Progress Report on
Particle Dynamics in a Wave with Variable Amplitude

John R. Cary

Principal Investigator

Department of Astrophysical, Planetary, and Atmospheric Sciences

University of Colorado

Boulder, CO 80309-0391

January, 1992

TRANSFERRED TO
DOE/ER/40302-6
JAN 11 1992

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

Abstract

Our past research efforts led to the derivation of the adiabatic invariant in spatially varying accelerator structures, to the calculation of the loss of the invariant due to trapping, and to a method for determining transverse invariants using a nonperturbative approach to the Hamilton-Jacobi equation. These research efforts resulted in the training of two graduate students who are now working in the area of accelerator physics.

Table of Contents

Abstract	ii
I. Introduction.....	iii
II. Spatial adiabatic trapping and emittance growth due to separatrix crossing: applications to free-electron lasers	7
A. Background and Accomplishments.....	7
B. Recent Accomplishments	12
III. Diffusive models of tracking results	19
A. Background	19
B. Accomplishments.	21
IV. Educational aspects	27
V. Contacts with the accelerator community.....	28
VI. References	29
Appendix A. Particle Dynamics in a Large-Amplitude Wavepacket	
Appendix B. Adiabatic Invariance for Spatially Dependent Accelerating Structures	
Appendix C. Report of the Working Group on the Nonlinear Analysis of Beams	
Appendix D. The Plot widget	

I. Introduction

Our past research efforts led to several advances in the field of the dynamics of particles in accelerators. We derived the adiabatic invariant for accelerating structures with slow spatial variation.^{1,3} This invariant is useful for understanding trapping in, e.g., radiofrequency quadrupoles. We further analyzed the loss of the adiabatic invariant due to the trapping process (separatrix crossing) in spatially varying systems.^{1,4} We also developed nonperturbative Hamilton-Jacobi methods for determining transverse invariants in nonlinear transport systems.⁵ These research efforts resulted in the training of two graduate students, David Bruhwiler and Bill Gabella, who are now working in the area of accelerator physics.

The research program is broad. Some of our efforts, such as that to determine whether tracking data for particle loss can be modeled by a diffusion equation, are direct applications to accelerators. Other efforts, such as the investigation of symplectic integrators, are more fundamental, but still relevant to accelerator physics. Discussing all of our projects would make this report unwieldy. So instead we concentrate on two of our projects: development of a free-electron laser experiment to test adiabatic trapping theory and modeling SSC tracking data by a diffusive model.

Our recent work on adiabatic theory has been to design a free-electron laser experiment to test adiabatic trapping theory. Separatrix crossing theory predicts that the emittance growth due to trapping in a free-electron laser can be greatly diminished by trapping adiabatically. This would be especially advantageous in a recirculating free-electron laser, where the longitudinal emittance growth due to each interaction of the beam with the wiggler must be eliminated by cooling.⁶ Less cooling would be needed if adiabatic trapping were used.

A possible location for such an experiment would be at the Accelerator Test Facility at Brookhaven National Laboratory, where there is a high-brightness electron beam of energy up to 50 MeV. As there is also a CO₂ laser at this facility, only an appropriate wiggler would be needed. Preliminary considerations show that adiabatic trapping and detrapping wiggler can be made that would be of the order of 2 meters long.

Our research to model tracking data by a diffusive model has the goal of understanding whether simple chaotic diffusion underlies the results of tracking calculations. In the tracking calculations, particles are launched at various distances from the design orbit. On average the confinement time increases as the launch point of the particles is moved towards that of the design orbit. However, the confinement time is not unique, because trajectories launched at the same distance, but at different locations around the ring, have different confinement times. The presence of this statistical aspect leads one to believe that the loss may be described by a chaotic diffusion, as is the case in other dynamical systems. We have been developing a method for determining whether chaotic diffusion is consistent with SSC tracking data. We show that there is a consistency relation among the first three moments of the confinement time if the loss is describable by a diffusion equation. We are in the midst of developing computer codes to see whether this consistency relation is satisfied for the given tracking data.

The ultimate goal of this research is to seek a dynamical basis for the diffusion equation, as has been found in other cases.⁷ Such a dynamical basis would allow one to derive the diffusion coefficient. Use of the diffusion equation rather than tracking data would greatly decrease the computational requirements for accelerator simulation. Moreover, with an understanding of the basic process, methods for diminishing the chaotic loss might be found.

To maintain strong ties between our university based accelerator research program

and the accelerator research at the national laboratories and other universities, we have continued to maintain a high level of contact with the accelerator community. An example is this year's sabbatical of the principal investigator at Fermilab. This sabbatical enabled the P.I. to interact with the accelerator physicists of the Accelerator Physics group led by Steve Peggs. In conjunction with Leo Michelotti and Jim Holt, the P.I. developed software for mouse-click interaction with tracking codes. With Bill Gabe'lla, this was used to study noise extraction of the beam periphery for a possible parasitic fixed-target experiment at the SSC. Documentation of this software is included in an appendix.

This research program has also been instrumental in training new accelerator physicists. As mentioned above, past funding allowed two graduate students to receive their Ph.D.'s in accelerator physics. Two new graduate students began this last year. Both are likely to finish their theses in 2-4 years. The current funding also provides for one postdoctoral fellow. This person, Carson Chow, completed his thesis in plasma physics and nonlinear dynamics. During his fellowship at the University of Colorado, Carson Chow will be able to learn the techniques and jargon of accelerator physics while doing research in this area. All three of these researchers will be attending the school on accelerator physics this summer.

In the following three sections we provide brief overviews of two areas of the proposed research: experimental tests of adiabatic trapping theory and diffusion models for SSC tracking data. Section IV discusses the educational aspects of this research program, while Sec. V notes the contacts made between the present research group and the accelerator community at large. The references are contained in Sec. VI. The appendices contain reports produced under past years of this research program.

II. Spatial adiabatic trapping and emittance growth due to separatrix crossing: applications to free-electron lasers

In previous years of this research project we developed the theory of spatial adiabaticity. We calculated the adiabatic invariant applying to the situation when an accelerating structure varies slowly in space rather than time. We also showed how to calculate the increase of emittance due to separatrix crossing. We have begun to design experiments to test these analyses. Our preliminary calculations show that these tests could be carried out in a free-electron laser experiment at the Accelerator Test Facility at Brookhaven National Laboratory.⁸ If successful, such experiments would point to a way of designing a recirculating free-electron laser with diminished beam cooling requirements, as sufficiently adiabatic trapping gives arbitrarily small emittance increase.

A. Background and Accomplishments

For a Hamiltonian $H(q,p,\epsilon t)$ depending slowly on the time variable, the action,

$$I(E,\epsilon t) \equiv \oint p|_{H=E} dq, \quad (2.1)$$

the phase-space area enclosed by a closed contour of H at fixed time, is the adiabatic invariant. *Slowly* means that the particle executes many oscillations before the parameters of the Hamiltonian change significantly. Invariance of the action, $I(E_i, \epsilon t_i) = I(E_f, \epsilon t_f)$, allows one to solve for the energy at one time in terms of the energy at another time. Adiabatic theory has also been used to predict the phase-space area occupied by a beam adiabatically trapped in an accelerating potential.

