Molecular Dynamics Simulation of a Lipid Bilayer with Dissolved Solutes

Amy Elizabeth Aussiker
Senior Thesis
Department of Chemistry
May, 1991

Molecular Dynamics Simulation of a Lipid Bilayer with Dissolved Solutes

Amy Elizabeth Aussiker
Senior Thesis
Department of Chemistry
May, 1991

Molecular Dynamics Simulation of a Lipid Bilayer with Dissolved Solutes

By

Amy Elizabeth Aussiker

Abstract of "Molecular Dynamics Simulation of a Lipid Bilayer with Dissolved Solutes" by Amy E. Aussiker.

This paper considers a computer simulation of a lipid bilayer with solute particles dissolved between the monolayers. The objective is to examine the bilayer system and to determine what effect, if any, the solutes have on the bilayer. Specifically, I consider the radial distribution function of the layers composing the bilayer, the z displacement of the solutes, and the bilayer width. It is expected that the bilayer width will increase as the bilayer swells to include the additional particles, and this is indeed observed. Results for the neat bilayer show, by comparison of a previous simulation by Berendsen et. al.[1], that the freely jointed chain model is a reasonable representation of the lipid molecules composing the monolayers. The simulations including solutes clearly demonstrate the presence of a solute layer which behaves as a two dimensional liquid.

Acknowledgements

I wish to thank both Washington and Lee University and the Thomas F. Jeffress and Kate Miller Jeffress Memorial Trust for their support which funded my project. I would like to thank my family for their support and for the many times they pretended to understand my project. I would also like to thank Dr. Steven Desjardins for all of his help and guidance throughout the past three years, both as a professor and a friend.

Table of Contents

I. Introduction

Computers are being used more and more frequently today in science and technology. They are used in research and development, in chemical analyses, and in other fields too numerous to mention. One use in particular is computer simulation projects. Here, instead of an actual model of a system or prototype being made, the system to be studied is first simulated on a computer and tests are run to ensure that the model is viable. After tests are run, the simulation from the computer can then be used as a blueprint for the actual working model.

Computer simulations are very useful. They can be used to test different theories without the expense of having to build many different prototypes or run lengthy experiments. This latter case is a major concern in the scientific world. Many of the hypotheses tested in science are on the molecular level and are difficult if not impossible to test in the laboratory. Even if lab testing were an option, it would be very difficult (close to impossible) to devise different tests that would examine certain aspects of a system while still having all other variables remain constant. Using a computer solves these problems by allowing visualization of the molecular interactions which one cannot see with the naked eye. The only way to visualize these types of systems experimentally is to use invasive procedures which modifies system behavior. Two such technique are X-ray crystallography and electron microscopy. From these procedures we can see molecules in different stages of interaction. However, we cannot see a continuous flow of action (which we can with the computer) and we do not know for certain if the

techniques used to prepare the molecules for analysis have changed the molecules. With a computer, the molecules are modeled to be in what we believe to be their naturally occurring states, which is based upon experimental data. We can observe particle interactions on a computer without having to stop the molecules at each stage. We can even observe the interactions at consecutive time steps or at different time scales difficult to achieve in a laboratory. This gives the computer observer a great advantage of being able to observe all the infinitesimal changes a system undergoes without having to destroy the molecules in the process.

Computer simulations also allow testing of different aspects of a system with just one simulation, while separate experiments may be needed in a laboratory setting. All one has to do to test a different aspect of a computer model is to add a couple of lines to the computer code which will solve for the new variable. This is much simpler than a laboratory experiment because we are ensured of having otherwise duplicate experimental conditions for each observation. This allows us to compare changes in a single property without having to take into consideration different testing conditions. Another useful feature of computer simulations is the ability to analyze interactions with much less expense than a lab experiment. In a lab experiment one must have all of the necessary equipment and chemicals. Since each experiment can only test one aspect of the system, many different experiments must be performed. With a computer simulation, one only needs the computer and the software (all of which can be used for other simulations, unlike used chemicals) and of course the (wo)manpower and devotion required for any type of experiment, computer or otherwise.

There are, of course, some drawbacks to using a computer simulation. One must have some experimental data in order to construct, i.e. parameterize, a viable model. In most cases, characteristics of the species to be studied and their interactions are available or can be estimated. Even with the availability of sufficiently detailed data, the devising of a reasonable model is often the most difficult part of the computer simulation approach. Testing of the program is usually accomplished by comparison with related simulations available in the literature (in our case, a comparison of a simulation of a 32 particle bilayer by Berendsen et. al.[1]). Another drawback is the availability of adequate computer resources. Computer time is expensive and, in a main frame setting, time sharing makes it difficult to find time during which to work on or run a lengthy simulation. Despite these difficulties, however, computer simulations remain an excellent way to view and study molecular interactions.

Exactly what is a computer simulation? Simply, it is a series of consecutive configurations (states) of the system. There are different ways to generate configurations. One way is the Monte Carlo method, so named because it is based on a matter of chance like a roll of the dice. Here the next configuration of the particles in the system is chosen randomly. The thermal probability, i.e. the Boltzman factor for the energy change, is determined for this trial configuration which, after a comparison with a random number, is either accepted or rejected. This calculation takes a relatively short period of time because only the value of the configurational energy is required. The drawback is that, being a statistical method, many configurations (10^6) are needed for adequate sampling.

An alternative method is the Molecular Dynamics simulation. In this method, Newton's Laws of Motion are used to calculate the position and velocity of each particle over a ceratin time period. Since integration of the classical equations of motion requires knowledge of forces, which are vector quantities, the generation of a new configuration takes more time than in the Monte Carlo method. Since, however, all configurations are then accepted in the Molecular Dynamics approach, far less configurations are needed (10^4) to give an adequate representation of system behavior. In addition, Molecular Dynamics allows the calculation of time dependent properties, a feature totally absent from its canonically sampled counterpart.

Our simulation uses the Molecular Dynamics method. In this particular simulation we are simulating a lipid bilayer with solute particles dissolved between the monolayers (see Fig. 1). A lipid bilayer is best known as the structural basis of cell membranes. It is composed of fatty acid chains which have a hydrophilic (water-loving) head group and a hydrophobic (water-hating) tail. Since the lipid molecules are in an aqueous environment under physiological conditions, the hydrophilic head groups form the surface of the bilayer (facing the aqueous layer) while the hydrophobic tails face the inside. This configuration resembles a sandwich with the head groups analogous to the bread and the tails analogous to the meat of the sandwich. The solute particles are the interesting part of this simulation. Being non-polar, the solutes dissolve in the interior (hydrophobic) section of the bilayer. The behavior of the solute particles can be characterized by a lateral diffusion in the x,y plane and a transverse swelling of the bilayer in the z direction. Swelling would occur to accommodate the added particles, just as a sandwich would swell

when more meat is added.

The lipid bilayer is an interesting object of study because of the important role of bilayer dynamics in living systems. One possible ramification of bilayer modification due to the presence of solutes is the cellular response to anesthesia. When a person is administered certain anesthetics, the anesthesia molecules enter the cell membranes of the nerves. These cell membranes contain many protein channels through which ions can pass, causing the change in the potential gradient along the cell which is associated with the "firing" of the nerve. The anesthetic particles travel through this lipid bilayer until they diffuse out. However, while the particles are in the bilayer, they cause the bilayer to swell. This swelling of the membrane causes the protein channels to close, disrupting the active transport of ions, and thus impeding proper nerve function. This causes a lack of sensation, i.e. the anesthetic effect. My experiment can be used to simulate the membrane swelling due to many types of anesthetics by simply altering the size, mass, and number of solute particles.

From this one computer simulation, many different aspects of the system can be calculated and studied. These include diffusion constants, amount of swelling in the bilayer, as well as the essentially two dimensional structure of the various layers. In addition this simulation is able to observe the movement of the particles through the bilayer without any invasive techniques. This experiment would be very difficult to do in the lab. First of all, it would be impossible to observe the movement of the anesthetic particles through the membrane. It would also require extensive experiments with live subjects that would have to undergo invasive techniques. However, one computer run

can be substituted for the many experiments and would (1) be able to have variables held constant (impossible to do in a lab setting) and (2) not use live subjects (which would be necessary in lab experiments).

There are drawbacks to using a computer simulation, however. First, nothing compares to using an actual living system. There can always be a small fluctuation that occurs in a living system that we either do not know about or cannot simulate. Also, there are a few simplifications that are made. For one, we are using freely jointed hydrocarbon chains instead of imposing gauche-trans interactions and angular constraints. However, this can be rectified by modifying the computer code. It also has the drawback of being a fairly extensive program which requires a long period of time for one simulation on the present type of computer. This could be shortened by running the program on a larger and faster computer such as a CRAY. However, the time spent running the program is still much less than the time that would be spent to experimentally test all the aspects that the computer program tests.

We were finally able to write and run a simulation of this system that, for all indications, works properly. This took a lot of time and often was a process of trial and error. In the following pages, the process by which the program was written and how it works will be discussed in great detail. The results of the final program will then be discussed including the methods of analysis. Here figures and tables summarizing my findings will be presented.

II. Methods and Model

Our system uses two layers, each of 16 decanoate chains, to create a bilayer. The chains consist of different types of particles including head groups (COOH), methylene (CH_2) groups, and methyl (CH_3) groups for the end particles of the tails. Each CH_3 and CH_2 groups are considered as having one center; that is, each group is represented as a sphere with the hydrogens included in the carbons' electronic cloud. Given the natural geometry of the bilayer, it is simplest to work in Cartesian coordinates. Movement occurs in all three directions; however, motion in the z direction (perpendicular to the plane of the bilayer) is obviously expected to be different from motion in the x,y plane.

The particles of the system use the Lennard-Jones potential for intermolecular interactions[2]. The form is:

$$v(r_{ij}) = 4\varepsilon[(\sigma/r_{ij})^{12} - (\sigma/r_{ij})^{6}]$$

where σ and ε are .392 nm and 124 K respectively (see Fig. 2). There are two extra forces applied in the z direction. The first is a very stiff (k, the force constant = 8.5 N) harmonic potential applied only to the head groups, which allows them to oscillate about their average z position. This force mimics the hydrophobic interaction with the aqueous environment maintaining head group plane integrity in the absence of an explicit aqueous solvent. The second force mimics the pressure applied by the aqueous solvent by applying a constant force equivalent to one atmosphere on each head group in the z direction. There is also a restoring force that is applied to inner particles if they should try to escape beyond the plane of the head groups. This keeps the tails of the chains or

the solutes from escaping and also mimics the hydrophobic interaction that they would naturally experience if outside a bilayer membrane in a biological system.

There are six versions of the simulation that differ only in the number (0,2,4,8,12, or 16) of solute particles that are dissolved within the bilayer. The solute particles have a mass of 20 and are similar to Argon atoms. The Lennard-Jones interaction is truncated at 2.5 σ, since the potential has become negligible at that distance. However, no cut-off is applied to interactions between particles in separate monolayers. This is important since the net effect of many small but orientationally coherent interactions is significant. The box size in the x,y plane is 15.4 Angstroms with periodic boundary conditions applied only in the x and y directions.

Our molecular dynamics simulation uses the velocity form of the Verlet algorithm[3] to solve for the positions and velocities of the particles over time. The form of this algorithm is

$$r_{n+1} = r_n + v_n\Delta t + 1/2\, a_n(\Delta t)^2$$

$$v_{n+1} = v_n + 1/2(a_{n+1} + a_n)\Delta t$$

where r is the position, t is time, a is acceleration, v is the velocity, and m is the mass. The program also uses a modified version of RATTLE[4], which imposes constraints on the distance between adjacent particles in the chains (i.e. fixes the bond lengths at 1.54 Angstroms). This routine allows the particles to first move without constraints. After the initial movement, the particles are brought back into positions which satisfy the bond length constraints. There are other routines in the program which impose restrictions on the system. One such routine fixed the temperature around 310 K, which is normal body

temperature. This is accomplished through scaling the velocities of the particles to maintain a kinetic energy value around the desired temperature. This restriction is used only during equilibration. When data is collected, the temperature restriction is removed so that the kinetic energy can fluctuate and velocity dependent functions can be calculated.

We allowed the system to equilibrate for 10,000 time steps or 12 ps (each time step is .0012 ps). The molecular dynamics run continued for 2,000 time steps or 2.4 ps during which data was gathered.

The program creates many different data files. This outpouring of data results from the different aspects of the system we are studying. One such file is the energy file. This file consists of the kinetic, potential, total energy, and the bilayer width of the system over the period of the computer run. We then plot this data over time to see (1) if there is conservation of energy, (2) to visualize how the energies and bilayer width are fluctuating over time, and (3) to obtain an average of the fluctuations in the system. We also check for conservation of energy by looking for any overall drift in the energy over time. This is our primary check to see if the system is working properly. Since our system behaves as a microcanonical ensemble, the total energy should remain constant (i.e. no drift) with only small fluctuations due to round-off error. The average fluctuations which are calculated are another check to make sure the system is behaving properly. We would expect a certain amount of random error (due to round off) in the total energy and a larger fluctuation in temperature (average kinetic energy) which is expected in the microcanonical ensemble. We have seen about .037 fluctuations which is

reasonable given the standard $1/(n)^{1/2}$ estimate[5] for a system of n=320.

The radial distribution function[5], g(r), can also be calculated from the simulation data. This analysis determines the structure of the system based on the probability of finding a particle a certain distance from a certain tagged particle. One particle is selected (tagged) and all particles located within a shell of width dr a distance r away are counted. This allows us to see how the particles in the head groups are distributed. The formula used is:

$$g(r) = \sum_{r1<r<r2} [4\pi r^2 \rho(2\Delta r)]^{-1}.$$

The velocities and positions of the particles are also saved over time. From this file, the diffusion constant[6] for the solute particles and the head groups can be calculated. When there is no net force on the system, the particles should move linearly with no net displacement. However this system includes interactions between particles so the mean square displacement is calculated using:

$$R(t)^2 = \langle |r_i(t_2) - r_i(t_i)|^2 \rangle$$

$$R(t)^2 = 2dDt$$

where D is the diffusion constant, $R(t)^2$ is the mean square displacement, t is time, and d is the dimensionality of the system. This formula looks at how the density of the bilayer changes over time due to solutes moving. From this average flux, the diffusion constants for the solutes and the head groups can be determined.

The velocities of the particles that are saved can be used to determine the velocity autocorrelation function[6]. In the absence of forces on a particle, its velocity through time

will remain constant. However, as a result of particle interactions, the correlation between the velocities at two different times will decrease. We analyze for the amount of correlation between the velocities over time. As the system reaches equilibrium, the correlation should become closer and the graph should diminish to zero. In other words, this function examines how long it takes for the particle velocities in the head group and solute layers to become uncorrelated. The formula used to calculate this is:

$$z(t) = \langle v_i(t_2) - v_i(t_1) \rangle$$

where z(t) is the velocity autocorrelation function, v is the velocity, and t is time. The velocity autocorrelation functions for both the solutes and the head groups are calculated. In this way we can discover if the solutes cause extra drag on the head groups because of solute interaction with the hydrocarbon chains.

Other data files are generated by the system which are not used in analysis. Of these are the two start-up files which contain the positions and velocities of the particles. Using these files, which are updated to current positions and velocities every twenty five time steps, the program can be stopped and restarted without having to start from the very beginning again. This is very useful if the program takes a long time to run and other people are using the computer.

There are many ways in which to check to make sure the program is running properly and behaving in the correct way for the system under study. One is the use of the energy plot discussed previously. Another is the graphical display of the bilayer and solute particles moving over time allows us to check for any unusual movements that might not occur in the real system. One such occurrence is the escape of one of the

solute particles. This would happen if there was a hole in the bilayer or if the particle was moved outside of the bilayer in between time steps so the constraints of the program could not be imposed to keep the particle within the bilayer. The radial distribution function, velocity autocorrelation, and diffusion constant plots also allow us to analyze the behavior of our system over time and compare them to experimental data. A good indication that the computer is running properly is if the computer generated plots correspond closely to the experimental data.

The simulations were performed using a Dell Computer System 325 which is a 386 microprocessor at Washington and Lee University, Department of Chemistry.

III. Results and Discussion

As mentioned above, six different bilayer simulations were performed, differing only in the number of solute particles. The energetics for the simulations are seen in Figs. 3A-F. As we can see, the fluctuations in total energy are small and there is no base line drift, indicating a correctly functioning simulation. As can be seen in Table 1, the temperature fluctuations were also within acceptable range.

The solutes did have the expected effect on the bilayer width. Fig.4 shows the linear dependence of the swelling of the bilayer on the number of solutes. It is interesting to consider the bilayer width for a sixteen solute system plotted versus time (Fig. 5). The bilayer experiences relatively large swells in width which could be due to the presence of the solutes. Also from Fig.6 we see there is a gradual increase of width fluctuations as solute number increases. Thus, in addition to increasing the width of the bilayer, the solutes also cause an increase in the average width fluctuation.

As depicted in Fig. 7, the decanoate molecules that compose the two monolayers tilt away from the norm of the head group plane. As seen in Table 2 the average tilt remains constant, with small fluctuations (Fig. 8). It is assumed that a concerted rotation about the norm is possible but that the time scale for this simulation was simply too short to observe such an effect. We note that this tilt has been previously observed in other simulations and experiments[1]. The solutes apparently had no effect on the tilt.

The radial distribution function (g(r)) for the head group plane can be seen in Figs. 9A-G and 10A-G. As expected, these results indicate that the structure of the head

group is that of a two dimensional liquid. Again, these results are in agreement with the neat bilayer simulation of Berendsen[1]. There was no obvious correlation between these results and different solute number, so that we assume the observed differences in peak heights (the four solute case is notable) are purely statistical and that longer runs would diminish these differences.

The root mean square z displacement of the solutes was also calculated. As seen in Fig. 11, the movement in the z direction is small but does increase with an increasing number of solute particles. This is most likely due to the increased crowding in the solute layer. Plots of the z displacement for each of the simulations over time are shown in Figs. 12A-E. One can see that the values around which the z displacement fluctuates does increase with the number of solutes which we would expect from Fig. 11. However it is also interesting to note that the overall range (y-axis i.e. 0-.06 or .005-.045 for example) through which the z displacement fluctuates decreases with the addition of solutes. Thus, although the average z displacement increases, the fluctuations about this average decrease. This seems to indicate that the solutes increasingly act as a coordinated body and that the solute-solute interactions restrict deviations from a two dimensional liquid structure.

An interesting fact can be drawn at this point in the discussion. From the bilayer width data and the z displacement data, the solutes appear to behave as a two dimensional liquid. The solutes interact within themselves while being solvated above and below by the monolayers. A result of this is that the linear progression seen in Fig. 4 will probably not continue once the two dimensional solute liquid becomes saturated. At

this point a second two dimensional liquid would form in between the monolayers. The two solute liquids would interact with themselves and each other. This would cause a large increase in the bilayer width (i.e. the linear progression will be destroyed) as the bilayer swells to accommodate the second liquid. The additional liquid would also cause a reduction of the z displacement of the solutes. This reduction would be caused by the presence of additional particles making it more difficult for a solute to move in the z direction.

The velocity autocorrelation function was computed and plotted for the two head group planes and the solutes. Figs. 13A-F, 14A-F, and 15A-E show the plots for the first fifty time steps of the simulations. As can be seen, the solute velocity autocorrelation function will diminish faster than the top or bottom head group plots. This could be due to the fact that the head groups are much larger and therefore have a greater momentum. This fact and the fact that the head groups have hydrocarbon tails attached that increase the momentum and also produce a drag on the head groups will cause the head groups to take longer to slow down and begin to move in the opposite direction. This trend was seen to continue upon examination of the velocity autocorrelation function over fifteen hundred time steps (Figs. 16A-F(a), 17A-F(a), and 18A-E(a)). The plots for the respective groups are very similar so there does not seem to be any effect caused by the addition of solutes.

The self diffusion constants for the solutes and the two head groups were also determined. In Fig. 19 the self diffusion constants for the groups are plotted against the number of solutes. There does not seem to be any trend that can be attributed to the

addition of solutes to the bilayer. The self diffusion constant was calculated by integration and the integrated values were plotted over time (Figs. 16A-F(b), 17A-F(b), and 18A-E(b)). The fluctuations do not indicate any solute effect.

IV. Conclusions

The most obvious result of our simulations is the layered character of bilayer solvation. We have clearly seen that a collection of solute particles in the interlayer region acts like a two dimensional liquid. The only changes in the bilayer itself due to the presence of solutes seems to be in the z displacement and the bilayer width. The addition of solutes to a lipid bilayer is a common occurrence in nature and if the bilayer was drastically changed, our cellular structure would also change. The only changes were the swelling of the bilayer, which is the expected change due to the steric effects of the solutes, and the increase in the width fluctuations which can be attributed to the crowding effect. Overall, these results indicate the viability of the Molecular Dynamics method as a means of investigation for membrane dynamics.

REFERENCES

1. P. van der Ploeg and H.J.C. Berendsen, J. Chem. Phys., **76** (1982) 3271.

2. J.P. Ryckaert, G. Ciccotti, and H.J.C. Berendsen, J. Comput. Phys., **23** (1977) 327.

3. W.C. Swope, H.C. Anderson, P.H. Berens, and K.R. Wilson, J. Chem. Phys., **76** (1982) 637.

4. H.C. Anderson, J. Chem. Phys., **52** (1983) 24.

5. D. Chandler, Introduction to Modern Statistical Mechanics (Oxford University Press, NY, 1987).

6. H. Gould and J. Tobochnik, An Introduction to Computer Simulation Methods, Applications to Physical Systems (Addison-Wesley Publishing Co., NY, 1988).

Appendix A.

FORTRAN Code for Lipid Bilayer Simulation

```
*************************************************************************
*                                                                       *
*                    FORTRAN Code for Lipid Bilayer with                *
*                  Sixteen Solute Particles Dissolved Within            *
*                                                                       *
*************************************************************************
*
*                                        main routine
      include 'fgraph.fi'
      implicit real*8 (a-h,o-z)
      include 'fgraph.fd'
      parameter(n=32,na=10,np=n*na,nb=na-1,ns=16)
      common /coord/ rx(n,na),ry(n,na),rz(n,na)
      common /speed/ vx(n,na),vy(n,na),vz(n,na)
      common /farce/ fx(n,na),fy(n,na),fz(n,na)
      common /scoord/ sx(ns),sy(ns),sz(ns)
      common /sspeed/svx(ns),svy(ns),svz(ns)
      common /sfarce/ sfx(ns),sfy(ns),sfz(ns)
      common /bonds/ dsq(na),xm(na)
      common /cnsts/ tol,bl,maxit,pe,ek,nsnap,itime,dt,vc,w,
     +                sm,zt,zb
      common /grval/ gtcum(2000),gbcum(2000)
      dimension angx(n),angy(n)
      open(unit=12,file='biwide.dat',status='old')
      open(unit=5,file='ssbene16.dat',status='old')
      open(unit=9,file='vels16.dat',status='old')
      open(unit=10,file='velthg16.dat',status='old')
      open(unit=11,file='velbhg16.dat',status='old')
      tol=1.0e-5
      bl=1.54/15.4
      do 111 i=1,nb
         dsq(i)=bl**2
111   continue
      maxit=1000
      eps=124
      sig=3.42/15.4
      ch2m=14
*      dt=0.002*sig*dsqrt(ch2m/eps)
      dt=.001
      dtsq=dt**2
      tol2=2*tol
      xm(1)=44
      xm(2)=14
      xm(3)=14
      xm(4)=14
      xm(5)=14
      xm(6)=14
      xm(7)=14
      xm(8)=14
      xm(9)=14
      xm(10)=15
      sm=20
```

```
      nsnap=10
      call grafinit
      open (unit=2,file='startr16.dat',status='old')
      open (unit=3,file='startv16.dat',status='old')
      do 10 i=1,n
         do 15 j=1,na
            read(2,*)itime,rx(i,j),ry(i,j),rz(i,j)
            read(3,*)itime,vx(i,j),vy(i,j),vz(i,j)
15       continue
10     continue
      do 16 i=1,ns
         read(2,*)itime, sx(i),sy(i),sz(i)
         read(3,*)itime,svx(i),svy(i),svz(i)
16     continue
       close(2,status='keep')
       close(3,status='keep')
       open (unit=4,file='gc16.dat',status='old')
          do 35 i=1,2000
             read(4,*)thyme,acont,gtcum(i),gbcum(i)
35       continue
       close(4,status='keep')
       call flj
       call sflj
      ek=310
       do 999 imd=1,100000
          do 100 isnap=1,nsnap
*            call thermostat
            call movea
            call moves1
            call flj
            call sflj
            call moveb
            call moves2
            itime=itime+1
            call outpt
            call snapshot
            open (unit=8,file='tilt16.dat',status='old')
            do 45 i=1,n
              xdisp=abs(rx(i,1)-rx(i,10))
              ydisp=abs(ry(i,1)-ry(i,10))
              zdisp=abs(rz(i,1)-rz(i,10))
              angx(i)=angx(i)+datan(xdisp/zdisp)
              angy(i)=angy(i)+datan(ydisp/zdisp)
              write(8,*) itime,angx(i),angy(i)
45          continue
            close(8,status='keep')
            do 30 js=1,ns
               write(9,13) itime,svx(js),svy(js),svz(js)
               write(9,13) itime,sx(js),sy(js),sz(js)
30          continue
            do 40 js=1,n
            if (js.le.n/2) then
```

```
               write(10,13) itime,vx(js,1),vy(js,1),vz(js,1)
               write(10,13) itime,rx(js,1),ry(js,1),rz(js,1)
            else
               write(11,13) itime,vx(js,1),vy(js,1),vz(js,1)
               write(11,13) itime,rx(js,1),ry(js,1),rz(js,1)
            end if
40          continue
100      continue
         if (mod(itime,25).eq.0) then
               call savecon
         end if
         if(mod(itime,100).eq.0) then
               open(unit=4,file='gc16.dat',status='old')
               do 37 i=1,2000
                  write(4,*)itime,i,gtcum(i),gbcum(i)
37             continue
               close(4,status='keep')
          end if
999   continue
13    format(1x,i6,2x,3(f15.9,2x))
      close(12,status='keep')
      close(5,status='keep')
      close(9,status='keep')
      close(10,status='keep')
      close(11,status='keep')
      end
*
*                                          Verlet and RATTLE algorithms
*
* Part 1 of RATTLE (update positions)
*
      subroutine movea
      implicit real*8(a-h,o-z)
      parameter (rptol=1.0e-6)
      parameter (n=32,na=10,np=n*na,nb=na-1,ns=16)
      common /coord/ rx(n,na),ry(n,na),rz(n,na)
      common /speed/ vx(n,na),vy(n,na),vz(n,na)
      common /farce/ fx(n,na),fy(n,na),fz(n,na)
      common /bonds/ dsq(na),xm(na)
      common /cnsts/ tol,bl,maxit,pe,ek,nsnap,itime,dt,vc,w,
     +                  sm,zt,zb
      logical done
      logical moving(na),moved(na)
      dimension rxi(na),ryi(na),rzi(na)
      dimension pxi(na),pyi(na),pzi(na)
      dimension vxi(na),vyi(na),vzi(na)
      integer i,a,b,it
      if((nb.ne.na).and.(nb.ne.na-1)) stop 'nb in error'
      tol2=2.0*tol
      dt2=dt/2.0
      dtsq2=dt*dt2
*                 loop over molecules
```

```
      do 2000 i=1,n
*                 velocity verlet algorithm part a
      do 100 a=1,na
            axia=fx(i,a)/xm(a)
            ayia=fy(i,a)/xm(a)
            azia=fz(i,a)/xm(a)
            rxi(a)=rx(i,a)
            ryi(a)=ry(i,a)
            rzi(a)=rz(i,a)
            pxi(a)=rx(i,a)+dt*vx(i,a)+dtsq2*axia
            pyi(a)=ry(i,a)+dt*vy(i,a)+dtsq2*ayia
            pzi(a)=rz(i,a)+dt*vz(i,a)+dtsq2*azia
            call image(pxi(a),pyi(a))
            vxi(a)=vx(i,a)+dt2*axia
            vyi(a)=vy(i,a)+dt2*ayia
            vzi(a)=vz(i,a)+dt2*azia
            moving(a)=.false.
            moved(a)=.true.
100   continue
      it=0
      done=.false.
*                 start of iterative loop
1000  if((.not.done).and.(it.le.maxit)) then
      done=.true.
      do 300 a=1,nb
            b=a+1
            if(b.gt.na) b=1
            if(moved(a).or.moved(b)) then
                  pxab=pxi(a)-pxi(b)
                  pyab=pyi(a)-pyi(b)
                  pzab=pzi(a)-pzi(b)
                  call image(pxab,pyab)
                  pabsq=pxab**2+pyab**2+pzab**2
                  rabsq=dsq(a)
                  diffsq=rabsq-pabsq
*                  print*,a,b,diffsq,rabsq*tol2
                  if(abs(diffsq).gt.(rabsq*tol2)) then
                        rxab=rxi(a)-rxi(b)
                        ryab=ryi(a)-ryi(b)
                        rzab=rzi(a)-rzi(b)
                        call image(rxab,ryab)
                        rpab=rxab*pxab+ryab*pyab+rzab*pzab
                        if (rpab.lt.(rabsq*rptol)) then
                              print*,'constraintl failure'
                              print*, a,rzi(a),pzi(a)
                              print*, b,rzi(b),pzi(b)
                              print*, rzab,pzab
                              print*, rpab,(rabsq*rptol)
                              print*, i
                              stop
                        end if
                        rma=1.0/xm(a)
```

```
                        rmb=1.0/xm(b)
                        gab=diffsq/(2.0*(rma+rmb)*rpab)
                        dx=rxab*gab
                        dy=ryab*gab
                        dz=rzab*gab
                        pxi(a)=pxi(a)+rma*dx
                        pyi(a)=pyi(a)+rma*dy
                        pzi(a)=pzi(a)+rma*dz
                        pxi(b)=pxi(b)-rmb*dx
                        pyi(b)=pyi(b)-rmb*dy
                        pzi(b)=pzi(b)-rmb*dz
                        dx=dx/dt
                        dy=dy/dt
                        dz=dz/dt
                        vxi(a)=vxi(a)+rma*dx
                        vyi(a)=vyi(a)+rma*dy
                        vzi(a)=vzi(a)+rma*dz
                        vxi(b)=vxi(b)-rmb*dx
                        vyi(b)=vyi(b)-rmb*dy
                        vzi(b)=vzi(b)-rmb*dz
                        moving(a)=.true.
                        moving(b)=.true.
                        done=.false.
                  end if
            end if
300         continue
            do 500 a=1,na
                  moved(a)=moving(a)
                  moving(a)=.false.
500         continue
            it=it+1
            goto 1000
      end if
*                 end of iterative loop
      if (.not.done) then
            print*,'too many constraint2 iterations in movea'
            print*,'molecule',i
            stop
      end if
*                 store away any new values
      do 600 a=1,na
            rx(i,a)=pxi(a)
            ry(i,a)=pyi(a)
            rz(i,a)=pzi(a)
            vx(i,a)=vxi(a)
            vy(i,a)=vyi(a)
            vz(i,a)=vzi(a)
600   continue
2000  continue
*                 end of loop over molecules
      return
      end
```

```
*
* Part 2 of RATTLE (update velocities)
*
      subroutine moveb
      implicit real*8(a-h,o-z)
      parameter (rptol=1.0e-6)
      parameter (n=32,na=10,np=n*na,nb=na-1,ns=16)
      common /coord/ rx(n,na),ry(n,na),rz(n,na)
      common /speed/ vx(n,na),vy(n,na),vz(n,na)
      common /farce/ fx(n,na),fy(n,na),fz(n,na)
      common /bonds/ dsq(na),xm(na)
      common /cnsts/ tol,bl,maxit,pe,ek,nsnap,itime,dt,vc,w,
     +                  sm,zt,zb
      logical done
      logical moving(na),moved(na)
      dimension rxi(na),ryi(na),rzi(na)
      dimension vxi(na),vyi(na),vzi(na)
      integer i,a,b,it
      dt2=dt/2.0
      ek=0.0
      w=0.0
*               loop over all molecules
      do 2000 i=1,n
*               velocity verlet algorithm part b
            do 100 a=1,na
                  rxi(a)=rx(i,a)
                  ryi(a)=ry(i,a)
                  rzi(a)=rz(i,a)
                  vxi(a)=vx(i,a)+dt2*fx(i,a)/xm(a)
                  vyi(a)=vy(i,a)+dt2*fy(i,a)/xm(a)
                  vzi(a)=vz(i,a)+dt2*fz(i,a)/xm(a)
                  moving(a)=.false.
                  moved(a)=.true.
100         continue
*                       start of iterative loop
            it=0
            done=.false.
1000        if((.not.done).and.(it.le.maxit)) then
      done =.true.
      do 300 a=1,nb
            b=a+1
            if(b.gt.na)b=1
            if (moved(a).or.moved(b)) then
                  vxab=vxi(a)-vxi(b)
                  vyab=vyi(a)-vyi(b)
                  vzab=vzi(a)-vzi(b)
                  rxab=rxi(a)-rxi(b)
                  ryab=ryi(a)-ryi(b)
                  rzab=rzi(a)-rzi(b)
                  call image(rxab,ryab)
                  rvab=rxab*vxab+ryab*vyab+rzab*vzab
                  rma=1.0/xm(a)
```

```
                  rmb=1.0/xm(b)
                  gab=-rvab/((rma+rmb)*dsq(a))
                  if (abs(gab).gt.tol) then
                        w=w+gab*dsq(a)
                        dx=rxab*gab
                        dy=ryab*gab
                        dz=rzab*gab
                        vxi(a)=vxi(a)+rma*dx
                        vyi(a)=vyi(a)+rma*dy
                        vzi(a)=vzi(a)+rma*dz
                        vxi(b)=vxi(b)-rmb*dx
                        vyi(b)=vyi(b)-rmb*dy
                        vzi(b)=vzi(b)-rmb*dz
                        moving(a)=.true.
                        moving(b)=.true.
                        done=.false.
                  end if
            end if
300         continue
            do 500 a=1,na
                  moved(a)=moving(a)
                  moving(a)=.false.
500         continue
            it=it+1
            goto 1000
      end if
*                 end of iterative loop
      if (.not.done) then
            print*,'too many constraint3 iterations in movea'
            print*,'molecule',i
            stop
      endif
      do 600 a=1,na
            vx(i,a)=vxi(a)
            vy(i,a)=vyi(a)
            vz(i,a)=vzi(a)
            ek=ek+(xm(a)*(vxi(a)**2+vyi(a)**2+vzi(a)**2))*0.5
600   continue
2000  continue
*                 end of loop over molecules
      w=w/dt2/3.0
      return
      end
*
*                 First move routine for solutes
*
      subroutine moves1
      parameter(n=32,na=10,np=n*na,nb=na-1,ns=16)
      implicit real*8 (a-h,o-z)
      common /coord/ rx(n,na),ry(n,na),rz(n,na)
      common /speed/ vx(n,na),vy(n,na),vz(n,na)
      common /farce/ fx(n,na),fy(n,na),fz(n,na)
```

```
      common /scoord/ sx(ns),sy(ns),sz(ns)
      common /sspeed/svx(ns),svy(ns),svz(ns)
      common /sfarce/ sfx(ns),sfy(ns),sfz(ns)
      common /bonds/ dsq(na),xm(na)
      common /cnsts/ tol,bl,maxit,pe,ek,nsnap,itime,dt,vc,w,
     +               sm,zt,zb
      integer i,it
      tol2=2.0*tol
      dt2=dt/2.0
      dtsq2=dt*dt2
      do 2000 i=1,ns
*             velocity verlet algorithm part 1
            saxi=sfx(i)/sm
            sayi=sfy(i)/sm
            sazi=sfz(i)/sm
            sx(i)=sx(i)+dt*svx(i)+dtsq2*saxi
            sy(i)=sy(i)+dt*svy(i)+dtsq2*sayi
            sz(i)=sz(i)+dt*svz(i)+dtsq2*sazi
            if(sz(i).ge.zt) then
                  if(svz(i).gt.0) then
                        svz(i)=-(svz(i))
                  end if
            end if
            if(sz(i).le.zb) then
                  if(svz(i).lt.0) then
                        svz(i)=-(svz(i))
                  end if
            end if
            call image(sx(i),sy(i))
            svx(i)=svx(i)+dt2*saxi
            svy(i)=svy(i)+dt2*sayi
            svz(i)=svz(i)+dt2*sazi
            it=0
2000  continue
      return
      end
*
*             Second move routine for solutes
*
      subroutine moves2
      parameter(n=32,na=10,np=n*na,nb=na-1,ns=16)
      implicit real*8 (a-h,o-z)
      common /coord/ rx(n,na),ry(n,na),rz(n,na)
      common /speed/ vx(n,na),vy(n,na),vz(n,na)
      common /farce/ fx(n,na),fy(n,na),fz(n,na)
      common /scoord/ sx(ns),sy(ns),sz(ns)
      common /sspeed/svx(ns),svy(ns),svz(ns)
      common /sfarce/ sfx(ns),sfy(ns),sfz(ns)
      common /bonds/ dsq(na),xm(na)
      common /cnsts/ tol,bl,maxit,pe,ek,nsnap,itime,dt,vc,w,
     +               sm,zt,zb
      integer i,it
```

```
      dt2=dt/2.0
      do 2000 i=1,ns
*                velocity verlet algorithm part 2
         svx(i)=svx(i)+dt2*sfx(i)/sm
         svy(i)=svy(i)+dt2*sfy(i)/sm
         svz(i)=svz(i)+dt2*sfz(i)/sm
         it=0
         ek=ek+(sm*(svx(i)**2+svy(i)**2+svz(i)**2))*0.5
2000  continue
      return
      end
*
*              force subroutine
*
      subroutine flj
      parameter(n=32,na=10,np=n*na,nb=na-1,ns=16)
      implicit real*8 (a-h,o-z)
      common /coord/ rx(n,na),ry(n,na),rz(n,na)
      common /farce/ fx(n,na),fy(n,na),fz(n,na)
      common /cnsts/ tol,bl,maxit,pe,ek,nsnap,itime,dt,vc,w,
     +                  sm,zt,zb
      common /grval/ gtcum(2000),gbcum(2000)
      eps=124
      sigma=3.42/15.4
      fhg=3600*(15.4**2)
      press=.1526
      sigsq=sigma**2
      eps4=4*eps
      eps24=24*eps
      pe=0
      w=0
      zto=zt
      zbo=zb
      zt=0
      zb=0
      do 100 i=1,n
         do 200 j=1,na
            fx(i,j)=0
            fy(i,j)=0
            fz(i,j)=0
            if (j.eq.1) then
               if (i.le.(n/2)) then
                  zt=zt+rz(i,j)
               else
                  zb=zb+rz(i,j)
               endif
            endif
200      continue
100   continue
      zt=2*zt/n
      zb=2*zb/n
      do 300 i=1,n
```

```
      do 400 j=1,n
         do 500 ia=1,na
            do 600 ja=1,na
               if((i .ne. j) .or. (abs(ia-ja) .gt.2)) then
                  xdiff=rx(i,ia)-rx(j,ja)
                  ydiff=ry(i,ia)-ry(j,ja)
                  zdiff=rz(i,ia)-rz(j,ja)
                  call image(xdiff,ydiff)
                  rdiffsq=xdiff**2+ydiff**2+zdiff**2
                  if (dabs(rdiffsq) .lt. 1.0e-6) then
                     print*,i,j,ia,ja,xdiff,ydiff,zdiff
                  end if
                  if ((ia.eq.1).and.(ja.eq.1)) then
                     ir=anint(125*dsqrt(rdiffsq))
                     if ((i.le.16).and.(j.le.16)) then
                       gtcum(ir)=gtcum(ir)+1
                     else
                       gbcum(ir)=gbcum(ir)+1
                     end if
                  end if
                  sr2=sigsq/rdiffsq
                     sr6=sr2**3
                     sr12=sr6**2
                     vdiff=sr12-sr6
                     pe=pe+vdiff
                     wdiff=vdiff+sr12
                     w=w+wdiff
                     fdiff=wdiff/rdiffsq
                     fxdiff=fdiff*xdiff
                     fydiff=fdiff*ydiff
                     fzdiff=fdiff*zdiff
                     fx(i,ia)=fx(i,ia)+fxdiff
                     fy(i,ia)=fy(i,ia)+fydiff
                     fz(i,ia)=fz(i,ia)+fzdiff
                     fx(j,ja)=fx(j,ja)-fxdiff
                     fy(j,ja)=fy(j,ja)-fydiff
                     fz(j,ja)=fz(j,ja)-fzdiff
               end if
600         continue
500      continue
400   continue
300   continue
      pe=pe*eps4
      do 800 i=1,n
         do 900 j=1, na
            fx(i,j)=fx(i,j)*eps24
            fy(i,j)=fy(i,j)*eps24
            fz(i,j)=fz(i,j)*eps24
            if (j.eq.1) then
               if (i.le.(n/2)) then
                  fz(i,j)=fz(i,j)-fhg*(rz(i,j)-zt)-press
                  pe=pe+0.5*fhg*(rz(i,j)-zt)**2-(zt-zto)*press
```

```
               else
                  fz(i,j)=fz(i,j)-fhg*(rz(i,j)-zb)+press
                  pe=pe+0.5*fhg*(rz(i,j)-zb)**2-(zb-zbo)*press
               endif
            endif
900      continue
800   continue
      return
      end
*
*              solute force subroutine
*
      subroutine sflj
      parameter(n=32,na=10,np=n*na,nb=na-1,ns=16)
      implicit real*8 (a-h,o-z)
      common /coord/ rx(n,na),ry(n,na),rz(n,na)
      common /farce/ fx(n,na),fy(n,na),fz(n,na)
      common /scoord/ sx(ns),sy(ns),sz(ns)
      common /sspeed/svx(ns),svy(ns),svz(ns)
      common /sfarce/ sfx(ns),sfy(ns),sfz(ns)
      common /cnsts/ tol,bl,maxit,pe,ek,nsnap,itime,dt,vc,w,
     +                sm,zt,zb
      eps=124
      sigma=3.42/15.4
      sigsq=sigma**2
      spe=0
      eps4=4*eps
      eps24=24*eps
      do 100 i=1,ns
            sfx(i)=0
            sfy(i)=0
            sfz(i)=0
100   continue
      do 300 i=1,ns-1
         do 400 j=i+1,ns
            sxdiff=sx(i)-sx(j)
            sydiff=sy(i)-sy(j)
            szdiff=sz(i)-sz(j)
            call image(sxdiff,sydiff)
            srdiffsq=sxdiff**2+sydiff**2+szdiff**2
            sssr2=sigsq/srdiffsq
               sssr6=sssr2**3
               sssr12=sssr6**2
               svdiff=sssr12-sssr6
               spe=spe+svdiff
               swdiff=svdiff+sssr12
               w=w+swdiff
               sfdiff=swdiff/srdiffsq
               sfxdiff=sfdiff*sxdiff
               sfydiff=sfdiff*sydiff
               sfzdiff=sfdiff*szdiff
               sfx(i)=sfx(i)+sfxdiff
```

```
                sfy(i)=sfy(i)+sfydiff
                sfz(i)=sfz(i)+sfzdiff
                sfx(j)=sfx(j)-sfxdiff
                sfy(j)=sfy(j)-sfydiff
                sfz(j)=sfz(j)-sfzdiff
400          continue
300      continue
         do 500 i=1,n
            do 600 j=1,na
               do 700 a=1,ns
                  xsdiff=sx(a)-rx(i,j)
                  ysdiff=sy(a)-ry(i,j)
                  zsdiff=sz(a)-rz(i,j)
                  call image(xsdiff,ysdiff)
                  sdiffsq=xsdiff**2+ysdiff**2+zsdiff**2
                  if(dabs(sdiffsq).lt.1.0e-6) then
                     print*,a,xsdiff,ysdiff,zsdiff
                  end if
                  ssr2=sigsq/sdiffsq
                     ssr6=ssr2**3
                     ssr12=ssr6**2
                     svd=ssr12-ssr6
                     spe=spe+svd
                     swdiff=svd+ssr12
                     w=w+swdiff
                     sfd=swdiff/sdiffsq
                     sfxd=sfd*xsdiff
                     sfyd=sfd*ysdiff
                     sfzd=sfd*zsdiff
                     sfx(a)=sfx(a)+sfxd
                     sfy(a)=sfy(a)+sfyd
                     sfz(a)=sfz(a)+sfzd
                     fx(i,j)=fx(i,j)-(sfxd*eps24)
                     fy(i,j)=fy(i,j)-(sfyd*eps24)
                     fz(i,j)=fz(i,j)-(sfzd*eps24)
700            continue
600         continue
500      continue
         spe=spe*eps4
         do 750 i=1,ns
            sfx(i)=sfx(i)*eps24
            sfy(i)=sfy(i)*eps24
            sfz(i)=sfz(i)*eps24
750      continue
         pe=pe+spe
         w=w*eps24/3
         return
         end

*
*                output subroutine
*
```

```
      subroutine outpt
      parameter(n=32,na=10,np=n*na,nb=na-1,ns=16)
      implicit real*8 (a-h,o-z)
      common /cnsts/ tol,bl,maxit,pe,ek,nsnap,itime,dt,vc,w,
     +               sm,zt,zb
      nt=np+ns
      ek=1.2023*ek/nt
      pe=pe/nt
      e=(ek+pe)
      bw=zt-zb
      write (12,*) itime,(bw*15.4)
      write(5,15)itime,e,ek,pe,bw
15    format(1x,i6,2x,4(d12.6,2x))
      vc=0
      return
      end
*
*              saving configurations
*
      subroutine savecon
      parameter(n=32,na=10,np=n*na,nb=na-1,ns=16)
      parameter(rptol=1.0e-6)
      implicit real*8 (a-h,o-z)
      common /coord/ rx(n,na),ry(n,na),rz(n,na)
      common /speed/ vx(n,na),vy(n,na),vz(n,na)
      common /scoord/ sx(ns),sy(ns),sz(ns)
      common /sspeed/svx(ns),svy(ns),svz(ns)
      common /sfarce/ sfx(ns),sfy(ns),sfz(ns)
      common /cnsts/ tol,bl,maxit,pe,ek,nsnap,itime,dt,vc,w,
     +               sm,zt,zb
      open (unit=2,file='startr16.dat',status='old')
      open (unit=3,file='startv16.dat',status='old')
      do 100 i=1,n
         do 150 j=1,na
         write(2,*)itime,rx(i,j),ry(i,j),rz(i,j)
         write(3,*)itime,vx(i,j),vy(i,j),vz(i,j)
150      continue
100   continue
      do 200 i=1, ns
         write(2,*)itime,sx(i),sy(i),sz(i)
         write(3,*)itime,svx(i),svy(i),svz(i)
200   continue
      close (2,status='keep')
      close(3,status='keep')
      return
      end
*
*                    GRAFINIT SUB
*
      subroutine grafinit
      implicit real*8 (a-h,o-z)
      include 'fgraph.fd'
```

```
      parameter(n=32,na=10,np=n*na,nb=na-1,ns=16)
      parameter(rptol=1.0e-6)
      common /coord/ rx(n,na),ry(n,na),rz(n,na)
      common /speed/ vx(n,na),vy(n,na),vz(n,na)
      common /cnsts/ tol,bl,maxit,pe,ek,nsnap,itime,dt,vc,w,
     +               sm,zt,zb
      common /farce/ fx(n,na),fy(n,na),fz(n,na)
      common /scoord/ sx(ns),sy(ns),sz(ns)
      common /sspeed/svx(ns),svy(ns),svz(ns)
      common /sfarce/ sfx(ns),sfy(ns),sfz(ns)
      common /bonds/ dsq(na),xm(na)
      dummy=setvideomode($vres16color)
      dummy = setcolor(15)
      call setviewport(430, 1, 630, 300)
      call settextwindow(1, 1, 28, 40)
      wx1 = -1
      wy1 = 2
      wx2 = 1
      wy2 = -2
      dummy=setwindow(.true.,wx1,wy1,wx2,wy2)
      call clearscreen($gclearscreen)
      return
      end
*
*                     SNAPSHOT SUB
*
      subroutine snapshot
      implicit real*8 (a-h,o-z)
      include 'fgraph.fd'
      parameter(n=32,na=10,np=n*na,nb=na-1,ns=16)
      parameter(rptol=1.0e-6)
      record /wxycoord/ wxy
      common /coord/ rx(n,na),ry(n,na),rz(n,na)
      common /speed/ vx(n,na),vy(n,na),vz(n,na)
      common /cnsts/ tol,bl,maxit,pe,ek,nsnap,itime,dt,vc,w,
     +               sm,zt,zb
      common /farce/ fx(n,na),fy(n,na),fz(n,na)
      common /scoord/ sx(ns),sy(ns),sz(ns)
      common /sspeed/svx(ns),svy(ns),svz(ns)
      common /sfarce/ sfx(ns),sfy(ns),sfz(ns)
      common /bonds/ dsq(na),xm(na)
      radius = .1
      e=ek+pe
      bw=zt-zb
      call clearscreen($gviewport)
      write(*,15)itime,e,ek,pe,(bw*15.4)
15    format(1x,i6,2x,4(d12.6,2x))
      do 50 i=1,ns
         wx1 = sy(i) - .5 * radius
         wx2 = sy(i) + .5 * radius
         wz1 = sz(i) + radius
         wz2 = sz(i) - radius
```

```
         dummy=ellipse_w($gborder,wx1,wz1,wx2,wz2)
50    continue
      do 100 i=1,n
         wx1 = ry(i,1) - .5 * radius
         wx2 = ry(i,1) + .5 * radius
         wz1 = rz(i,1) + radius
         wz2 = rz(i,1) - radius
         dummy=ellipse_w($gborder,wx1,wz1,wx2,wz2)
         dummy=setpixel_w(ry(i,1),rz(i,1))
         call moveto_w(ry(i,1),rz(i,1),wxy)
         do 200 j=1,na
            if(j.eq.1) then
               dummy=lineto_w(ry(i,j),rz(i,j))
            else
               rl=(ry(i,j)-ry(i,j-1))**2
               if(rl.lt.5) then
                  dummy=lineto_w(ry(i,j),rz(i,j))
               end if
            end if
200      continue
100   continue
      return
      end
*
*                        THERMOSTAT SUB
*
      subroutine thermostat
      implicit real*8 (a-h,o-z)
      parameter(n=32,na=10,np=n*na,nb=na-1,ns=16)
      parameter(rptol=1.0e-6)
      common /coord/ rx(n,na),ry(n,na),rz(n,na)
      common /speed/ vx(n,na),vy(n,na),vz(n,na)
      common /cnsts/ tol,bl,maxit,pe,ek,nsnap,itime,dt,vc,w,
     +                  sm,zt,zb
      common /farce/ fx(n,na),fy(n,na),fz(n,na)
      common /scoord/ sx(ns),sy(ns),sz(ns)
      common /sspeed/svx(ns),svy(ns),svz(ns)
      common /sfarce/ sfx(ns),sfy(ns),sfz(ns)
      common /bonds/ dsq(na),xm(na)
      tnow=ek
      tt=.4
      zavg=zt+zb
      ttrue=310
      chi=dsqrt(1+(dt/tt)*(ttrue/tnow-1))
*      chi=dsqrt(ttrue/tnow)
      do 10 i=1,n
         do 12 j=1,na
*           rz(i,j)=rz(i,j)-zavg
           vx(i,j)=vx(i,j)*chi
           vy(i,j)=vy(i,j)*chi
           vz(i,j)=vz(i,j)*chi
12       continue
```

```
10    continue
      do 15 i=1,ns
*        sz(i)=sz(i)-zavg
         svx(i)=svx(i)*chi
         svy(i)=svy(i)*chi
         svz(i)=svz(i)*chi
15    continue
      return
      end
*
*                        IMAGE SUB
*
      subroutine image(x,y)
      implicit real*8 (a-h,o-z)
      rt3=dsqrt(3.0)
      rrt3=1/rt3
      rt32=rt3/2
      rrt32=1/rt32
*      x=x
*      y=y
*      x=x-anint(x-rrt3*y)-anint(rrt32*y)*0.5
*      y=y-anint(y*rrt32)*rt32
      x=x-anint(x)
      y=y-anint(y)
      return
      end
```

