErrorValue(D,V,PreLength);
        IF V<Vmin THEN
          Vmin:=V;
          D0:=D;
          th1:=th;
          X:=0;
        ELSE
          X:=X+1;
        END;
        IF (X>=20) OR (D>Limit) THEN
          Ready:=TRUE;
          SetReady(TRUE);
          Change:=TRUE;
          SetDelay(D0);
        END;
        D:=D+1;
        CheckPlotEstim(Change);
      UNTIL Ready OR NOT Run;
      th:=th1;
    END;
    IF Ready THEN
      Error('estimate param');     (* estimate parameters *)
      k:=1;
      WHILE Run DO
        GetEstim(Run);
        EstimAB(A,B,th);
        CheckPlotEstim(Change);
```

```
      END;
    END;
 END PreEstim;


 (* Process*) PROCEDURE Estimate;

 BEGIN
    SetPriority(EstPri);
    NoSend:=FALSE;
    PlotCount:=1;
    LOOP
      GetCLSpec(A0,B0,MinErr,P0,Lambda,CLDeg,Reset,Ready,PreFilled,Run);
      IF Reset THEN
        InitEstim;
        ResetEstim;
      END;
      IF PreFilled AND Run THEN
          PreEstim;
      END;
    END;
 END Estimate;
END EstimProcess.
```

```
DEFINITION MODULE HelpProcess;

EXPORT QUALIFIED HelpProc;

PROCEDURE HelpProc;

END HelpProcess.
```

```
IMPLEMENTATION MODULE HelpProcess;

IMPORT RTMouse;

FROM Kernel    IMPORT CreateProcess,SetPriority;

FROM Graphics IMPORT handle,rectangle,color,point,SetWindow,SetViewPort,
                     VirtualScreen,SetFillColor,FillRectangle,
                     DrawRectangle,WaitMouseRectangle,HideCursor,
                     SetMouseRectangle,SetLineColor,SetTextColor,
                     WriteString,ReadString,ShowCursor,buttonset,
                     GetMouseRectangle,WaitForMouse;

FROM Storage   IMPORT ALLOCATE,DEALLOCATE;

FROM MathLib   IMPORT float;

FROM NumMenu   IMPORT MakeNumMenu,SetNumMenuEntry,ShowNumMenu,InitNumMenu,
                     GetNumMenuState,HideNumMenu,SetNumMenuColors,
                     GetNumMenuStateWait,NumMenuType;

FROM LogMenu   IMPORT MakeLogMenu,SetLogMenuEntry,ShowLogMenu,InitLogMenu,
                     GetLogMenuState,HideLogMenu,SetLogMenuColors,
                     GetLogMenuStateWait,LogMenuType;

FROM Decl      IMPORT ChangeWhat,WhatType,Error,CheckPlotWhat,GetNr;

CONST HideColor = black;
      HelpColor = grey;       HelpTextColor = yellow;




TYPE  TextStr         = ARRAY[0..16] OF CHAR;
      RealVectorType = ARRAY[0..10] OF REAL;
      WindowType      = (Ord,Num,Log);

      CommandWindowType = POINTER TO RECORD
                             H : handle;
                             CASE Kind : WindowType OF
                                Ord  : WindowRectangle :rectangle|
                                Num  : NM : NumMenuType|
                                Log  : LM : LogMenuType|
                             END;
                          END;



PROCEDURE SetRectangle(VAR R : rectangle; Xlo,Ylo,Xhi,Yhi : REAL);

BEGIN
   R.xlo:=Xlo;
   R.ylo:=Ylo;
   R.xhi:=Xhi;
   R.yhi:=Yhi;
END SetRectangle;


PROCEDURE DefineMouseWindow(H : handle;y : INTEGER);

VAR     R : rectangle;

BEGIN
```

```
   SetRectangle(R, 0.0, float(y-1), 1.0, float(y));
   SetMouseRectangle(H,R,y);
END DefineMouseWindow;


PROCEDURE ShowText(H :handle; y :REAL; str :TextStr);

VAR     Textpoint : point;
        R         : rectangle;

BEGIN
   Textpoint.h:=0.05;
   Textpoint.v:=y+0.15;
   SetRectangle(R,0.0,y,1.0,y+1.0);
   HideCursor;
   DrawRectangle(H,R);
   WriteString(H,Textpoint,str);
   ShowCursor;
END ShowText;


PROCEDURE HideMenu(VAR Window : CommandWindowType);

BEGIN
   WITH Window^ DO
      SetFillColor(H, HideColor);
      HideCursor;
      FillRectangle(H,WindowRectangle);
      ShowCursor;
   END;
END HideMenu;


(*************************************************************************)

(********************* Procedures for helpmenu ***************************)


PROCEDURE InitHelpButton(VAR HelpButton : CommandWindowType);
(* makes the helpbutton *)

VAR     HelpBox : rectangle;
        TextPos : point;

BEGIN
  NEW(HelpButton);
  WITH HelpButton^ DO
    VirtualScreen(H);
    Kind:=Ord;
    SetRectangle(HelpBox,0.05,0.17,0.45,0.25);
    SetViewPort(H,HelpBox);
    SetRectangle(WindowRectangle,0.0,0.0,1.0,1.0);
    SetWindow(H,WindowRectangle);
    SetTextColor(H,HelpTextColor);
    SetLineColor(H,HelpTextColor);
    SetFillColor(H,HelpColor);
    DefineMouseWindow(H,1);
    TextPos.h:=0.30;
    TextPos.v:=0.20;
    FillRectangle(H,WindowRectangle);
    DrawRectangle(H,WindowRectangle);
    WriteString(H,TextPos,'Help');
  END;
END InitHelpButton;
```

```
PROCEDURE InitHelpTextWindow(VAR HelpWindow : CommandWindowType);
(* initiates the helpwindow *)

VAR     HelpBox : rectangle;

BEGIN
  NEW(HelpWindow);
  WITH HelpWindow^ DO
    VirtualScreen(H);
    Kind:=Ord;
    SetRectangle(HelpBox,0.60,0.05,1.50,1.0);
    SetViewPort(H,HelpBox);
    SetRectangle(WindowRectangle,0.0,0.0,1.0,1.0);
    SetWindow(H,WindowRectangle);
    SetTextColor(H,HelpTextColor);
    SetLineColor(H,HelpTextColor);
    SetFillColor(H,HelpColor);
  END;
END InitHelpTextWindow;


PROCEDURE ShowHelpTextWindow(VAR HelpWindow : CommandWindowType);
(* shows the window where the helptexts are written *)

BEGIN
  WITH HelpWindow^ DO
    HideCursor;
    SetFillColor(H,HelpColor);
    FillRectangle(H,WindowRectangle);
    DrawRectangle(H,WindowRectangle);
    ShowCursor;
  END;
END ShowHelpTextWindow;


PROCEDURE HelpText1(VAR TextWindow : CommandWindowType);
(* headmenu help *)

VAR      Pos,var : point;
         butt    : buttonset;