1. Spatial adiabatic invariance

We have analyzed the case of adiabatic, spatially varying, accelerating structures.

We have shown that the usual adiabatic invariants for the temporally varying case are not invariant in the spatially varying case. This affects, in particular, the arguments that give the area occupied by a beam that has been adiabatically trapped in a *spatially* varying structure. Our results apply to a large number of accelerator and plasma physics problems such as trapping in the accelerating potential of an RFQ,⁹ in the ponderomotive (decelerating) potential of free-electron lasers,¹⁰ in plasma heating¹¹ and in plasma turbulence.¹²

For such structures, the potential instead has an amplitude that varies spatially:

$$H(q,p,t) = \frac{p^2}{2m} + e\Phi(\epsilon q) \cos(k(q - ut)). \quad (2.2)$$

Now the particles see an amplitude growing as they enter the accelerating structure. After being trapped in a stable bucket they may be accelerated or decelerated depending on how the phase velocity u changes with position. A model potential is shown in Fig. 2.1.

Fig. 2.1. Potential for (2.2) versus coordinate.

Temporal adiabatic theory does not apply to this situation. One can see this by considering the case $u=0$, where the Hamiltonian is time independent and so H is conserved. Temporal adiabatic theory would say that for passing particles, the actions $I_{\pm}(E, \epsilon q)$, obtained by integrating along a contour of H either above (+) or below (-) the bucket separatrix at $H = e\Phi$, are invariants. However, if both $I_{\pm}$ and E are constant, then q must be constant if $I_{\pm}(E, \epsilon q)$ has a nontrivial dependence on q . This contradiction shows the previous adiabatic theory to be false.

In previous work ^{2,3} we showed that one could find the new adiabatic invariant by transforming to the new fast variable $q - ut$, which has the conjugate momentum $z \equiv p + K/u$, where

$$K(q-ut, p, \epsilon q) \equiv \frac{(p-\mu)^2}{2m} + e\Phi(\epsilon q) \cos(k(q-ut)) \quad (2.3)$$

is the new Hamiltonian for these variables, and ϵq is the new slowly varying time. One can see that K is the Hamiltonian in the frame moving with the phase of the wave. The adiabatic invariant is then given by the integral,

$$J = \int (p + K/u) d(q - ut) = \int p d(q - ut) + (K/u) \int d(q - ut) , \quad (2.4)$$

along a contour of constant K with the slow spatial dependence ϵq held fixed.

For trapped particles, the phase $(q - ut)$ begins and ends at the same point. Hence, the second term vanishes and we have,

$$J_T = \oint p d(q - ut) \equiv 2I(\epsilon q, K). \quad (2.5)$$

The last equality defines the variable I , which for trapped particles is simply half the enclosed area. (This factor of $1/2$ makes I a continuous phase-space variable.) Thus, the

trapped-particle spatial adiabatic invariant is also the trapped-particle temporal adiabatic invariant. This must be, since a trapped particle trajectory, being confined to a single bucket, does not sample enough of the potential to determine whether the amplitude is spatially or temporally varying. Because I is half the temporal adiabatic invariant for trapped particles, and it is the temporal adiabatic invariant for passing particles, we will call I the action.

For passing particles, the phase $q-ut$ does not begin and end at the same value, but changes by $2\pi/k$. Thus, the adiabatic invariant for $\pm$ passing particles is given by

$$J_{\pm} = I + 2\pi H/ku, \quad (2.6)$$

with

$$I(\epsilon q, H) \equiv \int_0^{2\pi} (p - u) d(q-ut) \quad (2.7)$$

being the integral along a contour of constant K . The convention chosen here of integrating in the direction of increasing phase $q-ut$ makes I a signed quantity; it is positive (negative) for positive (negative) passing particles.

2. *Adiabatic invariant change due to separatrix crossing*

In the early years of this line of research, we pioneered the theory of adiabaticity loss due to separatrix crossing in adiabatic systems. We review this theory first for the case of a Hamiltonian, $H(q,p,\epsilon t)$, with one degree of freedom and slow time-dependence, i.e. small ϵ . Adiabatic invariance theory states as the contours of H change in the course of time, for small ϵ a given trajectory can always be found on a contour containing the same area as did the contour of the trajectory at the initial time. The requirement for ϵ is that it must be small compared with the orbit frequency ν_0 calculated by neglecting the change of the Hamiltonian function with time.

Unfortunately, adiabatic invariance breaks down for particles near the separatrix. (A separatrix is a contour of H containing an x-point, a saddle point, which is a fixed point of the dynamics for $\epsilon = 0$. Contours of a typical Hamiltonian with a separatrix are shown in Fig. 2.2.) This causes adiabatic invariance to break down in a large number of cases, because a separatrix can sweep across large regions of phase space in the course of time. For example, if the separatrix in Fig. 2.2 is growing in time, it collides with a trajectory in region-c at the time t_x given by $Y_c(\epsilon t_x) = J_i$, in which $Y_c(\epsilon t_x)$ is the area enclosed by the entire separatrix, and J_i is the initial value of the adiabatic invariant. This time t_x is called the pseudo-crossing time. In the vicinity of this time the adiabatic invariant is inapplicable.

Figure 2.2. Contours of a typical Hamiltonian with a separatrix.

We analyzed the consequences of crossing a separatrix in detail previously.¹³ It was found that the new value of the adiabatic invariant depends on the lobe of the separatrix into which the trajectory becomes trapped. For a trajectory becoming trapped into, say, lobe-a, (See Fig. 2.2) the new value of the adiabatic invariant, J_{fa} , is close to $Y_a(\epsilon t_x)$, the phase-space area enclosed by lobe-a of the separatrix at the pseudo-crossing time. The deviation, $\delta J_{fa}(M_c) \equiv J_{fa} - Y_a(\epsilon t_x)$, is found to depend on a crossing parameter M_c ,

which is related to the initial phase, or adiabatic angle, of the trajectory. This implies that an ensemble initially distributed on an adiabatic invariant ring, a single contour, is ultimately distributed throughout an adiabatic invariant annulus after a separatrix crossing. Thus, separatrix crossing leads to a spread of the adiabatic invariant.

In previous years of this research,⁴ separatrix crossing theory was used to calculate the increase of emittance of particles repeatedly trapping and detrapping in a wave. Separatrix crossing theory provides an analytic expression for the increase of emittance at each crossing. Summing the increases with the random-phase approximation gave an effective diffusion coefficient describing the long-time rate of emittance growth.

More recently, separatrix crossing theory was applied to problems with spatial adiabaticity.¹ We explicitly calculate the growth of emittance for a set of trajectories passing through accelerating fields of the form (2.2). We show, by comparison with direct numerical integration, that the analytic theory is accurate in the appropriate regime, i.e., when the typical synchrotron length is small compared with the scale length of the wavepacket.