Appendix B. Programs Necessary for Simulations

The following is a list of the FORTRAN files for the different lipid bilayers and the data files need to run the simulation. Each grouping is located on individual disks.

sratl0.for :
- startr0.dat
- startv0.dat
- gc0.dat
- ssbener0.dat
- tilt0.dat
- velthg0.dat
- velbhg0.dat

sratl2.for :
- startr2.dat
- startv2.dat
- gc2.dat
- ssbener2.dat
- vels2.dat
- velthg2.dat
- velbhg2.dat

sratl4.for :
- startr4.dat
- startv4.dat
- gc4.dat
- ssbener4.dat
- vels4.dat
- velthg4.dat
- velbhg4.dat

sratl8.for :
- startr8.dat
- startv8.dat
- gc8.dat
- ssbener8.dat
- tilt8.dat
- vels8.dat
- velthg8.dat
- velbhg8.dat

sratl12.for :
- startr12.dat
- startv12.dat
- gc12.dat
- ssbene12.dat
- vels12.dat
- velthg12.dat
- velbhg12.dat

sratl16.for :
- startr16.dat
- startv16.dat
- gc16.dat
- ssbene16.dat
- tilt16.dat
- biwide.dat
- vels16.dat
- velthg16.dat
- velbhg16.dat

Other files necessary for analysis of data are:

BASIC:
biwide.bas
chmoment.bas
displace.bas
gplot.bas
neplot.bas
position.bas
sdcon.bas
tilt.bas
tiltav.bas
velauto.bas
velrand.bas

FORTRAN:
gzero.for
restrt.for

BATCH:
flg.bat

DATA:
disp2.dat
disp4.dat
disp8.dat
disp12.dat
disp16.dat
sdisp2.dat
sdisp4.dat
sdisp8.dat
sdisp12.dat
sdisp16.dat

GRAPHS AND FIGURES:

gtotz.wp1 - average z-displacement for all solutes
bwidth.wp1 - average bilayer width for all solutes
selfdiff.wp1 - self diffusion constants for all solutes

zdisp2.wp1
zdisp4.wp1
zdisp8.wp1 ======>>
zdisp12.wp1
zdisp16.wp1

root mean square z displacement for each bilayer

gr0.wp1
gr2.wp1 radial distribution
gr4.wp1 ======>> function plots for
gr8.wp1 each bilayer
gr12.wp1
gr16.wp1

results - a Word Perfect table of overall results
tilt8.wp1 - tilt angle for 8 solutes vs. time
bw16.wp1 - bilayer width for 16 solutes vs. time
grtot.wp1 - average g(r) for all simulations
bwfluc.wp1 - average bilayer width fluctuations vs.solutes

Appendix C. Explanation of Simulation Programs

Each FORTRAN program is entitled sratl and a number designating the number of solutes that are dissolved in the lipid bilayer (i.e. sratl16.for is the title for a bilayer with 16 solute particles). Each of these programs requires two start up files. One file is startr plus a number and contains the positions of the particles in the system. The other is startv plus a number which contains the velocities of the particles. For example, sratl16.for requires startr16.dat and startv16.dat. The position and velocity files are saved every 25 time steps. This allows a program to be stopped and restarted without having to start from the very beginning.

The FORTRAN files also requires a file entitled ssbener plus a number. The energy of the system for a single run is saved to this file. The file can be analyzed to create a plot using the BASIC file neplot.bas. (Note: The 12 and 16 particle bilayers have energy files entitled ssbene plus number.) The FORTRAN program also requires the data file gc plus a number. This file contains the radial distribution function data and is updated every 100 time steps. The gc plus number data file can be used to generate a graph using the BASIC file gplot.bas.

Each FORTRAN file requires files for both sets of head groups and the solutes entitled vels plus number (for the solutes), velthg plus number (for the top head group), and velbhg plus number (for the bottom head group). These files contain the velocities and positions of the corresponding particles over the time period of the simulation. These files can be analyzed using the BASIC file velauto.bas. This analysis will result in

velocity autocorrelation plots and self diffusion plots and calculations.

The programs for 0, 8, and 16 particle bilayers also have a tilt data files. This file calculates the tilt of the chains of the bilayer. In the case of the 0 and 16 programs the angle for each chain is calculated and added to the grand total. The BASIC program tilt.bas analyzes the files tilt0.dat and tilt16.dat producing the average tilt for the monolayers (given in radians). The file tilt8.dat does not average the calculated angles but instead saves the individual angles over time. This can then be analyzed by the program tiltav.bas which calculates the average values for comparison with the 0 and 16 values but also plots the angles over time. This allows examination of the fluctuations in the tilt of the bilayer over time.

The 16 particle program also requires biwide.dat. This data files stores the average bilayer width during the simulation. This data can be plotted using the program bwide.bas which plots the bilayer width versus time. This allows examination of bilayer width fluctuations over time.

The z displacement of the solutes are determined using the positions in the vels data file. This program determines the root mean square displacement of the solutes and saves the values in the file titled disp plus number. The values were plotted using Quatro Pro. This graphics program will not read double precision values, so the program sdcon.bas was written to change the double precision values of the disp files to single precision values which were saved in the sdisp data files.

There are many other files that are used to check the programs. The BASIC file chmoment.bas uses the startv data files to check the momentum of the system. The file

velrand.bas is used to randomize the velocities to zero the momentum of the system. Gzero.for is a program which re-zeroes the radial distribution files when a new data run is started. Position.bas is a file that will reposition the particles about the x-axis. This is used only when the program experiences a net drift in the z direction and the positions have moved so that the entire bilayer is not able to be viewed on the output screen.

Two programs that are necessary to have in the directory are flg.bat and restrt.for. The batch file is needed to compile and link the sratl FORTRAN files to create working executable files. The restrt.for file will return the cursor to the screen after a program has been run. For some unknown reason FORTRAN graphics will turn off the cursor. By keeping an executable file of the restrt.for file, one can easily restore the cursor.

FIGURE CAPTIONS

Figure 1: Three Dimensional View of a Lipid Bilayer.

Figure 2: Graph of the Lennard-Jones Potential.
Energy is in ε units and distance in σ units.

Figure 3A-F: Simulation Energetics.
x-axis is in time steps.
The curves are, in order from bottom to top: bilayer width, kinetic energy, and total energy. These graphs are for simulations with 0,2,4,8,12, and 16 solutes, respectively.

Table 1: Compilation of Simulation Energetics Results.
E is total energy, T is temperature or kinetic energy, and BW is the bilayer width.

Figure 4: Graph of bilayer width vs solute number.

Figure 5: Graph of bilayer width fluctuations over time in a simulation with 16 solutes.

Figure 6: Graph of the fluctuations in bilayer width for all simulations.

Figure 7: Two dimensional view of bilayer demonstrating tilt effect.

Table 2: Compilation of the tilt angles for the monolayers of the simulation of 0,8,and 16 solutes.
Angles given in degrees.

Figure 8: Graph of the tilt fluctuation over time in a 8 solute simulation.
Top curves are the fluctuations in the y-z plane for both monolayers and the bottom curves are the x-z fluctuations.

Figure 9A-G: Radial Distribution Function.
Graphs are for all simulations averaged and simulations containing 0,2,4,8,12,and 16 solutes, respectively.
x-axis: 0 - 8 Angstroms.

Figure 10A-G: Radial Distribution Function.
Graphs are for all simulations averaged and simulations containing 0,2,4,8,12,and 16 solutes, respectively.
x-axis: 0 - 15 Angstroms.

Figure 11: Graph of the average RMS z displacement of the solutes.
y-axis in box lengths (1 box length=15.4 Angstroms).

Figure 12A-E: Graphs of the RMS z displacement over time for simulations containing 2,4,8,12,and 16 solutes respectively.
y-axis in box lengths (1 box length =15.4 Angstroms).

Figure 13A-F: Graph of the velocity autocorrelation function for the top head group plane for simulations containing 0,2,4,8,12,and 16 solutes, respectively.
x-axis: 0 - 50 time steps, y-axis: -1 - 1.

Figure 14A-F: Graph of the velocity autocorrelation function for the bottom head group plane for simulations containing 0,2,4,8,12,and 16 solutes, respectively.
x-axis: 0 - 50 time steps, y-axis: -1 - 1.

Figure 15A-E: Graph of the velocity autocorrelation function for the solutes for the simulations containing 2,4,8,12,and 16 solutes, respectively.
x-axis: 0 - 50 time steps, y-axis: -1 - 1.

Figure 16A-F(a)(b): Graphs of the (a) velocity autocorrelation function and (b) the self diffusion constant integrated over time. These are graphs for the top head group planes for simulations containing 0,2,4,8,12,and 16 solutes, respectively.
x-axis: 0 - 1500 time steps
(a) y-axis: -1 - 1 (b) y-axis: -.01 - .01

Figure 17A-F(a)(b): Graphs of the (a) velocity autocorrelation function and (b) the self diffusion constant integrated over time. These graphs are for the bottom head group planes for simulations containing 0,2,4,8,12,and 16 solutes, respectively.
x-axis: 0 - 1500 time steps
(a) y-axis: -1 - 1 (b) y-axis: -.01 - .01

Figure 18A-E(a)(b): Graphs of the (a) velocity autocorrelation function and (b) the self diffusion constant integrated over time. These graphs are for the solutes for simulations containing 2,4,8,12,and 16 solutes, respectively.
x-axis: 0 - 1500 time steps
(a) y-axis: -1 - 1 (b) y-axis: -.01 - .01

Figure 19: Graph of the self diffusion constant of the solutes.

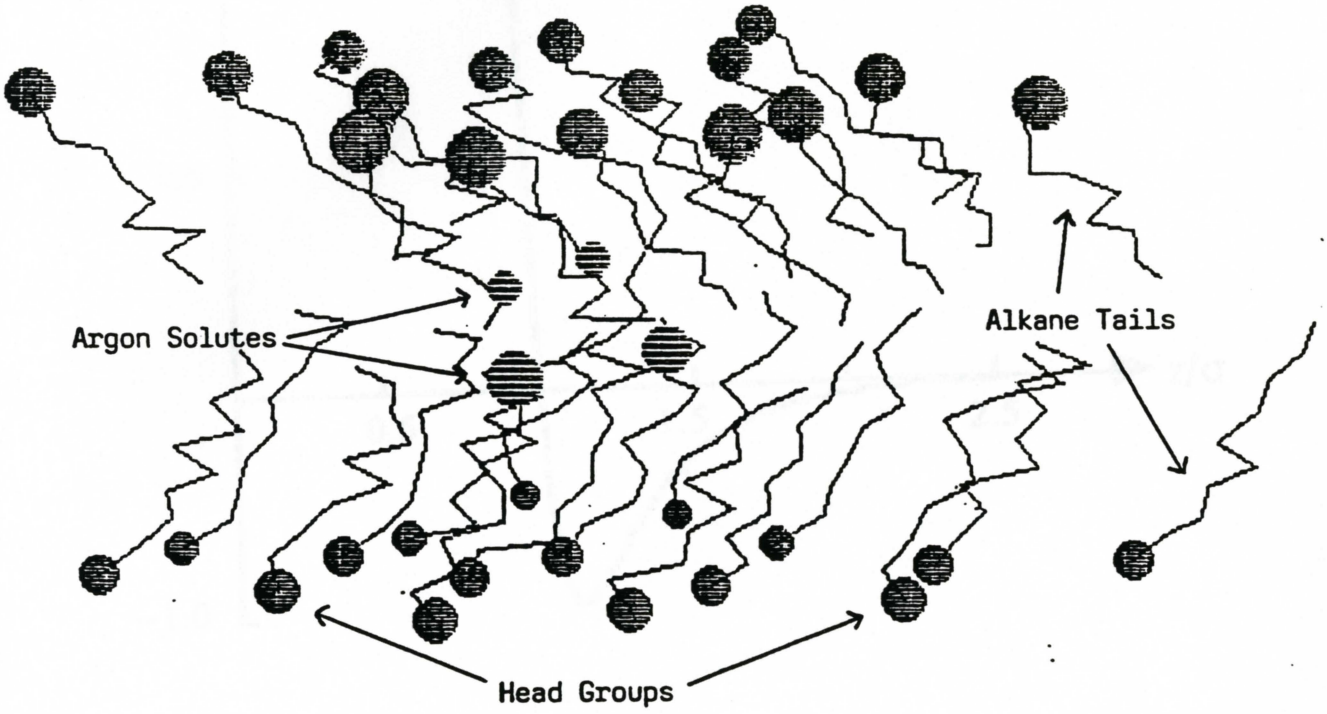


Figure 1

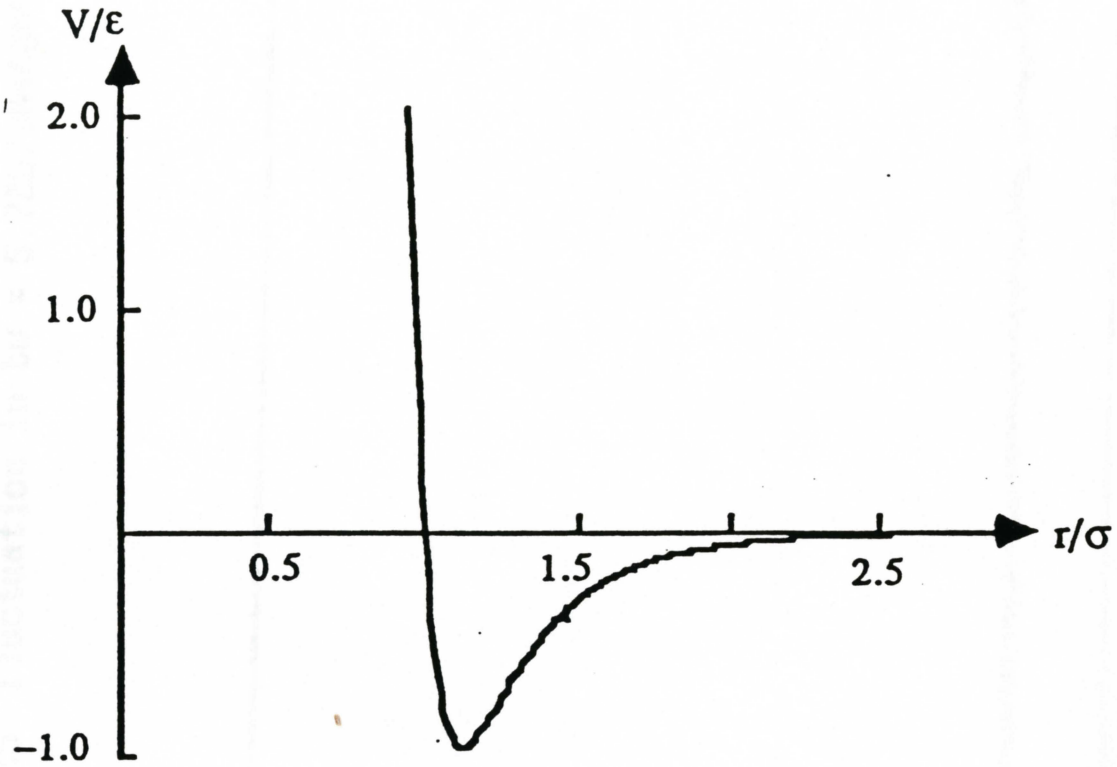


Figure 2

```
average e =-2629.86553913043   fluctuation in e = 7.07400130502514D-04
average pe =-2942.67174782609  fluctuation in ec= 3.51479025012648D-03
average solvent temp = 312.80  fluctuation in ke = 3.82500009162775D-02
average bw = 24.2537170479125  fluctuation in bw = 5.72870846809279D-03
xmin =  1  xmax =  2301
ymin =  0  ymax =  3000
```

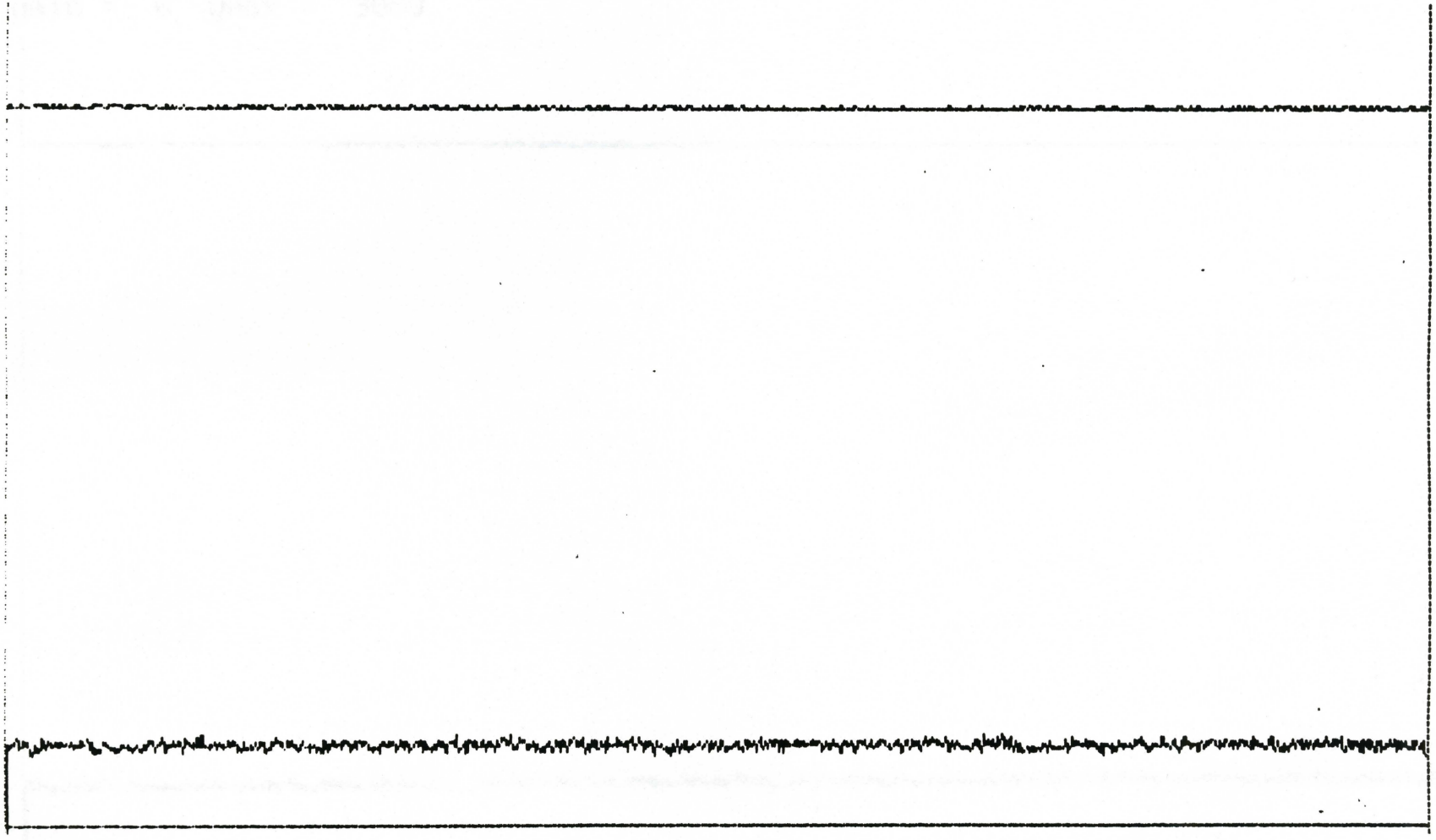


Figure 3A

```
average e =-2604.75635915493   fluctuation in e = 1.37720342004668D-03
average pe =-2911.42983450704  fluctuation in ec= 3.38330822547582D-03
average solvent temp = 306.67  fluctuation in ke = 3.82680064277512D-02
average bw = 24.5502970489127  fluctuation in bw = 5.58304379226978D-03
xmin =  1  xmax =  2841
ymin =  0  ymax =  3000
```

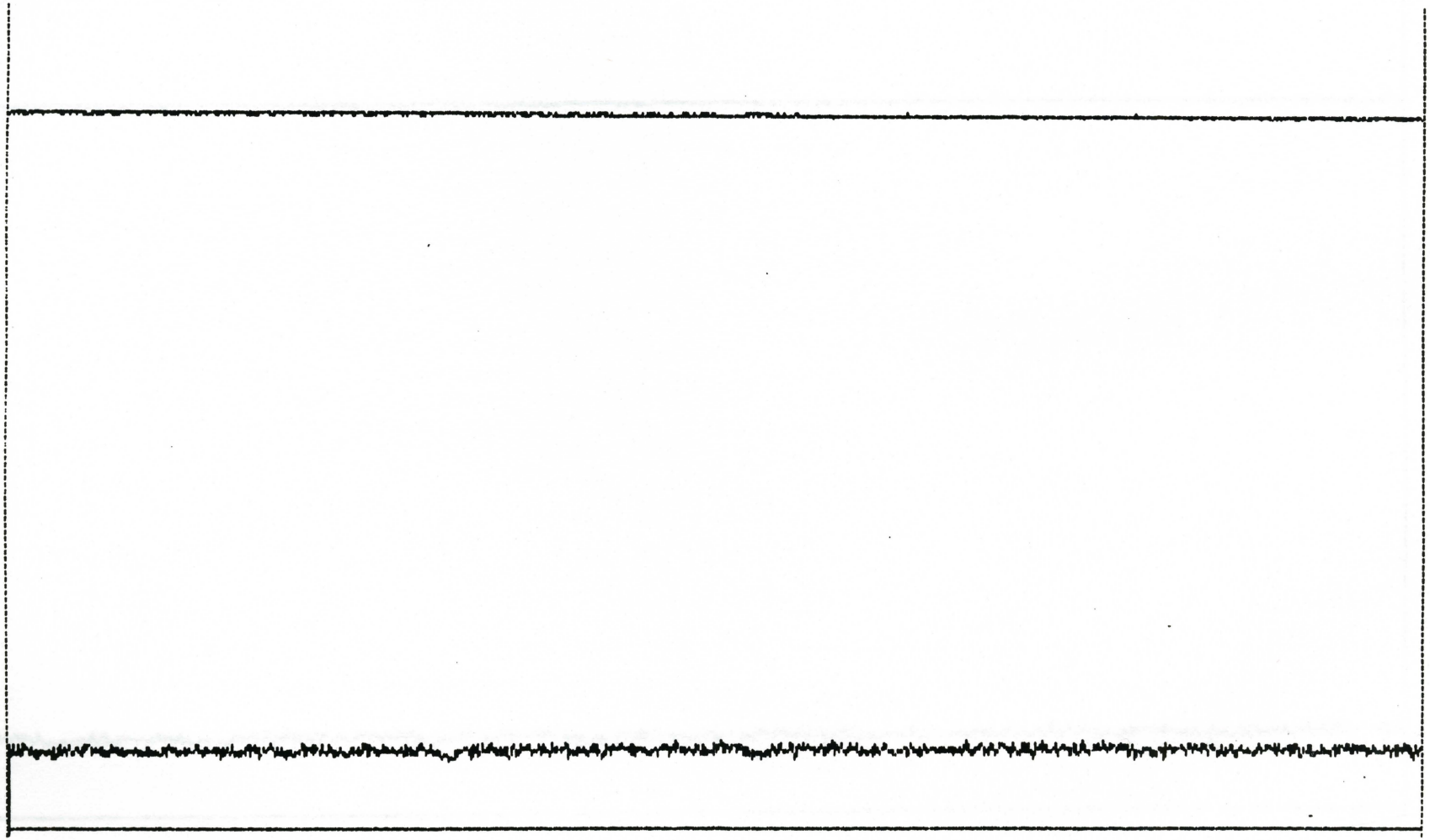


Figure 3B

```
average e =-2583.770395          fluctuation in e = 7.00638648149333D-04
average pe =-2888.461155         fluctuation in ec= 3.545931164994 29D-03
average solvent temp = 304.69    fluctuation in ke = 3.906637158848 91D-02
average bw = 24.8761740917982    fluctuation in bw = 5.89204557815345D-03
xmin =  1  xmax =  2001
ymin =  0  ymax =  3000
```

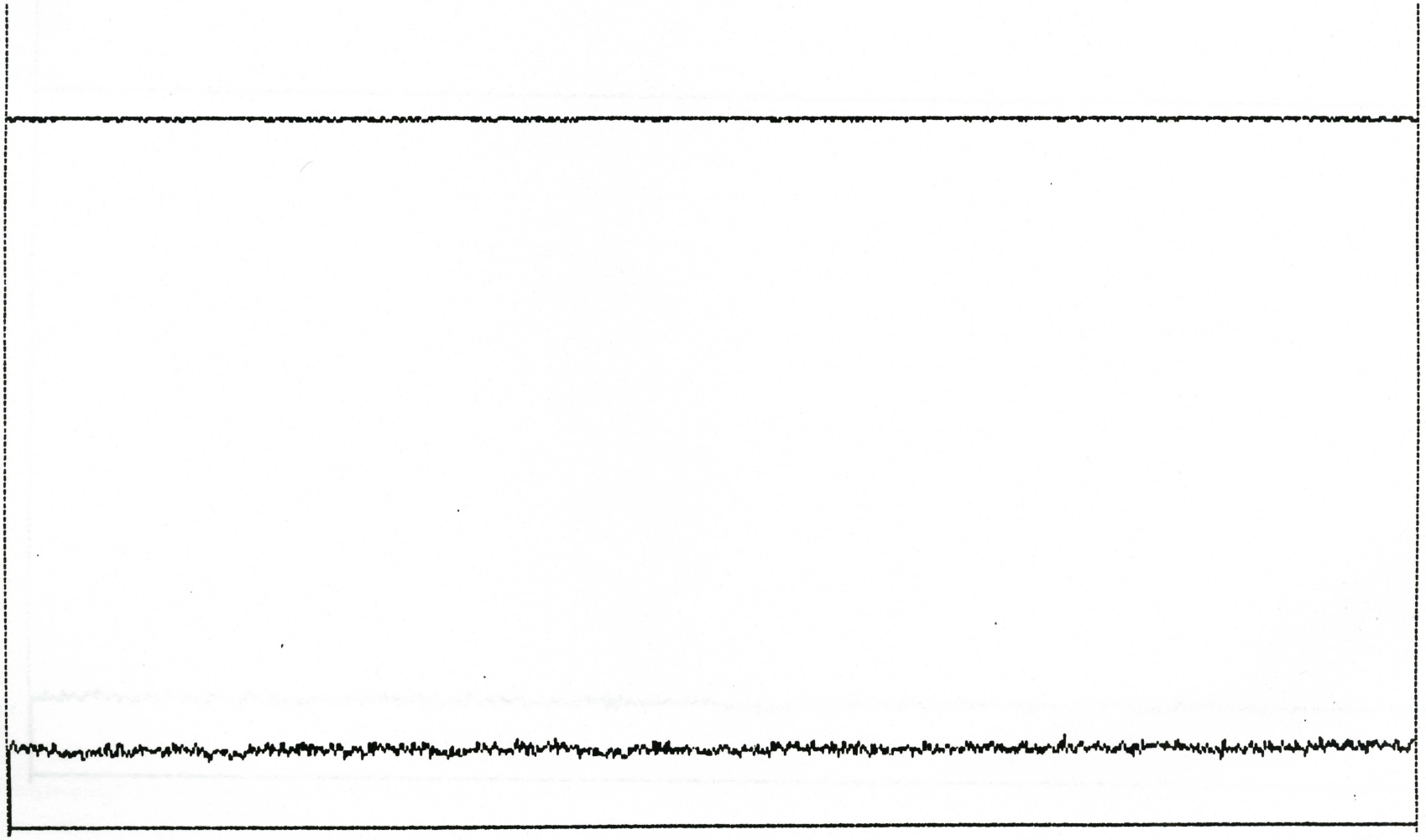


Figure 3C

```
average e =-2522.79972164948   fluctuation in e = 7.20906797419101D-04
average pe =-2827.52969072165  fluctuation in ec= 3.33784742889022D-03
average solvent temp = 304.72  fluctuation in ke = 3.59054609366398D-02
average bw = 25.6021566761243  fluctuation in bw = 5.10421672247439D-03
xmin =  1  xmax =  1941
ymin =  0  ymax =  3000
```