BEGIN
   WITH TextWindow^ DO
     Pos.h:=0.01;
     Pos.v:=0.95;
     WriteString(H,Pos,'- System parameters -');
     Pos.v:=Pos.v - 0.04;
     WriteString(H,Pos,' Change global parameters');
     Pos.v:=Pos.v - 0.06;
     WriteString(H,Pos,'- Test -');
     Pos.v:=Pos.v - 0.04;
     WriteString(H,Pos,' Choose test to run or change');
     Pos.v:=Pos.v - 0.06;
     WriteString(H,Pos,'- Estimator -');
     Pos.v:=Pos.v - 0.04;
     WriteString(H,Pos,' Configure & run the online estimation');
     Pos.v:=Pos.v - 0.06;
     WriteString(H,Pos,'- Plot -');
     Pos.v:=Pos.v - 0.04;
     WriteString(H,Pos,' Choose what to plot');
     Pos.v:=Pos.v - 0.06;
     WriteString(H,Pos,'- Stop test -');
```

```
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Cancel test in progress');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- Quit program -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Leave VISCON');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText1;


PROCEDURE HelpText2(VAR TextWindow : CommandWindowType);
(* testmenu help *)

VAR     Pos,var : point;
        butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,'- BeepTest -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Ramp up, Random runtime, ramp down and ');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' gives a signal before the break starts  ');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- RampRandom -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Ramps up and dowm with random intervalls');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- Sinus -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' The speed varies as a sinus signal');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- PRBSSinus -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' A Sinus with psuedorandom phaseshifts');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- PRBS -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Runs Forwards and backwards with pseudo-');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' random shifts of direction');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- Parameters -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Change parameters of the tests');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- Run from file -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Load and run an old test');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- DONE -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Return to the headmenu');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText2;
```

```
PROCEDURE HelpText3(VAR TextWindow:CommandWindowType);
(* Testparametersmenu help *)

VAR         Pos,var : point;
            butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,'- Tests -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Change the parameters of the selected test ');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- DONE -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Return to the headmenu');
    Pos.v:=Pos.v - 0.06;
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText3;


PROCEDURE HelpText4(VAR TextWindow:CommandWindowType);
(* estimatormenu help *)

VAR         Pos,var : point;
            butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,'- A/B-Polynom -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Set initial values of the estimated polynomial ');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- EstimatorPar -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Set the estimation parameters');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- EstimatorStatus -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Turn the estimation on & off');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- Reset -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Resets the polynomial to initial values');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- DONE -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Return to the headmenu');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText4;


PROCEDURE HelpText5(VAR TextWindow:CommandWindowType);
(* plotmenu help *)
```

```
VAR       Pos,var : point;
          butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,'- Measure -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Plot the measured data from the platform');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- Estimator -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Plot the loss function during estimation');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- TestSeq -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Plot the test sequences, without the motor');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' running.');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText5;


PROCEDURE HelpText6(VAR TextWindow:CommandWindowType);
(* plottestmenu help *)

VAR       Pos,var : point;
          butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,'- Tests -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Plot the testsequences, without the motor');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' running');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- DONE -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Return to the headmenu');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText6;


PROCEDURE HelpText7(VAR TextWindow:CommandWindowType);
(* questionwindow help *)

VAR      Pos,var : point;
         butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
```

```
    WriteString(H,Pos,' Write the name of the file you want to');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' save ( or load ). if no data is to be saved');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' ( or loaded ) just press return.');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText7;


PROCEDURE HelpText8(VAR TextWindow:CommandWindowType);
(* testpar1 help *)

VAR       Pos,var : point;
          butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,' RampTime, the acceleration time.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' BreakTime, the retardation time.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' RandomTime, maximum added random time.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' MinTime, the minimum time of constant speed.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' MaxSpeed, the maximum control signal to');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' the motor. this cannot be set to more ');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' than ten.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' BeepDelay, the time between the beep and');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' the beginning of the retardation.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Max testtime, self explainatory');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText8;


PROCEDURE HelpText9(VAR TextWindow:CommandWindowType);
(* testpar2 help *)

VAR     Pos,var : point;
        butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,' RampTime, the acceleration time.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' BreakTime, the retardation time.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' MinTime, the minimum time of constant speed.');
    Pos.v:=Pos.v - 0.06;
```

```
    WriteString(H,Pos,' Center speed, the speed around which the ');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' changes in speed are centered.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' RandomTime, the maximum added random speed.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' RandomSpeed, the value of the changes');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' in speed.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Max testtime, the total time of the test.');
    Pos.v:=Pos.v - 0.08;
    WriteString(H,Pos,' Center speed + RandomSpeed must not be');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' larger than 10.');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText9;


PROCEDURE HelpText10(VAR TextWindow:CommandWindowType);
(* testpar3 help*)

VAR      Pos,var : point;
         butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,' RampTime, the time to reach centerspeed.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' WaitPause, the time with centerspeed before');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' the sinewave starts.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Center speed, the speed around which the ');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' changes are centered.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Period Time, the period of the sinewave.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Amplitude, the amplitude of the changes.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Max testtime, the total time of the test');
    Pos.v:=Pos.v - 0.08;
    WriteString(H,Pos,' Center speed + RandomSpeed must not be');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' larger than 10.');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText10;


PROCEDURE HelpText11(VAR TextWindow:CommandWindowType);
(* testpar4 help *)

VAR      Pos,var : point;
         butt    : buttonset;
```

```
BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,' RampTime, the time to reach centerspeed.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' WaitPause, the time with centerspeed before');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' the sinewave starts.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Center speed, the speed around which the ');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' changes are centered.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Amplitude, the amplitude of the changes.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Period Time, the period of the sinewave.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' PRBSPeriod, the period of the PRBS sequence.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Max testtime, the total time of the test');
    Pos.v:=Pos.v - 0.08;
    WriteString(H,Pos,' Center speed + RandomSpeed must not be');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' larger than 10.');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText11;


PROCEDURE HelpText12(VAR TextWindow:CommandWindowType);
(* testpar5 help *)

VAR      Pos,var : point;
         butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,' RampTime, the time to reach centerspeed.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' WaitPause, the time with centerspeed before');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' the sequence starts.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Center speed, the speed around which the ');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' changes are centered.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Amplitude, the amplitude of the changes.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' PRBSPeriod, the period of the PRBS sequence.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Max testtime, the total time of the test');
    Pos.v:=Pos.v - 0.08;
    WriteString(H,Pos,' Center speed + RandomSpeed must not be');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' larger than 10, and the amplitude should not');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' be larger than 2.');
    Pos.v:=0.04;
```

```
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText12;


PROCEDURE HelpText13(VAR TextWindow:CommandWindowType);
(* Systemparameters help *)

VAR       Pos,var : point;
          butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,' SampleTime, how often the program collects');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' data.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Plot const, how often the program sends');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' data to plot.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' File const, how many sets of data the ');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' program collects before writing to disk.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Foot-front, the distance between the foot');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' and the front edge of the platform.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Foot-back, the distance between the foot');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' and the rear edge of the platform.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Load koeff, the relationship between the');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' load on the platform and the measured voltage');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' StabTime, The stabilizing time at the');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' beginning of each test.');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText13;


PROCEDURE HelpText14(VAR TextWindow:CommandWindowType);
(* Estimatorstatus help *)

VAR       Pos,var : point;
          butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,' Estimator, turns the esimator on or off');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' PreEstim, is used to let the computer');
    Pos.v:=Pos.v - 0.04;
```