B. Recent Accomplishments

1. Predictions of spatial adiabatic theory for longitudinal acceptance

In the past two years we investigated in detail the consequences of spatial adiabatic theory for trapping in spatially varying structures. We found that a beam of particles of incoming momentum does not trap at the same amplitude as is found from temporal adiabatic theory. Instead, both the location and the trapping capability of a wave of some given amplitude are affected. In particular, there exist values of incoming momentum which cannot be trapped. These results are discussed in some detail in Refs. 12 and 13. Below we summarize these results.

We consider a beam of particles with momentum $p_i = m(u + \Delta)$ far from the accelerating structure. Such particles are in the positive passing state and have initial value of the adiabatic invariant,

$$J_{+i} = (2\pi m/k)(\Delta + K/\mu) = (2\pi m/k)\left(\Delta + \frac{1}{2}\Delta^2/u\right). \quad (2.8)$$

The last equality follows from the fact that $K = (p - \mu)^2/2m$ far from the accelerating structure. These particles trap when the value,

$$Y_+ = 8m A^{1/2}/k + 2\pi m A/ku, \quad (2.9)$$

of the positive passing adiabatic invariant on the separatrix equals the initial value of the adiabatic invariant. Equating (2.8) and (2.9) shows that these particles become trapped at amplitude A_x satisfying

$$\Delta + \frac{1}{2}\Delta^2/u = 4 A_x^{1/2}/\pi + A_x/u, \quad (2.10)$$

which implies that these particles are spread throughout an area of

$$a_{q+}(\Delta) = 16mA_x^{1/2}/k = (32mu/k\pi)\left(\sqrt{1 + \pi^2\Delta/4u + (\pi\Delta/2u)^2/2} - 1\right), \quad (2.11)$$

which is the area enclosed by the accelerating bucket at the position of trapping. For small values of the momentum difference Δ (and, therefore, trapping amplitude), the result is identical to that obtained in the temporally varying case. However, for large positive values of Δ/u one finds a correction of the order of 10%.

For particles with negative values of Δ , the difference between the spatially and temporally varying cases is much greater. Such particles trap when the initial value, J_{-i} , of the negative-passing-state adiabatic invariant, which is also given by (2.8) far from the

accelerating structure, matches the value,

$$Y_- = -8mA^{1/2}/k + 2\pi mA/ku, \quad (2.12)$$

of the negative-passing-state adiabatic invariant on the separatrix. Equating (2.8) and (2.12) gives

$$a_{q-}(\Delta) = 16mA_x^{1/2}/k = (32mu/k\pi) \left(1 \pm \sqrt{1 + \pi^2 \Delta/4u + (\pi\Delta/2u)^2/2} \right). \quad (2.13)$$

The sign of the radical must be chosen such that the area is positive.

For small, negative Δ , the condition of positive area indicates that the minus sign must be chosen in (13). This sign continues to be valid until the radicand becomes negative, just beyond which there is no solution. The range in which there is no solution is

$$|p| < (1 - 8/\pi^2)mu. \quad (2.14)$$

Thus, initial momenta in this range can never be trapped. They are always reflected by the ponderomotive potential before colliding with the separatrix. This effect never occurs in the temporally varying case.

For still more negative values of the momentum, $p < (8/\pi^2 - 1)mu$, particles entering the accelerating structure from the left will interact with the separatrix if the potential amplitude is large enough, but only in becoming untrapped there, as a more detailed analysis¹¹ shows. The untrapped particles originate from positive-passing incident particles that pass through the separatrix to become reflected, negative-passing particles. We call this *resonant reflection*.

We have seen that spatially adiabatic systems can be quite different from temporally adiabatic systems. In temporally adiabatic systems all particles can be trapped if the

amplitude is increased sufficiently. In contrast, no particles in a spatially adiabatic system with momentum less than $(1 - 8/\pi^2)\mu$ can be trapped by the potential, no matter how large it becomes. Moreover, in spatially adiabatic systems, there is the new phenomenon of resonant reflection.

2. *Estimates of experimental parameters needed to observe adiabatic trapping*

We would now like to see if experimental tests of adiabatic trapping theory can be performed. Adiabatic theory predicts that the increase of longitudinal emittance of a beam in trapping process is proportional to the adiabaticity parameter. The adiabaticity parameter is proportional to the characteristic orbit period divided by the characteristic time over which the trapping takes place. For example, for trapping in a longitudinal bucket with synchrotron bounce period λ_b over a length L , the emittance increase is proportional to λ_b/L .

Experimental tests are needed to see whether such theory is relevant to a real situation, in which many real-world effects, such as transverse motion and experimental errors and unknowns, come into play. As the Accelerator Test Facility at Brookhaven National Laboratory is nearing completion, and it provides many of facilities needed for such tests, we decided to investigate whether tests of adiabatic theory could be carried out there.

The Accelerator Test Facility (ATF) has both a high-power CO_2 laser and an electron linac capable of producing electrons with energies of five to 50 MeV. Combining these two items with an appropriate wiggler produces a free-electron laser amplifier with a large ponderomotive potential. The adiabaticity requirement is that the synchrotron length be short compared with the scale length of the ponderomotive potential. Thus, our calculations have been geared towards finding an experimental setup within the restrictions of available laser power and beam energy such that an adiabatic wiggler could be

constructed.

In a free-electron laser, an electron beam of relativistic factor γ passes through a spatially oscillating magnetic field of wavelength λ_w . The strength of the magnetic field is represented by the dimensionless vector potential,

$$a_w = \frac{eB_w}{mk_w c^2}, \quad (2.15)$$

where $k_w \equiv 2\pi/\lambda_w$ and B_w is the peak wiggler field. If a laser of wavelength λ_s is present, a ponderomotive potential is produced. This potential is resonant with the electrons when¹⁴ the condition,

$$\lambda_s = \lambda_w / 2\gamma_r^2, \quad (2.16)$$

is satisfied, where the reduced relativistic factor is given by

$$\gamma_r = \gamma / (1 + \frac{1}{2} a_w^2)^{1/2} \quad (2.17)$$

for a linearly polarized wiggler. If the electrons are trapped, the wavelength λ_w can be decreased, while adiabaticity guarantees that Eq. (2.16) remains satisfied. Thus, the electrons lose energy, which is transferred to the laser light.

The adiabaticity requirement is satisfied if the electrons execute many synchrotron-bounce oscillations in the time for the beam to slow down. The synchrotron bounce wavenumber is given by

$$k_b^2 = \frac{e^2 B_w E_s}{m^2 c^4 \gamma_r^2}, \quad (2.18)$$

where E_s is the peak laser electric field in cgs units. Thus, adiabaticity is satisfied when

$\lambda_b \ll L$, where L is the characteristic length over which the system changes, and $\lambda_b \equiv 2\pi/k_b$ is the synchrotron wavelength.

This same condition is required for adiabatic trapping. To satisfy this condition, we imagine a constant laser power along the electron beam. The ponderomotive potential, proportional to the product of the laser amplitude and the wiggler amplitude, is constructed to increase slowly over a length L by having the wiggler amplitude vary slowly. This implies that we want λ_b to be as small as possible. Thus, we need large laser power and wiggler strength, but low γ .