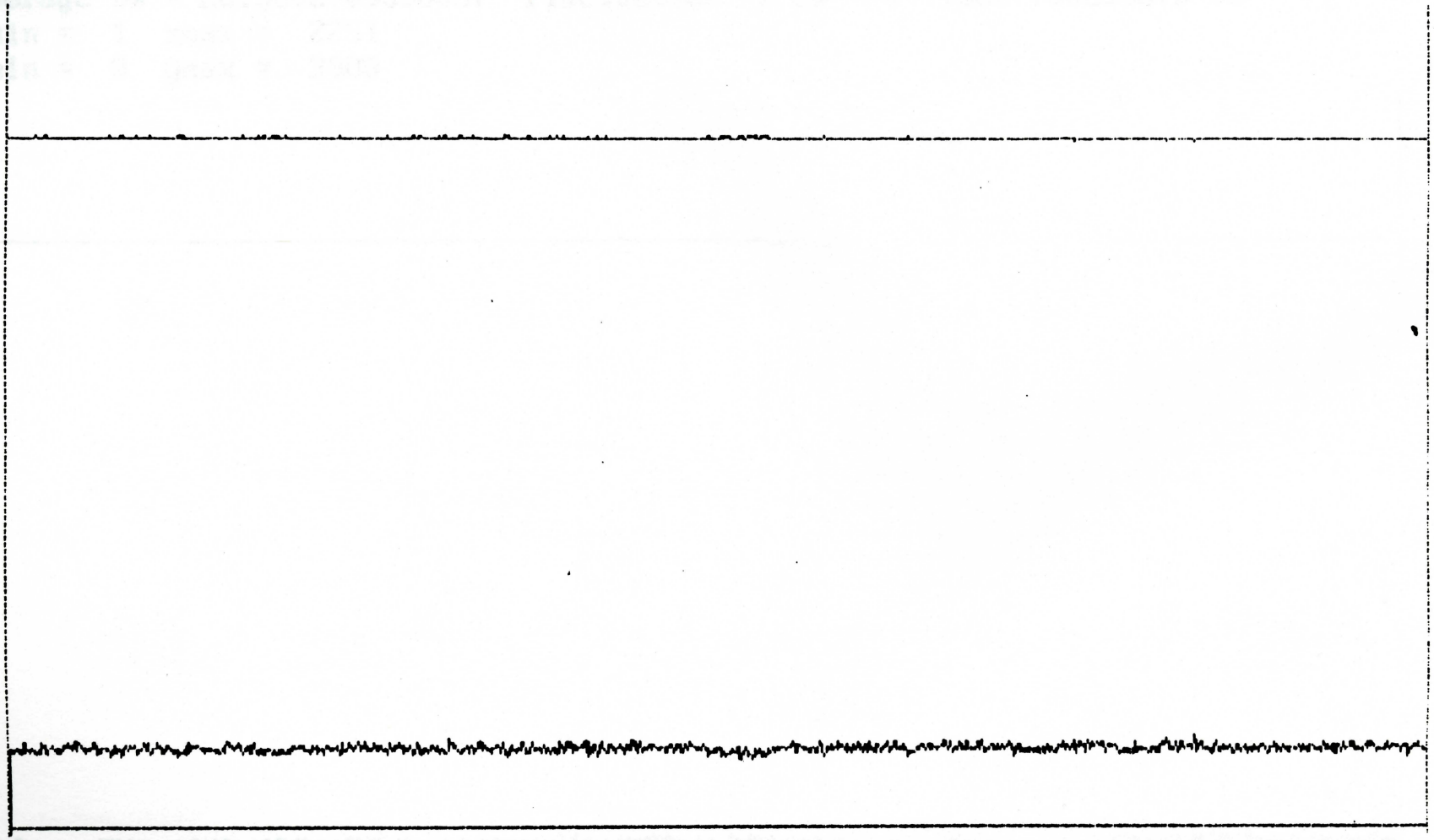


Figure 3D

```
average e =-2473.02328          fluctuation in e = 6.97249690843713D-04
average pe =-2779.08813333333   fluctuation in ec= 3.51434819091153D-03
average solvent temp = 306.06   fluctuation in ke = 3.70192389677266D-02
average bw = 26.5802709398087   fluctuation in bw = 6.776287738253870-03
xmin =  1  xmax =  2251
ymin =  0  ymax =  3000
```

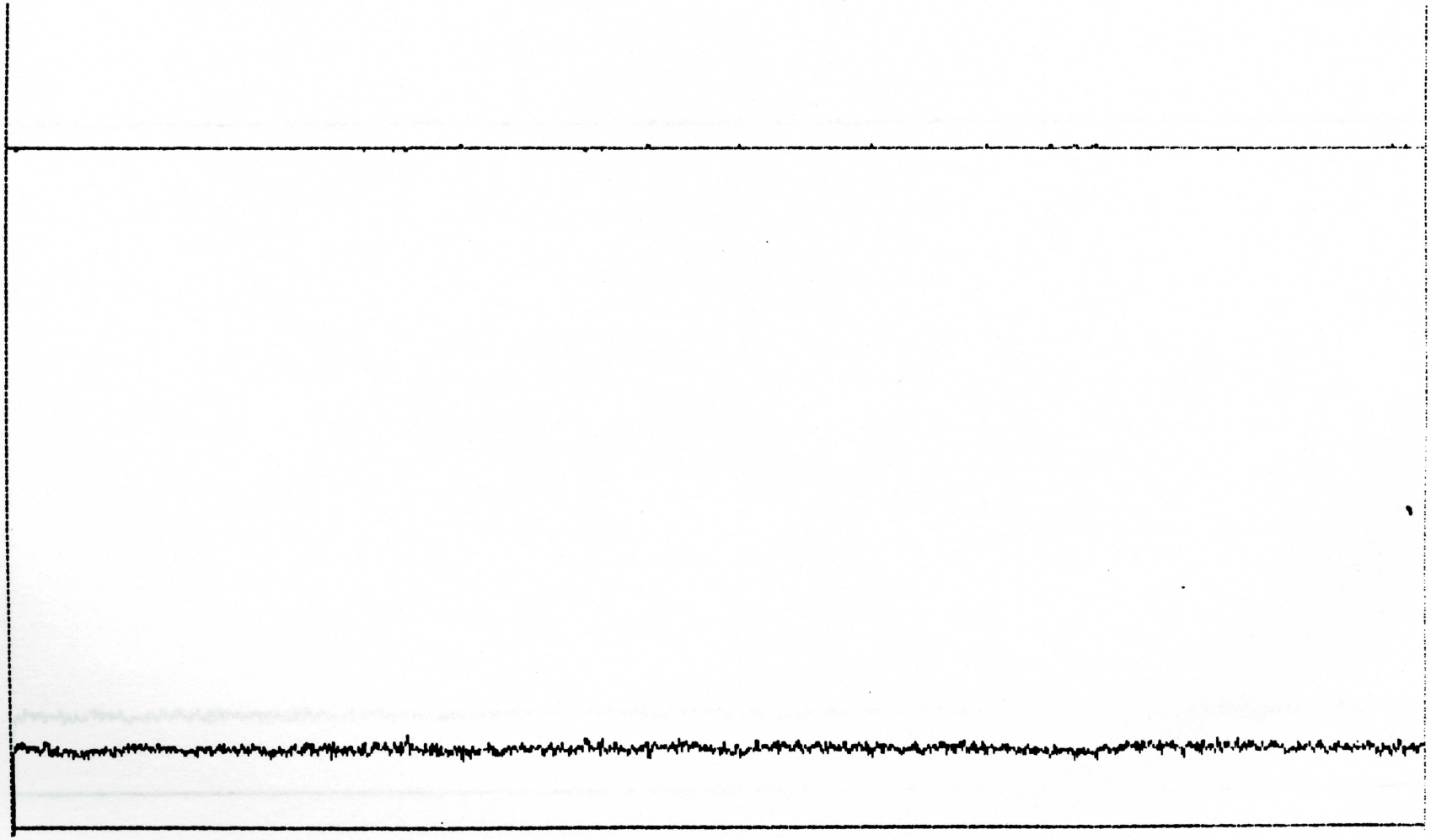


Figure 3E

```
average e =-2434.295985          fluctuation in e = 7.0311033917551D-04
average pe =-2740.735975         fluctuation in ec= 3.3983956592186D-03
average solvent temp = 306.44    fluctuation in ke = 3.5449122687420BD-02
average bw = 27.7327846240378    fluctuation in bw = 7.739145862846 32D-03
xmin =  1  xmax =  2001
ymin =  0  ymax =  3000
```

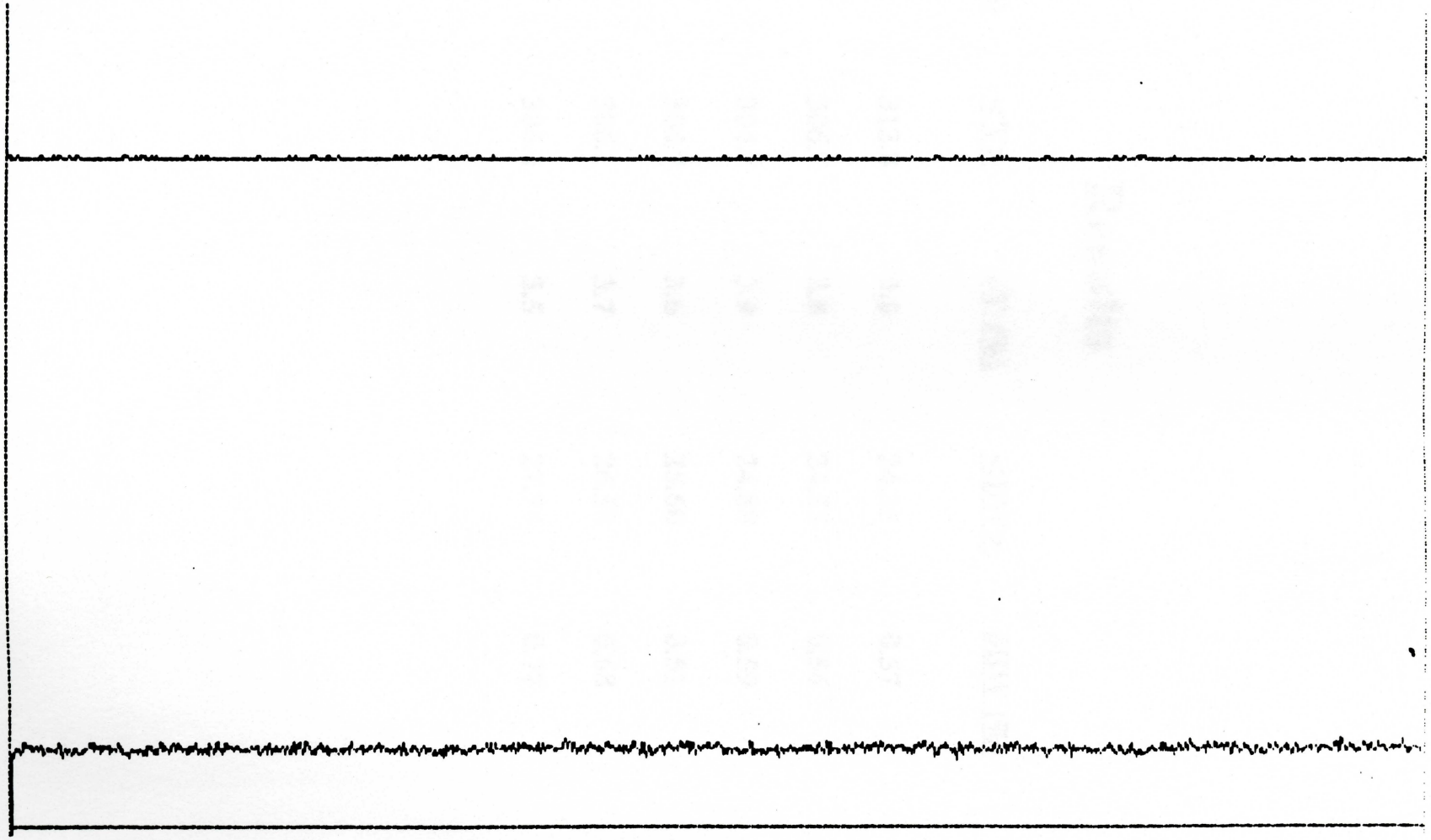


Figure 3F

Results

Solutes	δE (%)	<T>	δT (%)	<BW>	δBW (%)
0	0.071	312.8	3.8	24.25	0.57
2	0.14	306.7	3.8	24.55	0.56
4	0.070	304.7	3.9	24.88	0.59
8	0.072	304.7	3.6	25.60	0.51
12	0.070	306.1	3.7	26.58	0.68
16	0.070	306.4	3.5	27.74	0.77

Table 1

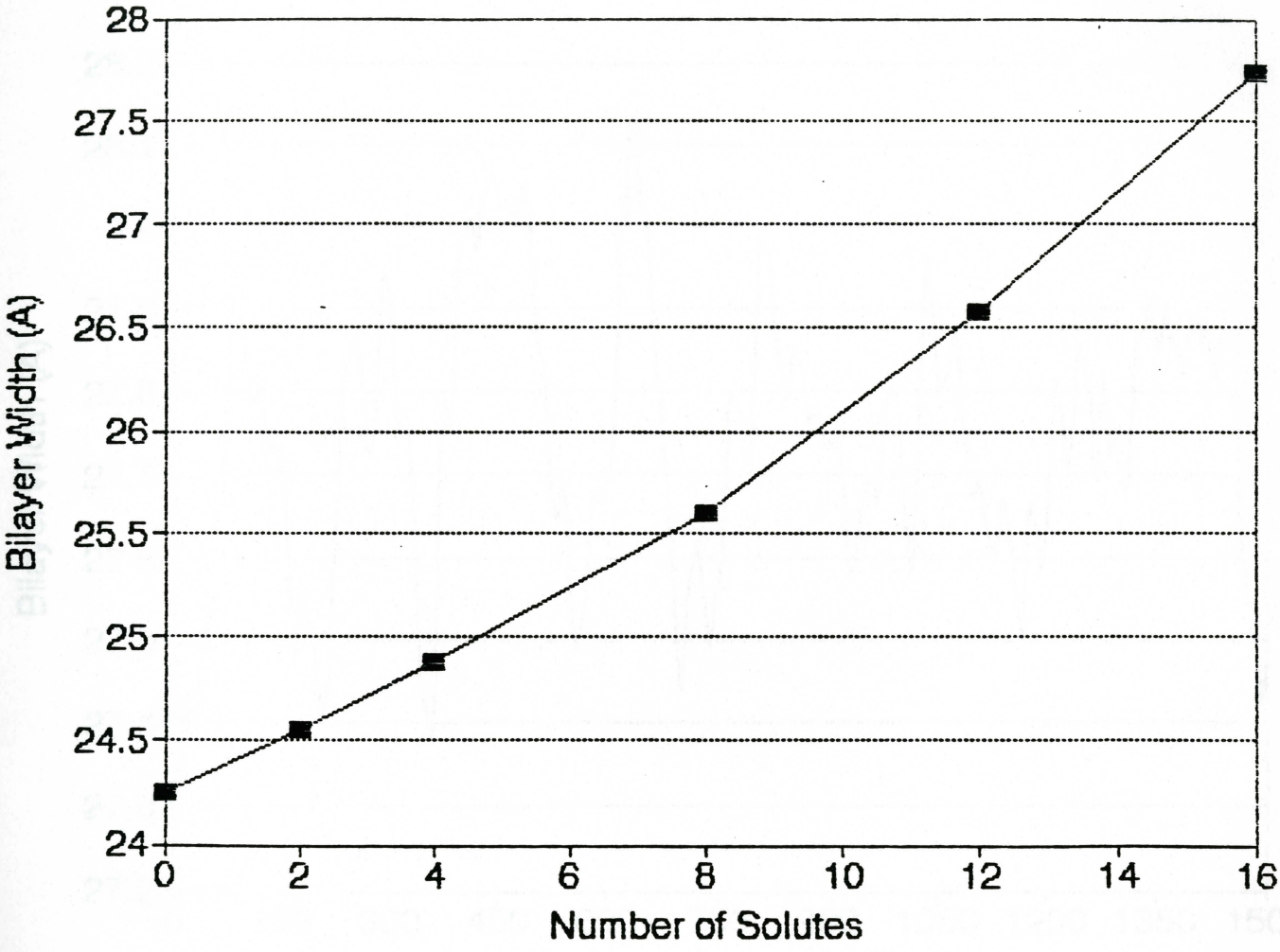

28
27.5
27
26.5
26
25.5
25
24.5
24
Bilayer Width (A)
0
2
4
6
8
10
12
14
16
Number of Solutes

Figure 4

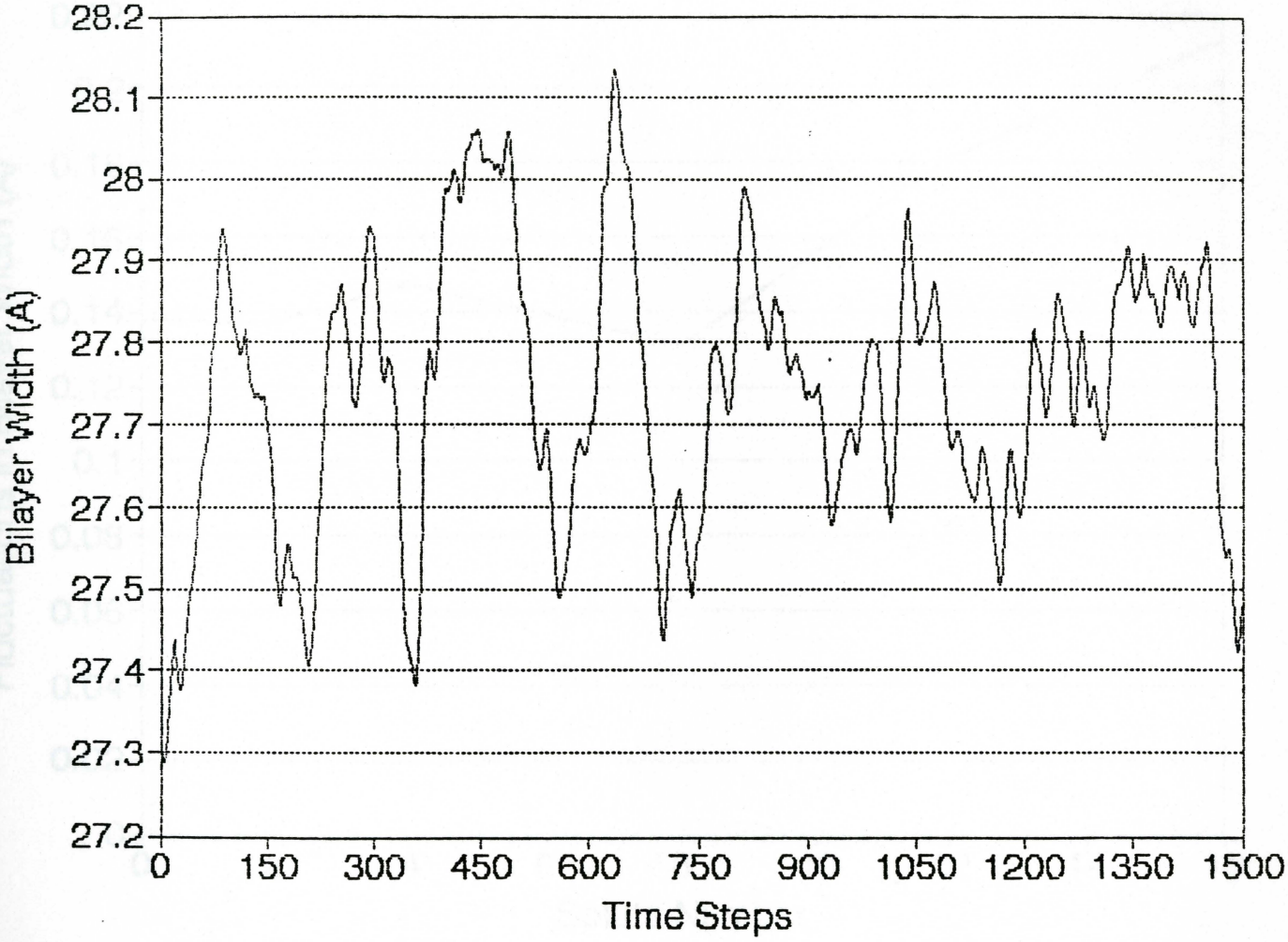

28.2
28.1
28
27.9
27.8
27.7
27.6
27.5
27.4
27.3
27.2
Bilayer Width (A)
0
150
300
450
600
750
900
1050
1200
1350
1500
Time Steps

Figure 5

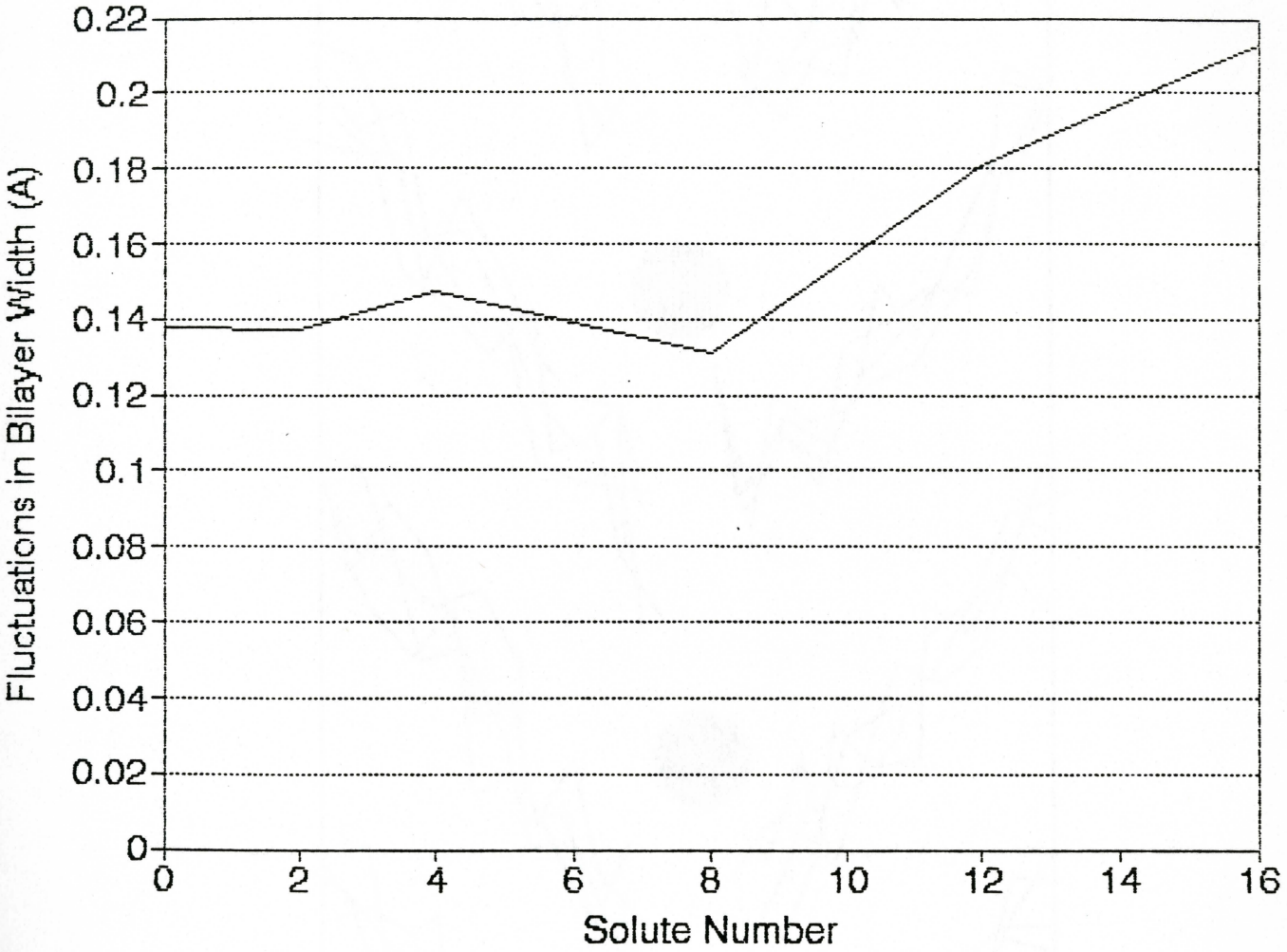

0.22
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
Fluctuations in Bilayer Width (A)
0
2
4
6
8
10
12
14
16
Solute Number

Figure 6

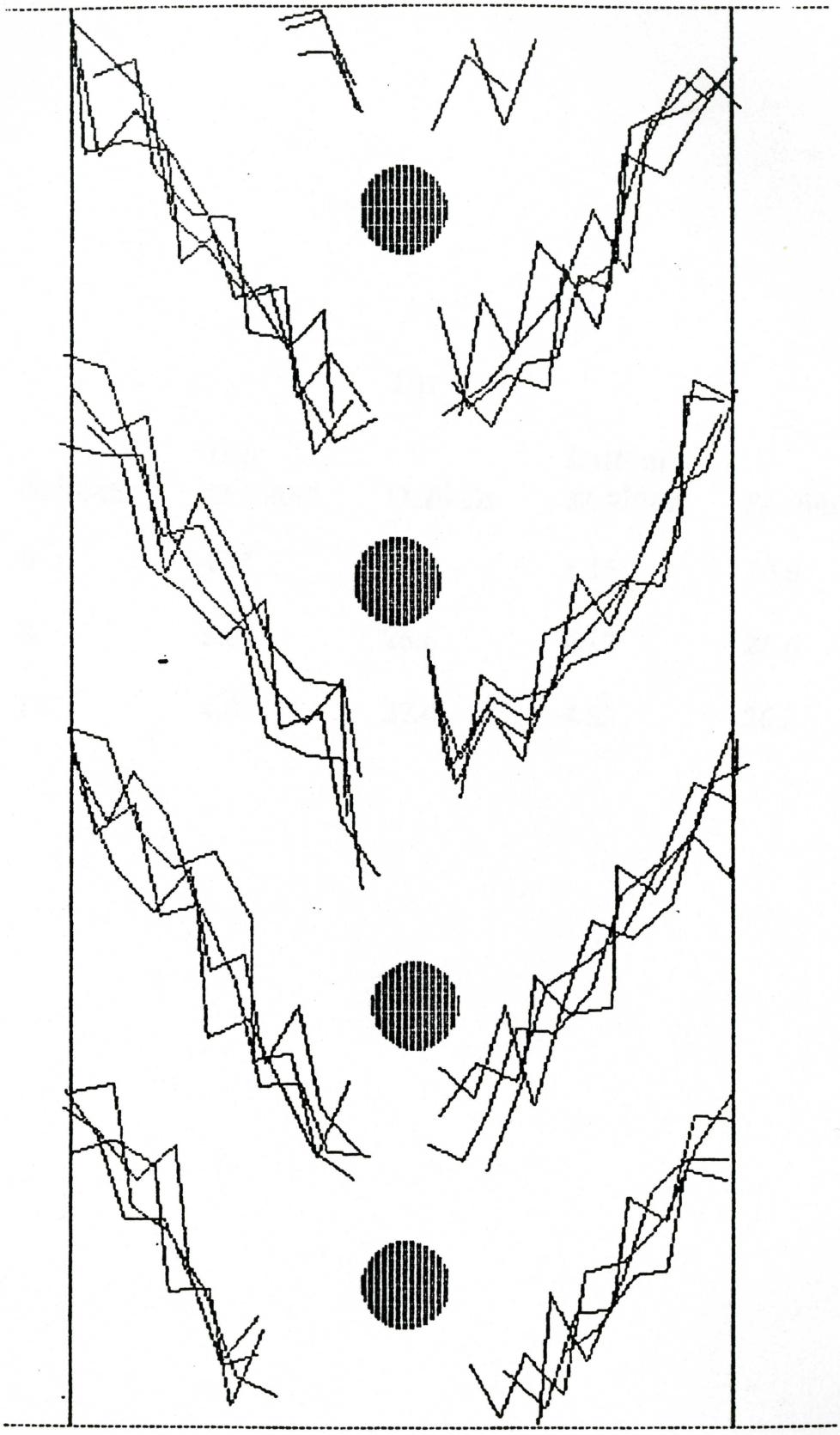


Figure 7

	Tilt			
	Top		Bottom	
Solutes	xz plane	yz plane	xz plane	yz plane
0	4.41	26.9	5.15	26.5
8	5.04	26.6	4.17	26.6
16	4.41	27.4	4.53	26.2

Table 2

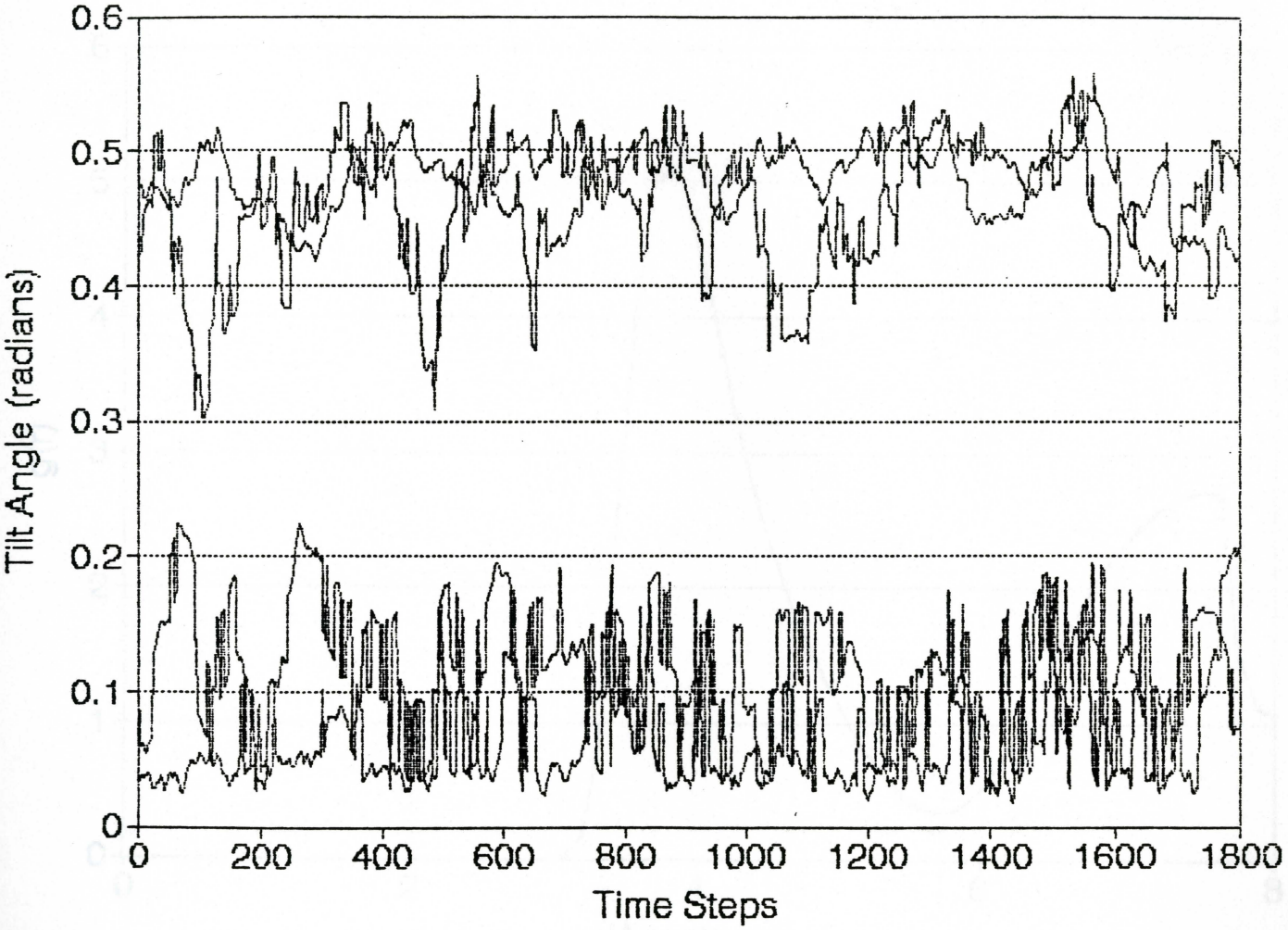

0.6
0.5
0.4
0.3
0.2
0.1
0
Tilt Angle (radians)
0
200
400
600
800
1000
1200
1400
1600
1800
Time Steps