```
    WriteString(H,Pos,' find the delay.');
    Pos.v:=Pos.v - 0.08;
    WriteString(H,Pos,' A selection is enabled if the button is ');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' green, and disabled if it has the same colour');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' as the Estimator menu. To switch between ');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' enabled and disbled, click the button when ');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' you have made your choice, click done.');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' ');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText14;


PROCEDURE HelpText15(VAR TextWindow:CommandWindowType);
(* estimatorparameters help *)

VAR      Pos,var : point;
         butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,' Proc degree, the degree of the estimated');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' polynomial.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' P0, constant for estimation performance');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Lambda, the forgetting factor, how much ');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' previous values are considered');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' MinErr, the smallest difference between the');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' estimated value and the measured value that');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' affects the polynomial estimation.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Delay, the time difference between stimulation');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' and resulting response. this is measured in ');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' sample intervalls.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Nr in preestim, the number of samples used to ');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' determine the delay.');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText15;


PROCEDURE HelpText16(VAR TextWindow:CommandWindowType);
(* AB-polynom help *)
```

```
VAR      Pos,var : point;
         butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,' This is the initial values of the estimated');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' polynomial.');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText16;


(*************************************************************************

(******************** Procedure for HelpProc ******************************


(*Process*)PROCEDURE HelpProc;

VAR          HelpButt,HelpWindow : CommandWindowType;
             n                   : CARDINAL;
             Ret                 : ARRAY[0..1]OF CHAR;
             Plotting            : WhatType;
             Change              : BOOLEAN;
             Helpnr              : INTEGER;

BEGIN
  SetPriority(89);
  InitHelpButton(HelpButt);
  InitHelpTextWindow(HelpWindow);
  WITH HelpWindow^ DO
    LOOP
      n:=WaitMouseRectangle(HelpButt^.H);
      GetNr(Helpnr);
      IF (Helpnr>0) THEN
        CheckPlotWhat(Plotting,Change);
        ChangeWhat(Oper);
        CASE Helpnr OF
           1:  ShowHelpTextWindow(HelpWindow);
               HelpText1(HelpWindow);
               HideMenu(HelpWindow)|
           2:  ShowHelpTextWindow(HelpWindow);
               HelpText2(HelpWindow);
               HideMenu(HelpWindow)|
           3:  ShowHelpTextWindow(HelpWindow);
               HelpText3(HelpWindow);
               HideMenu(HelpWindow)|
           4:  ShowHelpTextWindow(HelpWindow);
               HelpText4(HelpWindow);
               HideMenu(HelpWindow)|
           5:  ShowHelpTextWindow(HelpWindow);
               HelpText5(HelpWindow);
               HideMenu(HelpWindow)|
           6:  ShowHelpTextWindow(HelpWindow);
               HelpText6(HelpWindow);
               HideMenu(HelpWindow)|
           7:  ShowHelpTextWindow(HelpWindow);
               HelpText7(HelpWindow);
```

```
                HideMenu(HelpWindow)|
            8:  ShowHelpTextWindow(HelpWindow);
                HelpText8(HelpWindow);
                HideMenu(HelpWindow)|
            9:  ShowHelpTextWindow(HelpWindow);
                HelpText9(HelpWindow);
                HideMenu(HelpWindow)|
            10: ShowHelpTextWindow(HelpWindow);
                HelpText10(HelpWindow);
                HideMenu(HelpWindow)|
            11: ShowHelpTextWindow(HelpWindow);
                HelpText11(HelpWindow);
                HideMenu(HelpWindow)|
            12: ShowHelpTextWindow(HelpWindow);
                HelpText12(HelpWindow);
                HideMenu(HelpWindow)|
            13: ShowHelpTextWindow(HelpWindow);
                HelpText13(HelpWindow);
                HideMenu(HelpWindow)|
            14: ShowHelpTextWindow(HelpWindow);
                HelpText14(HelpWindow);
                HideMenu(HelpWindow)|
            15: ShowHelpTextWindow(HelpWindow);
                HelpText15(HelpWindow);
                HideMenu(HelpWindow)|
            16: ShowHelpTextWindow(HelpWindow);
                HelpText16(HelpWindow);
                HideMenu(HelpWindow);
        ELSE;
        END;
        IF Helpnr <> 5 THEN
          ChangeWhat(Plotting);
        END;
      END;
    END;
  END;
END HelpProc;


END HelpProcess.
```

```
DEFINITION MODULE TestGenerator;

EXPORT QUALIFIED TestManager;

PROCEDURE TestManager;

END TestGenerator.
```

```
IMPLEMENTATION MODULE TestGenerator;

FROM Kernel   IMPORT   SetPriority,WaitUntil,GetTime,IncTime,Time;

FROM Decl     IMPORT   PausTest,GetMotorData,MotorType,GetSample,
                       SetEnabled,SetBeep,PutSpeed,EndMeasure,GetTest,
                       StopMeasure,Error,ErrorType,MeasureType,
                       GetMeasWork,GetTStart;

FROM Random   IMPORT   RandomReal;

FROM MathLib IMPORT    round,float,sin;

 CONST              MakePointTime = 0.1;
                    PI            = 3.1416;
                    Dangle        = 0.6580;          (* radian *)

 VAR                t1,t2,t3,tdelta,
                    hTime,Speed,TStart : REAL;
                    t                  : Time;
                    MotorData          : MotorType;
                    h                  : INTEGER;
                    Test               : CARDINAL;
                    TestEnd            : BOOLEAN;
                    Work               : MeasureType;
                    Sequence           : ARRAY[1..10] OF INTEGER;


(*************************************************************************)

(*****************  initiation  procedure       **********************)


 PROCEDURE InitPar(Number : INTEGER);

 VAR            i : INTEGER;

 BEGIN
    GetTStart(TStart);
    hTime:=0.0;
    TestEnd:=FALSE;
    FOR i := 2 TO 10 DO
      Sequence[i]:=0;
    END;
    Sequence[1]:=1;
    GetMotorData(MotorData,Number);
    tdelta:=MotorData.RandomTime*RandomReal();
    t1:=MotorData.RampTime+TStart;
    IF Number<3 THEN
      t2:=tdelta+MotorData.WaitPaus+t1;
      t3:=MotorData.BreakTime+t2;
    ELSE
      t2:=MotorData.WaitPaus+t1;
      t3:=MotorData.BreakTime+t2;
    END;
    GetTime(t);
 END InitPar;


(*************************************************************************)

(*****************       PRBS    procedure       **********************)
```

```
PROCEDURE PRBSSequencer( VAR PRBSOut : INTEGER);

VAR          i,PRBS : INTEGER;

BEGIN
   FOR i := 0 TO 8 DO              (* makes sequence PRBS *)
     Sequence[10-i]:=Sequence[9-i];
   END;
   PRBS:=1;
   IF PRBS<>Sequence[3] THEN
     PRBS:=1;
   ELSE
     PRBS:=0;
   END;
   IF PRBS<>Sequence[10] THEN
     Sequence[1]:=1;
   ELSE
     Sequence[1]:=0;
   END;
   PRBSOut:=Sequence[10];
END PRBSSequencer;


(*****************************************************************)