A preliminary design was arrived by the following procedure. The CO_2 laser wavelength is $10.6\mu\text{m}$. To keep the relativistic factor low, we chose $\lambda_w = 0.8\text{ cm}$, consistent with the wiggler built by Stoner.¹⁵ Equation (2.16) then implies $\gamma_r = 19.8$, or that the electron beam have energy 9.6 MeV . We set the wiggler strength to be 5 kG , again consistent with Stoner's design. Next we decided on a synchrotron wavelength of 22 cm . This would allow 9 synchrotron wavelengths across a 2 meter wiggler. Equation (2.18) then implies that the laser field must be $1.8 \times 10^4\text{ statvolt/cm}$.

The laser field is limited by the laser power and the laser spot size. We chose the laser power to be 10 GW as is expected at ATF. This was used in the relation $P_s = c(aE_s)^2/8$ between the power, the laser spot size a , and field strength to determine the spot size. We obtained a spot size of 2 mm . This implies a Rayleigh length of 78 cm , which, since the ATF laser is diffraction limited, implies that the laser beam will not spread excessively from its minimum spot size over the length of the wiggler.

We also checked whether such a configuration would have a sufficiently large bucket for trapping the electrons. The ponderomotive potential amplitude V is related to the synchrotron bounce wavenumber by

$$k_b = (k_s^2 V / m \gamma_r^3)^{1/2} / c. \quad (2.19)$$

This is the standard result for a particle of mass $m \gamma_r^3$ moving at velocity c bouncing in a longitudinal potential of amplitude V and wavenumber k_s . The bucket height in momentum in this situation is given by $\Delta p^2 / 2 m \gamma_r^3 = V$. From this one can show that the relative bucket height is given by

$$\Delta p / p = 2 \gamma_r^2 \lambda_s / \lambda_b. \quad (2.20)$$

For the situation described above, the relative bucket height is 4%. This is much greater than the expected 0.3% energy spread of the ATF beam.

Lastly, we checked whether the beam's finite radius, which cause different beam particles to see different wiggler strengths, would cause significant variation of the reduced relativistic factor. For the case just discussed, the dimensionless vector potential is 0.37. Hence, even at 100% variation of the wiggler strength, the divisor in Eq. (2.17) varies by only 2%. This effect is therefore expected to be negligible.

III. Diffusive models of tracking results

The purpose for the following work is to discover whether particle loss in an accelerator can be described by existing models of chaotic diffusion. We plan to answer this question by testing the consistency of such models against tracking code results. The success of a diffusive model will give the properties of particle loss in terms of a diffusion coefficient. Therefore, a diffusive model will greatly reduce the effort and time needed to evaluate the quality of beam transport lines.

A. Background

At present tracking codes are used to determine the quality of beam transport lines. Tracking codes follow the particles element by element around the accelerator. In Fig. 3.1 is a sketch of tracking results for the SSC.¹⁶ The abscissa is the launch betatron amplitude. The ordinate is the confinement time of an orbit, measured in circuits around the machine. This figure shows that the confinement time of particles increases exponentially with launch betatron amplitude, A . This figure also shows that there are several values of the confinement time for each launch amplitude. This happens because particles may be launched at the same amplitude, but with different phases, i.e., at different locations along the machine.

This process is statistical in the same way that many chaotic systems are statistical: Nearby orbits can evolve very differently. So we now ask whether these results can be described by a diffusion equation,^{4,7} in parallel with previous work in nonlinear dynamics. In this line of research, there is an underlying approximate invariant, J . Perturbations cause the motion to look like diffusion in this approximate invariant, so that the distribution function obeys the equation,

$$\frac{\partial f}{\partial t} = \frac{\partial}{\partial J} \left(D(J) \frac{\partial f}{\partial J} \right). \quad (3.1)$$

In fact, this special form of the diffusion equation applies only when the variable J is one in which the steady-state solution for particles trapped in some region is of constant density. Thus, for an ergodic Hamiltonian system, from which one obtains (3.1) by integrating out other degrees of freedom, the differential element dJ must be proportional to the volume of phase space in the differential region between surfaces J and $J+dJ$.

Fig. 3.1. Sketch of confinement results from the tracking codes of the SSC group.

Two functions must be known to have a diffusion equation (3.1) for these tracking results. The first is the diffusion coefficient $D(J)$. In other dynamical problems⁷ this diffusion coefficient has been determined from first principles. Here the goal is to determine the coefficient from the data. The second unknown function $J(A)$ relates the phase-space variable J to the betatron launch amplitude A . Were the underlying approximate invariant the linear betatron invariant for two coupled transverse degrees of

freedom, the phase space volume, that of a four dimensional ellipsoid, would be proportional to A^4 . However, here we hope to determine $J(A)$ also from the data.

B. Accomplishments

Our goal is to determine whether the tracking data of Fig. 3.1 are consistent with the diffusion model (3.1). To do so, we must first determine the two arbitrary functions, $D(J)$ and $J(A)$. Then we must use these functions to make a prediction about the data. If the prediction is consistent, we would then consider trying to find such a diffusion model from first principles.

Our method will be to consider the confinement-time moments as a function of the launch amplitude. The diffusion model makes a prediction for all such moments. By using two such moments, we will be able to deduce the two functions, $D(J)$ and $J(A)$. Comparison of the prediction of the third moment with the result from tracking data will determine consistency. Hence, our first goal is to show how these moments rely on the chosen diffusion equation.

To solve the diffusion equation, we must decide on boundary conditions. Figure 3.1 shows that for smaller A the confinement time is very large. This implies that the diffusion coefficient is becoming very small. For the trajectories launched at larger amplitudes the precise small-amplitude dynamics are unimportant, so we can just as well assume that there is a "hard" wall at some minimum value $A_{\min}$ of the amplitude. This corresponds to choosing the boundary condition,

$$\left(D(J) \frac{\partial f}{\partial J} \right)_{J_{\min}} = 0. \quad (3.2)$$

At large amplitudes the trajectories are lost in just a few turns. Hence, we can assume there is a loss boundary at $J_{\max}$ at which the boundary condition,

$$f(J_{\max}) = 0, \quad (3.3)$$

holds. Here $J_{\min}^{\max} \equiv J(A_{\min}^{\max})$.

We are now ready to calculate the confinement-time moments. We define the function $F(J, t; J_0)$ to be the solution to Eq. (15) satisfying

$$F(J, 0; J_0) = \delta(J - J_0). \quad (3.4)$$

This is just the Green's function for this system. If we know F , then we can calculate an arbitrary evolution and, hence, all of the confinement-time moments. However, instead of calculating F , we will derive a recursive relation between successive moments.