Figure 8

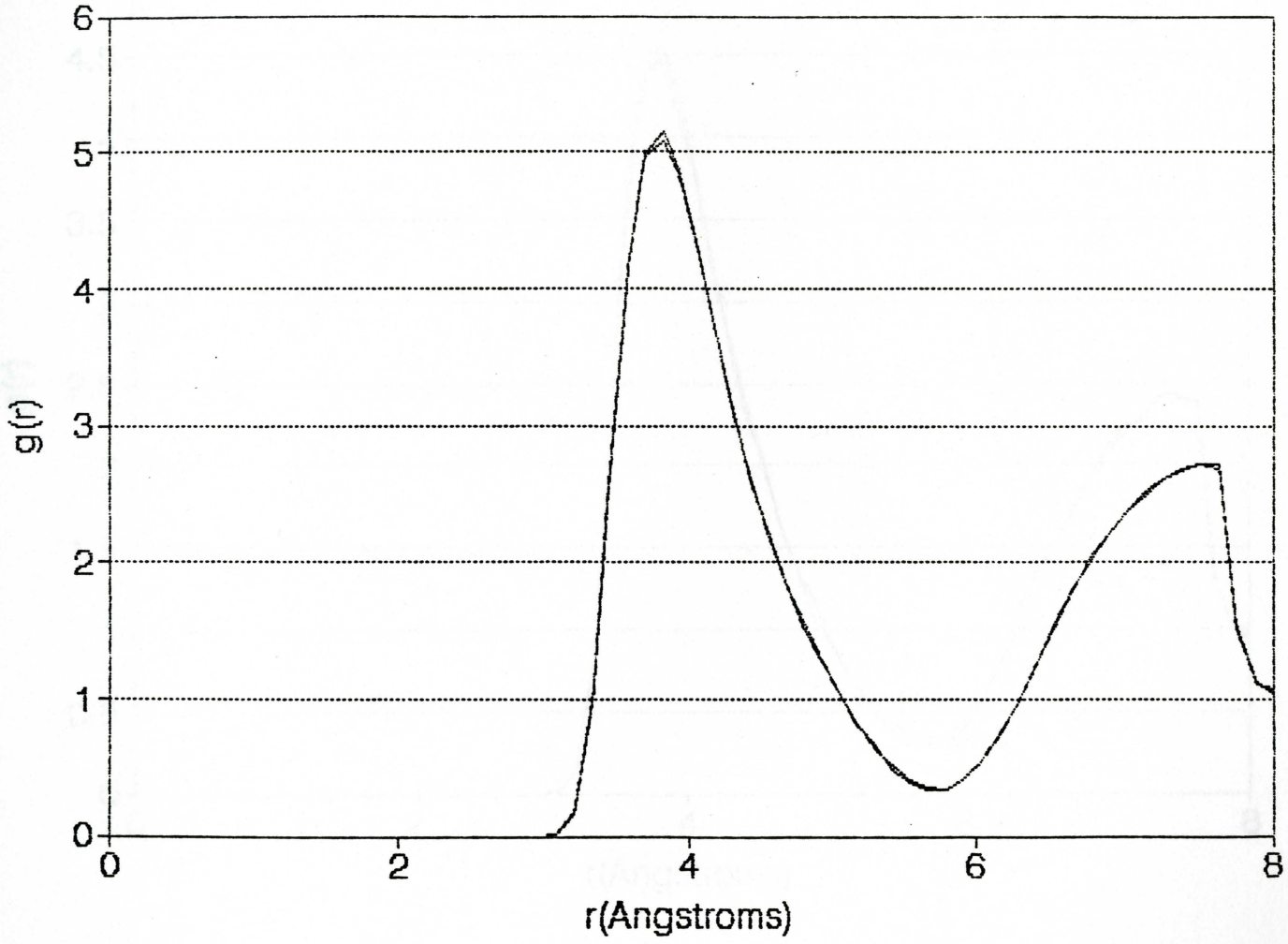


Figure 9A

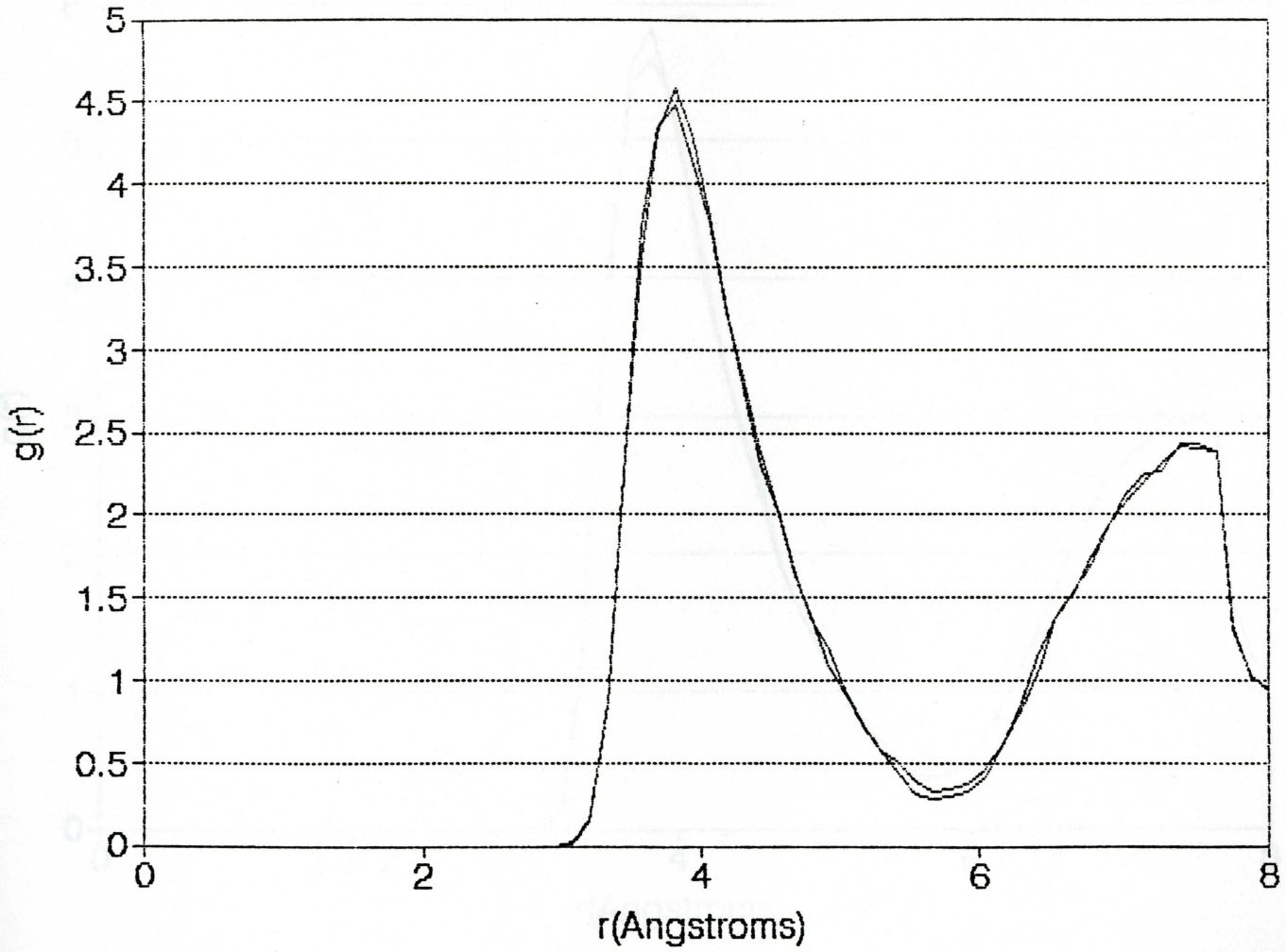


Figure 9B

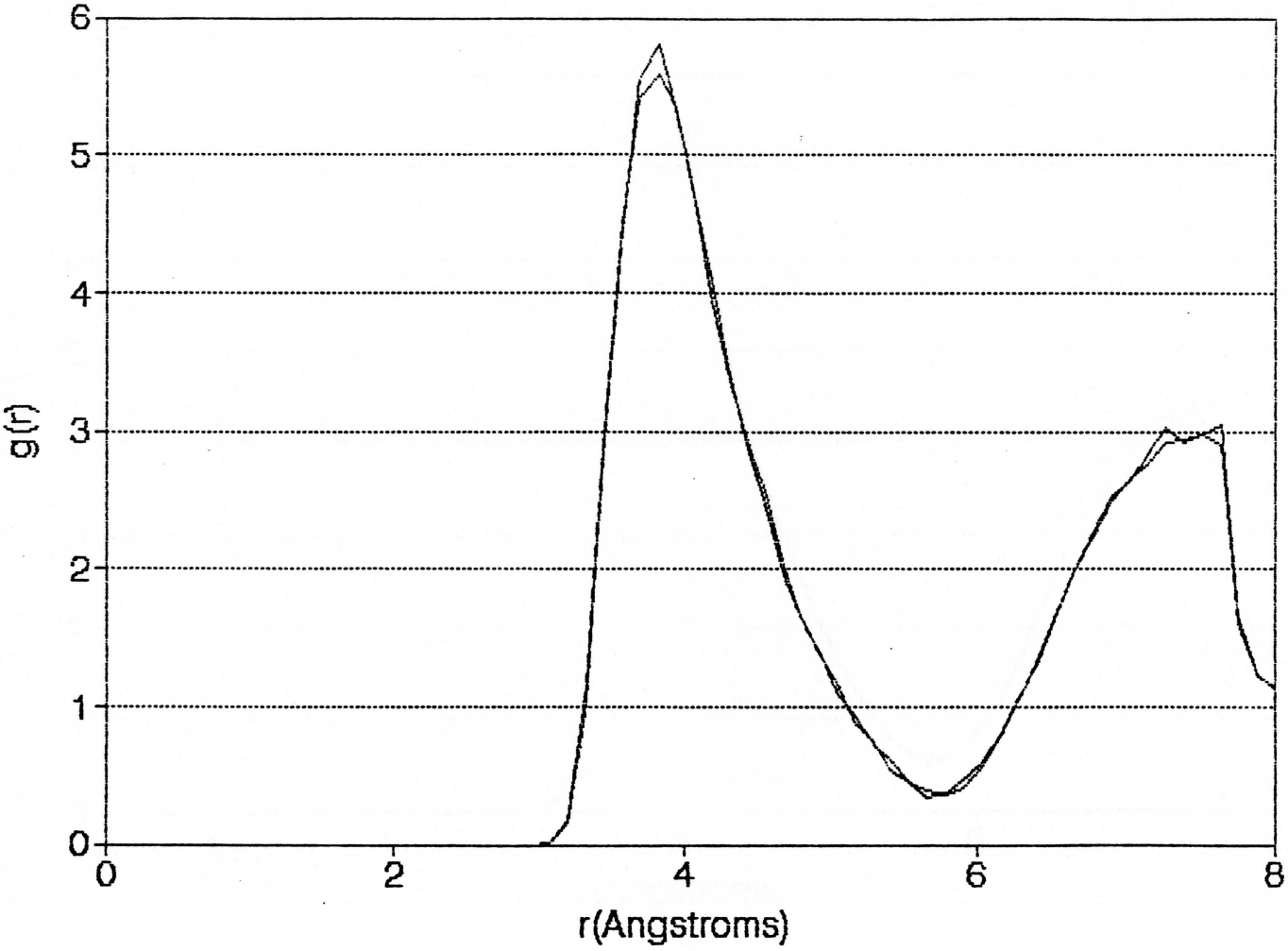


Figure 9C

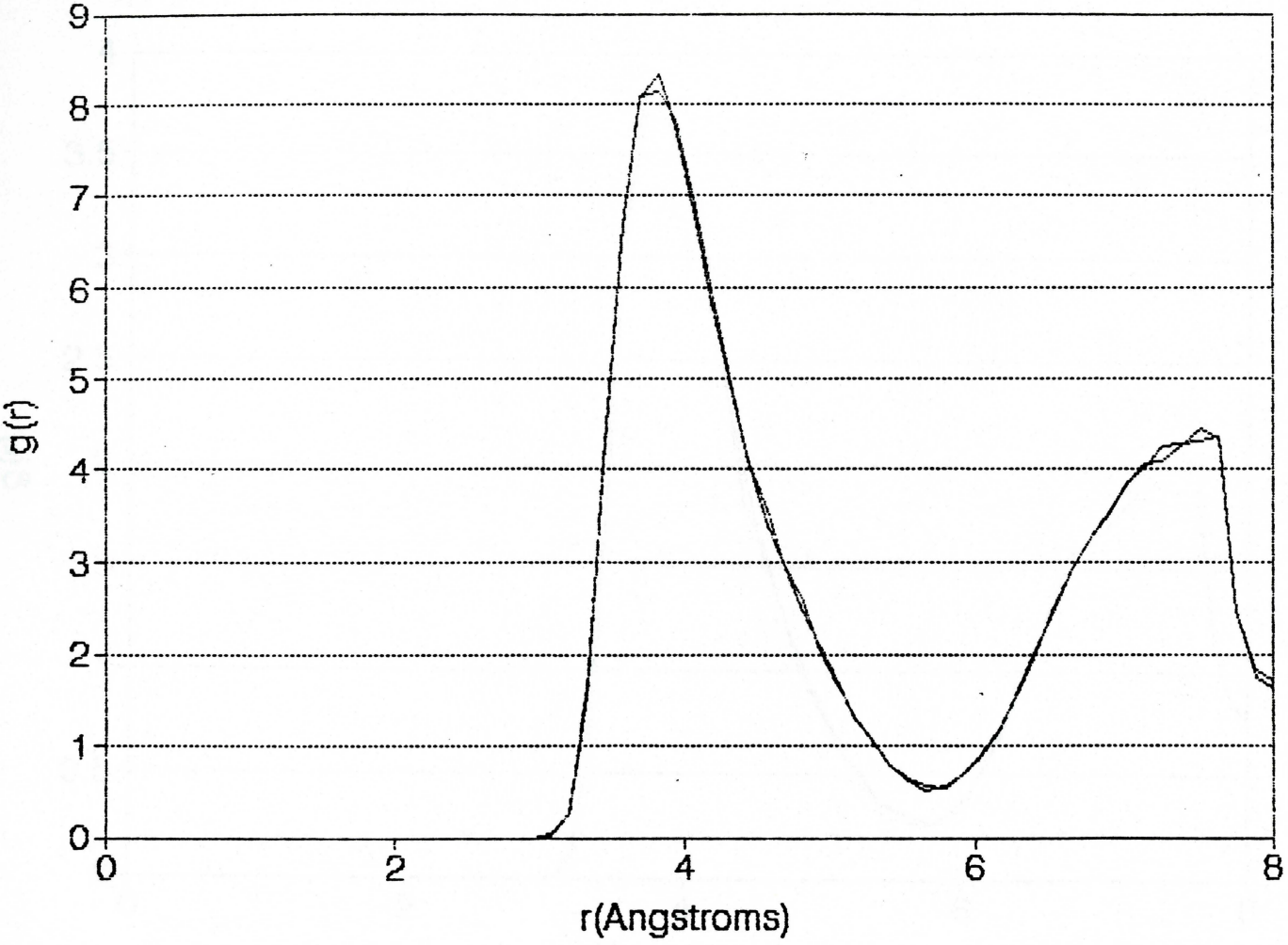


Figure 9D

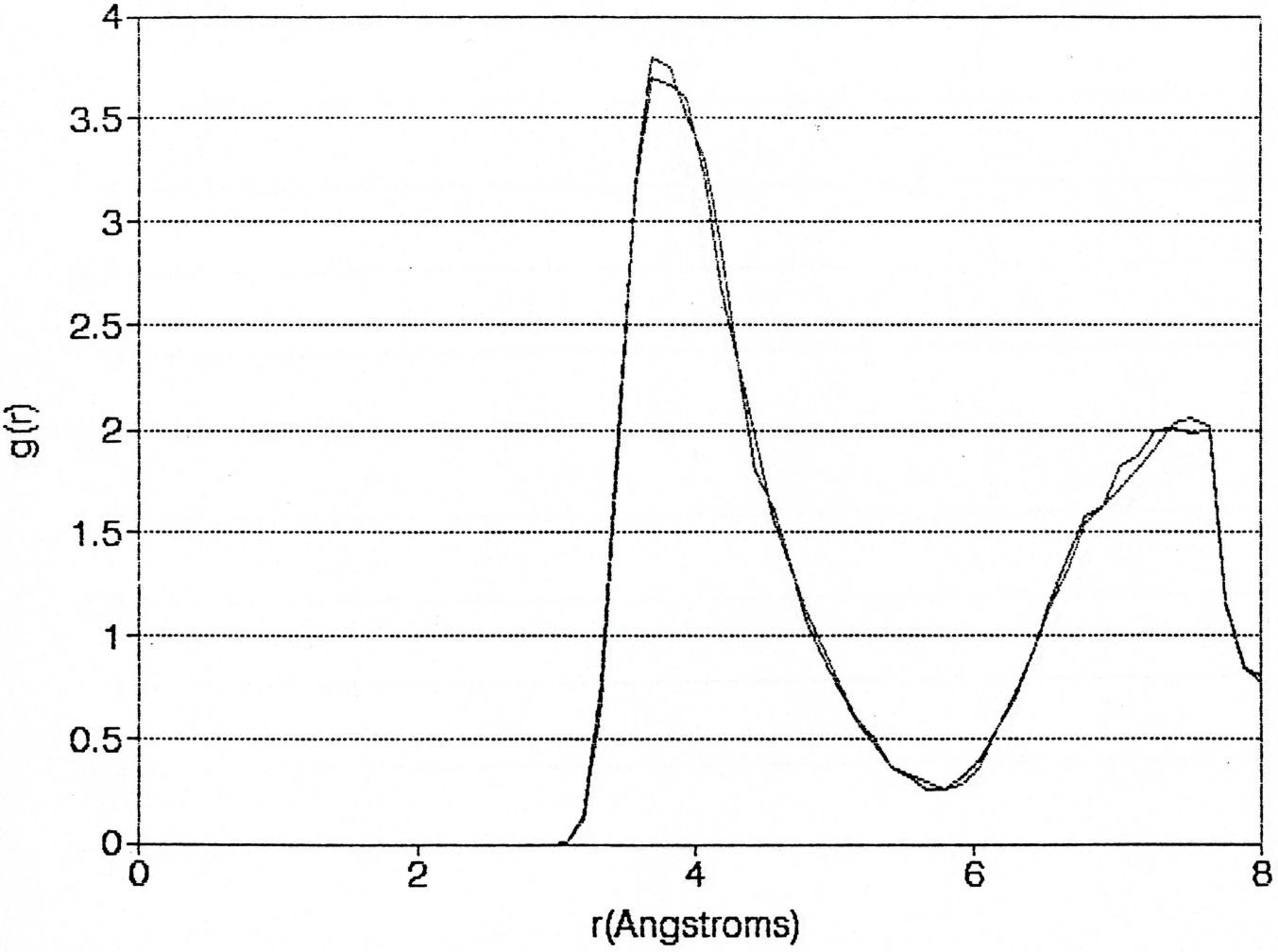


Figure 9E

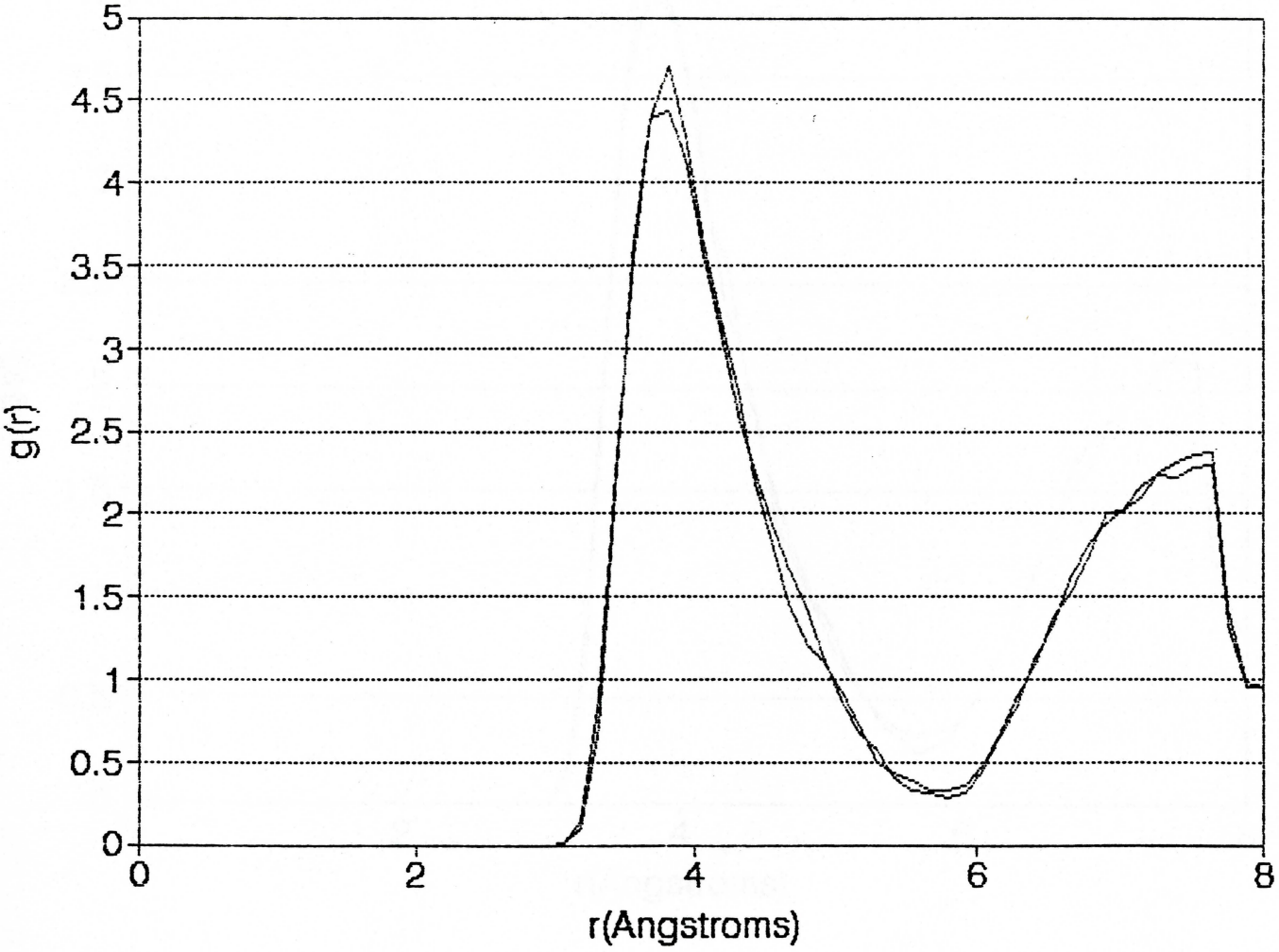


Figure 9F

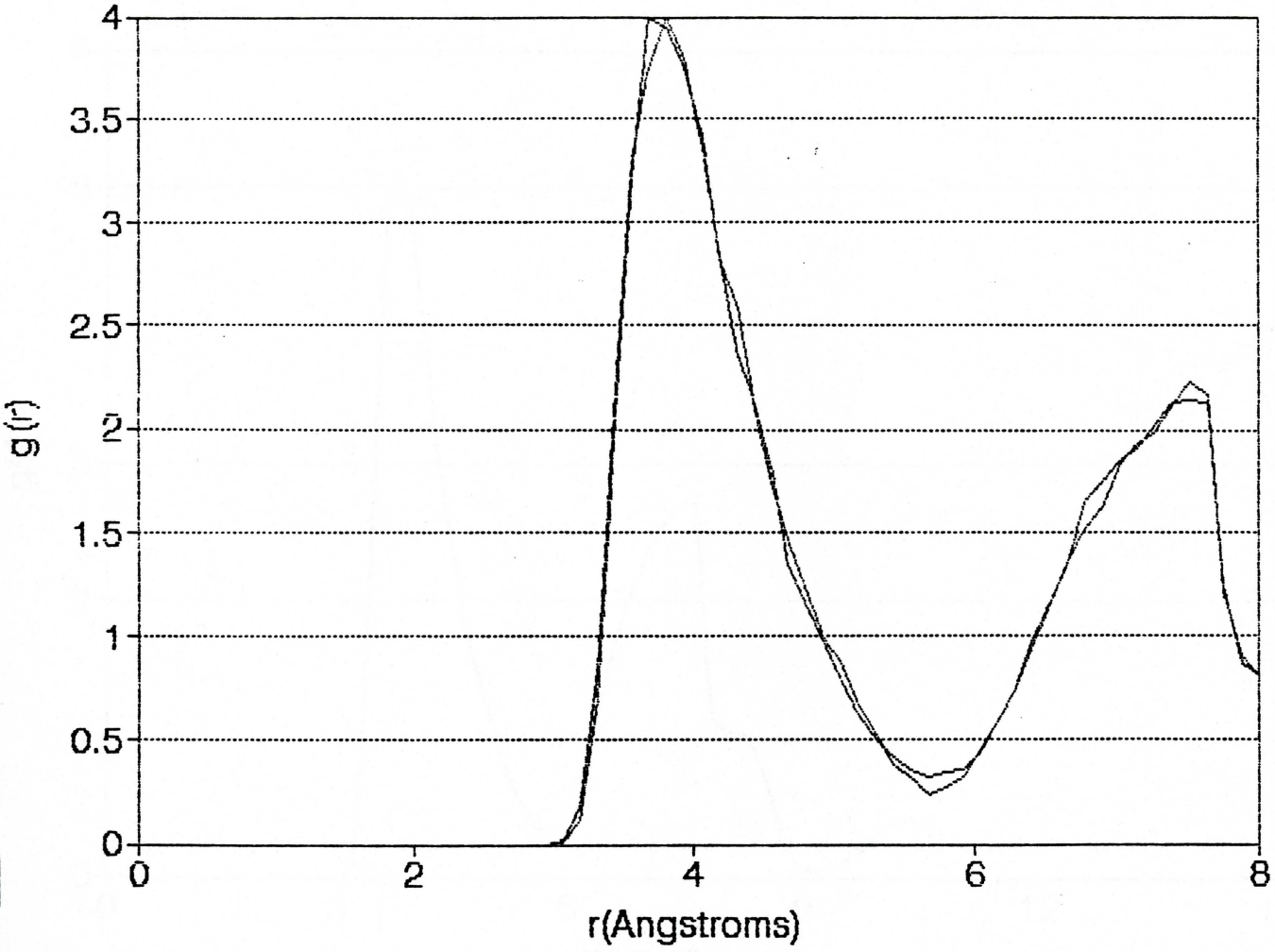


Figure 9G

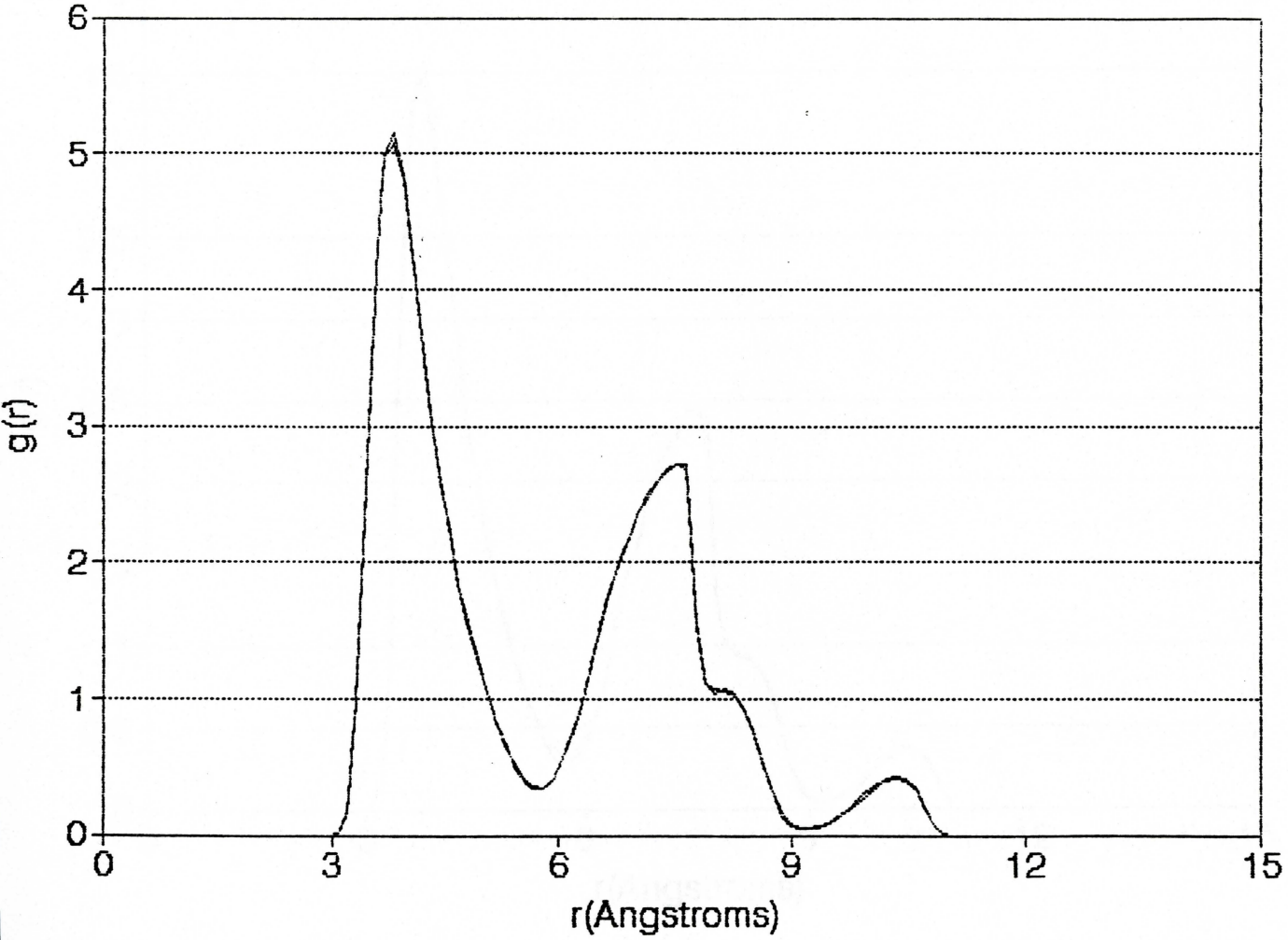


Figure 10A

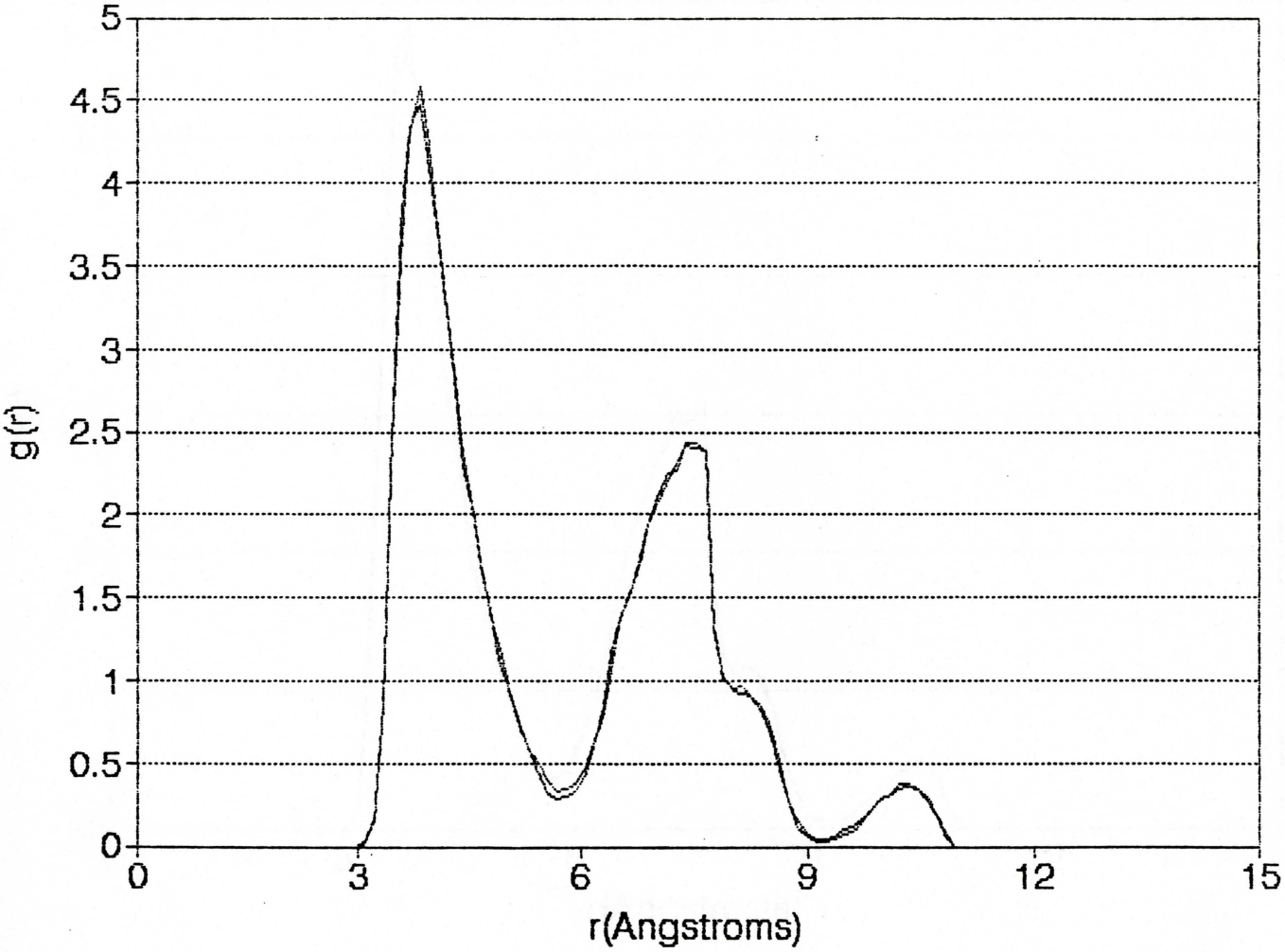


Figure 10B

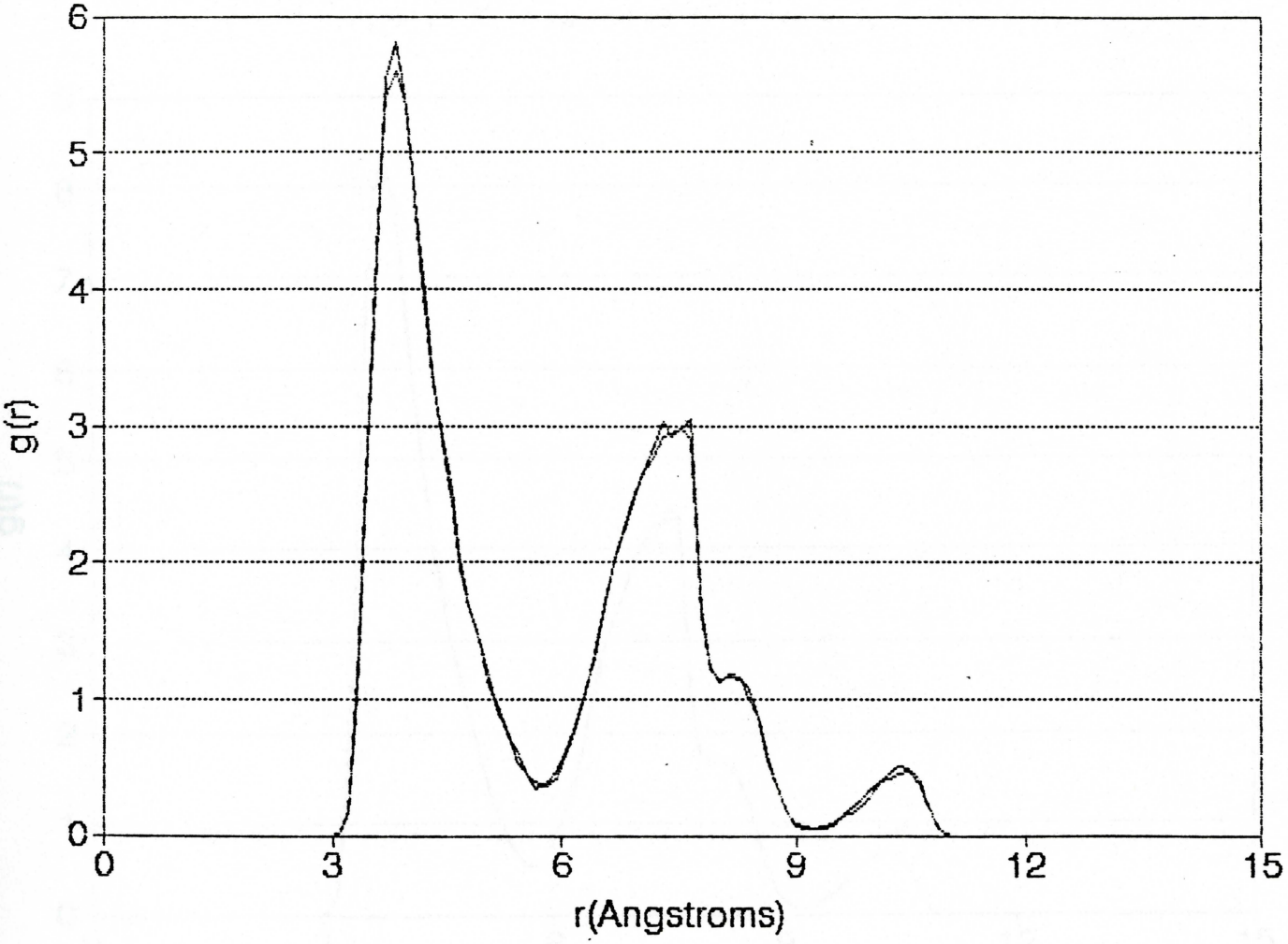


Figure 10C

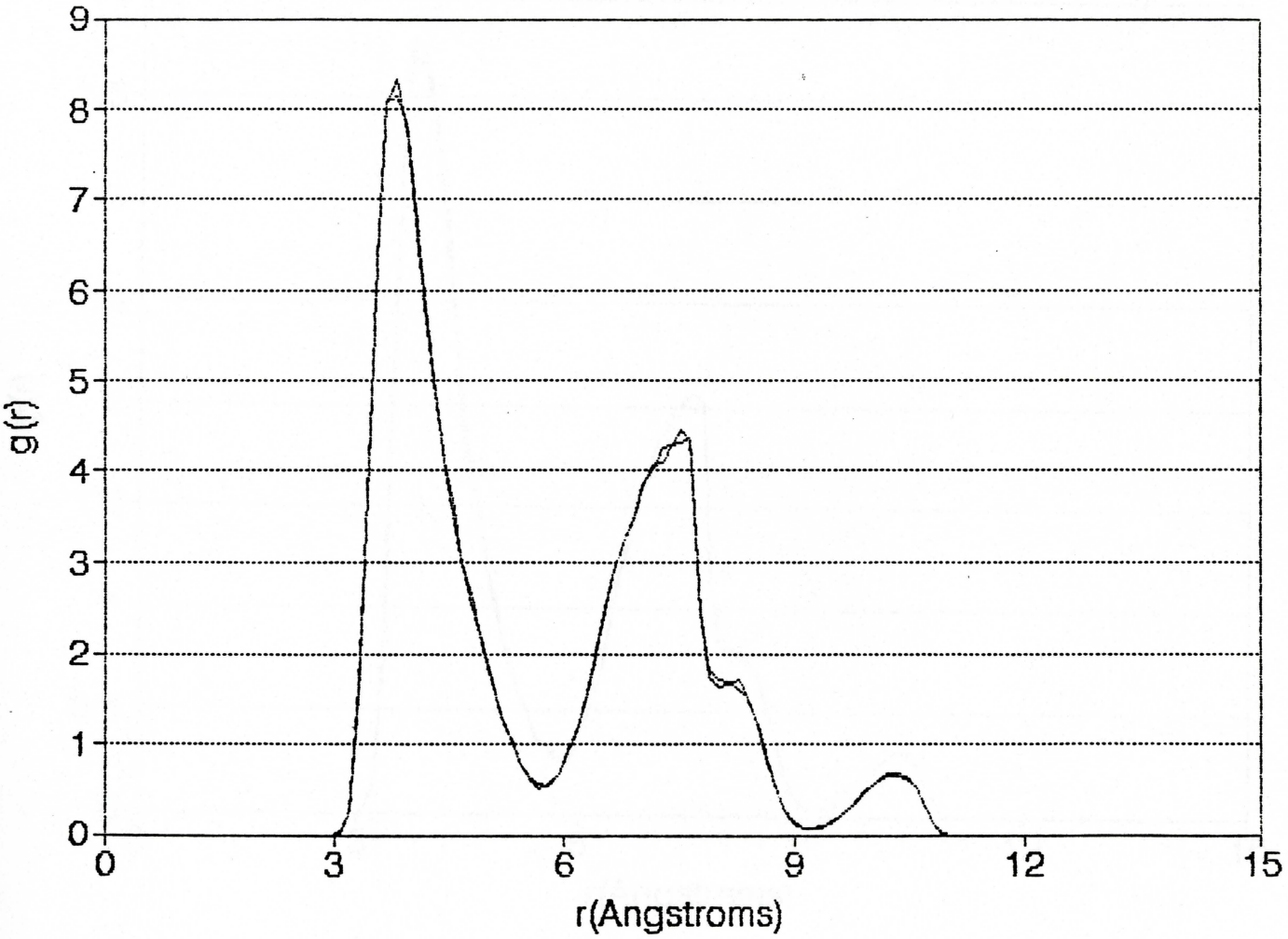


Figure 10D

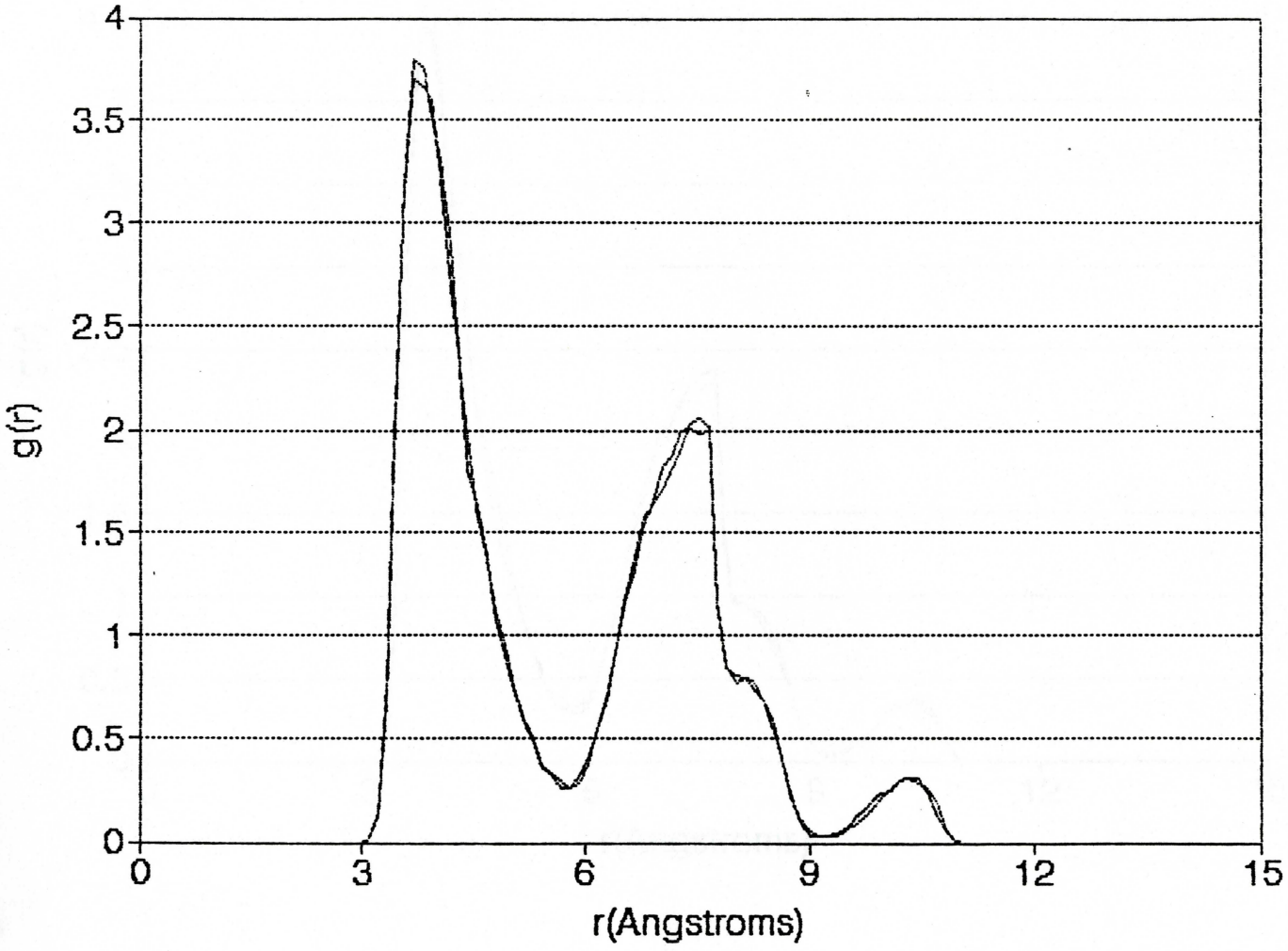


Figure 10E

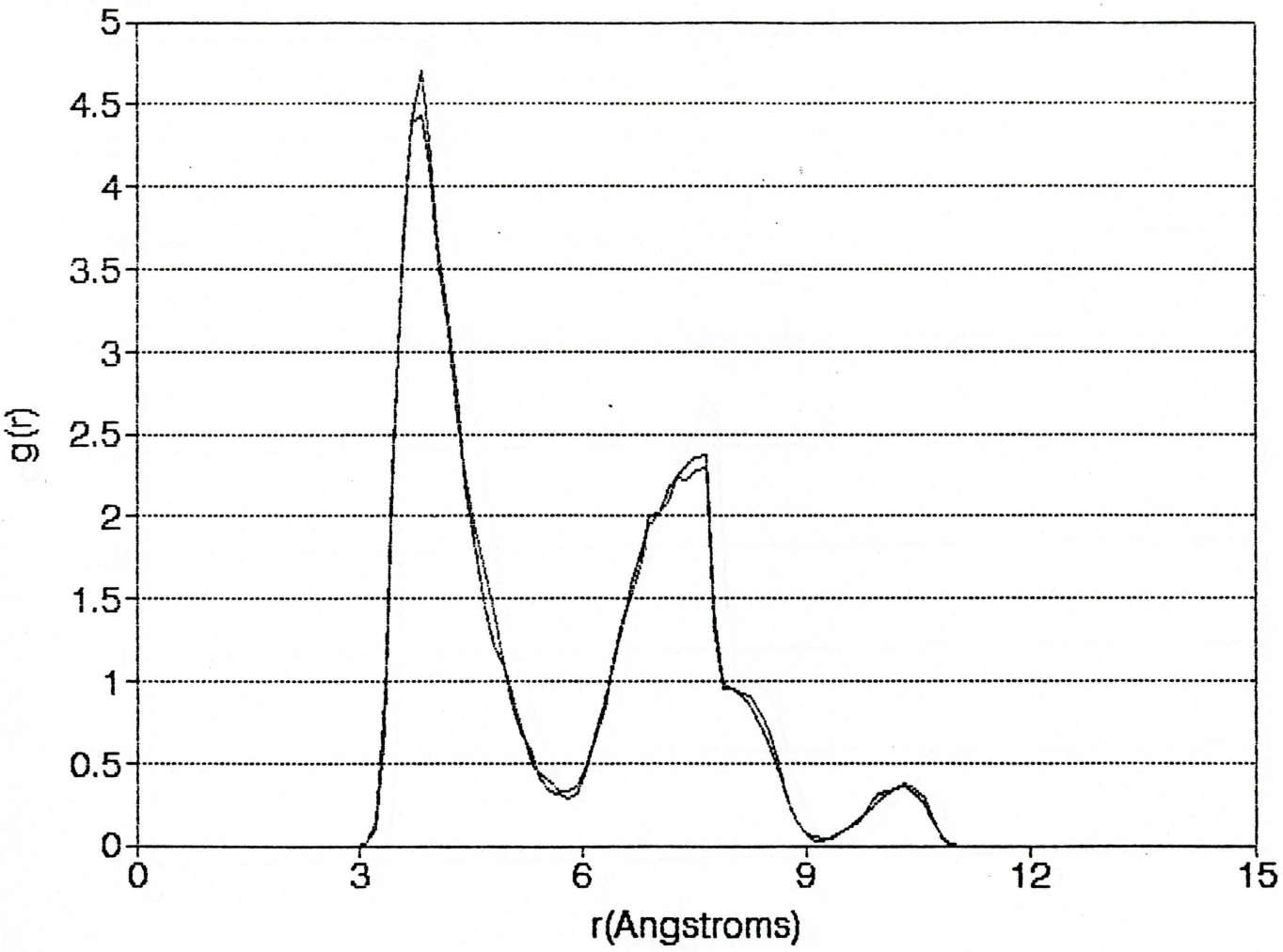


Figure 10F

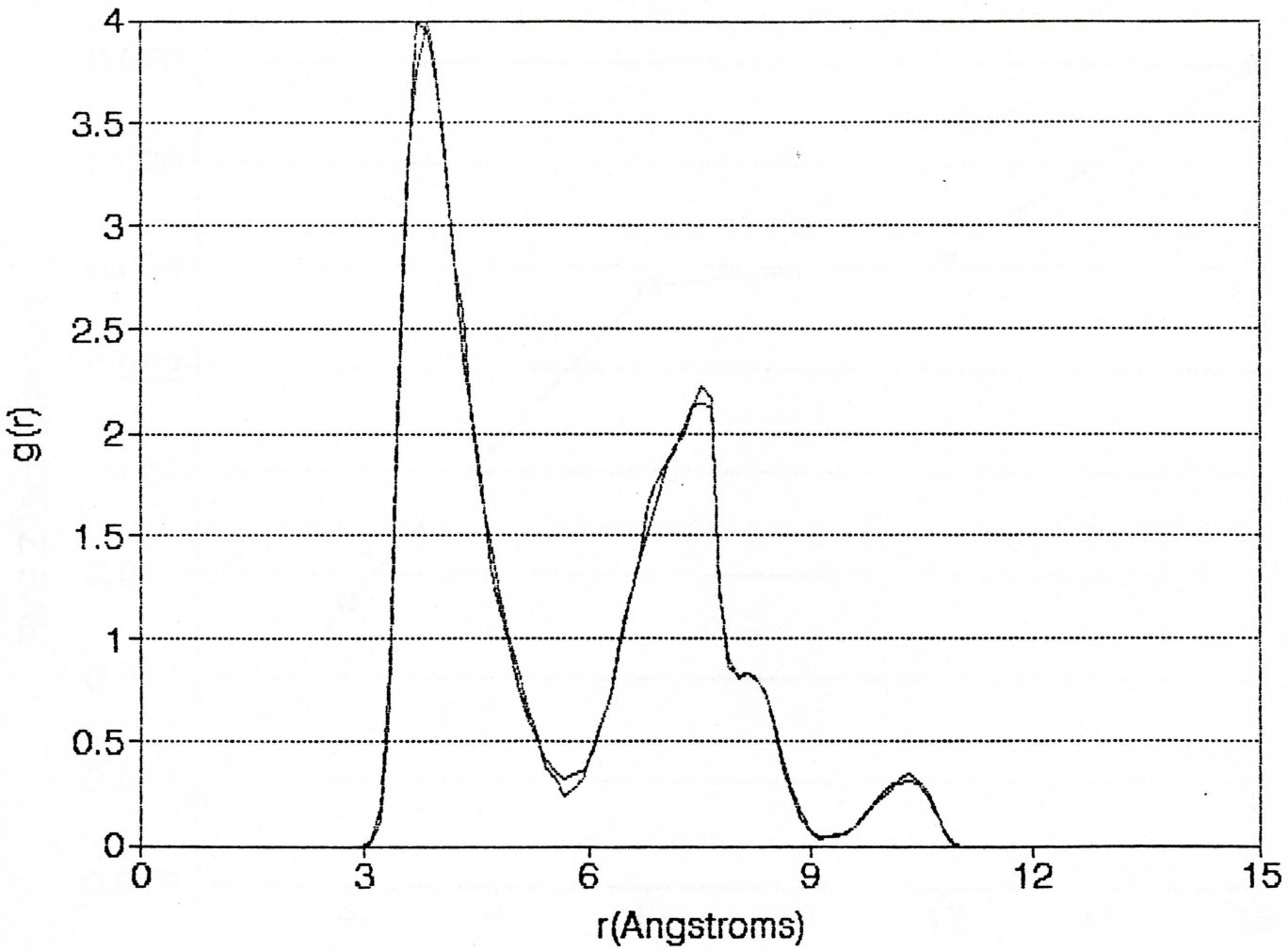


Figure 10G

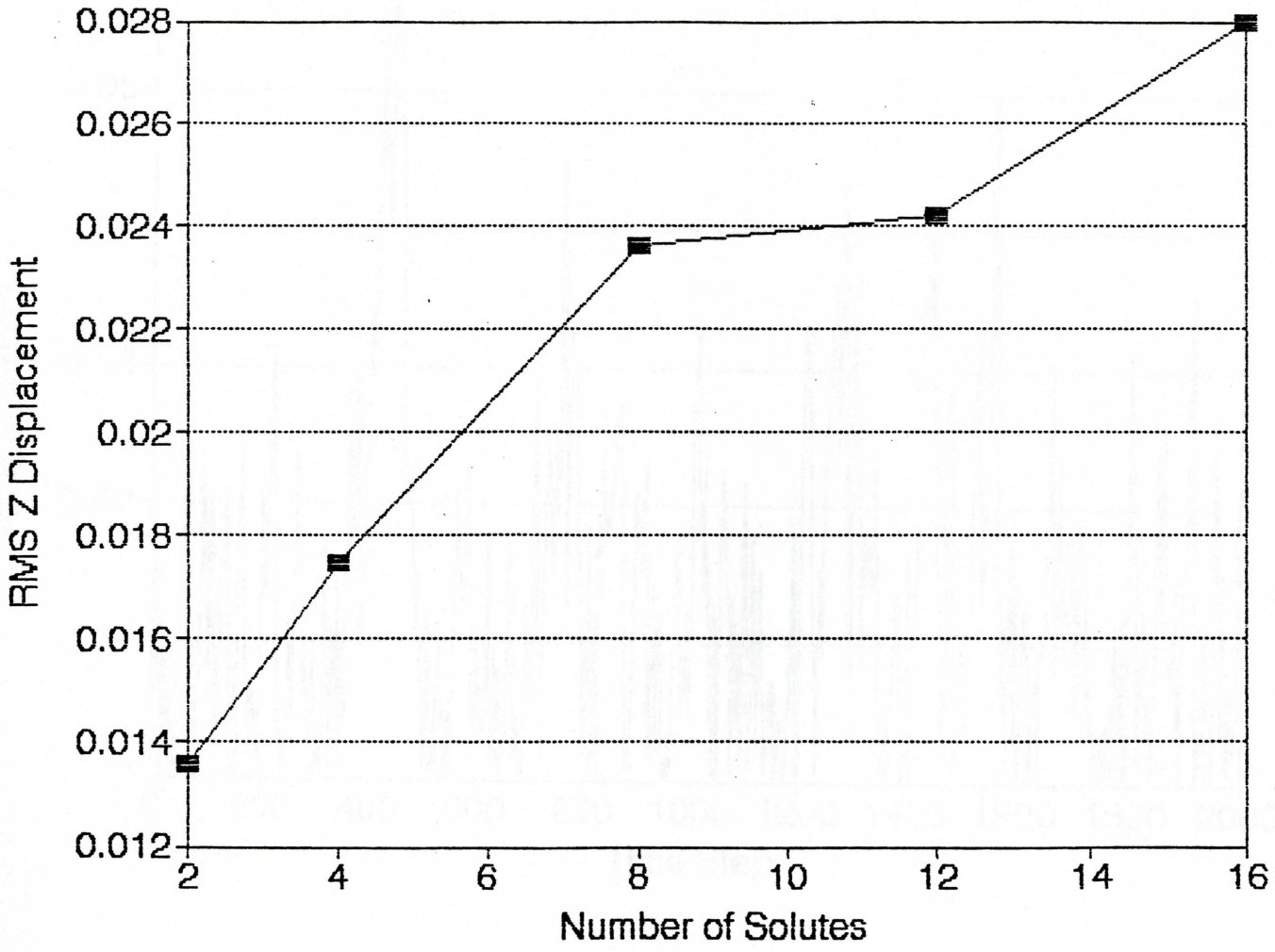

0.028
0.026
0.024
0.022
0.02
0.018
0.016
0.014
0.012
RMS Z Displacement
2
4
6
8
10
12
14
16