(*****************    Ramping  procedure        *********************)


PROCEDURE Ramping(from,to : REAL; steps : INTEGER);

VAR          K              : INTEGER;
             Speed1,DSpeed : REAL;

BEGIN
   DSpeed:=to-from;
   FOR K:=1 TO steps DO
     WaitUntil(t);
     IncTime(t,100);
     Speed1:=from+DSpeed*(float(K)/float(steps));
     PutSpeed(Speed1,0.0,5,TestEnd);
   END;
   WaitUntil(t);
   IncTime(t,100);
   WaitUntil(t);
END Ramping;


PROCEDURE RampDown;

VAR          K      : INTEGER;
             Speed1 : REAL;

BEGIN
   FOR K:=1 TO 60 DO
     WaitUntil(t);
     IncTime(t,100);
     Speed1:=Speed*(1.0-float(K)/60.0);
     PutSpeed(Speed1,0.0,1,TestEnd);
   END;
   WaitUntil(t);
   IncTime(t,1000);
   WaitUntil(t);
END RampDown;
```

```
(*********************************************************************)

(*****************   testsequence procedure     *********************)


 PROCEDURE BeepTest;

 VAR           Beep : BOOLEAN;

 BEGIN
    InitPar(1);         (* initiates control signaldata *)
    Beep:=FALSE;
    WHILE NOT EndMeasure() AND NOT TestEnd DO  (* start of beeptest *)
      h:=round(MakePointTime*1000.0);
      WaitUntil(t);
      IncTime(t,h);
      hTime:=hTime+MakePointTime;
      WITH MotorData DO
        IF hTime<=TStart THEN
          Speed:=0.0
        ELSIF hTime<=t1 THEN
          Speed:=MaxSpeed*(hTime-TStart)/RampTime (* speed function *)
        ELSIF hTime<=t2 THEN
          Speed:=MaxSpeed;
          IF ((t2-hTime)<=BeepDelay) THEN         (* beep sound     *)
            IF NOT Beep THEN
              Beep:=TRUE;
              SetBeep(TRUE);
            END;
          END
        ELSIF (hTime<=t3) THEN
          Speed:=MaxSpeed*(t3-hTime)/BreakTime; (* speed function *)
          IF Beep THEN                          (* downramping    *)
            Beep:=FALSE;
            SetBeep(FALSE);
          END
        ELSE
          Speed:=0.0
        END;
      END;
      PutSpeed(Speed,hTime,1,TestEnd); (* sends speed to measure *)
    END;
    SetBeep(FALSE);        (* ramping function if the test *)
    IF (Work=Meas) THEN    (* was stopped before selected  *)
      IF Speed<>0.0 THEN
        RampDown;
      END;
    END;
    StopMeasure;
    SetEnabled(FALSE);
 END BeepTest;


 PROCEDURE RandomTest;

 VAR          RSpeed,OldSpeed,
              TestSpeed,hTime1 : REAL;
              RStart           : BOOLEAN;

 BEGIN
    InitPar(2);
    RStart:=TRUE;
```

```
OldSpeed:=MotorData.MaxSpeed;
WHILE NOT EndMeasure() AND RStart DO
  h:=round(MakePointTime*1000.0);
  WaitUntil(t);
  IncTime(t,h);
  hTime:=hTime+MakePointTime;
  WITH MotorData DO
    IF hTime<=TStart THEN
      Speed:=0.0
    ELSIF hTime<=t1 THEN
      Speed:=OldSpeed*(hTime-TStart)/RampTime
    ELSE
      RStart:=FALSE
    END;
    PutSpeed(Speed,hTime,2,TestEnd);
  END;
END;
t1:=MotorData.RampTime;
t2:=tdelta+MotorData.WaitPaus+t1;
t3:=MotorData.BreakTime+t2;
hTime1:=t2+t3;
WHILE NOT EndMeasure() AND NOT TestEnd DO
  h:=round(MakePointTime*1000.0);
  WaitUntil(t);
  IncTime(t,h);
  hTime:=hTime+MakePointTime;
  hTime1:=hTime1+MakePointTime;
  WITH MotorData DO
    IF hTime1>t3 THEN
      OldSpeed:=Speed;
      tdelta:=RandomTime*RandomReal();
      RSpeed:=RandomReal();
      IF RSpeed<0.5 THEN
        RSpeed:=-RandomSpeed
      ELSE
        RSpeed:=RandomSpeed
      END;
      TestSpeed:=MaxSpeed+RSpeed;
      IF TestSpeed>10.0 THEN
        RSpeed:=-RSpeed;
      END;
      IF TestSpeed<-10.0 THEN
        RSpeed:=-RSpeed;
      END;
      tdelta:=MotorData.RandomTime*RandomReal();
      hTime1:=0.0;
      t2:=WaitPaus+tdelta+t1;
      t3:=BreakTime+t2;
    END;
    IF hTime1<=WaitPaus THEN
      Speed:=OldSpeed
    ELSIF hTime1<=(t1+WaitPaus) THEN
      Speed:=OldSpeed-RSpeed*(hTime1-WaitPaus)/RampTime
    ELSIF hTime1<=t2 THEN
      Speed:=OldSpeed-RSpeed
    ELSIF hTime1<=t3 THEN
      Speed:=OldSpeed-RSpeed*(t3-hTime1)/BreakTime
    END;
  END;
  PutSpeed(Speed,hTime,2,TestEnd);
END;
IF (Work=Meas) THEN
  IF Speed<>0.0 THEN
    RampDown;
```

```
      END;
    END;
    StopMeasure;
    SetEnabled(FALSE);
END RandomTest;


PROCEDURE SinusTest;

VAR           Rest : BOOLEAN;
            hTime1 : REAL;

BEGIN
    InitPar(3);
    Rest:=TRUE;
    WHILE NOT EndMeasure() AND Rest DO
      h:=round(MakePointTime*1000.0);
      WaitUntil(t);
      IncTime(t,h);
      hTime:=hTime+MakePointTime;
      WITH MotorData DO
        IF hTime<=TStart THEN
          Speed:=0.0
        ELSIF hTime<=t1 THEN
          Speed:=MaxSpeed*(hTime-TStart)/RampTime;
        ELSIF hTime<=t2 THEN
          Speed:=MaxSpeed;
        ELSE
          Rest:=FALSE;
        END;
        PutSpeed(Speed,hTime,3,TestEnd);
      END;
    END;
    hTime1:=0.0;
    WHILE NOT EndMeasure() AND NOT TestEnd DO
      h:=round(MakePointTime*1000.0);
      WaitUntil(t);
      IncTime(t,h);
      WITH MotorData DO
        hTime:=hTime+MakePointTime;
        hTime1:=hTime1+MakePointTime;
        Speed:=MaxSpeed+RandomSpeed*sin(hTime1*2.0*PI/Period);
      END;
      PutSpeed(Speed,hTime,3,TestEnd);
    END;
    IF (Work=Meas) THEN
      IF Speed<>0.0 THEN
        RampDown;
      END;
    END;
    StopMeasure;
    SetEnabled(FALSE);
END SinusTest;


PROCEDURE PRBSSinusTest;

VAR           Rest                              : BOOLEAN;
              PRBSRandom,RoundTime,PRBSTime     : INTEGER;
              hTime1                            : REAL;