The probability for particles launched at J_0 to remain in the system at time t is

$$P(t, J_0) \equiv \int_{J_{\min}}^{J_{\max}} dJ F(J, t; J_0). \quad (3.5)$$

(This probability decays exponentially fast provided the spectrum of the diffusion operator is discrete.) Thus, the loss rate is given by

$$v \equiv -\frac{\partial P}{\partial t} = -\left(D(J) \frac{\partial F}{\partial J}\right)_{J_{\max}}. \quad (3.6)$$

The mean loss time,

$$\tau(J_0; D) = \int_0^{\infty} dt \, t \, v(t; J_0) = \int_0^{\infty} dt \, P(t; J_0), \quad (3.7)$$

is simply the time weighted by the loss rate. The last equality here follows from integration

by parts. On left side of Eq. (3.7) we have indicated that the mean loss time is not only a function of the launch position, but also a functional of the diffusion coefficient.

Because Eq. (3.1) is linear and time independent, we introduce the Laplace transform. We get the relation,

$$s G = \frac{\partial}{\partial J} \left(D(J) \frac{\partial G}{\partial J} \right) + \delta(J - J_0) \quad (3.8)$$

for the Laplace transform G of the distribution F . Similarly, we introduce the Laplace transform,

$$\mu(s) \equiv \int_0^{\infty} dt e^{-st} v(t) = - \left(D(J) \frac{\partial G}{\partial J} \right)_{J_{\max}}, \quad (3.9)$$

of the loss rate. Differentiation of Eq. (3.9) with respect to s and setting s to zero gives the negative of the mean loss time,

$$\left. \frac{\partial \mu}{\partial s} \right|_{s=0} = - \int_0^{\infty} dt t v(t) = - \tau(J_0; D). \quad (3.10)$$

Combining (24) and (25) gives the mean loss time as

$$\tau(J_0; D) = \left(D(J) \frac{\partial^2 G}{\partial s \partial J} \right)_{J_{\max}, s=0}. \quad (3.11)$$

The quantity inside the parentheses of Eq. (3.11) can be found by integration of the appropriately differentiated form of Eq. (3.8). We will not repeat all of the details here. The result is

$$\tau(J_0; D) = \int_{J_0}^{J_{\max}} dJ \frac{(J - J_{\min})}{D(J)}. \quad (3.12)$$

Differentiating this with respect to J_0 gives the local relation,

$$\frac{\partial \tau}{\partial J_0} = -\frac{(J_0 - J_{\min})}{D(J_0)}, \quad (3.13)$$

between the diffusion coefficient and the mean confinement time.

Equation (3.13) is the first relation to use in determining whether a diffusion model is applicable. However, we have yet to derive the relation between the betatron amplitude and the invariant J , and we have yet to determine a consistency condition. For this we calculate the higher moments.

We use τ_n to denote the n^{th} moment of the confinement time. This moment is given by

$$\tau_n(J_0) \equiv \int_0^\infty dt t^n v(t; J_0) = \left(-\frac{\partial}{\partial s} \right)^n \mu(s) \Big|_{s=0}. \quad (3.14)$$

A recursive formula for τ_n can be deduced from Eq. (3.8), by differentiating with respect to s , setting s to zero, and using the moment generating function. The result is

$$\tau_n(J_0, D) = n \int_{J_0}^{J_{\max}} \frac{dJ'}{D(J')} \int_{J_{\min}}^{J'} dJ'' \tau_{n-1}(J_0, D) \quad (3.15)$$

Here we have derived a relation by which one can predict successive moments of the confinement time. Examining the consistency of these moments will determine whether a

diffusion model exists.

In the last year, we tested the diffusive model by examining higher moments. The procedure for testing the diffusive model is as follows. One fits the function $\tau_1(A_0)$ to the tracking results. From this function Eq. (3.13) can be used to determine the diffusion coefficient, $D(J_0)$. Next the relation $J(A)$ is inverted and substituted in Eq. (3.12) giving the predicted second moment. This is then compared to $\tau_2(A_0)$ from the tracking results.

Fig. 3.2 Sketch of second confinement time moments. The solid line shows predicted second moments for various J_0 with $m = 4$. The points are from a tracking code

In performing this test, we assumed a power relation between the variable J and the launch amplitude A ,

$$J = A^m \tag{3.16}$$

and repeated to test for $m = 1, 2, 3, 4$, and 6. Figure 3.2 shows the results for $m = 4$. It can

be seen from the figure that agreement for τ_2 is good for J_0 near $J_{\min}$, but becomes increasingly poor as J_0 approaches $J_{\max}$. Other values for m produce similar results. This indicates that if a diffusive model exists, a power relation between J and the launch amplitude A is not correct.

Therefore we proceed to determine the functional relation

$$J = J(A) \quad (3.17)$$

To determine this relationship we start with Eq. (3.12) and τ_2 derived from Eq. (3.15). Combining these equations we eliminate the diffusion coefficient and arrive at the following differential equation for $A(J)$,

$$\frac{dA}{dJ} = \frac{2\tau_1 - \left(\frac{d\tau_2}{d\tau_1}\right)_{J_{\max}} + \frac{d\tau_2}{d\tau_1}}{(J_{\max}-J_{\min}) \frac{d}{dA} \left(\frac{d\tau_2}{d\tau_1}\right)_{J_{\max}} - (J_0-J_{\min}) \frac{d}{dA} \left(\frac{d\tau_2}{d\tau_1}\right)}. \quad (3.18)$$

In practice, we fit the first and second confinement time moments to tracking results giving $\tau_n(A)$. The derivatives in Eq. (3.18) are then calculated as,

$$\frac{d\tau_2}{d\tau_1} = \left(\frac{\frac{d\tau_2}{dA}}{\frac{d\tau_1}{dA}} \right) \quad (3.19)$$

Once Eq. (3.18) is solved, we have the desired relation $A(J)$. Next $D(J)$ is calculated using Eq. (3.13). We then proceed with Eq. (3.15) to test the consistency of the third moment predicted by our theory with that given by the tracking code.

IV. Educational aspects

There has always been and will continue to be a strong educational aspect to this research grant. Past funding of this research enabled two students to obtain their Ph. D.'s. The first, David Bruhwiler, has recently taken a job at Grumman in Princeton. He will be working with Ira Lehrman on free-electron lasers. The second, Bill Gabella, is working as a postdoc for Professor Klein of UCLA. Bill is now interacting strongly with the Fermilab Accelerator Physics Department. He is also involved in calculations for the Super Fixed Target Proposal.

There are now two Ph. D. students associated with this research program. The first, Scott Hendrickson, began working with the PI in January 1991. He has been involved in the calculations on the diffusion description of tracking results and on the free-electron laser calculations. The second student, Peter Stoltz, began working on accelerator/dynamics research in June 1991. He has been involved in the calculation of Lyapunov exponents for randomly phased perturbations. He was supported in the summer of 1991 by this grant. He needed no support for this academic year, as he had a two-year fellowship. However, if he is to continue research in accelerator physics, funding for a research assistantship must be provided.

Additionally, this grant provides funding for Carson Chow, the postdoctoral fellow. Dr. Chow comes from MIT, where he completed his thesis in spatio-temporal chaos in plasma turbulence. He, therefore, has a good basic understanding of dynamical phenomena. Coming to Colorado will provide Dr. Chow with the opportunity to learn accelerator physics. It is planned that his initial project will be to understand nonlinear accelerator transport systems.