Number of Solutes

Figure 11

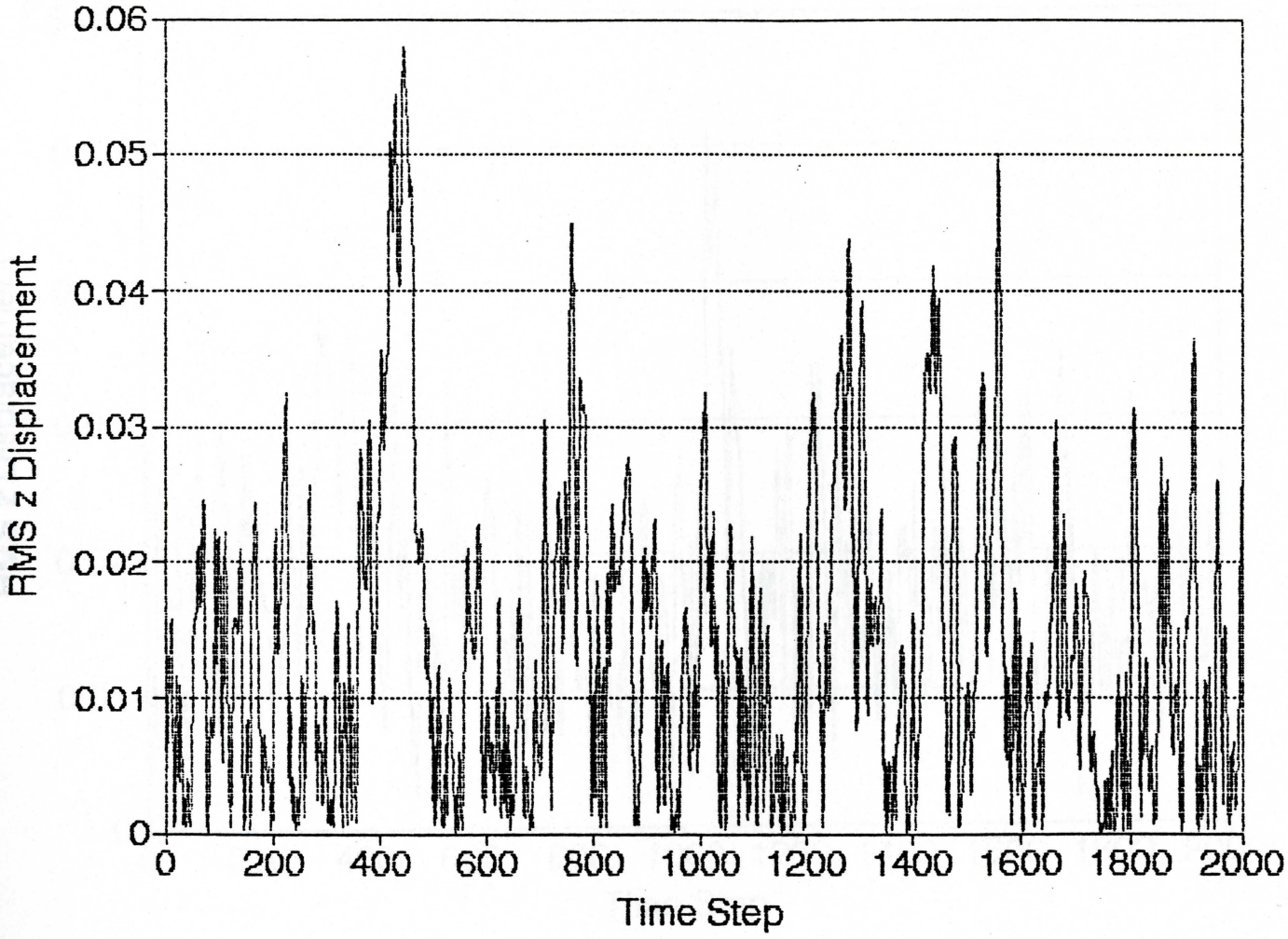

RMS z Displacement
0.06
0.05
0.04
0.03
0.02
0.01
0
0
200
400
600
800
1000
1200
1400
1600
1800
2000
Time Step

Figure 12A

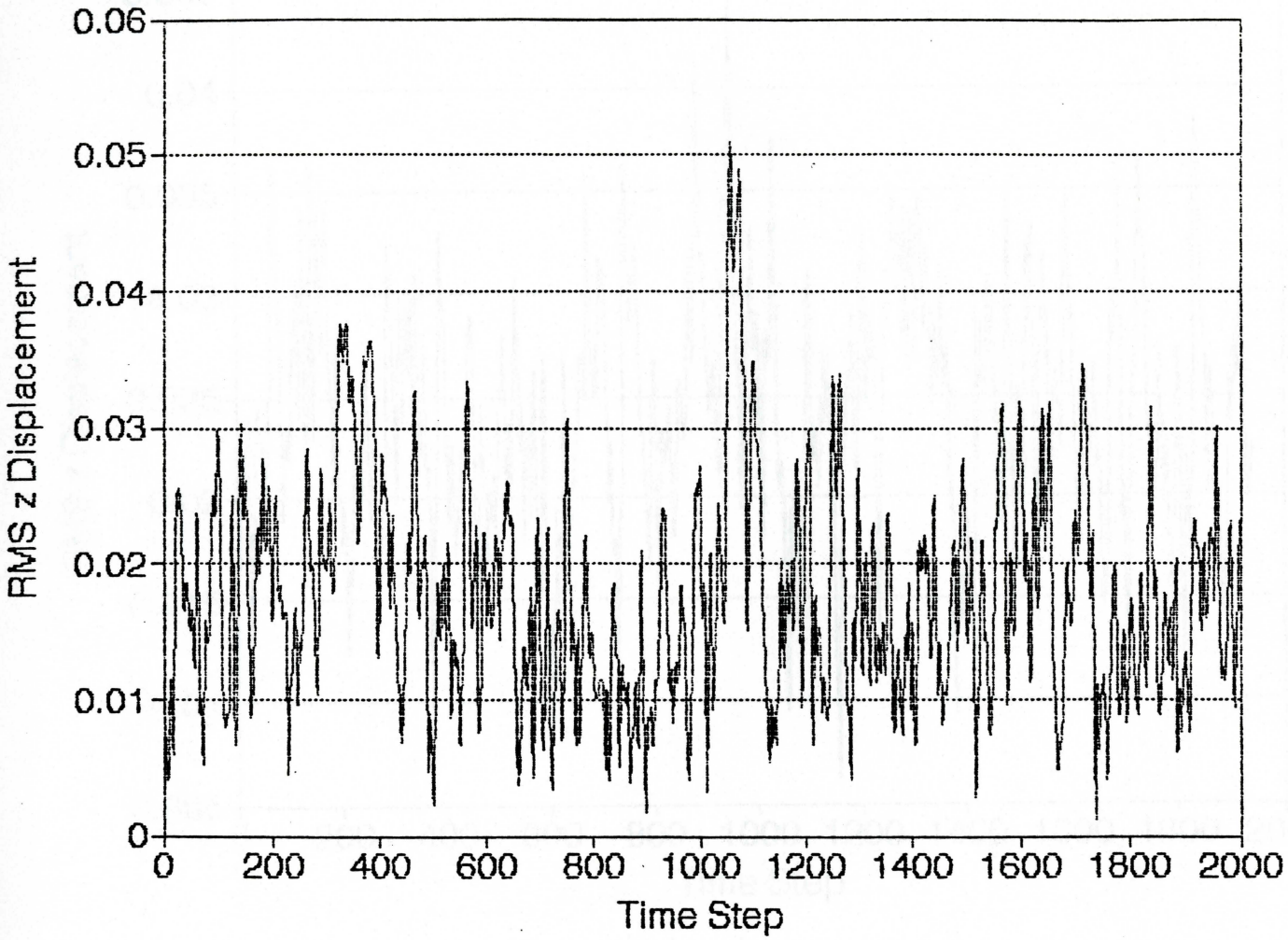

0.06
0.05
0.04
0.03
0.02
0.01
0
RMS z Displacement
0
200
400
600
800
1000
1200
1400
1600
1800
2000
Time Step

Figure 12B

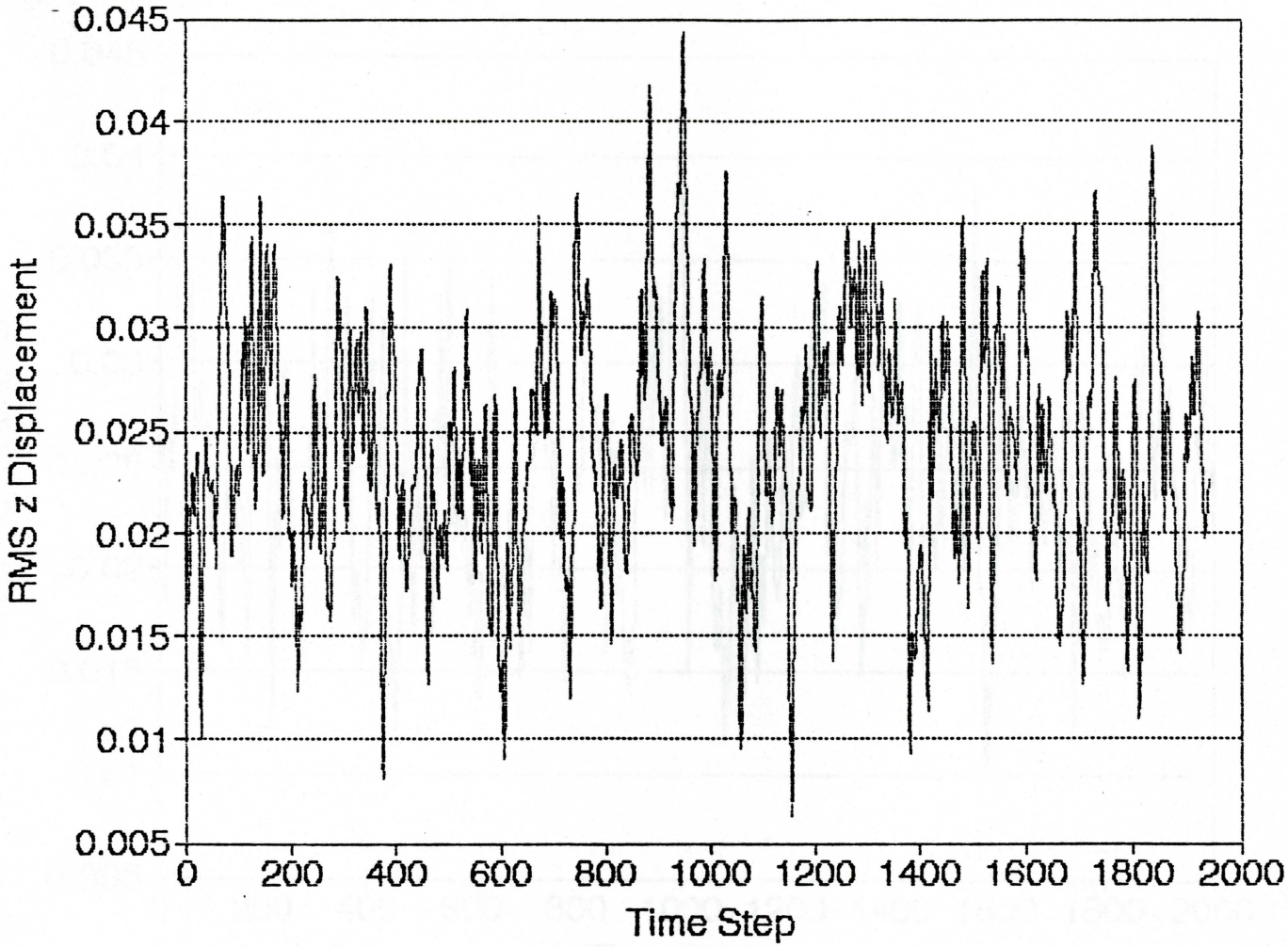

0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
RMS z Displacement
0
200
400
600
800
1000
1200
1400
1600
1800
2000
Time Step

Figure 12C

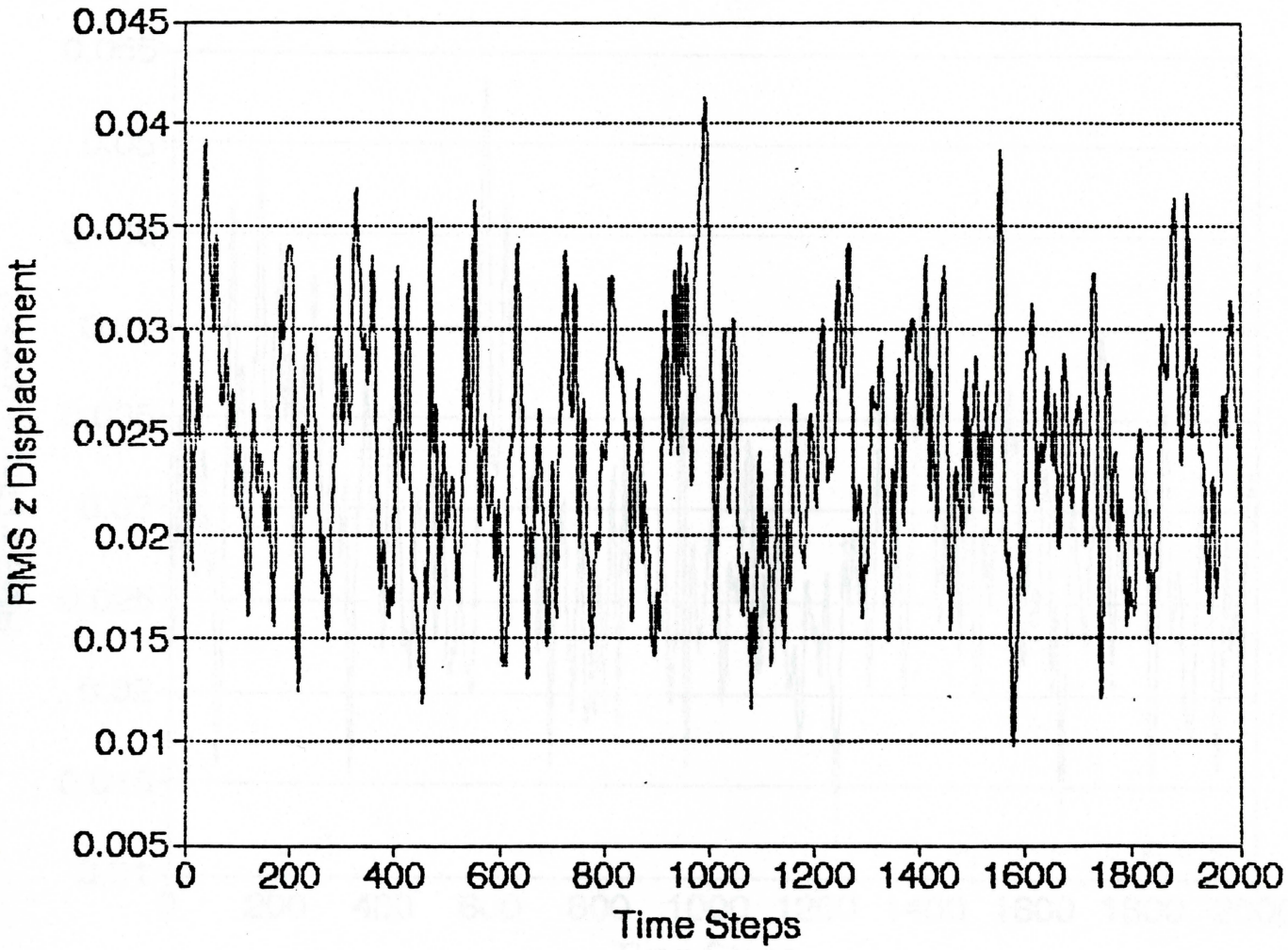

0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
RMS z Displacement
0
200
400
600
800
1000
1200
1400
1600
1800
2000
Time Steps

Figure 12D

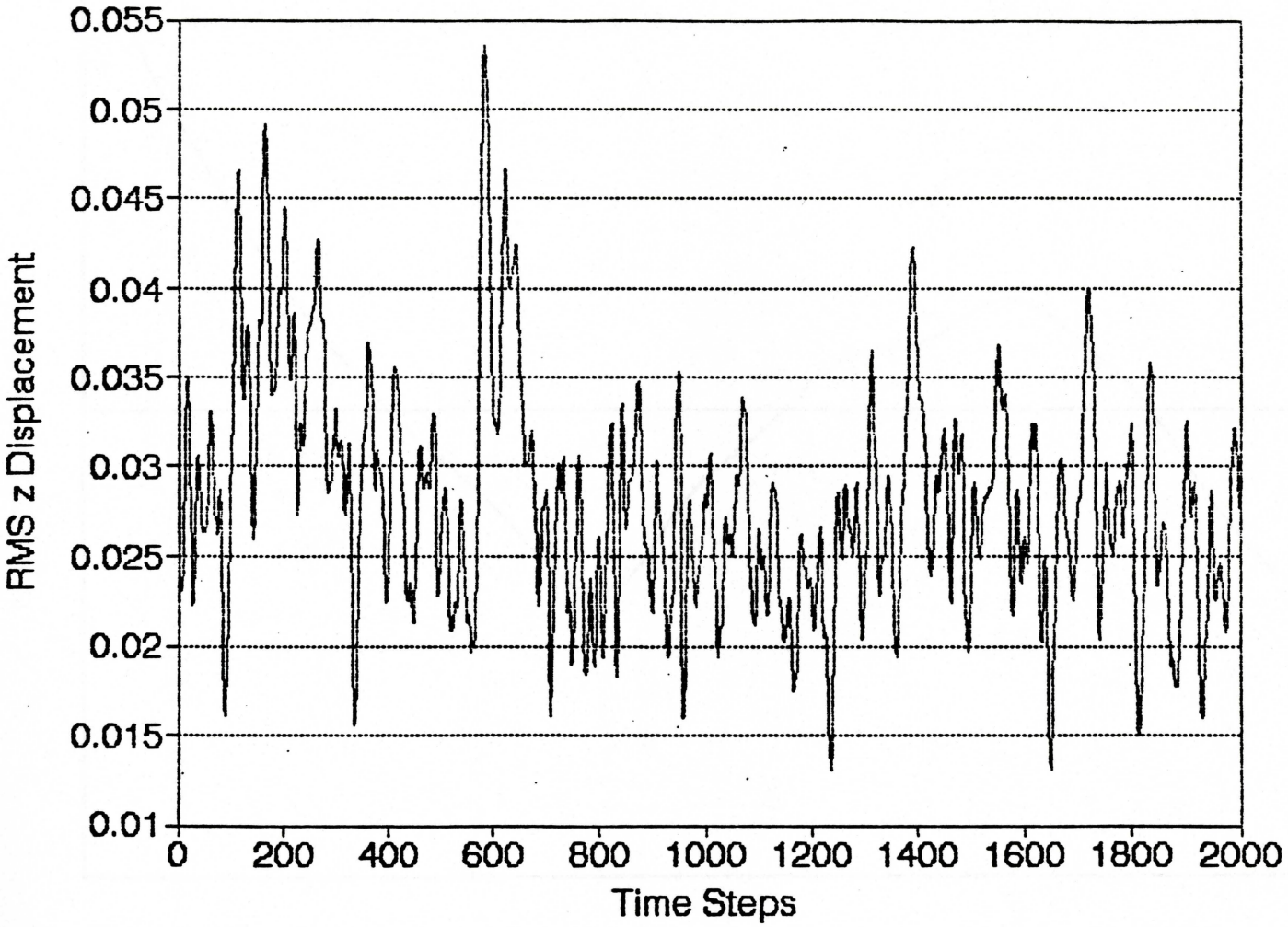

0.055
0.05
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
RMS z Displacement
0
200
400
600
800
1000
1200
1400
1600
1800
2000
Time Steps