BEGIN
    InitPar(4);
    Rest:=TRUE;
```

```
    PRBSTime:=round(10.0*MotorData.PRBSPeriod);
    WHILE NOT EndMeasure() AND Rest DO
      h:=round(MakePointTime*1000.0);
      WaitUntil(t);
      IncTime(t,h);
      hTime:=hTime+MakePointTime;
      WITH MotorData DO
        IF hTime<=TStart THEN
          Speed:=0.0
        ELSIF hTime<=t1 THEN
          Speed:=MaxSpeed*(hTime-TStart)/RampTime;
        ELSIF hTime<=t2 THEN
          Speed:=MaxSpeed;
        ELSE
          Rest:=FALSE;
        END;
        PutSpeed(Speed,hTime,4,TestEnd);
      END;
    END;
    hTime1:=0.0;
    WHILE NOT EndMeasure() AND NOT TestEnd DO
      h:=round(MakePointTime*1000.0);
      WaitUntil(t);
      IncTime(t,h);
      WITH MotorData DO
        hTime:=hTime+MakePointTime;
        hTime1:=hTime1+MakePointTime;
        RoundTime:=round(10.0*hTime);
        Speed:=MaxSpeed+RandomSpeed*sin(hTime1*2.0*PI/Period);
        IF (RoundTime MOD PRBSTime) = 0 THEN
          PRBSSequencer(PRBSRandom);
          IF PRBSRandom=1 THEN
            IncTime(t,2*h);
            hTime1:=hTime1+Dangle*Period/2.0*PI;
          END;
        END;
      END;
      PutSpeed(Speed,hTime,4,TestEnd);
    END;
    IF (Work=Meas) THEN
      IF Speed<>0.0 THEN
        RampDown;
      END;
    END;
    StopMeasure;
    SetEnabled(FALSE);
END PRBSSinusTest;


PROCEDURE PRBSTest;

VAR           Rest,up                          : BOOLEAN;
              PRBSRandom,RoundTime,PRBSTime    : INTEGER;

BEGIN
    InitPar(5);
    Rest:=TRUE;
    PRBSTime:=round(10.0*MotorData.PRBSPeriod);
    WHILE NOT EndMeasure() AND Rest DO
      h:=round(MakePointTime*1000.0);
      WaitUntil(t);
      IncTime(t,h);
      hTime:=hTime+MakePointTime;
      WITH MotorData DO
```

```
        IF hTime<=TStart THEN
          Speed:=0.0
        ELSIF hTime<=t1 THEN
          Speed:=(RandomSpeed+MaxSpeed)*(hTime-TStart)/RampTime;
        ELSIF hTime<=t2 THEN
          Speed:=RandomSpeed+MaxSpeed;
        ELSE
          Rest:=FALSE;
        END;
        PutSpeed(Speed,hTime,5,TestEnd);
      END;
    END;
    up:=TRUE;
    WHILE NOT EndMeasure() AND NOT TestEnd DO
      h:=round(MakePointTime*1000.0);
      WaitUntil(t);
      IncTime(t,h);
      WITH MotorData DO
        hTime:=hTime+MakePointTime;
        RoundTime:=round(10.0*hTime);
        IF (RoundTime MOD PRBSTime) = 0 THEN
          PRBSSequencer(PRBSRandom);
          IF PRBSRandom=1 THEN
            IF NOT up THEN
              Ramping(-RandomSpeed+MaxSpeed,RandomSpeed+MaxSpeed,6);
              up:=TRUE;
            END;
            Speed:=RandomSpeed+MaxSpeed;
          ELSE
            IF up THEN
              Ramping(RandomSpeed+MaxSpeed,-RandomSpeed+MaxSpeed,6);
              up:=FALSE;
            END;
            Speed:=-RandomSpeed+MaxSpeed;
          END;
        END;
      END;
      PutSpeed(Speed,hTime,5,TestEnd);
    END;
    IF (Work=Meas) THEN
      IF Speed<>0.0 THEN
        RampDown;
      END;
    END;
    StopMeasure;
    SetEnabled(FALSE);
 END PRBSTest;


(*process*) PROCEDURE TestManager;

 BEGIN
    SetPriority(95);
    LOOP
      GetTest(Test);          (* selects and run test *)
      GetMeasWork(Work);      (* from the menu        *)
      IF Work<>OldData THEN
        CASE Test OF
          0 : PausTest|
          1 : BeepTest|
          2 : RandomTest|
          3 : SinusTest|
          4 : PRBSSinusTest|
          5 : PRBSTest;
```

```
        END;
      END;
    END;
  END TestManager;

END TestGenerator.
```

```
DEFINITION MODULE PlotProcess;

EXPORT QUALIFIED Plot;

 PROCEDURE Plot;

END PlotProcess.
```

```
IMPLEMENTATION MODULE PlotProcess;

FROM Kernel      IMPORT SetPriority,Wait,Signal;

FROM Storage     IMPORT ALLOCATE,DEALLOCATE;

FROM Graphics    IMPORT rectangle,VirtualScreen,SetViewPort,SetWindow,
                        SetLineColor,SetFillColor,PolyLine,point,color,
                        SetTextColor,handle,CharacterSize,FillRectangle,
                        DrawRectangle,WriteString,HideCursor,ShowCursor;

FROM Messages    IMPORT ReceiveMessage,AcceptMessage,SendMessage;

FROM Decl        IMPORT WhatType,PMailType,CheckPlotWhat,EBox,PBox,MBox,
                        Error,PlotWindowType,ABType,PlotKinds,GetCLDeg;

FROM MathLib     IMPORT exp,float,entier,ln;

FROM ConvReal    IMPORT RealToString;

CONST          MaxWindow = 12;
               UpScale   = 10;
               DownScale = 10;
               Delta     = 0.01;
               Size      = 95;

TYPE           PointerType = ARRAY[1..8] OF point;
               TextStr     = ARRAY[0..15] OF CHAR;

VAR            PLWindow           : ARRAY[1..MaxWindow] OF PlotWindowType;
               upA,downA          : ARRAY[1..2] OF INTEGER;
               Start,Lines,Win,
               Nr,NumWin,CLDeg    : INTEGER;
               PlotMess           : PMailType;
               NewPoint,OldPoint  : PointerType;
               WhatKind           : WhatType;
               x                  : REAL;
               Change,OK,Loss     : BOOLEAN;
               ColorStr           : ARRAY[0..5] OF color;
               TextWin            : PlotWindowType;


(*****************************************************************)

(*****************   grafical  procedure        *********************)


 PROCEDURE InitColor;

 BEGIN
    ColorStr[0]:=lightblue;
    ColorStr[1]:=lightred;
    ColorStr[2]:=green;
    ColorStr[3]:=brown;
    ColorStr[4]:=black;
    ColorStr[5]:=grey;
 END InitColor;


 PROCEDURE SetPoint(VAR p : point; x,y : REAL);

 BEGIN                        (* place plotpoint *)
    p.h:=x;
    p.v:=y;
```

```
END SetPoint;


PROCEDURE PlotLine(Window : PlotWindowType; p1,p2 : point);

VAR         Line : ARRAY[0..1] OF point;