V. **Contacts with the accelerator community**

As the proposed research takes place at a University far from any of the large accelerator laboratories, we have continued to interact strongly through travel and electronic communication. For example, we have obtained SSC tracking data from e-mail from Yiton Yang of SSC. In addition to the regular accelerator conferences, we attended the Accelerator Test Facility review that took place in October of 1991 in order to find out from Drs. Kirk and Ben-Zvi what facilities are available.

At the writing of this report, the PI is on sabbatical at Fermilab, where he is working in the Accelerator Physics Department of the Accelerator Division. During this sabbatical the PI has had the opportunity to interact with many of the accelerator physicists and to learn about accelerator physics operations. The PI did a week in the control room. He became involved in discussions with Pat Colestock, Kathy Harkay, and Andre Gerasimov on collective instabilities in the booster and the Tevatron. He also discussed basic accelerator physics experiments with Todd Satogata and Bill Gabella, an accelerator physicist whose thesis research was supported under this grant.

By far the largest amount of effort was expended in learning about new workstation tools for computational analysis. The PI wrote an interface based on the X window system for tracking codes. This interface allows a researcher to specify an initial point in phase space by a mouse click in a phase space plot. Simple tracking codes have been grafted onto this interface to study the effects of power supply ripple and external noise on particle confinement. Perhaps unfortunately, because it detracted from physics research, the PI also became involved in installing the appropriate software (Release 4 of version 11 of X and the Motif widgets and window manager) on the workstations used by the Accelerator Physics Department. This code is included as App. D.

VI. References

- ¹ D. L. Bruhwiler, *Scattering and Diffusion of Particles in Slowly-Varying Large-Amplitude Waves*, Ph. D. thesis, University of Colorado, Boulder (1990).
- ² D. L. Bruhwiler and J. R. Cary, Phys. Rev. Lett. (1991), in press. Included as App. A.
- ³ J. R. Cary and D. L. Bruhwiler, AIP proceedings of 89 US-Japan workshop. Included as App. C.
- ⁴ D. L. Bruhwiler and J. R. Cary, Physica **40D**, 265 (1989).
- ⁵ W. E. Gabella, *Numerical Solution of the Hamilton-Jacobi Equation in 2 1/2 Degrees of Freedom*, Ph. D. thesis, University of Colorado, Boulder (1991).
- ⁶ A. Renieri, Nuovo Cimento **53B**, 160 (1979).
- ⁷ A. B. Rechester and R. B. White, Phys. Rev. Lett. **44**, 1586 (1980); A. B. Rechester, M. N. Rosenbluth and R. B. White, Phys. Rev. A **23**, 2664 (1981); J. R. Cary, J. D. Meiss and A. Bhattacharjee, Phys. Rev. A **23**, 2744 (1981).
- ⁸ I. Ben-Zvi, Bull. Am. Phys. Soc. **36**, 1512 (1991).
- ⁹ I. M. Kapchinskii and V. A. Teplyakov, Prib. Tek. Eksp. **119**, 19 (1970); S. O. Schriber, Proc. 1985 Particle Accelerator Conference (Vancouver, British Columbia, 1985) and references therein.
- ¹⁰ For example, C. W. Roberson and P. Sprangle, Phys. Fluids **1**, 3 (1989); N. M. Kroll, P. L. Morton, and M. N. Rosenbluth, IEEE J. Quant. Elec. **17**, 1436 (1981).
- ¹¹ W. M. Nevins, T. D. Rognlien, and B. I. Cohen, Phys. Rev. Lett. **59**, 60 (1980).
- ¹² V. Fuchs, V. Krapchev, A. Ram, and A. Bers, Physica **D14**, 141 (1985).
- ¹³ J. R. Cary, D. F. Escande, and J. L. Tennyson, Phys. Rev. **A34**, 4256 (1986); J. R. Cary and R. T. Skodje, Phys. Rev. Lett. **61**, 1795 (1988); J. R. Cary and R. T. Skodje, Physica **36D**, 287 (1989).

- ¹⁴ N. M. Kroll, P. L. Morton and M. N. Rosenbluth, IEEE J. Quantum Electron. **17**, (1981); T. J. Orzechowski, B. R. Anderson, J. C. Clark, W. M. Fawley, A. C. Paul, D. Proznitz, E. T. Scharlemann, S. M. Yarema, D. B. Hopkins, A. M. Sessler and J. S. Wurtele, Phys. Rev. Lett. **57**, 2172 (1986).
- ¹⁵ R. Stoner, S. C. Chen and G. Bekefi, Bull. Am. Phys. Soc. **36**, 2437 (1991).
- ¹⁶ A. Chao, in *Nonlinear Dynamics and Particle Acceleration* (AIP Conf. Proc. no. 230), (American Institute of Physics, New York, 1991) p. 203.

preprints removed.
cb

Appendix D. The plot widget

The following is the coding developed for interactive tracking of one and a half degrees of freedom. This code uses the libraries of release 4 of version 11 of the X window system. Clicking the mouse at a point on the graph initiates a trajectory. The iterates of the trajectory are displayed in the window.

```

/*
 * xmaps.c - Program to to surfaces of section
 */

/* Unix stuff */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/*
 * Include files required for all Toolkit programs
 */
#include "X11/Intrinsic.h"      /* Intrinsics Definitions */
#include <X11/StringDefs.h>      /* Standard Name-String definitions */
#include <X11/Core.h>           /* The core translations*/

/* The structures for xmaps*/
#include "xmaps.h"

/* Need to know structure passed back */
#include "PlotWindow.h"

/* The data for this program*/
POSITION pos=(0.001,0.001);      /* The position of present trajectory*/
POSITION initPos=(0.001,0.001); /* The initial position of this trajectory*/
Boolean going=FALSE;             /* Whether we are integrating*/
Boolean drawFirst=TRUE;          /* Whether to draw the first point*/
Boolean sosMade=FALSE;           /* Whether sos ever made*/
XPlot sosW=NULL;                 /* The surface of section plot*/

/* Map specification */
int imap=1;

/* The data for the Standard map (imap=0) */
double eps=.99,kStd;

/* Data for the Gabella Map (imap=1) */
double linTune=0.66000031415926535,sexStrength=.1,octStrength=0.;
double linFreq,cosFreq,sinFreq;
double noiseLev=0.001;

/* Universal constants*/
double pi,twopi;

/* My error routine */
void MyError(char* message)
/* My error routine to keep code in the machine*/
{
    char inChar;
    fprintf(stderr,"Error from X:\n%s",message);
    fprintf(stderr,"MyError: waiting for q to exit\n");
    while( inChar != 'q' ) inChar = getchar();
    fprintf(stderr,"MyError: q received.  Exiting.\n");
    exit(1);
}

void InitMap(){
    linFreq = twopi*linTune;
    cosFreq = cos(linFreq);
    sinFreq = sin(linFreq);
}

void SetConstants(){
/* Set the global constants*/
    pi = acos(-1.);
    twopi = 2.*pi;
}