Figure 12E

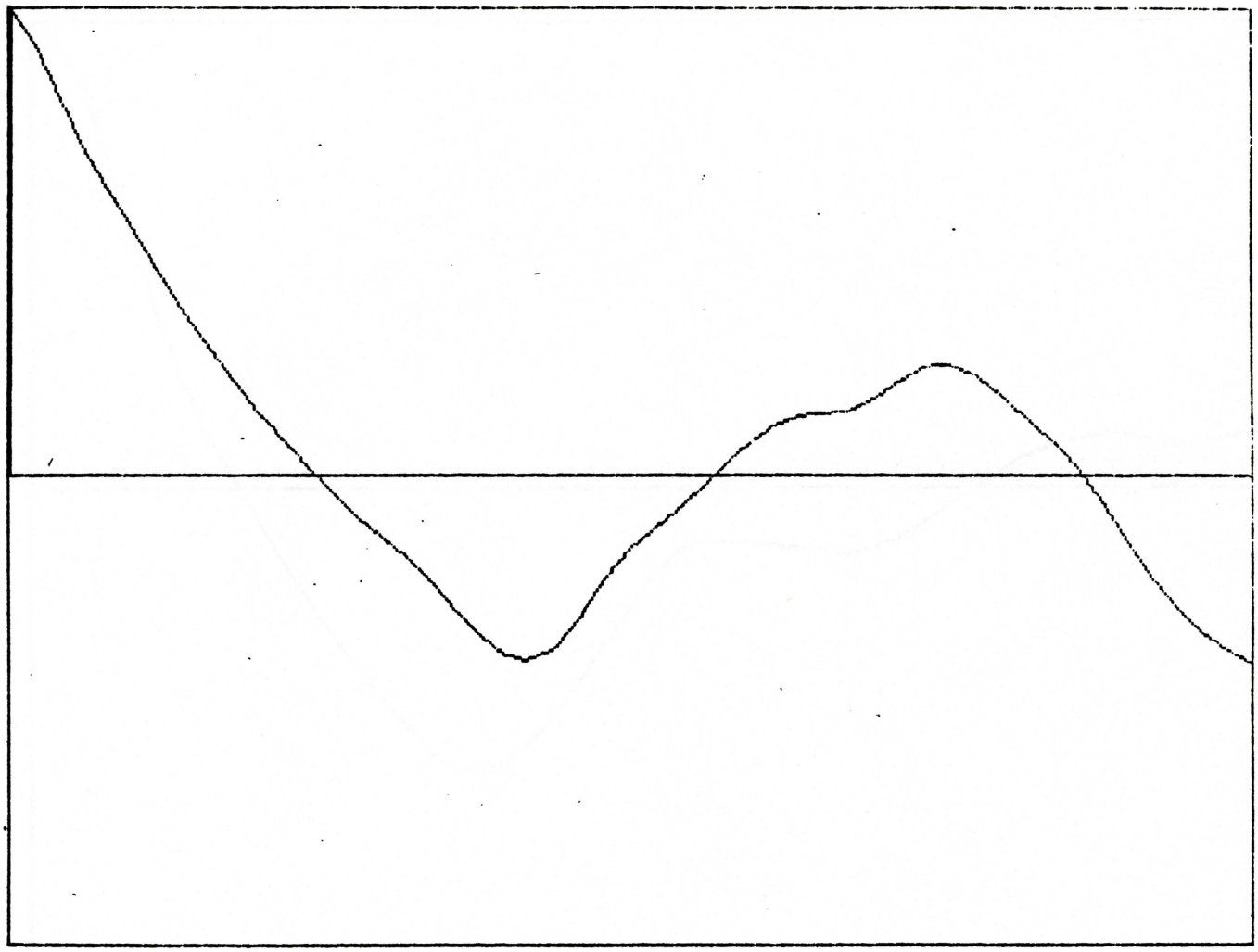


Figure 13A

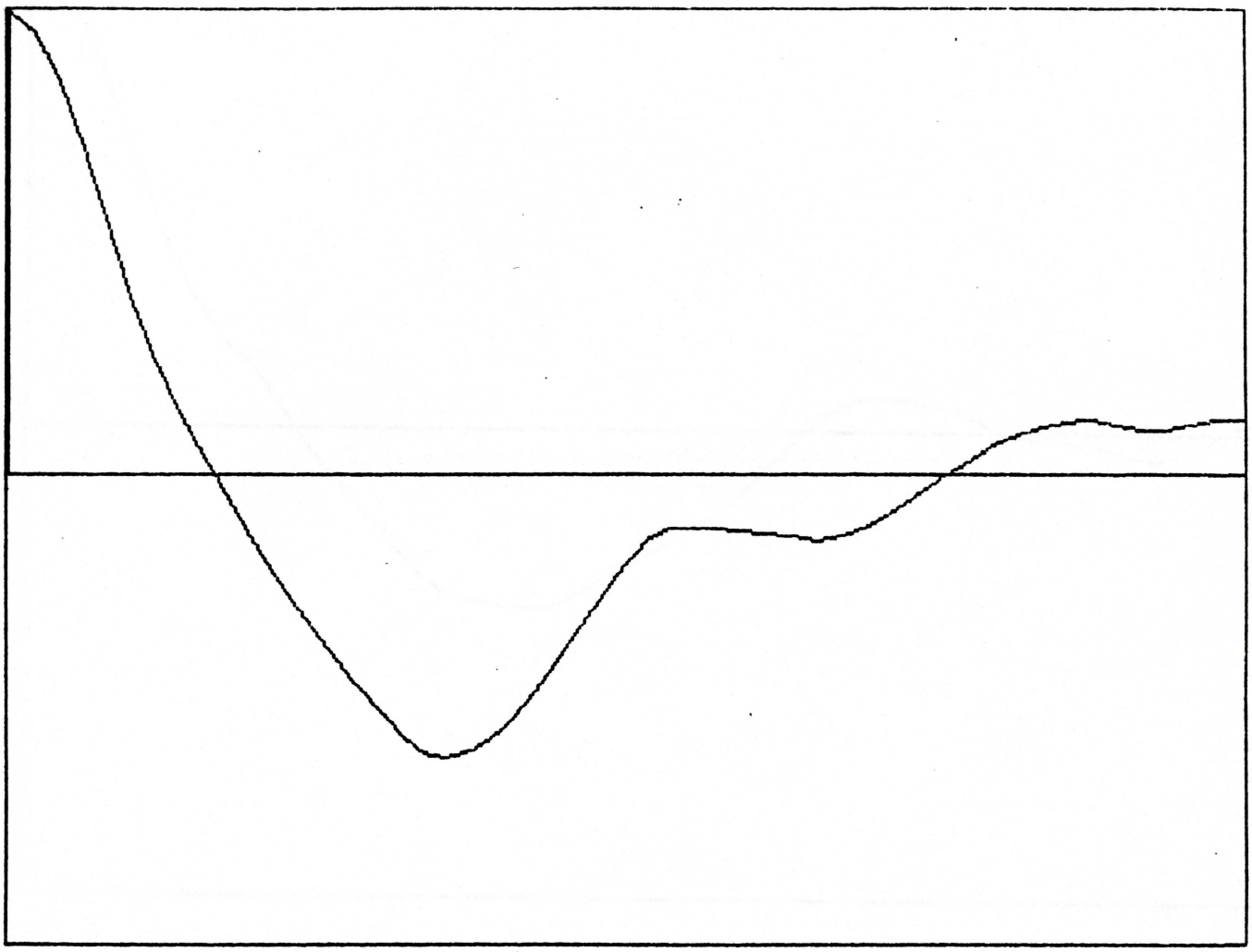


Figure 13B

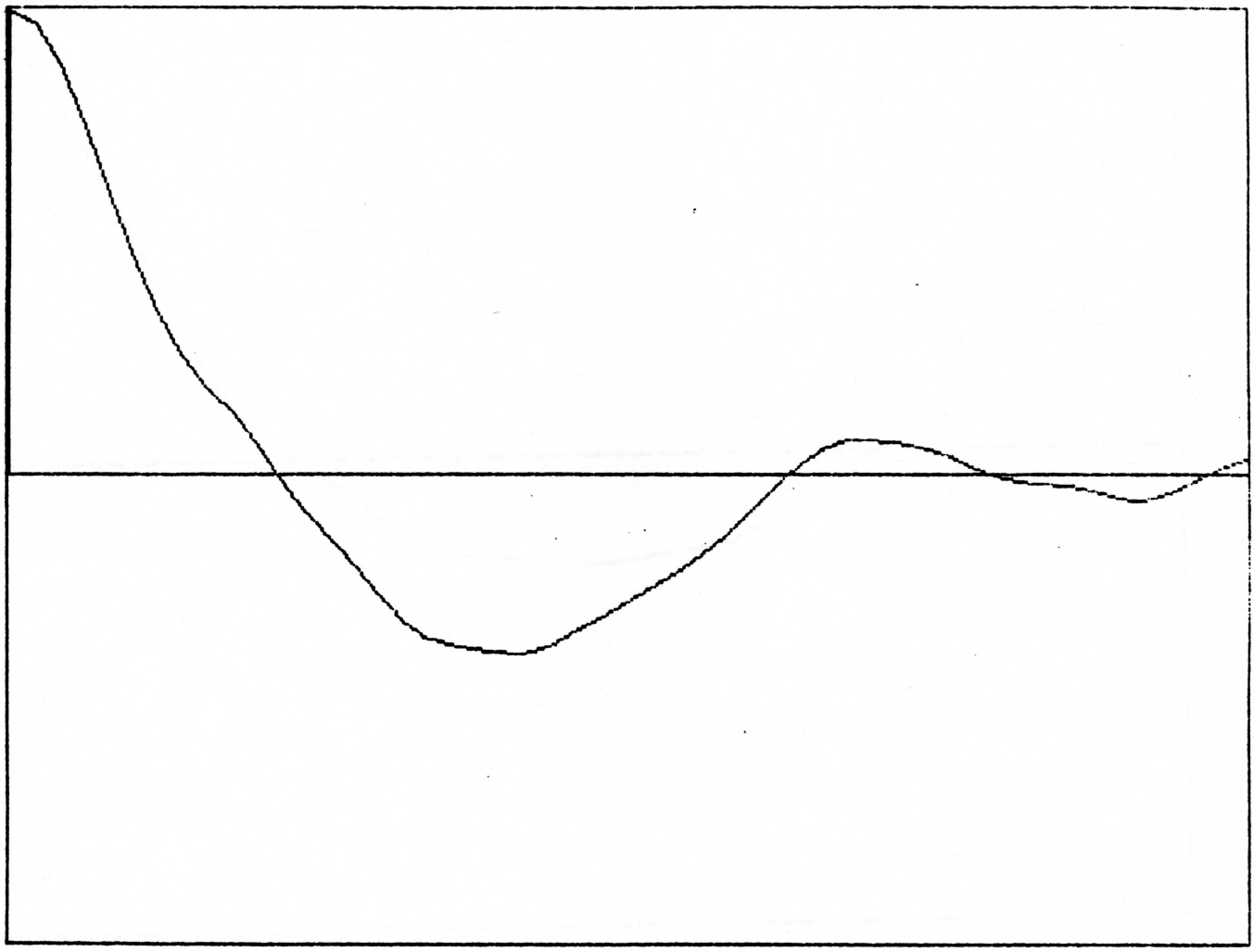


Figure 13C

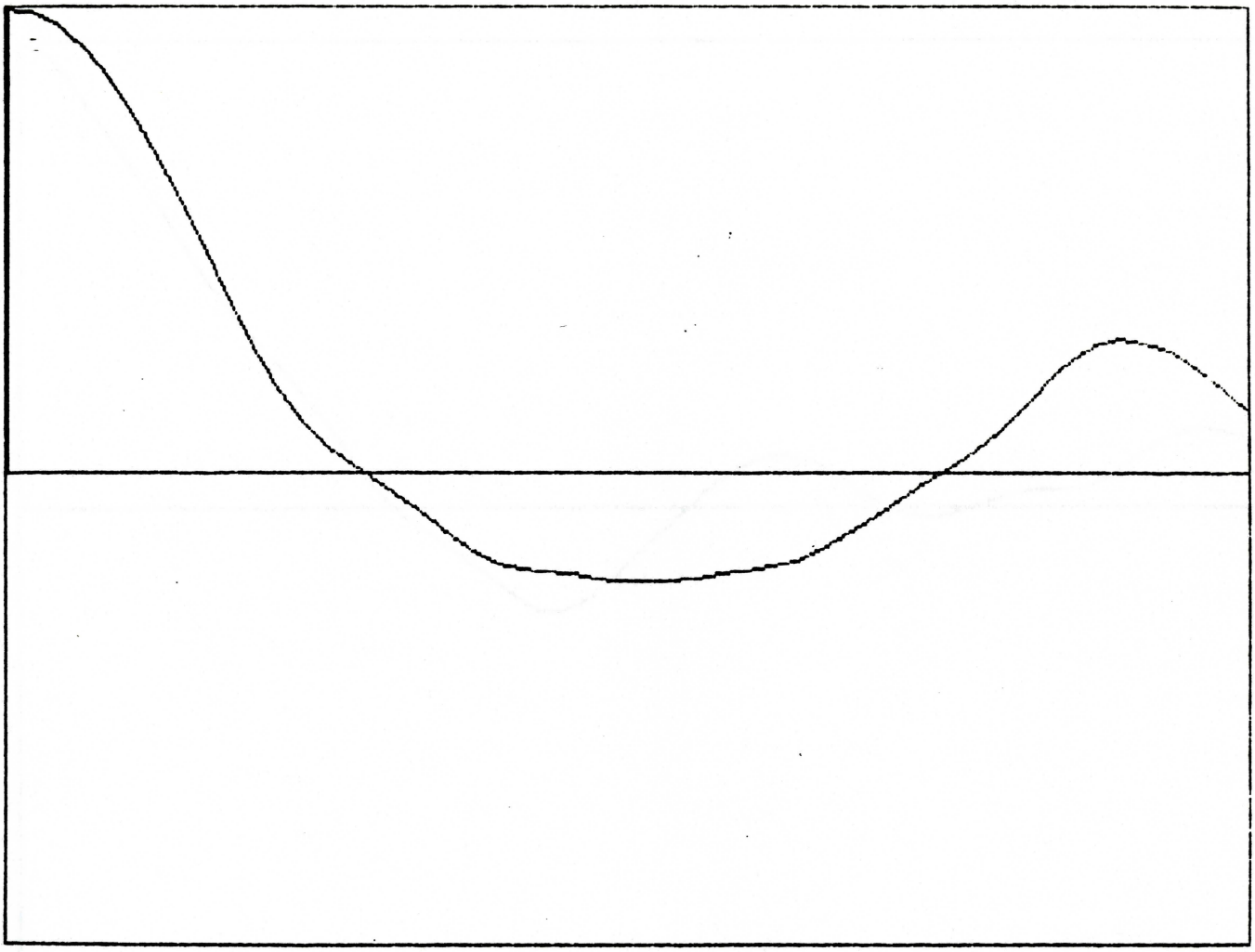


Figure 13D

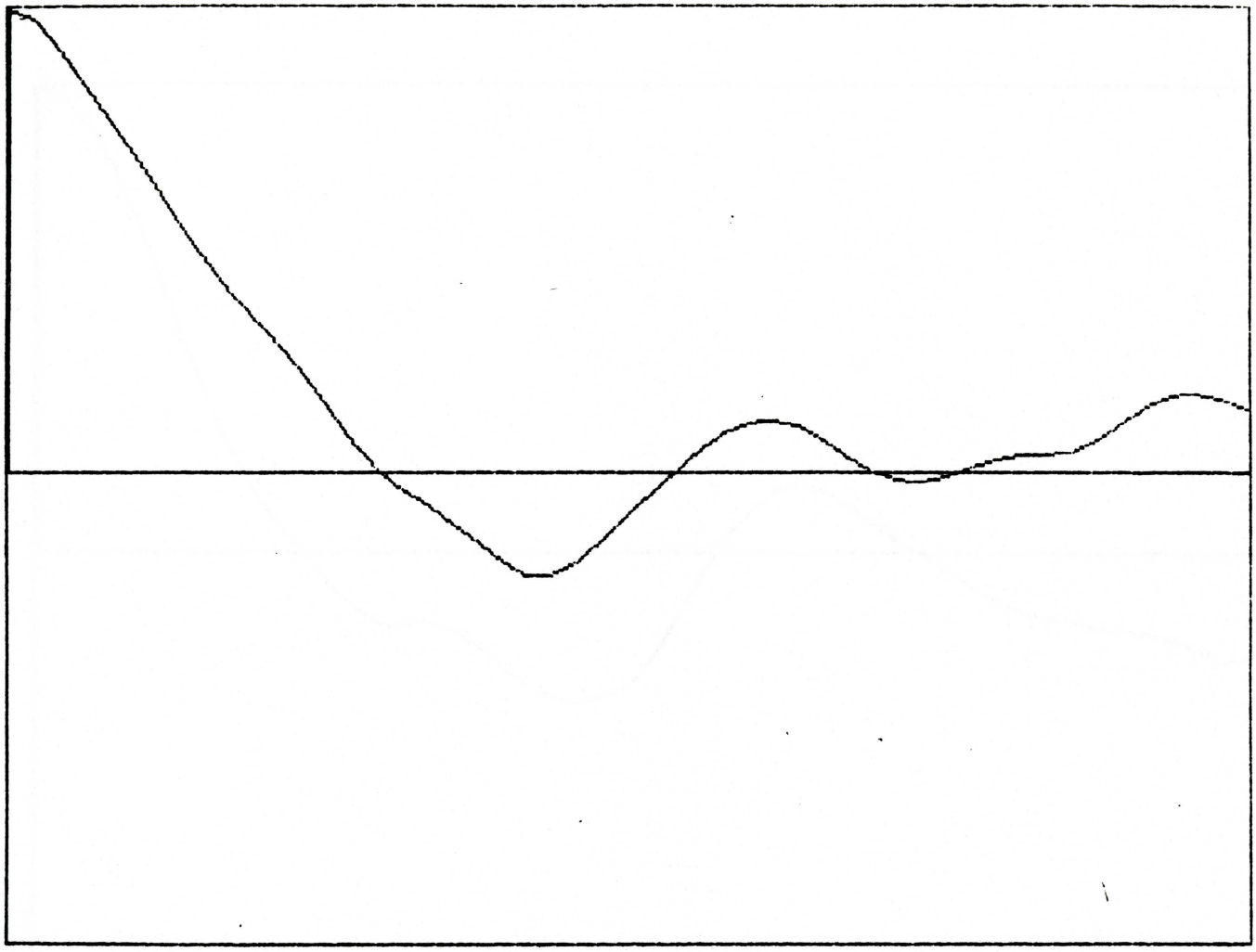


Figure 13E

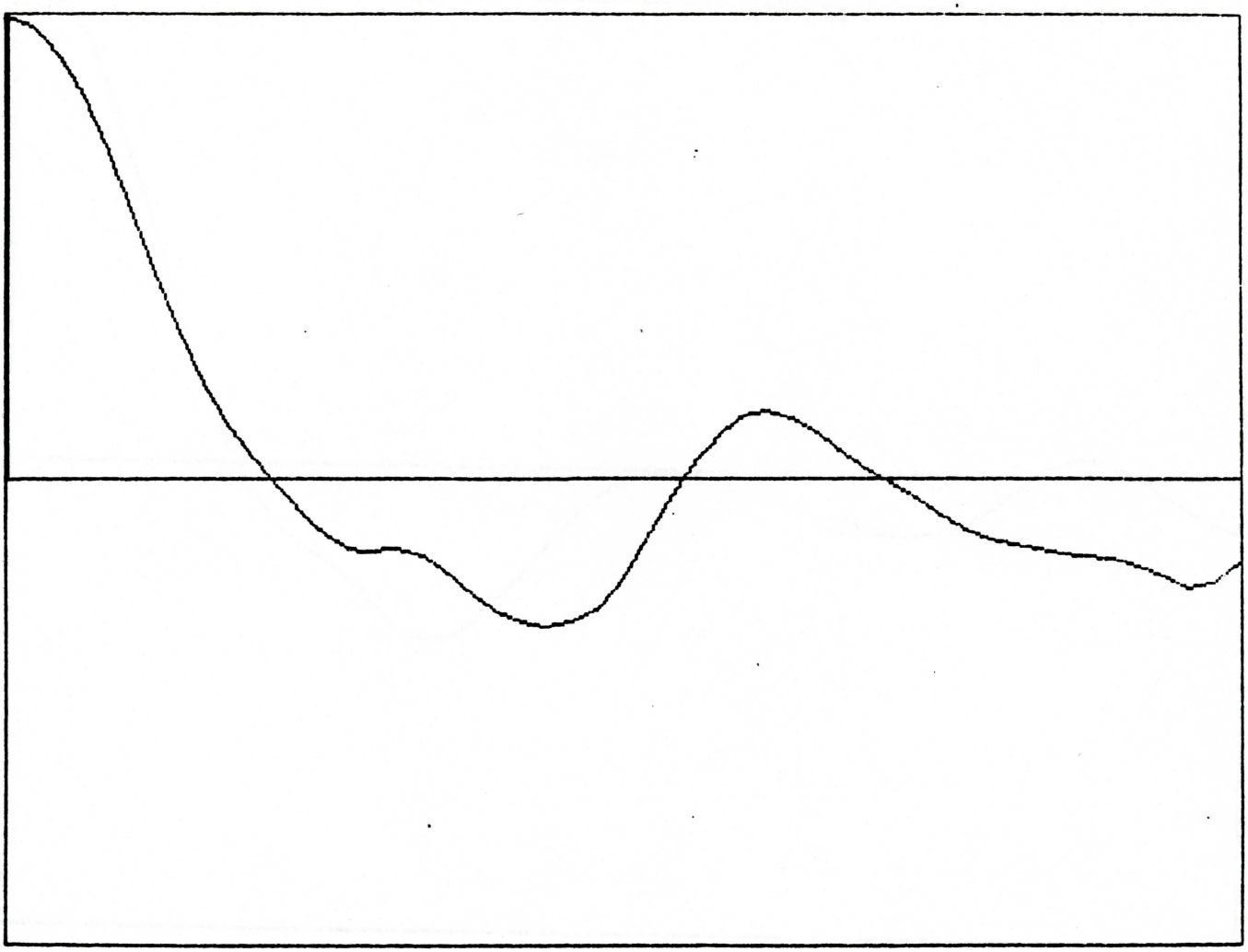


Figure 13F

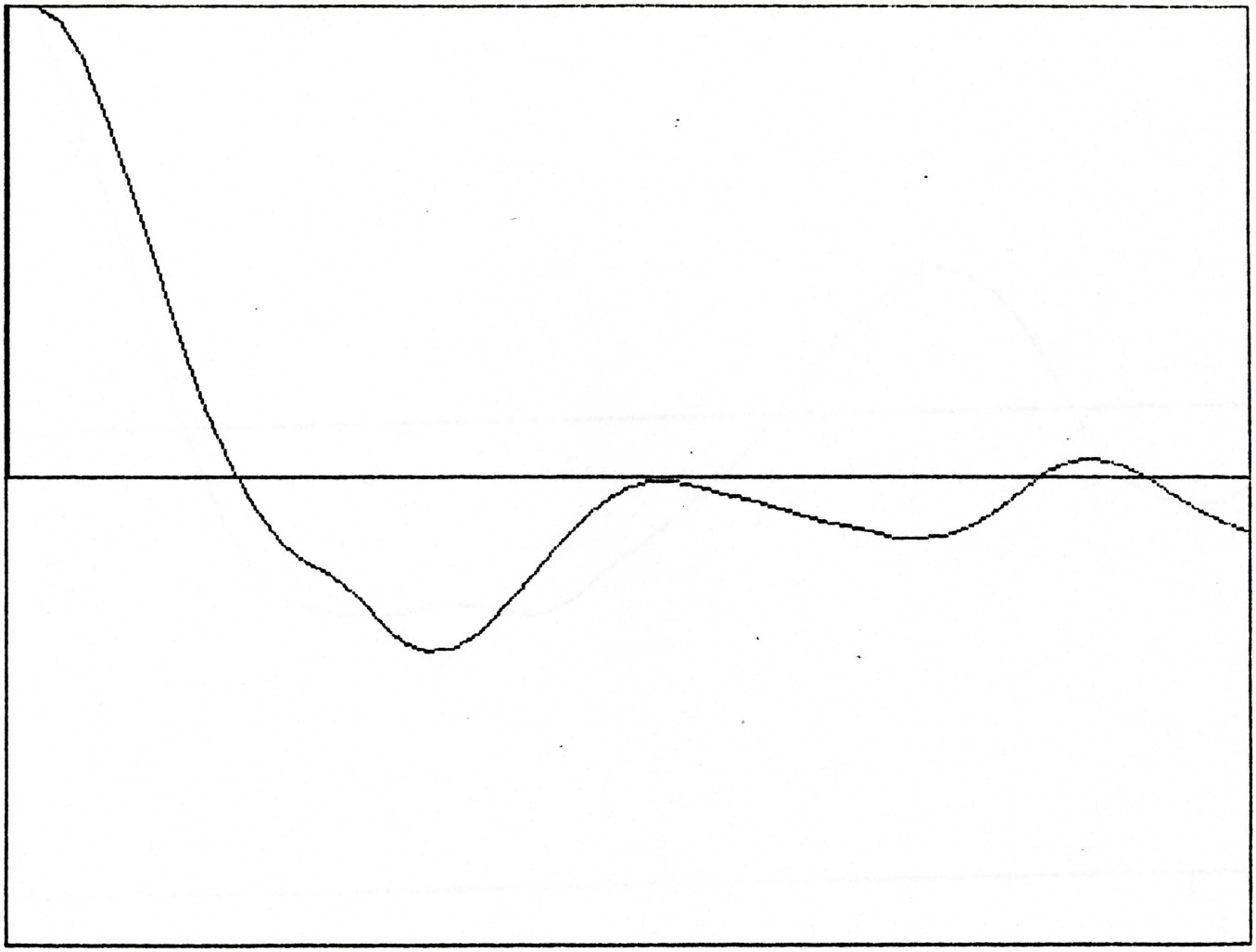


Figure 14A

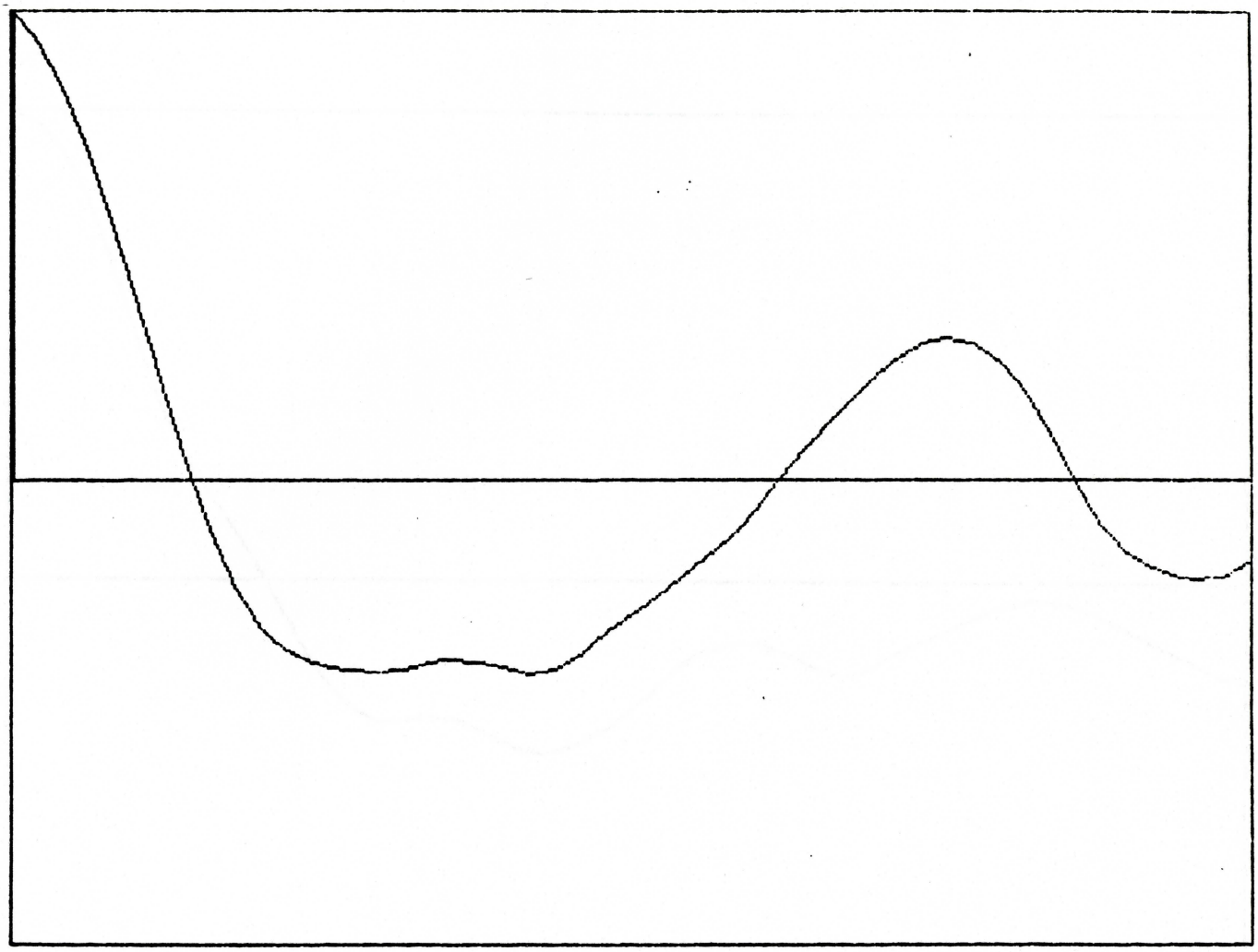


Figure 14B

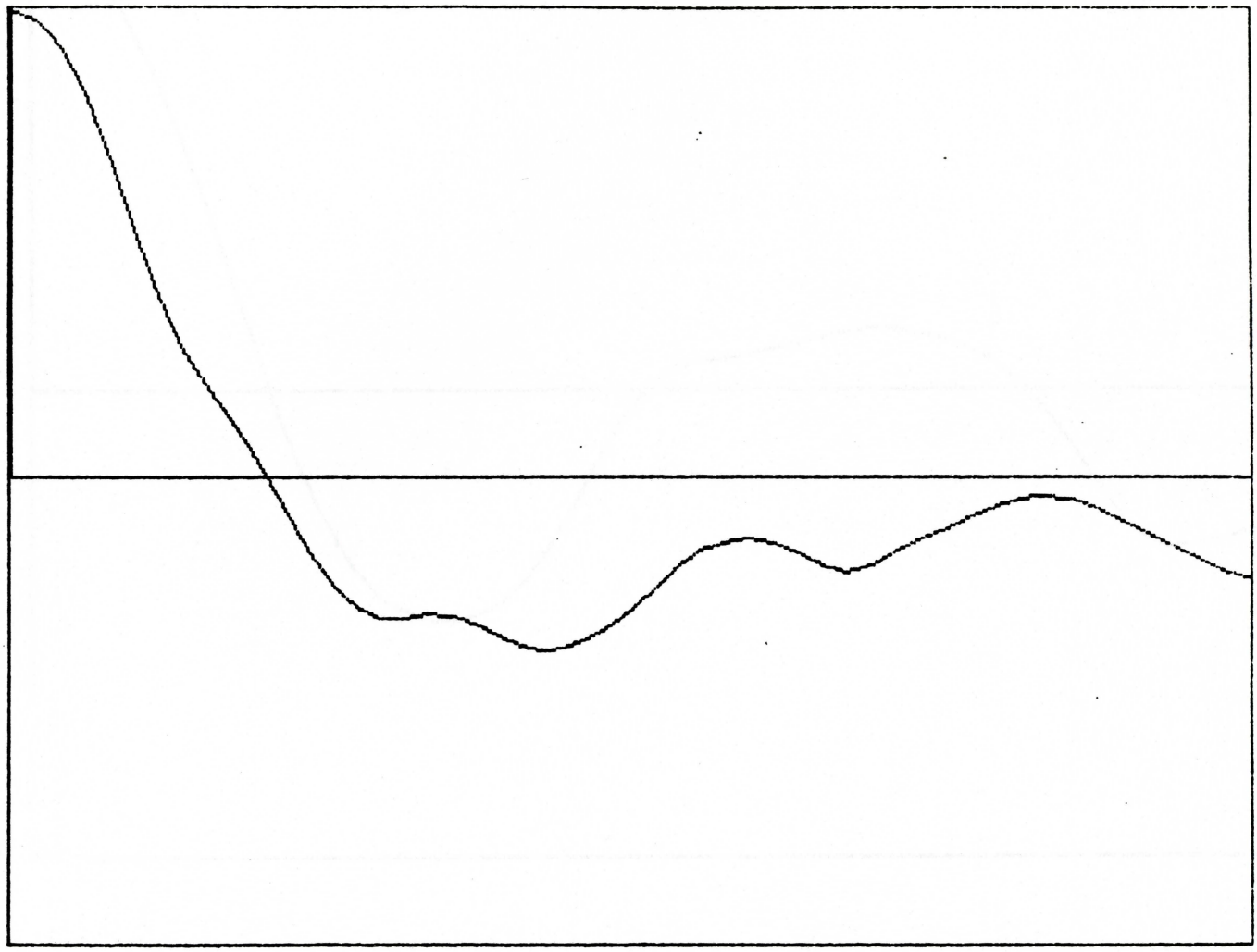


Figure 14C

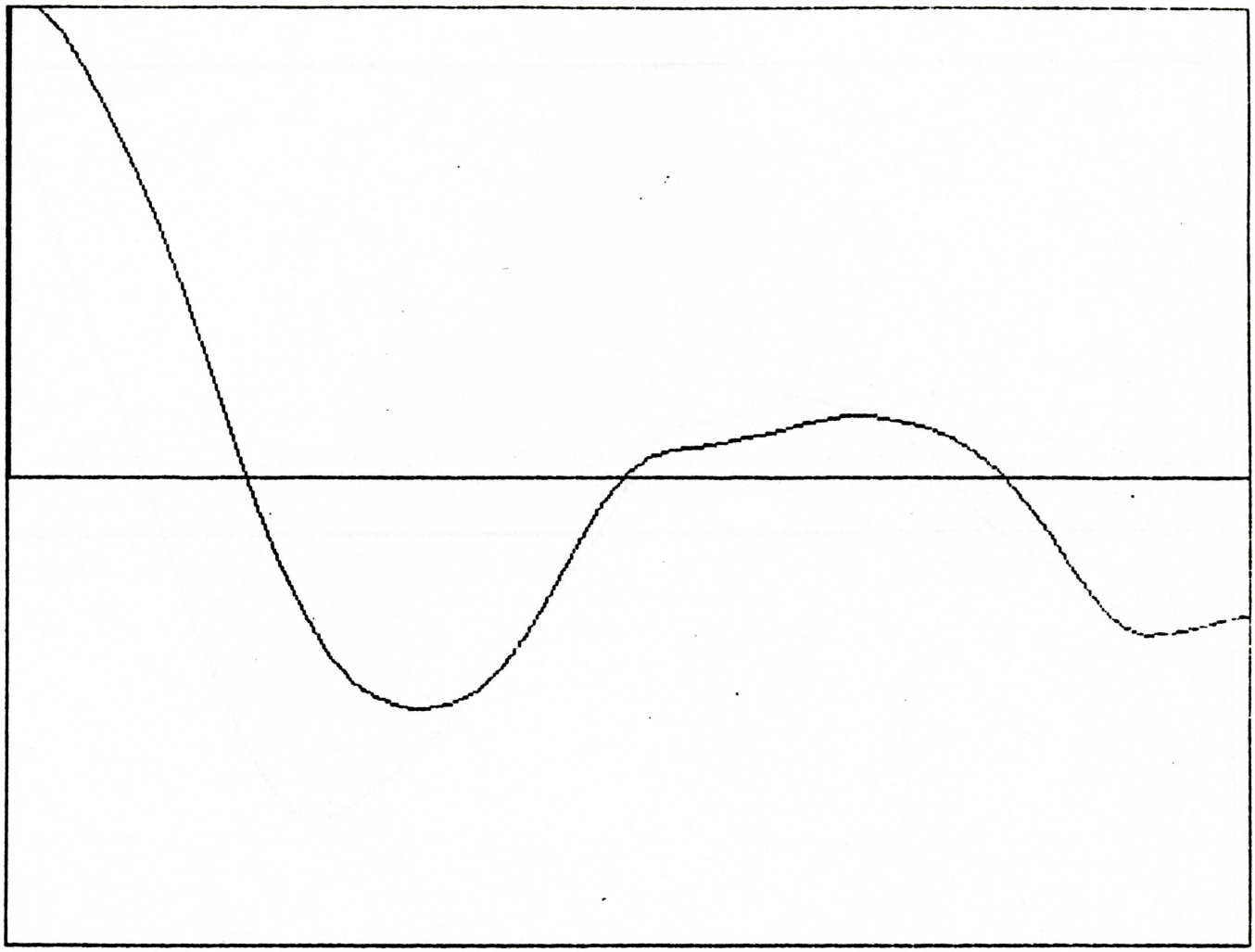


Figure 14D

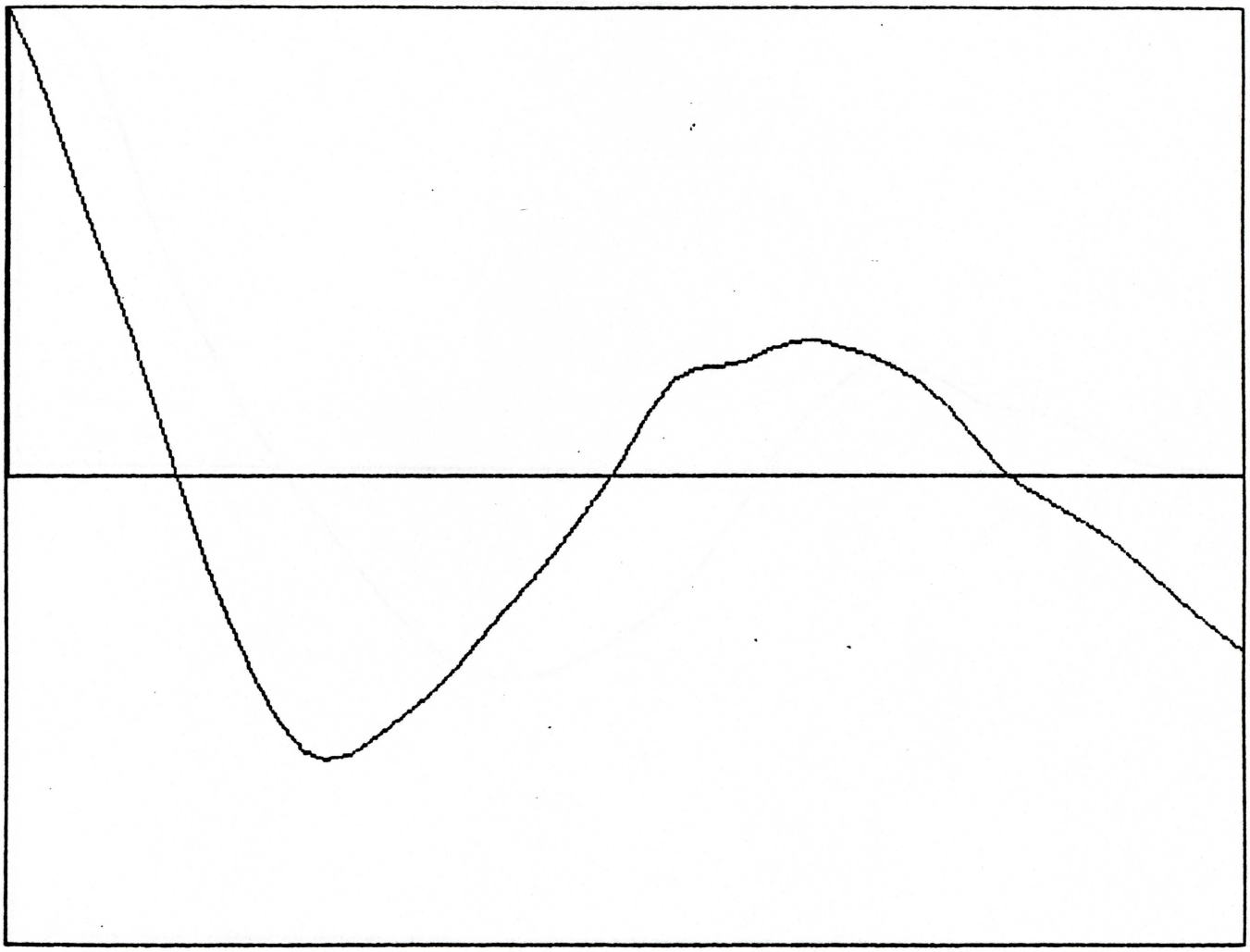


Figure 14E

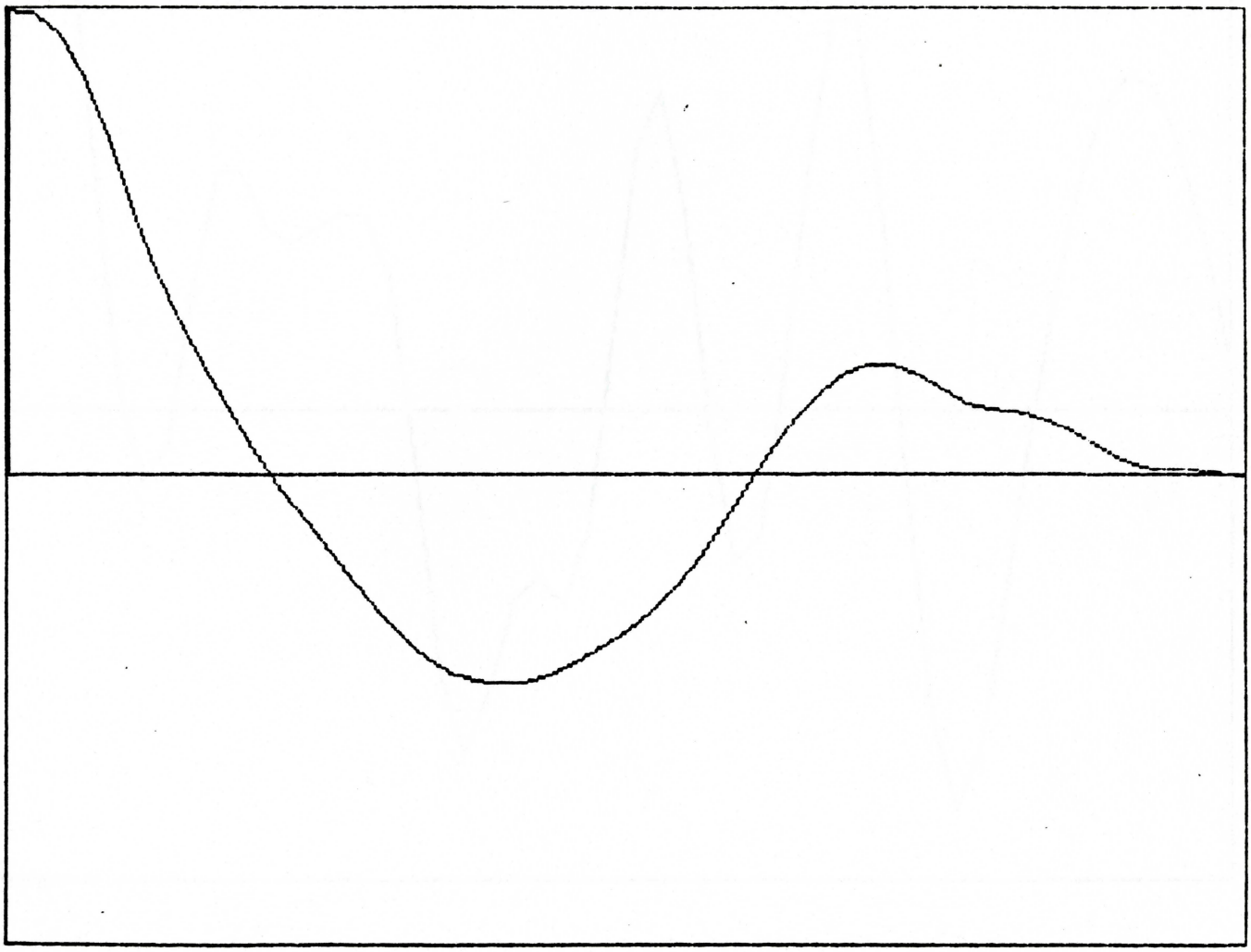


Figure 14F