BEGIN
   Line[0]:=p1;                (* makes line *)
   Line[1]:=p2;
   PolyLine(Window^.H,Line,2);
END PlotLine;


PROCEDURE SetRectangle(VAR R : rectangle ; XLo,YLo,XHi,YHi : REAL);

BEGIN
   R.xlo:=XLo;                    (* defines plotarea *)
   R.ylo:=YLo;
   R.xhi:=XHi;
   R.yhi:=YHi;
END SetRectangle;


PROCEDURE WriteReal(H:handle;X,Y,Val : REAL);

VAR            P : point;
             Str : ARRAY[1..8] OF CHAR;

BEGIN
   SetPoint(P,X,Y);
   IF ABS(Val)<0.0001 THEN
     Val:=0.0;
   END;
   RealToString(Val,Str,6);
   WriteString(H,P,Str);
END WriteReal;


PROCEDURE ShowText(H :handle; x,y,dx :REAL; str :TextStr;paint : color);

VAR Textpoint      : point;
    R              : rectangle;

BEGIN
   Textpoint.h:= x+0.3;   (* makes text in plotwindow *)
   Textpoint.v:= y+ 0.1;
   SetFillColor(H,paint);
   SetRectangle(R,x,y,x+dx,y+1.0);
   DrawRectangle(H,R);
   FillRectangle(H,R);
   WriteString(H,Textpoint,str);
END ShowText;


PROCEDURE InitPlotWindow(VAR Window : PlotWindowType;
       Xmin,Ymin,Xmax,Ymax,HLo,VLo,HHi,VHi : REAL; paint:color);

VAR       ViewPortRectangle : rectangle;

BEGIN
   NEW(Window);
   Window^.XMin:=Xmin;    (* initiate windowsize on the sceen   *)
   Window^.YMin:=Ymin;    (* and the size on the signals to be  *)
```

```
    Window^.XMax:=Xmax;    (* plotted                                    *)
    Window^.YMax:=Ymax;
    WITH  Window^ DO
     VirtualScreen(H);
     SetRectangle(ViewPortRectangle,HLo,VLo,HHi,VHi);
     SetViewPort(H,ViewPortRectangle);
     SetRectangle(WindowRectangle,Xmin,Ymin,Xmax,Ymax);
     SetWindow(H,WindowRectangle);
     SetLineColor(H,paint);
     IF paint=black THEN
       SetTextColor(H,black);
     ELSE
       SetTextColor(H,intensewhite);
     END;
     SetFillColor(H,intensewhite);
    END;
END InitPlotWindow;


PROCEDURE ClearWindow;

VAR           K : INTEGER;

BEGIN
    InitPlotWindow(PLWindow[1],-1.0,-1.0,1.0,1.0,0.6,0.0,1.5,1.0,lightblue);
    SetFillColor(PLWindow[1]^.H,black);
    FillRectangle(PLWindow[1]^.H,PLWindow[1]^.WindowRectangle);
    FOR K:= 1 TO NumWin DO
     DISPOSE(PLWindow[K]);           (* removes used windows *)
    END;
    IF NumWin>1 THEN
      DISPOSE(TextWin);
    END;
END ClearWindow;


(*********************************************************************)

(*****************   calculation  procedure     *********************)


PROCEDURE Lg(X : REAL):REAL;

BEGIN
    RETURN (ln(X)/10.0);          (* 10 logaritm *)
END Lg;


PROCEDURE ExpTen(X : REAL): REAL;

BEGIN
    RETURN (exp(X*ln(10.0)));
END ExpTen;


PROCEDURE ScalePoint(X : REAL;N : INTEGER): REAL;

VAR           Divisor : REAL;

BEGIN
    IF X<0.0 THEN
      Divisor:=ExpTen(float(entier(Lg(-X))+N-1));
      RETURN -(Divisor*float(entier((-X)/Divisor)));
    ELSIF X>0.0 THEN
```

```
      Divisor:=ExpTen(float(entier(Lg(X))+N-1));
      RETURN Divisor*float(entier((X)/Divisor));
    ELSE
      RETURN 0.0;
    END;
 END ScalePoint;


 PROCEDURE ComputeScalePoints(X,Y : REAL; VAR NewX,NewY : REAL);

 VAR      N : CARDINAL;

 BEGIN
    N:=1;
    REPEAT
      NewX:=ScalePoint(X,N);       (* rescale axelmarks *)
      NewY:=ScalePoint(Y,N);
      N:=N+1;
    UNTIL (NewX<>NewY) OR(N>4);
 END ComputeScalePoints;


(*****************************************************************)

(*****************    windowtext  procedure     *******************)


 PROCEDURE DataText;

 VAR           String : ARRAY [0..7] OF TextStr;
               i      : INTEGER;
               dx     : REAL;

 BEGIN
   InitPlotWindow(TextWin,0.0,0.0,9.0,21.0,0.6,0.0,1.5,0.84,lightblue);
   WITH TextWin^ DO
      String[0]:= "speed";          (* datawindow text *)
      String[1]:= "uref";
      String[2]:= "Fn1Force";
      String[3]:= "Fn2Force";
      String[4]:= "Fn3Force";
      String[5]:= "Moment";
      String[6]:= "FxForce";
      String[7]:= "FyForce";
      dx:=4.5;
      FOR i:= 0 TO 1 DO
         ShowText(H,4.5*float(i),20.0,dx,String[i],ColorStr[i]);
      END;
      dx:=3.0;
      FOR i:= 0 TO 2 DO
         ShowText(H,3.0*float(i),10.0,dx,String[i+2],ColorStr[i]);
         ShowText(H,3.0*float(i),0.0,dx,String[i+5],ColorStr[i]);
      END;
   END;
 END DataText;


 PROCEDURE EstimText1;

 VAR          String : ARRAY [0..8] OF TextStr;
              i      : INTEGER;
              dx     : REAL;

 BEGIN
```

```
   InitPlotWindow(TextWin,0.0,0.0,1.0,1.0,0.6,0.5,1.5,0.6,lightblue);
   WITH TextWin^ DO
      String[0]:= "loss funktion";
      ShowText(H,0.0,0.0,1.0,String[0],ColorStr[0]);
   END (*WITH*);              (* estimation  text    *)
END EstimText1;               (* for lossestimation *)


PROCEDURE EstimText2;

VAR          String : ARRAY [0..8] OF TextStr;
             i      : INTEGER;
             dx     : REAL;

BEGIN
   InitPlotWindow(TextWin,0.0,0.0,12.0,6.0,0.6,0.0,1.5,0.6,lightblue);
   WITH TextWin^ DO
      String[0]:= "param1 a0";     (* estimation text by *)
      String[1]:= "param1 a1";     (* parameterestimation *)
      String[2]:= "param1 a2";
      String[3]:= "param1 a3";
      String[4]:= "param1 b0";
      String[5]:= "param1 b1";
      String[6]:= "param1 b2";
      String[7]:= "param1 b3";
      dx:=12.0/float(CLDeg);
      FOR i:= 0 TO CLDeg DO
         ShowText(H,dx*float(i),5.0,dx,String[i],ColorStr[i]);
      END;
      FOR i:= 0 TO CLDeg DO
         ShowText(H,dx*float(i),0.0,dx,String[i+4],ColorStr[i]);
      END;
   END;
END EstimText2;