```



```

    kStd = eps/twopi;
    printf("kStd = %g\n",kStd);

}

void ReceiveClick(      /* This routine receives the click*/
    Widget      wdg,    /* from this widget */
    char*       name,   /* of this name */
    ClickData   cData) /* at this specified point */
{
#ifdef DEBUG
    fprintf(stderr,"ReceiveClick called with x = %g and y = %g\n",
        cData->x,cData->y);
#endif
    initPos.x = cData->x;
    initPos.y = cData->y;
    pos = initPos;
    going = !going;
}

void GabellaMap(POSITION *posPtr){ /* Iterate quadratic map */
    double dum;

    /* First do simple rotation */
    dum = posPtr->x*cosFreq + posPtr->y*sinFreq;
    posPtr->y = - posPtr->x*sinFreq + posPtr->y*cosFreq;
    posPtr->x = dum;

    /* Now add some nonlinearity */
    posPtr->y += (sexStrength + octStrength*dum)*dum*dum;

    /* Now add some random noise */
    posPtr->y += noiseLev*(2.*drand48() - 1.)*dum*dum*dum;
}

void StandardMap(POSITION *posPtr){/* Advance using standard map */

    posPtr->y += kStd*sin(twopi*posPtr->x);
    posPtr->y -= floor(posPtr->y);
    posPtr->x += posPtr->y;
    posPtr->x -= floor(posPtr->x);
}

void AdvAndPlot(      /* Advance and plot */
    POSITION *posPtr)   /* this point */
{
    /* Advance the points */
    switch(imap){
        case 0:
            StandardMap(posPtr);
            break;
        case 1:
            GabellaMap(posPtr);
            break;
    }

    /* Check for out of bounds */
    if( (fabs(posPtr->x) > 100.) || (fabs(posPtr->y) > 100.) ){
        going = FALSE;
        return;
    }

    /* Draw the points */

```

```

    DrawPoints(sosW, 1, &posPtr->x, &posPtr->y);
}

Boolean work(                                /* Do this procedure on every pass through
                                             through Xt's main loop */
    XtPointer    clientData)                /* with this data */
{
    static Boolean workCalled=FALSE;

    if(!workCalled) {
#ifdef DEBUG
        fprintf(stderr, "Work called first time.\n");
#endif
        workCalled = TRUE;
    }

    /* Advance the position and plot it */
    if(going) AdvAndPlot(&pos);

    /* Keep calling me back */
    return(FALSE);
}

main(                                /* Entry point for program */
    int  argc,                        /* number of arguments */
    char** argv)                     /* arguments */
{
    XtAppContext appContext=NULL;
    Widget topLevel;
    XtWorkProcId workID;
    char q = 'q', inChar='\0';

    /* Start by initializing the constants */
    SetConstants();

    /* Initialize the map */
    InitMap();

    /* The main widget for the program */
    topLevel = XtVaAppInitialize(
        &appContext,                /* Application context */
        "XMaps",                    /* Application class */
        (XrmOptionDescList)NULL,    /* Options */
        0,                          /* number of options */
        &argc,                      /* Number of command line options */
        argv,                      /* command line args */
        (char**)NULL,               /* Put other resources here */
        (XtPointer)NULL);          /* Terminate variable args list */

    /* Realize the top widget
       XtRealizeWidget(topLevel); */

#ifdef DEBUG
    fprintf(stderr, "This is a debugging run\n");
    XSynchronize(XtDisplay(topLevel), True);
    fprintf(stderr, "topLevel initialized\n");
#endif

    /* Create the main window for the surface of section. */
#ifdef DEBUG
    /* fprintf(stderr, "Sending %o to receive clicks\n", ReceiveClick); */
#endif
    sosW = CreatePlot(

```

```

        "sosw", /*The name of the plot*/
        topLevel, /* The parent of this window */
        ReceiveClick); /*Routine to receive click*/
#ifdef DEBUG
    fprintf(stderr, "Surface of section initialized and realized\n");
#endif

/* Register the error procedure
   XtAppSetErrorHandler(appContext, MyError); */

/* Register the work procedure*/
workID = XtAppAddWorkProc(appContext, work, (XtPointer) NULL);

/* Realize the top widget */
XtRealizeWidget(topLevel);

/* Loop for events.*/
XtAppMainLoop(appContext);
}

```

```

/* xmaps.h: the structures for xmaps */

#ifndef _XMAPS_H
#define _XMAPS_H

#include <stdlib.h>
#include "PlotWindow.h"
#include "Plot.h"

typedef struct _position {
    double      x;
    double      y;
} POSITION;

extern void SetConstants();

extern void ReceiveClick(      /* This routine receives the click*/
    Widget,      /* from this widget */
    char*,      /* of this name */
    ClickData); /* at this specified point */

extern void AdvAndPlot(
    POSITION
);

extern XtWorkProc work;

#endif

```

```

/*
 * PlotWindow: Functions for setting up a plot and for
 * sending plotting calls to the appropriate widgets
 */
#include <stdio.h>
#include <stdlib.h>
/*
 * Include files required for all Toolkit programs
 */
#include "X11/Intrinsic.h"      /* Intrinsics Definitions */
#include <X11/StringDefs.h>      /* Standard Name-String definitions */
#include <X11/Core.h>            /* The core translations*/

/*
 * Public include file for widgets we actually use in this file.
 */
#include <X11/Xaw/Viewport.h>    /* Athena Viewport Widget */
#include <X11/Xaw/Form.h>
#include <X11/Xaw/Command.h>
#include <X11/Xaw/MenuButton.h>
#include <X11/Xaw/SimpleMenu.h>
#include <X11/Xaw/Text.h>
#include <X11/Xaw/AsciiText.h>
#include "Plot.h"

/* The structures needed here */
#include "PlotWindow.h"

void ErasePlot( /* Routine to erase a plot */
    Widget wdg, /* called by this widget */
    XPlot xplPtr, /* to erase this widget */
    XEvent *evt /* Irrelevant event data */
)
{
#ifdef DEBUG
    fprintf(stderr, "ErasePlot called\n");
#endif
    PlErasePlot(xplPtr->plotWdg);
}

void DrawPoints( /* Translate points and send */
    XPlot xplPtr, /* to this plot widget */
    int npts, /* this many pairs */
    double *xarray, /* of x's */
    double *yarray) /* and y's */
{
    PlDrawPoints(xplPtr->plotWdg, npts, xarray, yarray);
}

XPlot CreatePlot( /* Creates a generic plot window */
    char* pnam, /* The name of this plot */
    Widget parWdg, /* The parent widget */
    void (*rcvr)() /* Routine to receive click */
)
{
    XPlot xplPtr; /* The structure for an x-plot */

#ifdef DEBUG
    fprintf(stderr, "CreatePlot entered\n");
#endif

    /* Create a structure for the Xplot */
    xplPtr = (XPlot) calloc(1, sizeof(XPlotRec));

    xplPtr->plotFormWdg = XtCreateManagedWidget(pnam, formWidgetClass,
        parWdg, NULL, 0);
}