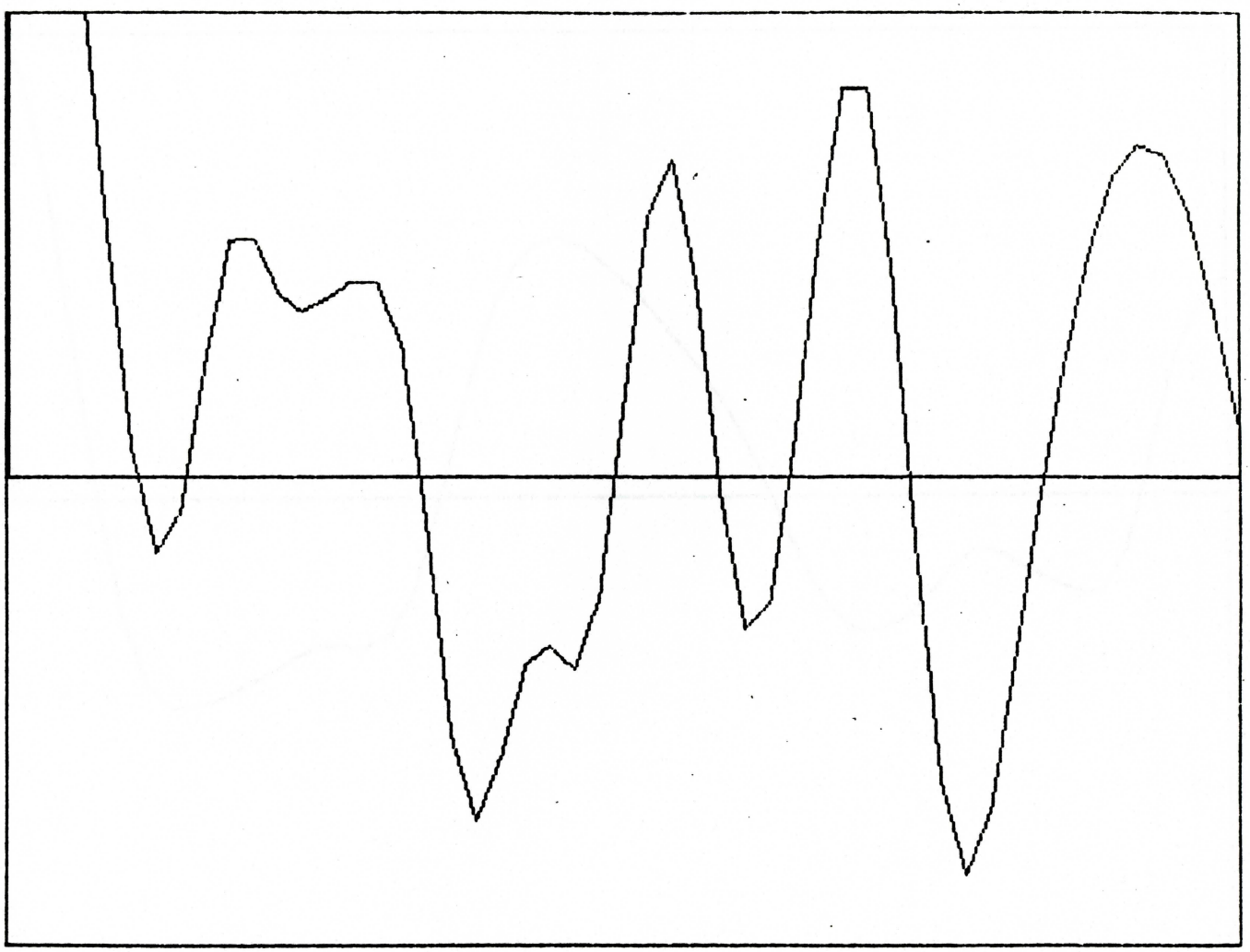


Figure 15A

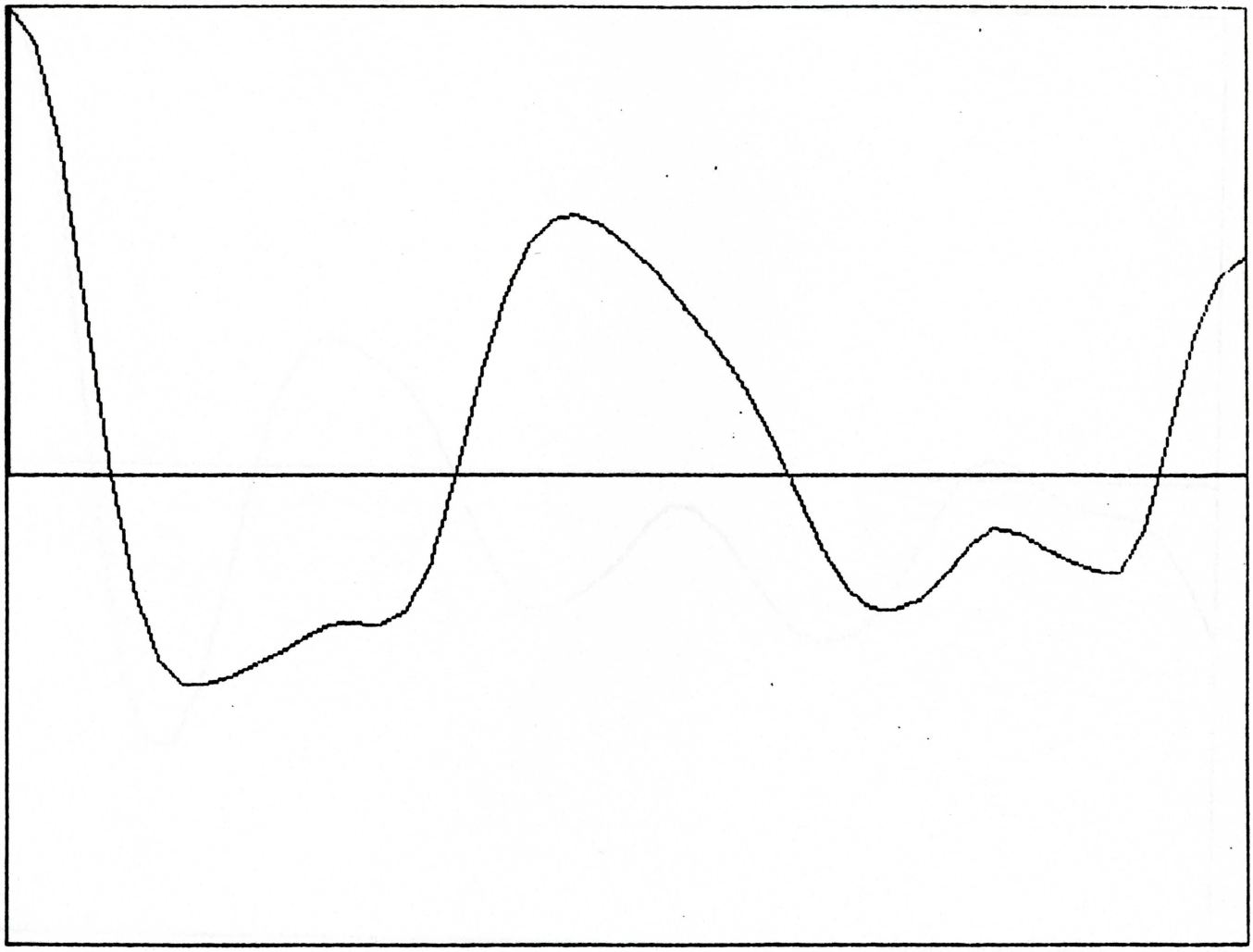


Figure 15B

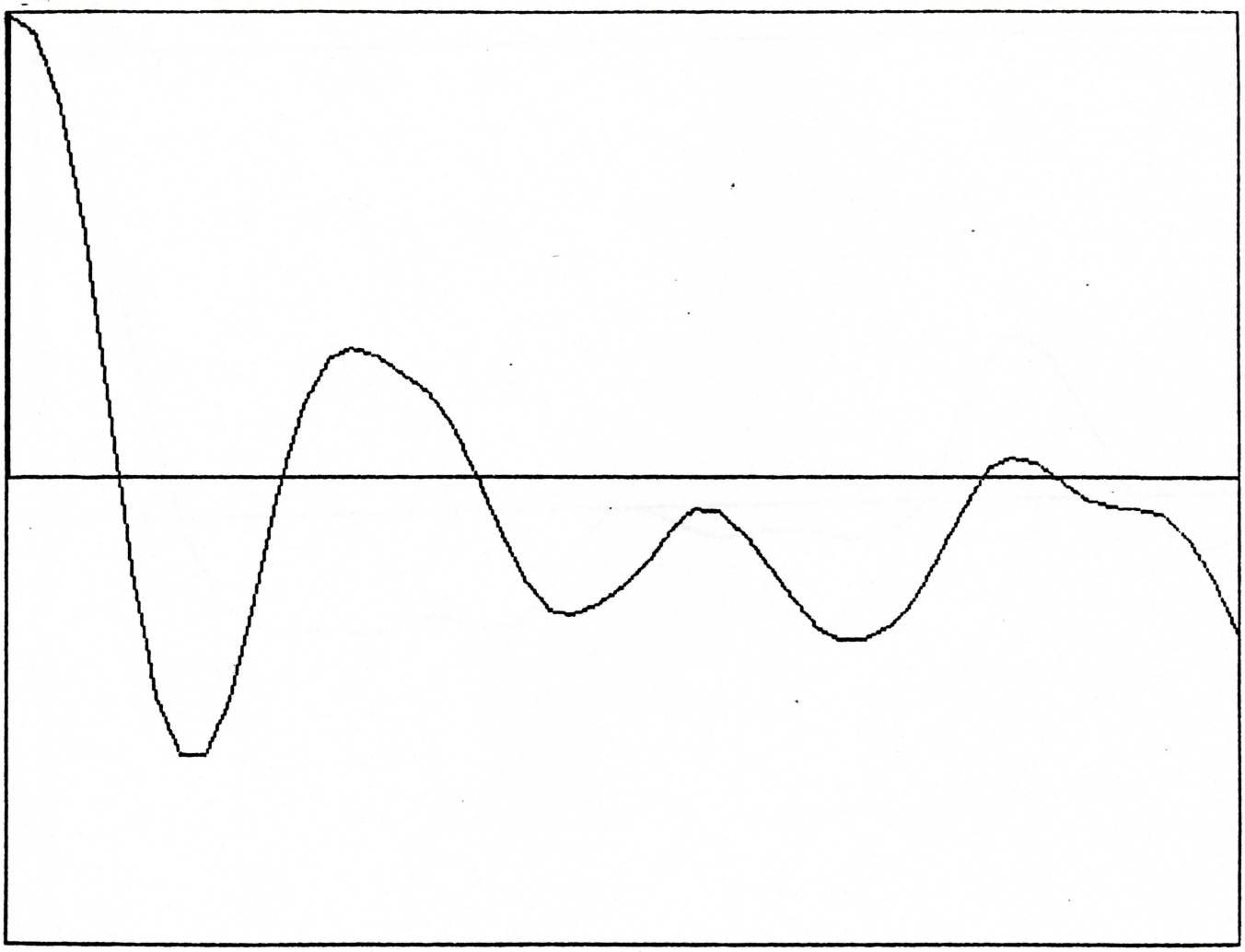


Figure 15C

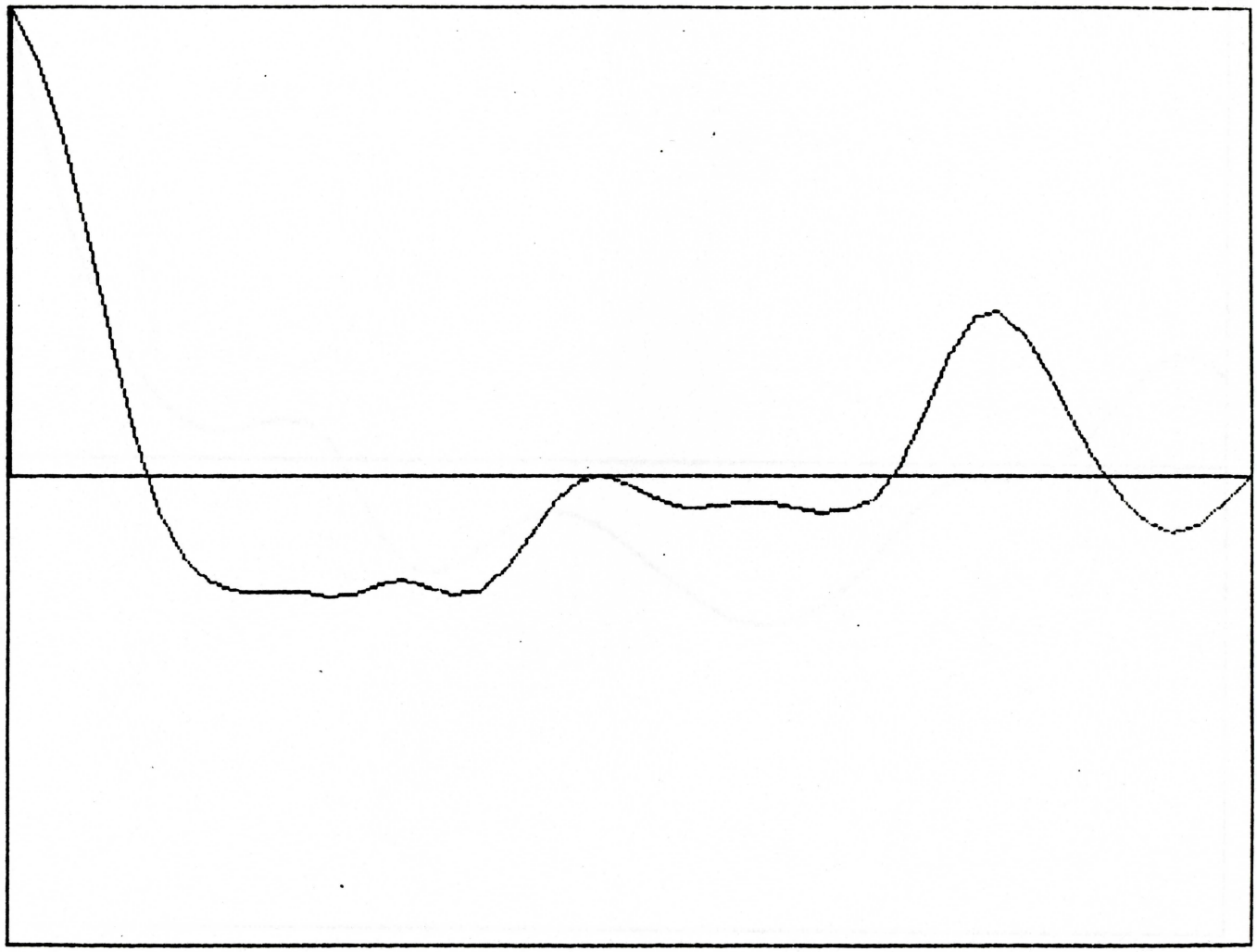


Figure 15D

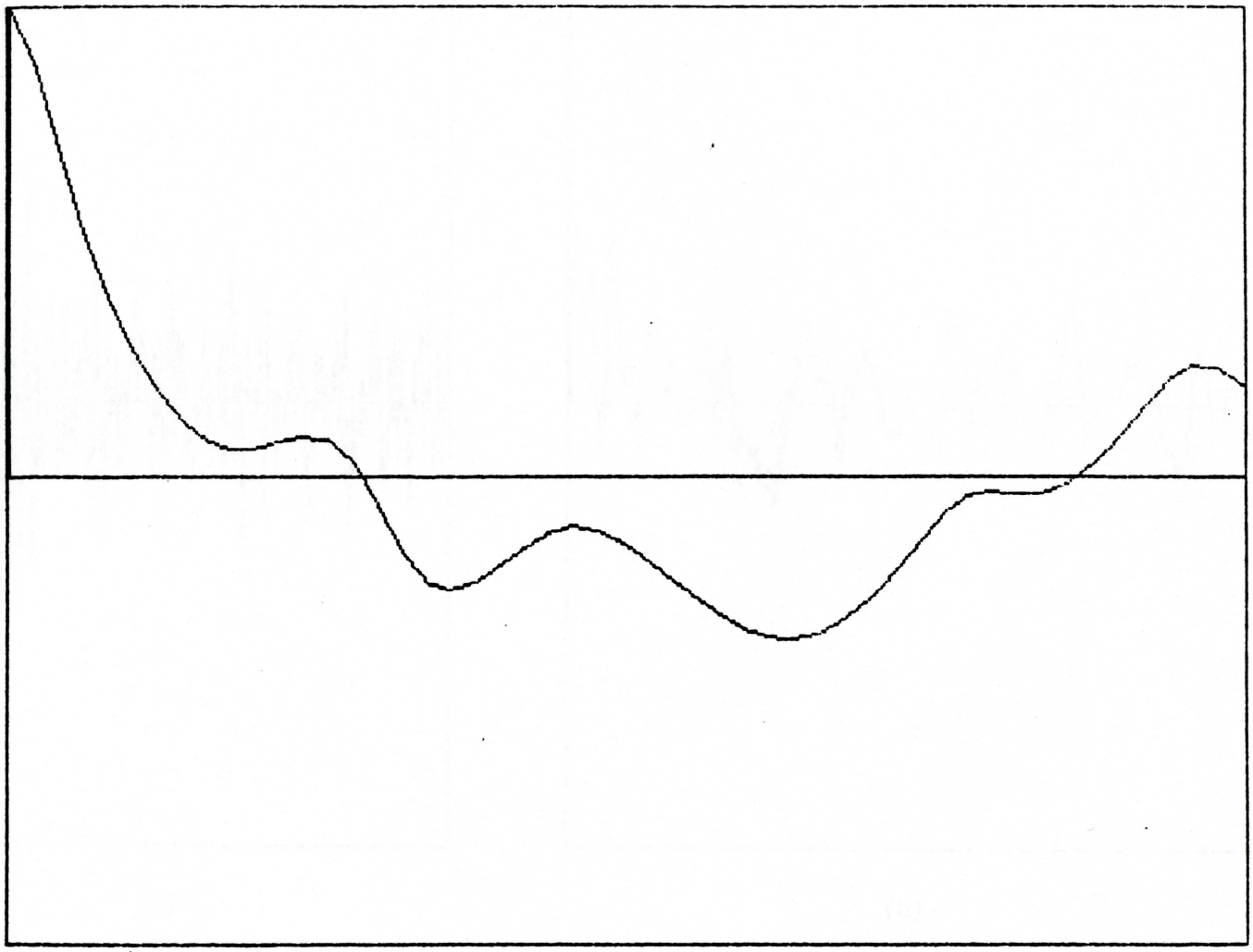
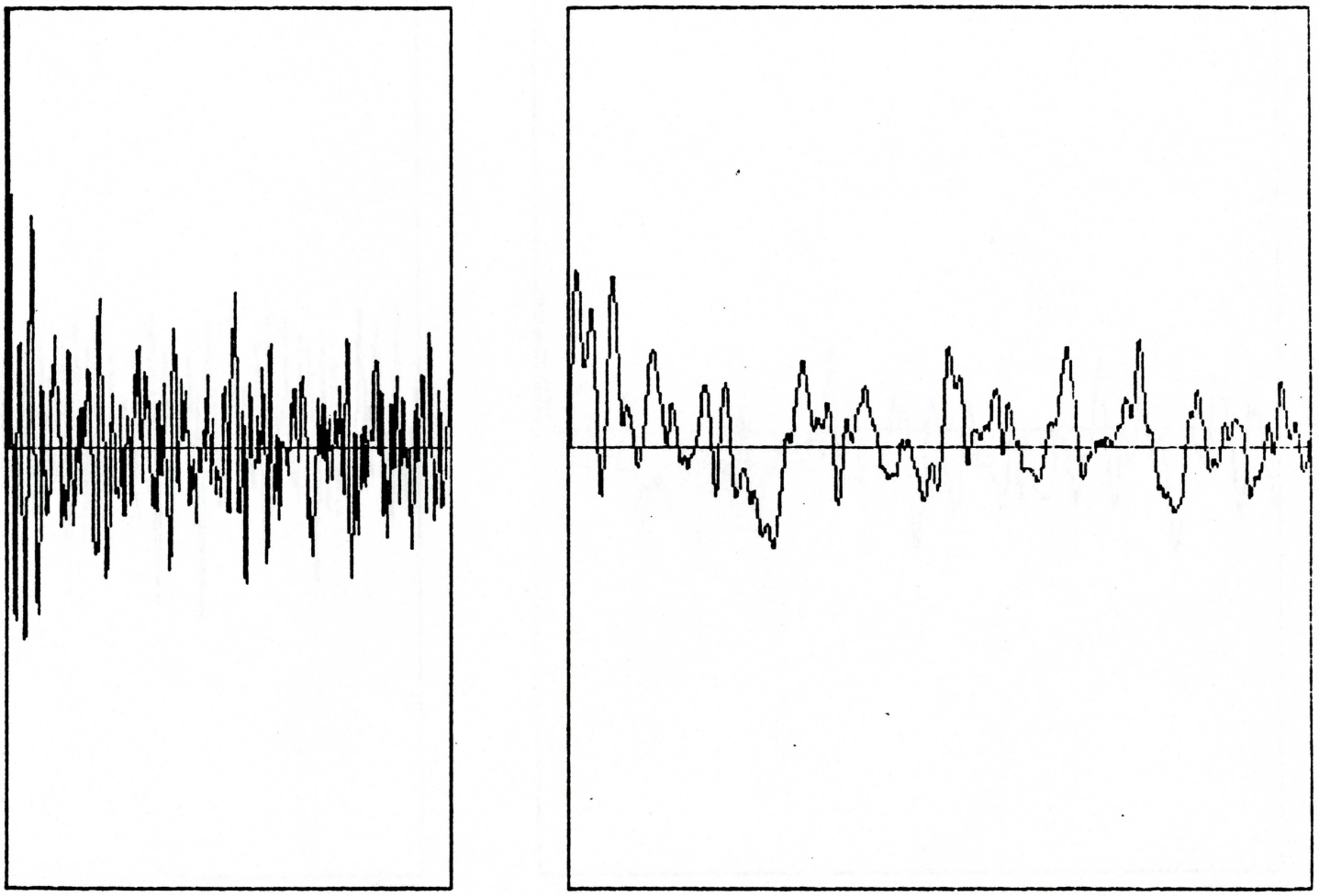


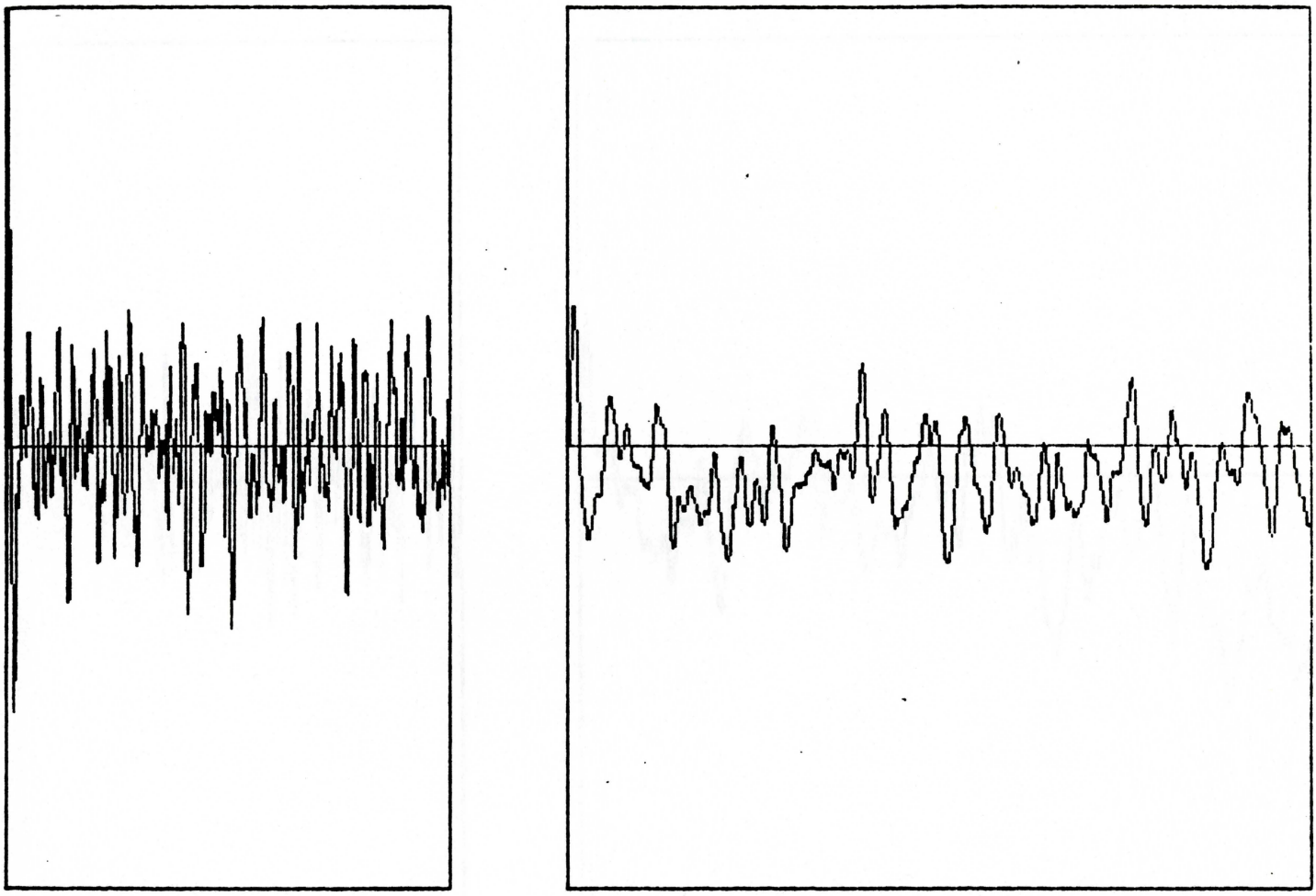
Figure 15E



(a)

(b)

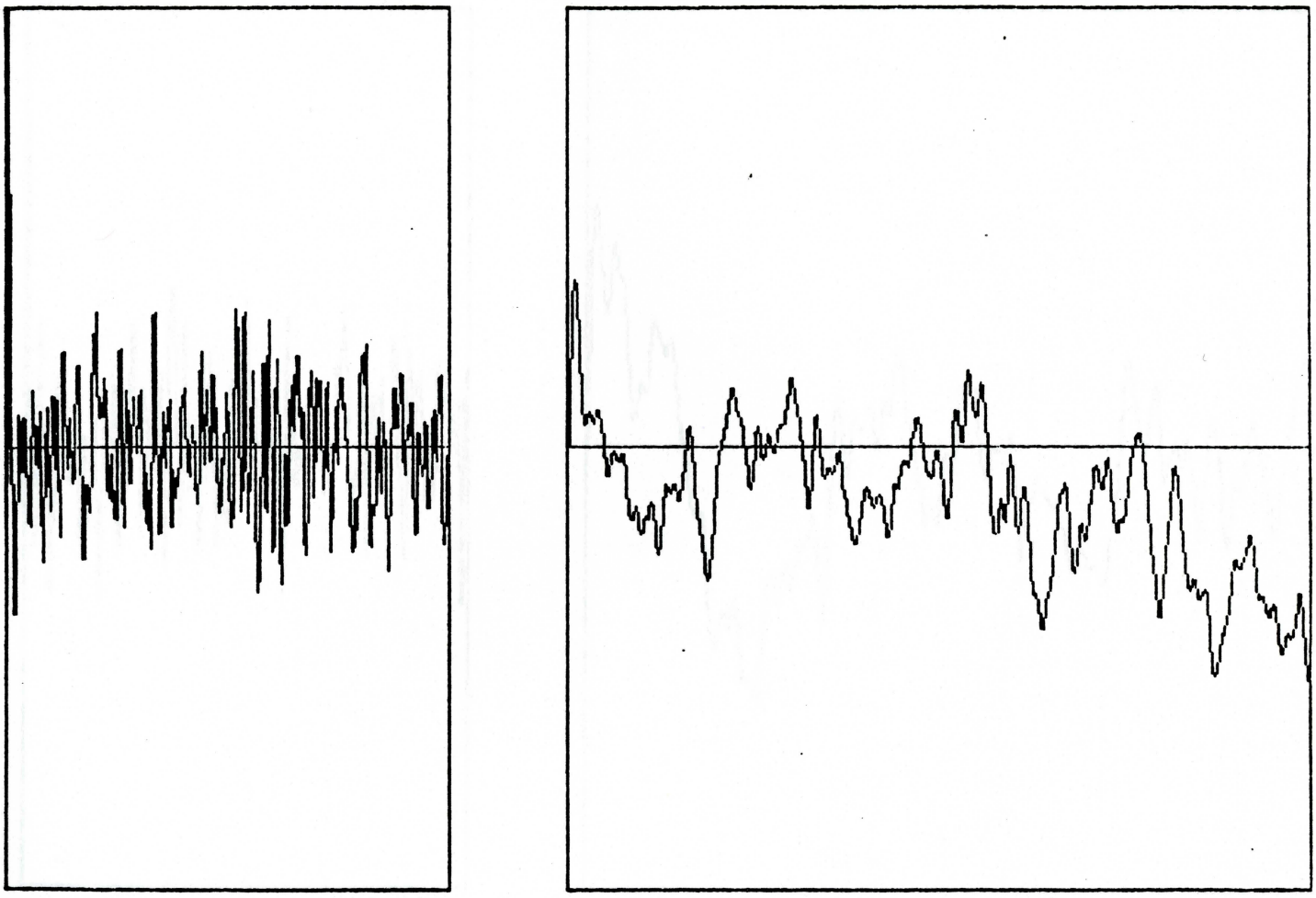
Figure 16A



(a)

(b)

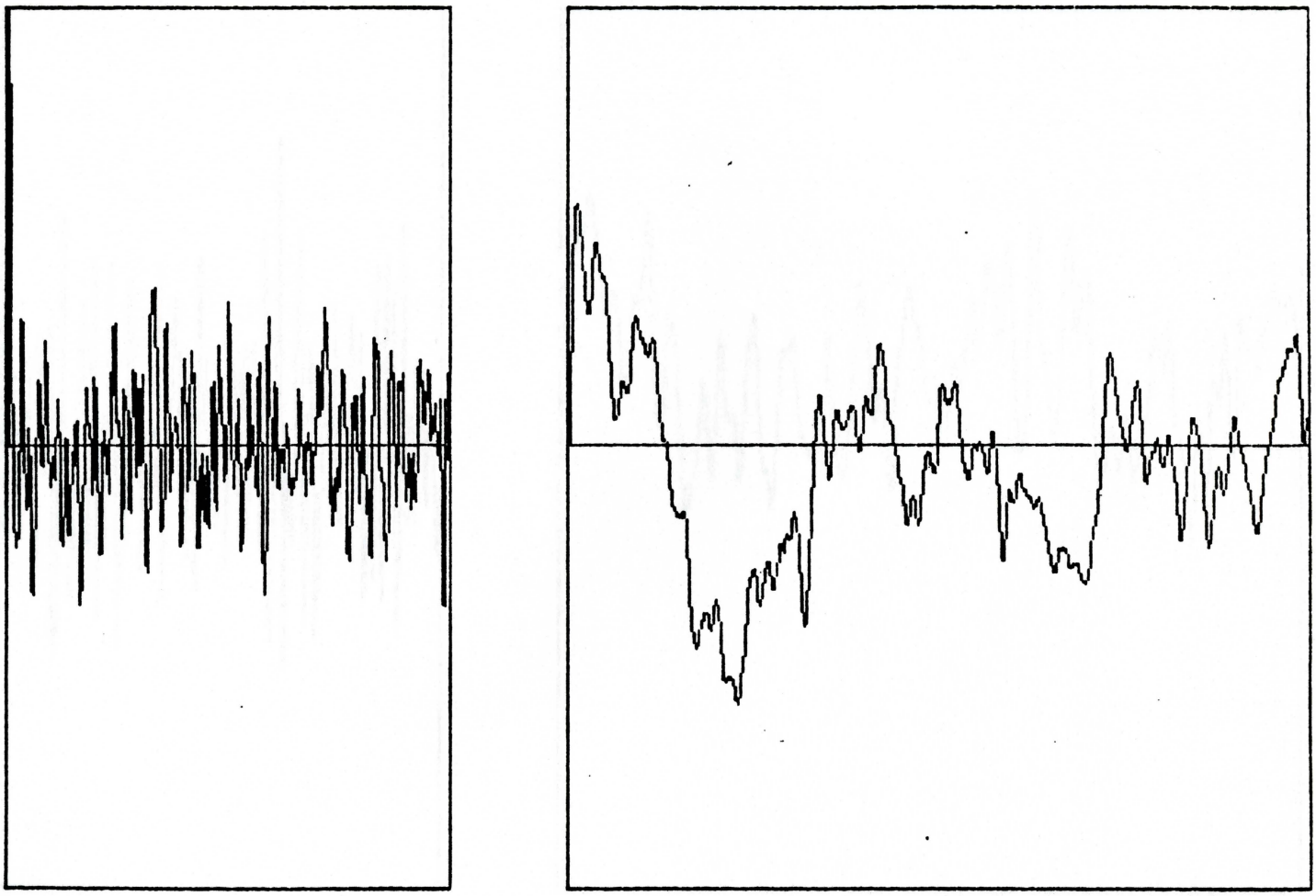
Figure 16B



(a)

(b)

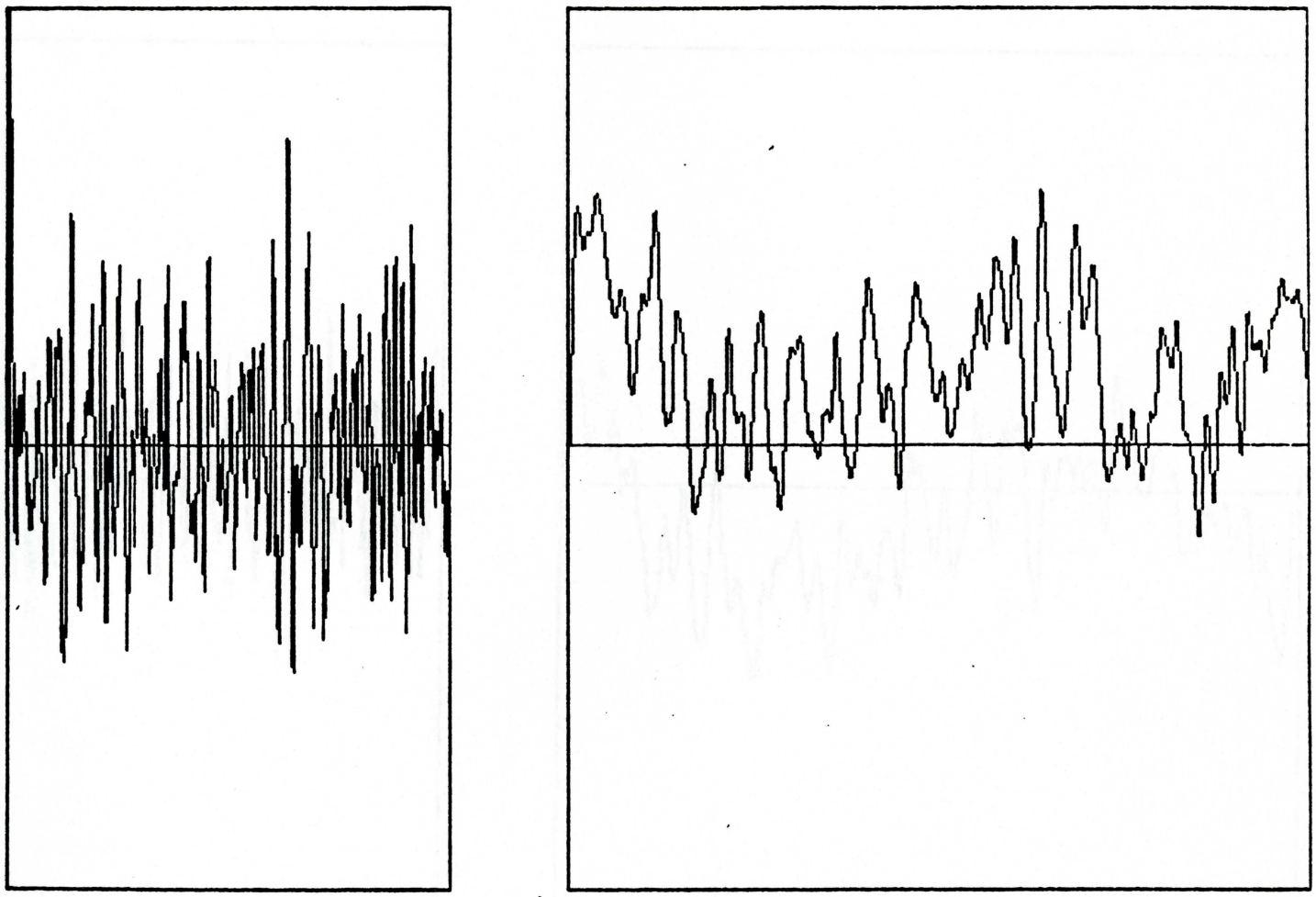
Figure 16C



(a)

(b)

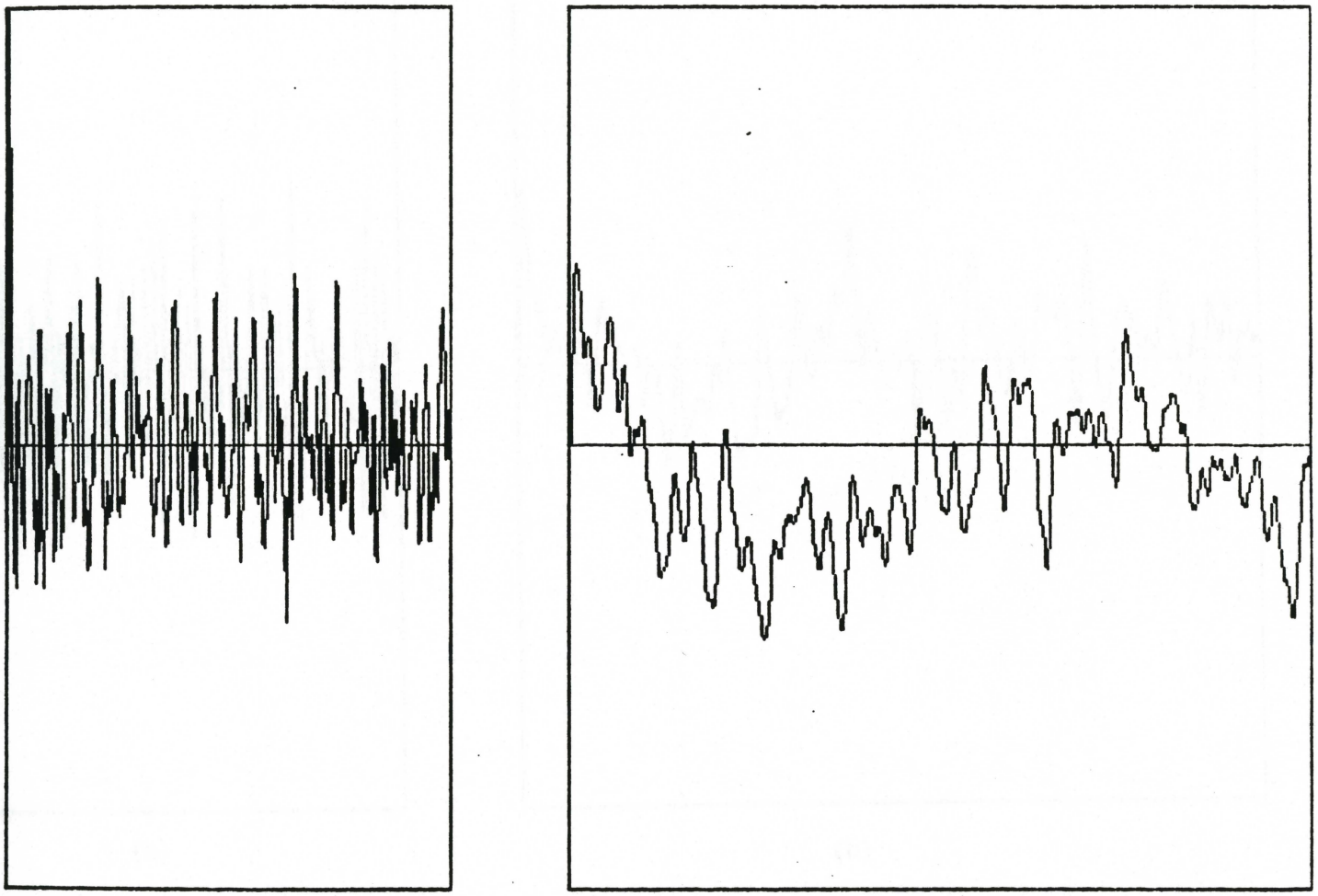
Figure 16D



(a)

(b)

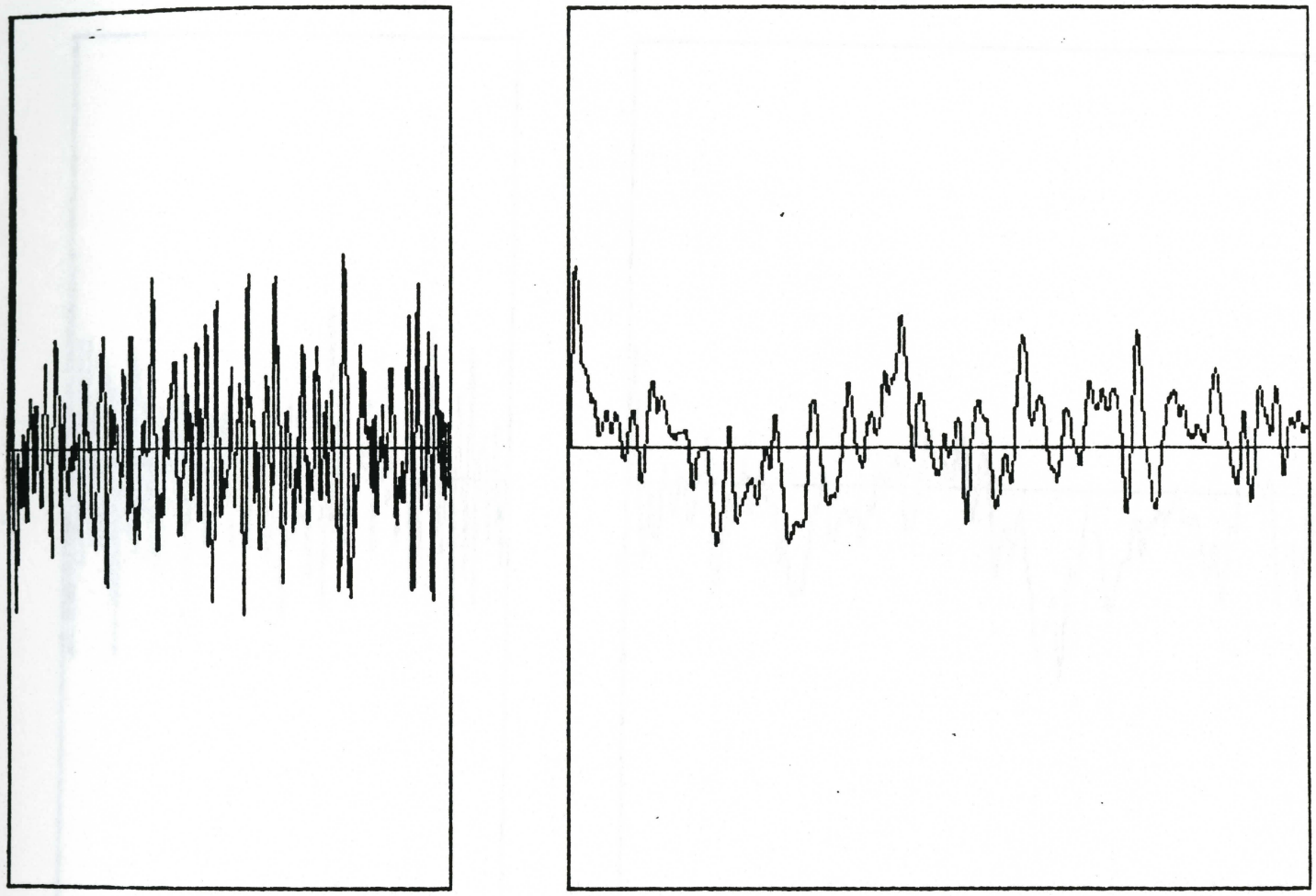
Figure 16E



(a)

(b)