PROCEDURE TestText;

VAR          String : ARRAY [0..6] OF TextStr;
             dx     : REAL;

BEGIN
   InitPlotWindow(TextWin,0.0,0.0,1.0,1.0,0.6,0.5,1.5,0.6,lightblue);
   WITH TextWin^ DO
      String[0]:= "Control Signal";
      ShowText(H,0.0,0.0,1.0,String[0],ColorStr[0]);
   END;
END TestText;              (* control signal text *)


(*****************************************************************)

(*****************   plotaxes  procedure        *******************)


PROCEDURE Axes(Window : PlotWindowType;textmax : REAL);

VAR          Line   : ARRAY [0..1] OF point;
             Lo,Hi  : REAL;
             CW,CH  : REAL;                (* Character sizes    *)

BEGIN
   HideCursor;
   WITH Window^ DO
```

```
    CharacterSize(H, CH, CW);
    FillRectangle(H, WindowRectangle);       (* Clear window       *)
    DrawRectangle(H, WindowRectangle);       (* Draw rims          *)
    SetPoint(Line [0], 0.05, 0.0);
    SetPoint(Line [1], 1.0, 0.0);
    PolyLine(H, Line, 2);                    (* Draw X-axis        *)
    SetPoint(Line [0], 0.05, YMin);
    SetPoint(Line [1], 0.05, YMax);
    PolyLine(H, Line, 2);                    (* Draw Y-axis        *)
    ComputeScalePoints(YMin,YMax,Lo,Hi);     (* Draw Y-axis  Scale      *)
    SetPoint(Line[0],Lo,0.0);
    SetPoint(Line[1],Hi,0.0);
    WriteReal(H,0.1,0.93*YMin,Lo);           (* Lo *)
    WriteReal(H,0.1,textmax*YMax,Hi);        (* Hi *)
    END;
    ShowCursor;
END Axes;


(*****************************************************************)

(*****************   rescale   procedure        *********************)


PROCEDURE Ashow(A1,A2,A3,A4 : REAL;M,Lo,Hi,Ax : INTEGER);
(* Automatic scaling of plot axes *)

VAR          Change              : BOOLEAN;
             K,L                 : INTEGER;
             Ymax,Xmax,Xmin,max  : REAL;
             A                   : ARRAY[1..4]OF REAL;

BEGIN
  A[1]:=A1;              (* automatcal rescaling of the windows  *)
  A[2]:=A2;              (* signallimits after the size on A1..4 *)
  A[3]:=A3;
  A[4]:=A4;
  Ymax:=PLWindow[Lo]^.YMax;
  Change:=FALSE;
  max:=0.0;
  FOR K:=1 TO 4 DO
    IF (ABS(A[K])>max) THEN
      max:=ABS(A[K]);
    END;
  END;
  IF max>=0.8*Ymax THEN
     upA[M]:=upA[M]+1;
  END;
  IF max<=0.2*Ymax THEN
    downA[M]:=downA[M]+ 1;
  ELSIF downA[M]>0 THEN
    downA[M]:=downA[M]-1;
  END;
  IF upA[M]>=UpScale THEN
    Change:=TRUE;
    upA[M]:=0;
    WHILE Ymax<=max*1.5 DO
      Ymax:=Ymax*2.0;
    END;
  END;
  IF downA[M]>=DownScale THEN
    Change:=TRUE;
    downA[M]:=0;
    WHILE Ymax>=3.0*max DO
```

```
      Ymax:=Ymax*0.5;
    END;
  END;
  IF Change THEN
    FOR K:=Lo TO Hi DO
      IF Loss THEN
        PLWindow[K]^.YMin:=0.0;
      ELSE
        PLWindow[K]^.YMin:=-Ymax;
      END;
      PLWindow[K]^.YMax:=Ymax;
      WITH PLWindow[K]^ DO
       SetRectangle(WindowRectangle,XMin,YMin,XMax,YMax);
       SetWindow(H,WindowRectangle);
      END;
    END;
    IF Loss THEN
      PLWindow[Ax]^.YMin:=0.0;
    ELSE
      PLWindow[Ax]^.YMin:=-Ymax;
    END;
    PLWindow[Ax]^.YMax:=Ymax;
    WITH PLWindow[Ax]^ DO
      SetRectangle(WindowRectangle,XMin,YMin,XMax,YMax);
      SetWindow(H,WindowRectangle);
    END;
    x:=0.05;
    Nr:=0;
  END;
END Ashow;


(*************************************************************************)

(*****************   plotwindow  procedure      *********************)


PROCEDURE DataWindow;

VAR          K : INTEGER;

BEGIN
   FOR K:=1 TO 2 DO
     InitPlotWindow(PLWindow[K],0.0,-10.0,1.0,10.0,0.6,0.84,1.5,1.0,ColorStr
   END;
    FOR K:=1 TO 3 DO
     InitPlotWindow(PLWindow[K+2],0.0,-1048.0,1.0,1048.0,0.6,0.44,1.5,0.8,Co
     InitPlotWindow(PLWindow[K+5],0.0,-64.0,1.0,64.0,0.6,0.04,1.5,0.4,ColorS
   END;
   InitPlotWindow(PLWindow[9],0.0,-10.0,1.0,10.0,0.6,0.84,1.5,1.0,black);
   InitPlotWindow(PLWindow[10],0.0,-1048.0,1.0,1048.0,0.6,0.44,1.5,0.8,black
   InitPlotWindow(PLWindow[11],0.0,-64.0,1.0,64.0,0.6,0.04,1.5,0.4,black);
   Nr:=0;
   x:=0.05;          (* windowsize on data window *)
   NumWin:=11;
   DataText;
   upA[1]:=0;
   downA[1]:=0;
   upA[2]:=0;
   downA[2]:=0;
   Lines:=8;
   Start:=9;
   Win:=3;
   Loss:=FALSE;
```

```
END DataWindow;


PROCEDURE TestWindow;

VAR             K : INTEGER;

BEGIN
   InitPlotWindow(PLWindow[1],0.0,-10.0,1.0,10.0,0.6,0.6,1.5,1.0,ColorStr[0]
   InitPlotWindow(PLWindow[2],0.0,-10.0,1.0,10.0,0.6,0.6,1.5,1.0,black);
   Nr:=0;
   x:=0.05;
   NumWin:=2;      (* windowsize on control window *)
   TestText;
   Lines:=1;
   Start:=2;
   Win:=1;
   Loss:=FALSE;
END TestWindow;


PROCEDURE EstimWindow1;

VAR             K : INTEGER;

BEGIN
   InitPlotWindow(PLWindow[1],0.0,0.0,1.0,5.0,0.6,0.6,1.5,1.0,ColorStr[0]);
   InitPlotWindow(PLWindow[2],0.0,0.0,1.0,5.0,0.6,0.6,1.5,1.0,black);
   Nr:=0;
   x:=0.05;          (* windowsize on loss window *)
   NumWin:=2;
   EstimText1;
   Lines:=1;
   Start:=2;
   Win:=1;
   upA[1]:=0;
   downA[1]:=0;
   upA[2]:=0;
   downA[2]:=0;
   Loss:=TRUE;
END EstimWindow1;


PROCEDURE EstimWindow2;

VAR        K : INTEGER;

BEGIN
   GetCLDeg(CLDeg);
   FOR K:=1 TO CLDeg DO
     InitPlotWindow(PLWindow[K],0.0,-3.0,1.0,3.0,0.6,0.6,1.5,1.0,ColorStr[K-
     InitPlotWindow(PLWindow[K+CLDeg],0.0,-0.2,1.0,0.2,0.6,0.1,1.5,0.5,Color
   END;
   InitPlotWindow(PLWindow[2*CLDeg+1],0.0,-3.0,1.0,3.0,0.6,0.6,1.5,1.0,black
   InitPlotWindow(PLWindow[2*CLDeg+2],0.0,-0.2,1.0,0.2,0.6,0.1,1.5,0.5,black
   Nr:=0;
   x:=0.05;
   upA[1]:=0;            (* windowsize on parameter window *)
   downA[1]:=0;
   upA[2]:=0;
   downA[2]:=0;
   NumWin:=2*CLDeg+2;
   Lines:=2*CLDeg;
   Start:=2*CLDeg+1;
```

```
   Win:=2;
   Loss:=FALSE;
   EstimText2;
END EstimWindow2;


(*****************************************************************)

(*****************     plotdata  procedure      *******************)


PROCEDURE DMailHandler(VAR NewPoint : PointerType; VAR OK : BOOLEAN);

BEGIN
   OK:=TRUE;
   IF PlotMess^.Kind = Es THEN    (* reform measured data      *)
    SendMessage(EBox,PlotMess);   (* for plotting on the sceen *)
    OK:=FALSE;
   ELSE
     WITH PlotMess^ DO
       Ashow(PForce[1],PForce[2],PForce[3],PForce[1],1,3,5,10);
       Ashow(TForce[1],TForce[2],TForce[3],TForce[1],2,6,8,11);
       SetPoint(NewPoint[1],x,u);
       SetPoint(NewPoint[2],x,uref);
       SetPoint(NewPoint[3],x,PForce[1]);
       SetPoint(NewPoint[4],x,PForce[2]);
       SetPoint(NewPoint[5],x,PForce[3]);
       SetPoint(NewPoint[6],x,TForce[1]);
       SetPoint(NewPoint[7],x,TForce[2]);
       SetPoint(NewPoint[8],x,TForce[3]);
     END;
    SendMessage(MBox,PlotMess);
   END;
END DMailHandler;


PROCEDURE SMailHandler(VAR NewPoint : PointerType; VAR OK : BOOLEAN);

BEGIN
   OK:=TRUE;
   IF PlotMess^.Kind = Es THEN    (* reform the controlsignal  *)
    SendMessage(EBox,PlotMess);   (* for plotting on the sceen *)
    OK:=FALSE;
   ELSE
    WITH PlotMess^ DO
     SetPoint(NewPoint[1],x,uref);
    END;
    SendMessage(MBox,PlotMess);
   END;
END SMailHandler;


PROCEDURE EMailHandler(VAR NewPoint : PointerType; VAR OK : BOOLEAN);

 VAR              K           : INTEGER;
                  ListA,ListB : ARRAY [1..4] OF REAL;


BEGIN
   OK:=TRUE;                        (* reform estimationdata for *)
   IF PlotMess^.Kind = Re THEN      (* plotting on the sceen     *)
    SendMessage(MBox,PlotMess);
    OK:=FALSE;
   ELSE
```

```
      WITH PlotMess^ DO
        IF Ready THEN
          IF ExChange OR Change THEN
            IF NOT Start AND NOT Change THEN
              ClearWindow;
            END;
            EstimWindow2;
          END;
          FOR K:=1 TO CLDeg DO
            ListA[K]:=A[K];
            ListB[K]:=B[K];
          END;
          IF CLDeg<4 THEN
            FOR K:=CLDeg+1 TO 4 DO
              ListA[K]:=A[1];
              ListB[K]:=B[1];
            END;
          END;
          Ashow(ListA[1],ListA[2],ListA[3],ListA[4],1,1,CLDeg,2*CLDeg+1);
          Ashow(ListB[1],ListB[2],ListB[3],ListB[4],2,CLDeg+1,2*CLDeg,2*CLDeg
          FOR K:= 1 TO CLDeg DO
            SetPoint(NewPoint[K],x,A[K]);
            SetPoint(NewPoint[CLDeg+K],x,B[K]);
          END;
        ELSE
          IF Start THEN
            EstimWindow1;
          END;
          Ashow(V,V,V,V,1,1,2,2);
          SetPoint(NewPoint[1],x,V);
        END;
      END;
      SendMessage(EBox,PlotMess);
    END;
  END EMailHandler;


(*********************************************************************)

(*****************   windowplot  procedure      *********************)


  PROCEDURE PlotCurve;

  VAR         K : INTEGER;

  BEGIN
     IF (Nr = 0) THEN                    (* Makes a x vs time plot *)
       IF Win=3 THEN
         Axes(PLWindow[Start],0.6);
         FOR K :=1 TO (Win-1) DO
           Axes(PLWindow[K+Start],0.8);
         END;
       ELSE
         FOR K :=0 TO (Win-1) DO
           Axes(PLWindow[K+Start],0.8);
         END;
       END;
       x:=x+Delta;              (* plots the values in a timediagram *)
       Nr:=Nr+1;
       FOR K:=1 TO Lines DO
         OldPoint[K]:=NewPoint[K];
       END;
     ELSE
```

```
      FOR K:= 1 TO Lines DO
        HideCursor;
        PlotLine(PLWindow[K],OldPoint[K],NewPoint[K]);
        ShowCursor;
        OldPoint[K]:=NewPoint[K];
      END;
      x:=x+Delta;
      Nr:=Nr+1;
      IF (Nr>=Size) THEN
        x:=0.05;
        Nr:=0;
      END;
    END;
 END PlotCurve;


(*Process*) PROCEDURE Plot;

 BEGIN
    SetPriority(98);
    NumWin:=1;
    InitColor;
    LOOP;
      ReceiveMessage(PBox,PlotMess);
      CheckPlotWhat(WhatKind,Change);
      IF WhatKind=Oper THEN
        CASE PlotMess^.Kind OF
          Re : SendMessage(MBox,PlotMess)|
          Es : SendMessage(EBox,PlotMess);
        END;
      END;
      IF Change THEN                (* selects plotobjekt  *)
        CASE WhatKind OF
          Data : ClearWindow; DataWindow|
          Test : ClearWindow; TestWindow|
           Par : ClearWindow;|
          Oper : ClearWindow;
        END;
      END;
      IF WhatKind = Data THEN
        DMailHandler(NewPoint,OK);        (* data value plotting *)
        IF OK THEN
         PlotCurve;
        END;
      ELSIF WhatKind = Test THEN          (* control plotting *)
        SMailHandler(NewPoint,OK);
        IF OK THEN
          PlotCurve;
        END;
      ELSIF WhatKind = Par THEN           (* parameter plotting *)
        EMailHandler(NewPoint,OK);
        IF OK THEN
          PlotCurve;
        END;
      END;
    END;
 END Plot;
END PlotProcess.
```