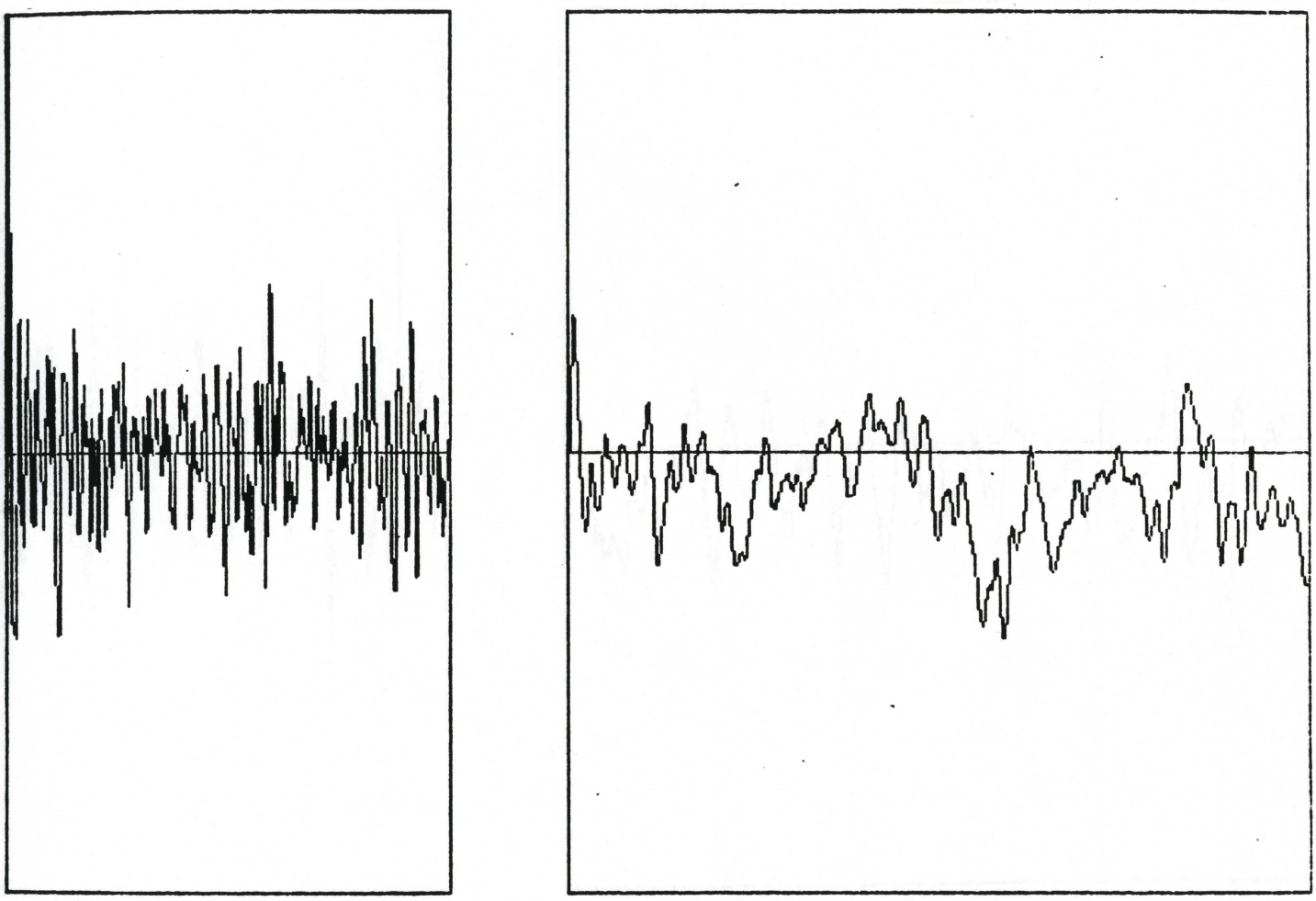
Figure 16F



(a)

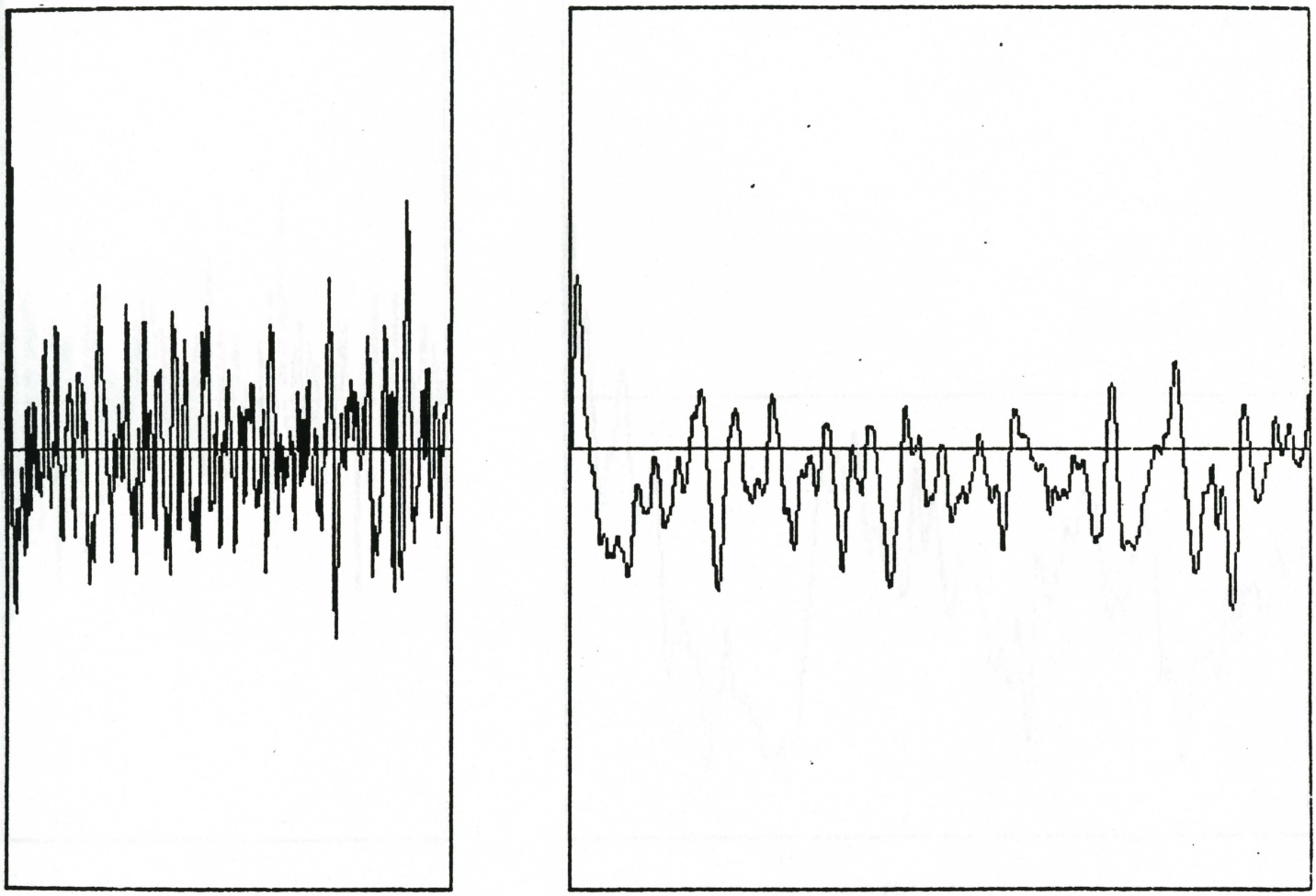
(b)

Figure 17A



(a) (b)

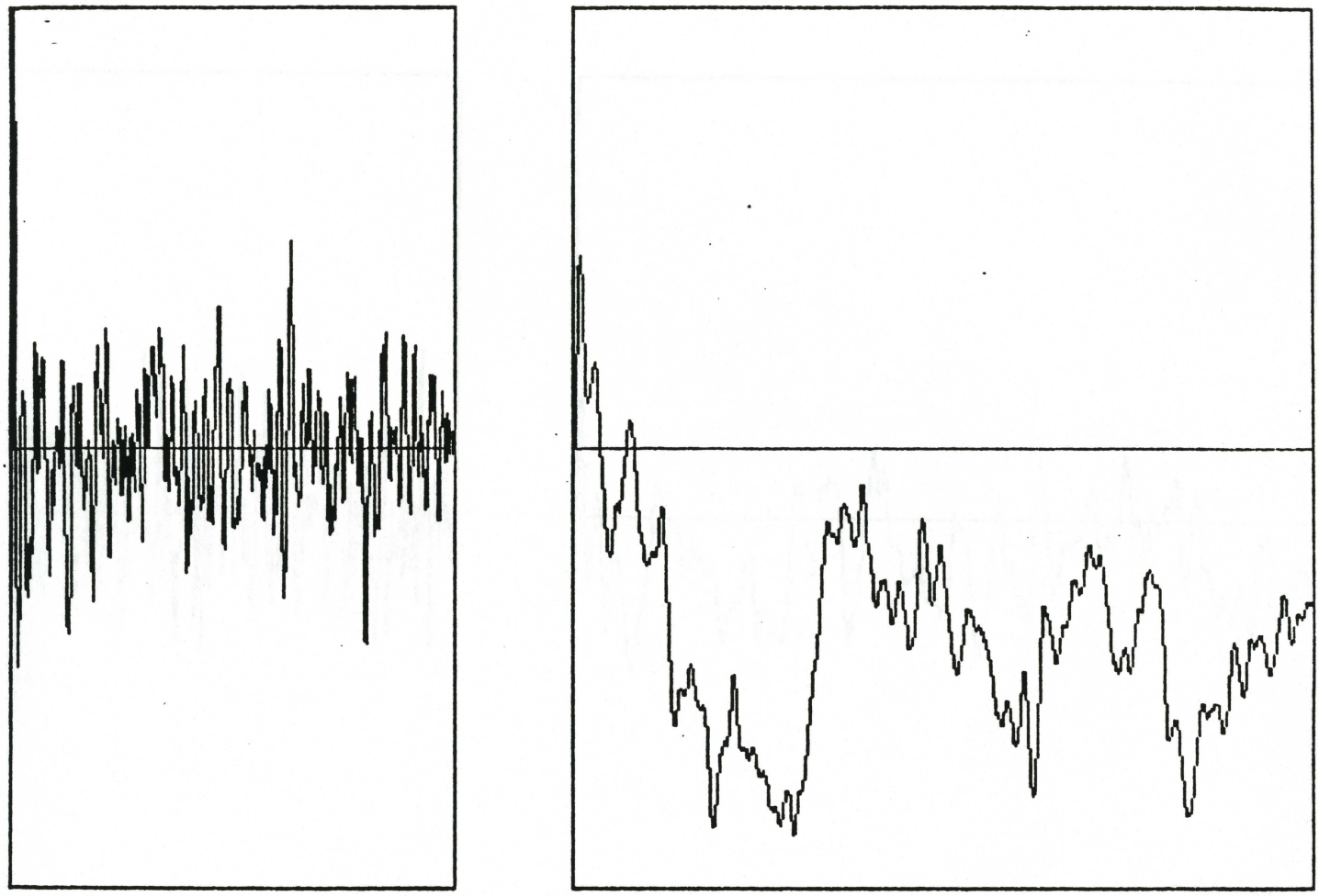
Figure 17B



(a)

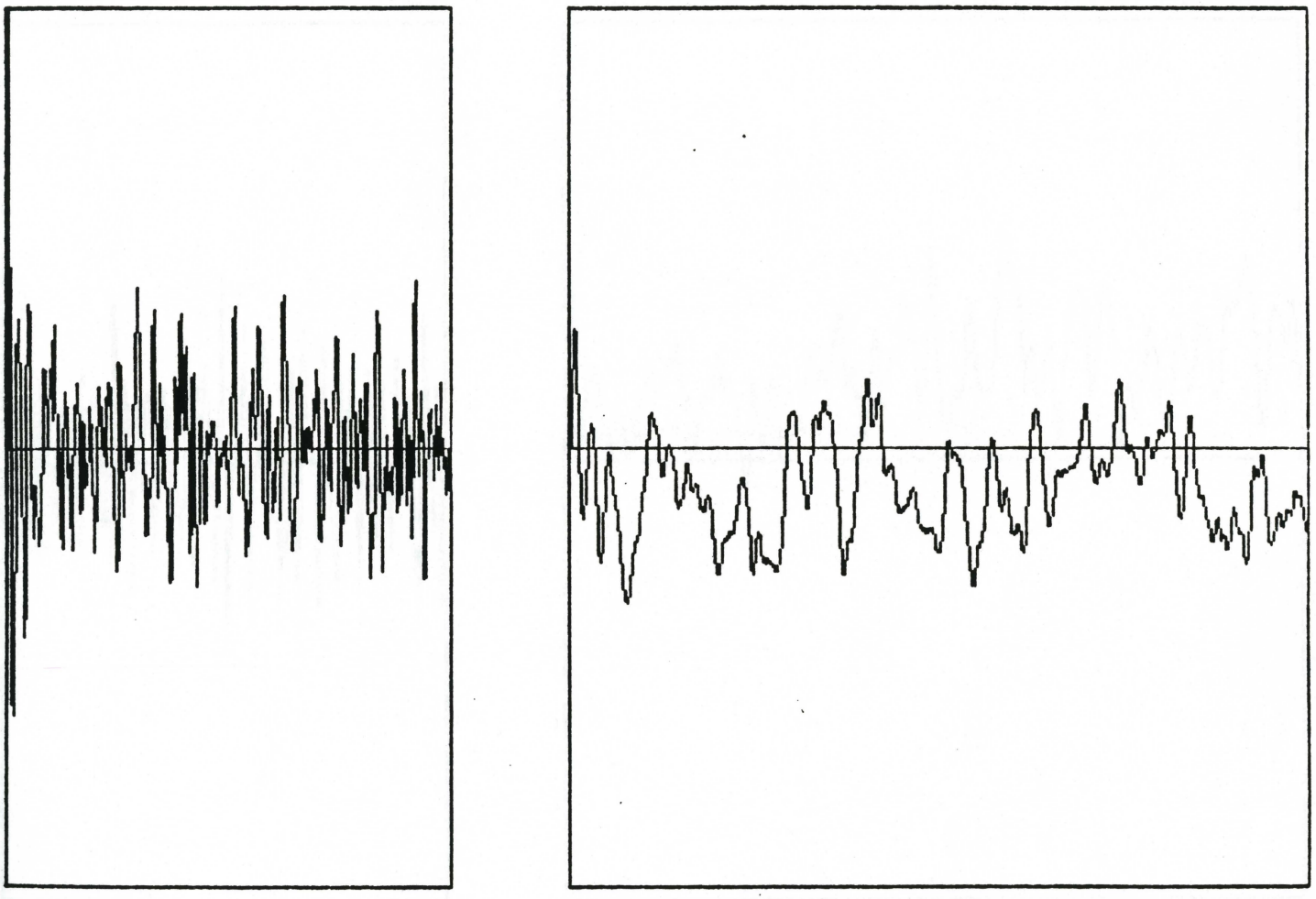
(b)

Figure 17C



(a) (b)

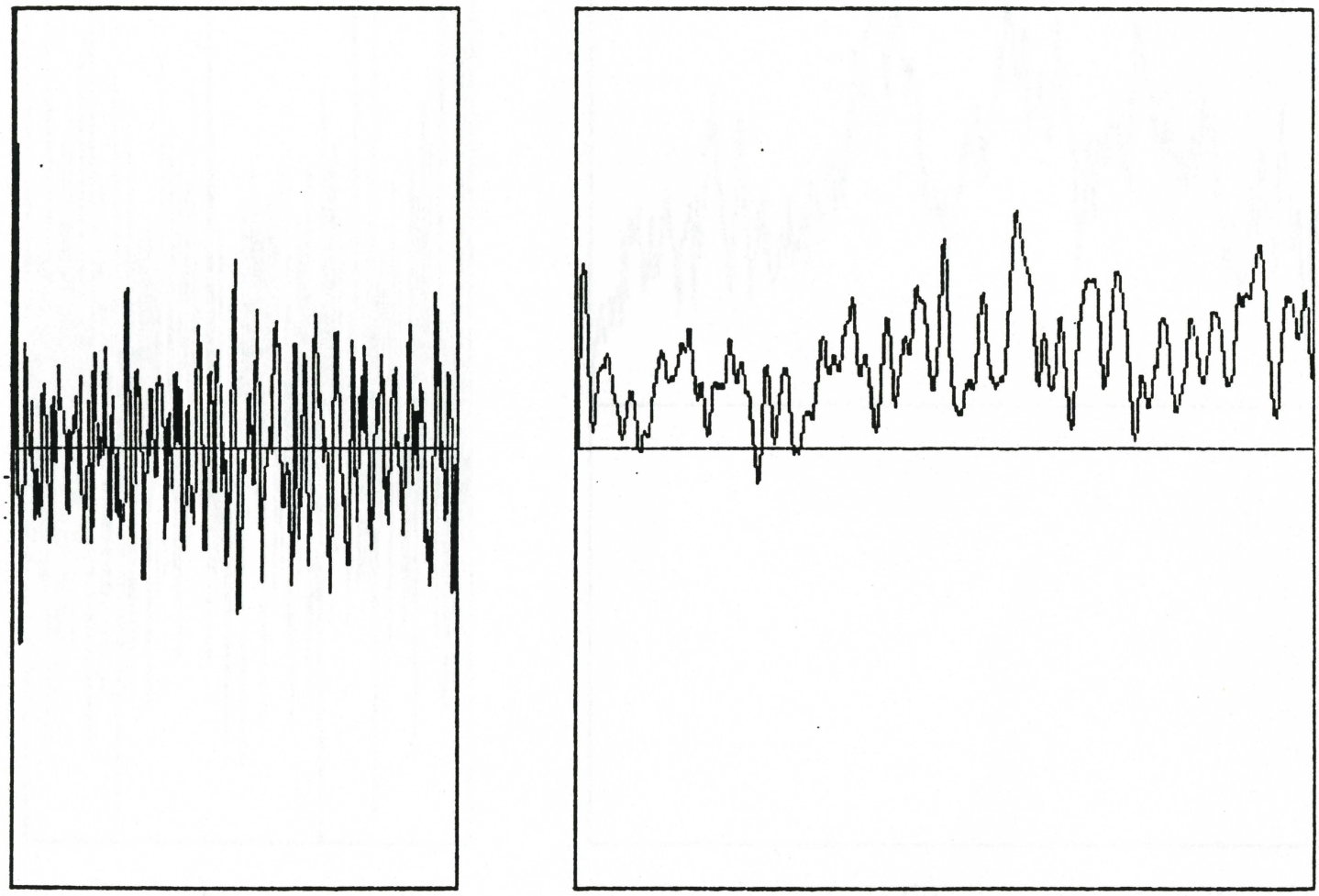
Figure 17D



(a)

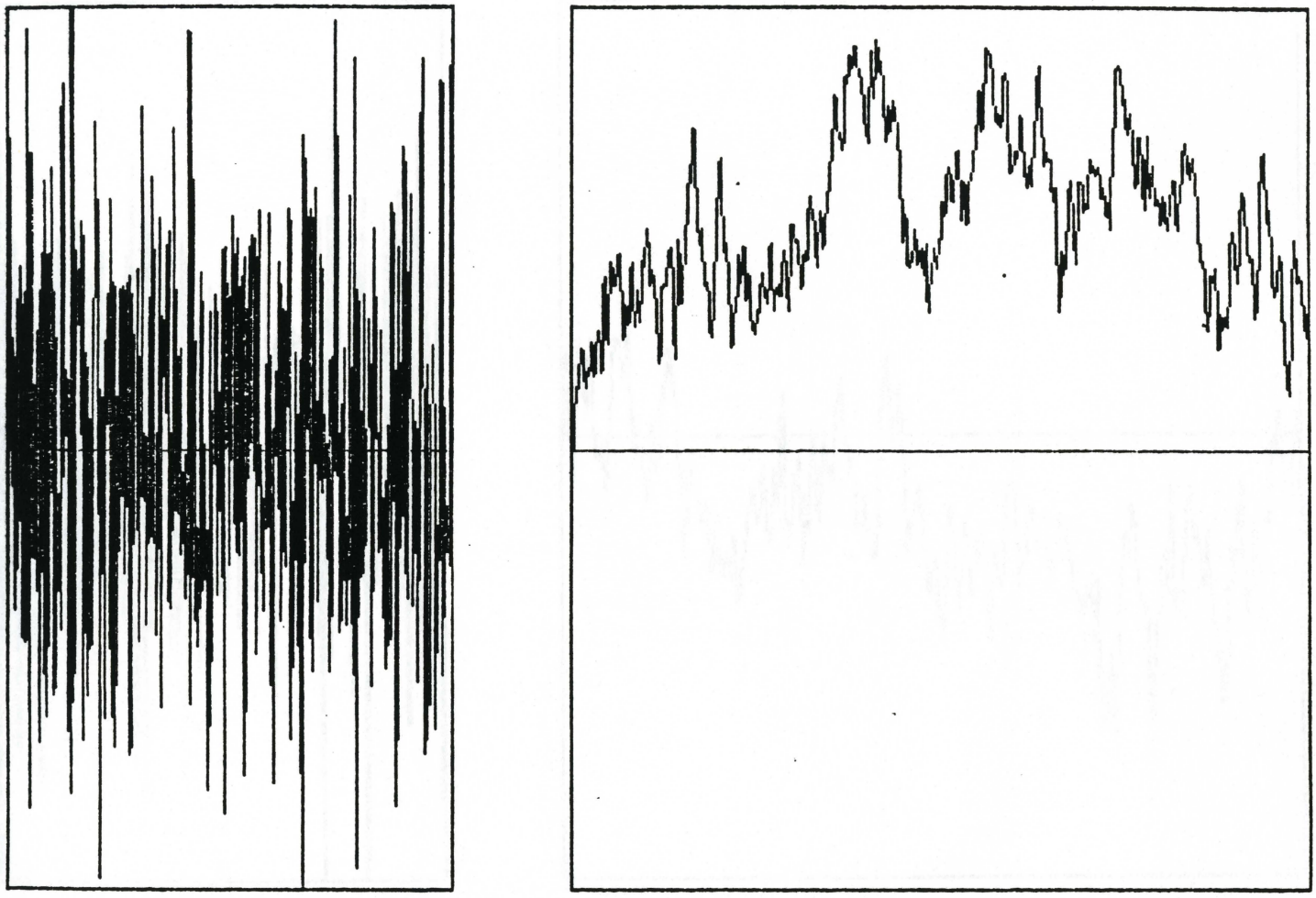
(b)

Figure 17E



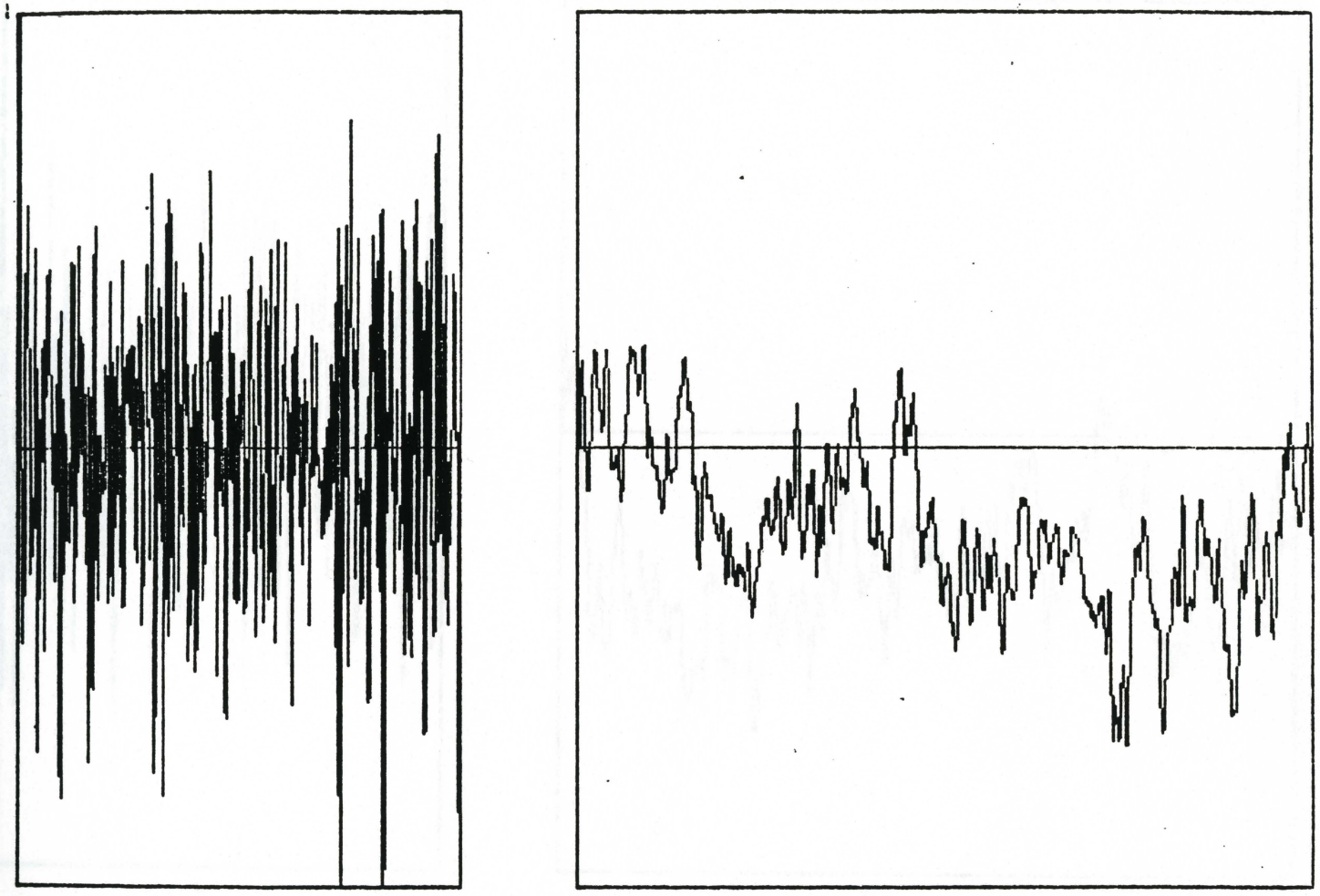
(a) (b)

Figure 17F.



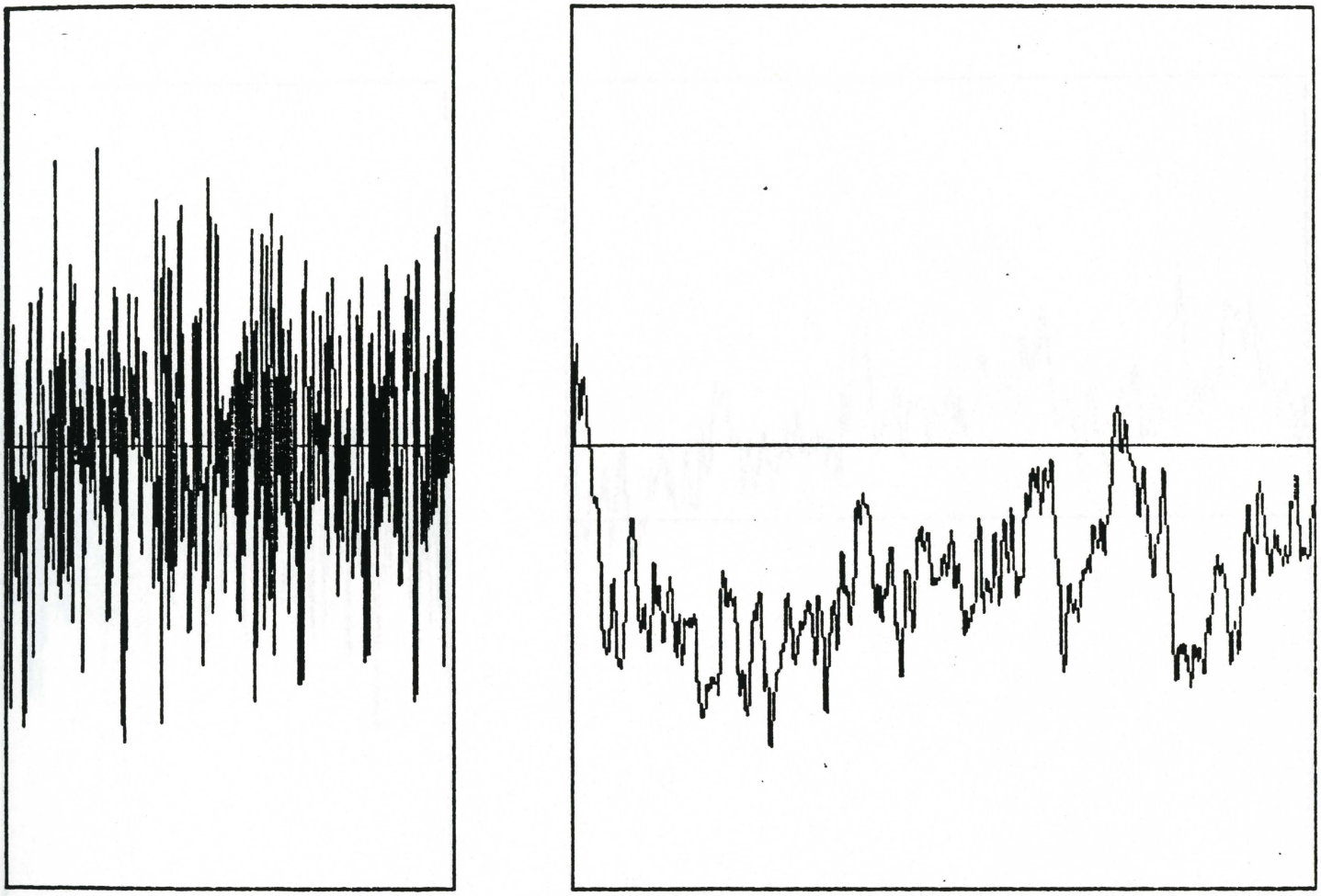
(a) (b)

Figure 18A



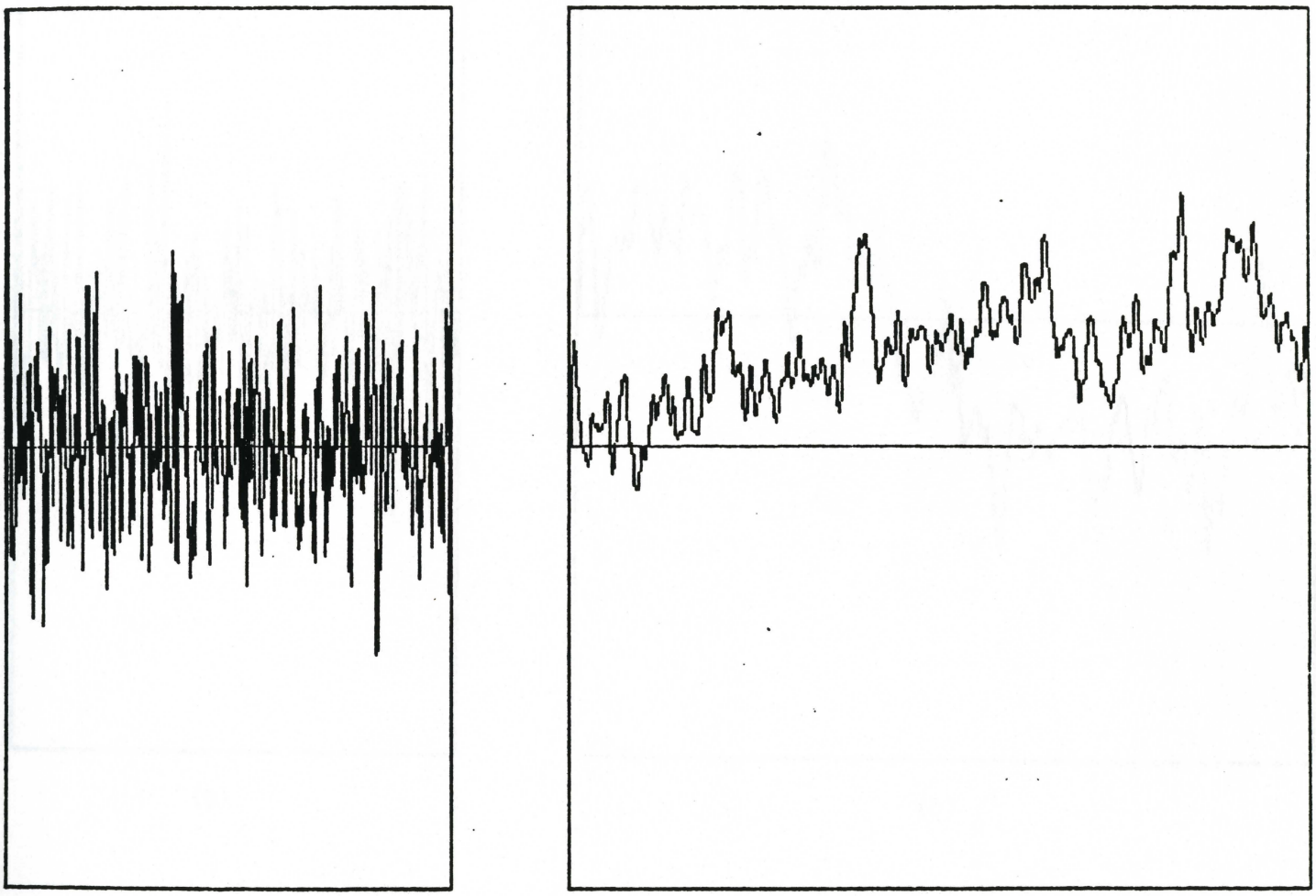
(a) (b)

Figure 18B



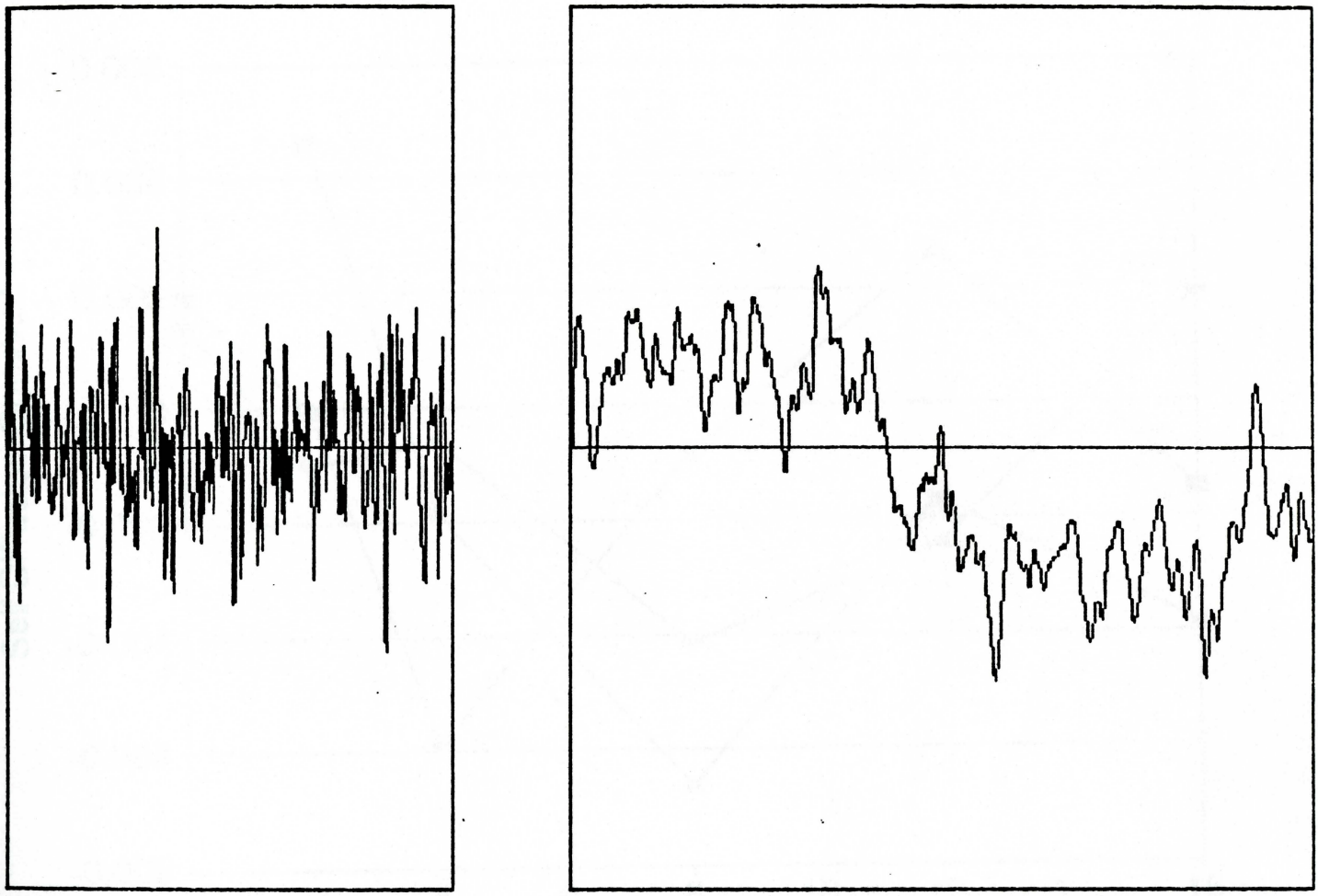
(a) (b)

Figure 18C



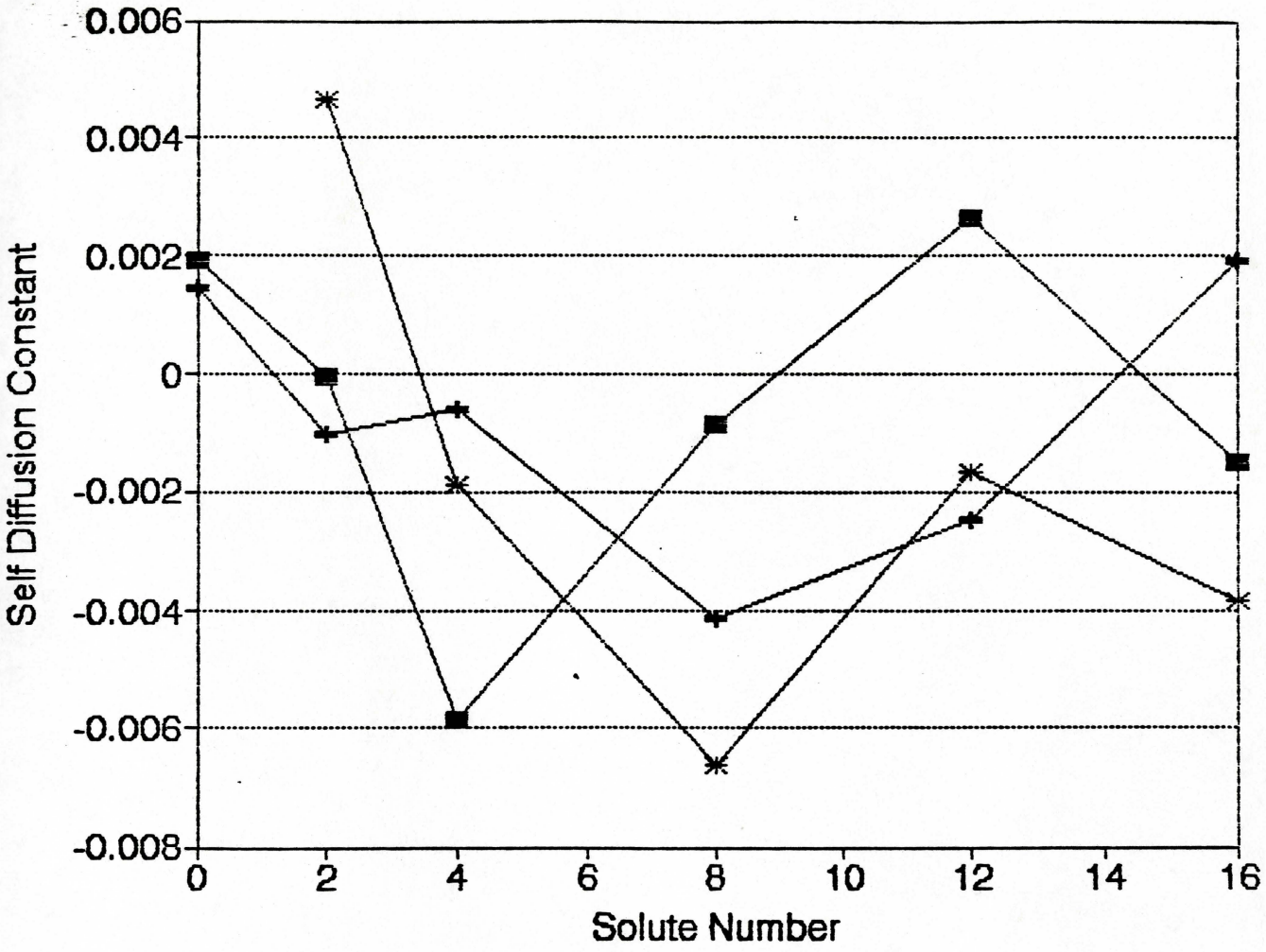
(a) (b)

Figure 18D



(a) (b)

Figure 18E



Bottom - cross

Top - Square

Solute - asterisk

Figure 19