```

```

#ifdef DEBUG
    fprintf(stderr, "CreatePlot: plotForm widget created\n");
#endif

/* Create a command widget inside this form */
xplPtr->cmdWdg = XtVaCreateManagedWidget("cmdWdg", commandWidgetClass,
    xplPtr->plotFormWdg, /* This is a child of plotFormWdg */
    XtNlabel, "Erase", /* of this name */
    XtNfromHoriz, NULL, /* Spaced to left side */
    XtNhorizDistance, 5, /* By 10 pixels */
    XtNleft, XtChainLeft, /* Keep this distance constant */
    XtNright, XtChainLeft, /* Keep this distance constant */
    XtNfromVert, NULL, /* Spaced from top */
    XtNvertDistance, 5, /* By 10 pixels */
    XtNtop, XtChainTop, /* Keep this distance constant */
    XtNbottom, XtChainTop, /* Keep this distance constant */
    NULL);

#ifdef DEBUG
    fprintf(stderr, "CreatePlot: cmdWdg created\n");
#endif

/* Create a viewport inside this form */
xplPtr->vportWdg = XtVaCreateManagedWidget("vportWdg", viewportWidgetClass,
    xplPtr->plotFormWdg, /* This is a child of plotFormWdg */
    XtNfromHoriz, NULL, /* Spaced to left side */
    XtNhorizDistance, 5, /* By * pixels */
    XtNleft, XtChainLeft, /* Keep this distance constant */
    XtNright, XtChainRight, /* Keep this distance constant */
    XtNfromVert, xplPtr->cmdWdg, /* Spaced from command widget */
    XtNvertDistance, 5, /* By * pixels */
    XtNtop, XtChainTop, /* Keep this distance constant */
    XtNbottom, XtChainBottom, /* Keep this distance constant */
    XtNallowHoriz, True, /* Allow horizontal scrollbars */
    XtNallowVert, True, /* Allow vertical scrollbars */
    XtNuseRight, True, /* Put vertical scroll bar on the right */
    XtNuseBottom, True, /* Put horizontal scroll bar on the right */
    NULL);

#ifdef DEBUG
    fprintf(stderr, "CreatePlot: vportWdg created\n");
#endif

/* Create a plot inside this viewport */
xplPtr->plotWdg = (PlotWidget)XtVaCreateManagedWidget("plotWdg",
    plotWidgetClass, xplPtr->vportWdg,
    NULL);

#ifdef DEBUG
    fprintf(stderr, "Sending %o to receive clicks\n", rcvr);
#endif

/* Put in the handler for the clicks */
XtAddCallback( /* Add a callback */
    (Widget)(xplPtr->plotWdg), /* to this widget */
    PlNclickHandler, /* to handle clicks */
    (XtCallbackProc)rcvr, /* by calling this function */
    (XtPointer)NULL); /* passing this data. */

#ifdef DEBUG
    fprintf(stderr, "CreatePlot: finished initialization of %s\n", pnam);
#endif

/* Now add a callback to the Erase button to send to the PlotWidget */
XtAddCallback( /* Add a callback */
    (Widget)(xplPtr->cmdWdg), /* to this widget */
    XtNcallback, /* to handle clicks */
    (XtCallbackProc)ErasePlot, /* by calling this function */
    (XtPointer)xplPtr); /* passing this data. */

/* Realize these widgets

```

```
    XtRealizeWidget(xplPtr->plotFormWdg); */
#ifdef DEBUG
    /* fprintf(stderr, "CreatePlot: plotWdg named %s realized\n", pnam); */
#endif

/* Return the value of the created widget */
return(xplPtr);
}
```

```

/*
 * PlotWindow.h: contains the data structures for use of the routines
 * in PlotWindow
 */

#ifndef _PLOT_WINDOW_H
#define _PLOT_WINDOW_H
#define FUNCPROTO

/* X11 headers */
#include "X11/Intrinsic.h"
#include "Plot.h"

/* The data structure of a plot */
typedef struct _XPlot {
    Widget      plotFormWdg;
    Widget      cmdWdg;
    Widget      vportWdg;
    PlotWidget  plotWdg;
} XPlotRec, *XPlot;

extern XPlot CreatePlot(
    char* pnam,
    Widget vportWdg,
    void (*rcvr)());

extern void DrawPoints( /* Translate points and send */
    XPlot      xplPtr, /* to this plot widget */
    int        npts,   /* this many pairs */
    double     *xarray, /* of x's */
    double     *yarray); /* and y's */

#endif /* _PLOT_WINDOW_H */

```



```

/* *****
 * Plot.c: The Plot Widget
 * by John R. Cary
 */

/* Basic Unix*/
#include <string.h>
#include <stdio.h>

/* X11 core */
#include "X11/IntrinsicP.h"
#include <X11/StringDefs.h>
#include <X11/Core.h>
#include <X11/cursorfont.h>

/* Private and public widget info */
#include "PlotP.h"

#define superclass (&coreClassRec)

#define MIN(a,b)      (((a) < (b)) ? (a) : (b))
#define MAX(a,b)      (((a) > (b)) ? (a) : (b))

/* The default values pointers*/
char defTitle[] = "";
char defXLabel[] = "x";
char defYLabel[] = "y";
char fontDef[] = "-*-new century schoolbook-bold-r-normal--*-120-100-100-*-*-*-*";
float mnDef[] = {-0.};
float mxDef[] = {1.};
float lbDef[] = {0.2};
float rbDef[] = {0.1};
float tbDef[] = {0.1};
float bbDef[] = {0.2};
#define DEFDEPTH 3
char defColor1[] = "red";
char defColor2[] = "green";
char defColor3[] = "blue";
char defColor4[] = "dark turquoise";
char defColor5[] = "orange";
char defColor6[] = "magenta";

/* The string stuff*/
#define NCHLIM 20

/* The resource list for this widget */
#define offset(field) XtOffsetOf(PlotRec, field)
static XtResource resources[] = {
    /* The strings for the plot */
    {
        PlNtitle,
        PlCTitle,
        XtRString,
        sizeof(XtPointer),
        offset(plot.title),
        XtRImmediate,
        (XtPointer)defTitle
    },
    {
        PlNxLabel,
        PlCXLabel,
        XtRString,
        sizeof(XtPointer),
        offset(plot.xLabel),
        XtRImmediate,

```

```

(XtPointer)defXLabel
{
    PlNyLabel,
    PlCYLabel,
    XtRString,
    sizeof(XtPointer),
    offset(plot.yLabel),
    XtRImmediate,
    (XtPointer)defYLabel
},

/* Resources for the plot limits*/
{
    PlNfont,
    PlCFont,
    XtRString,
    sizeof(XtPointer),
    offset(plot.font),
    XtRImmediate,
    (XtPointer)fontDef
},
{
    PlNprec,
    PlCPrec,
    XtRInt,
    sizeof(int),
    offset(plot.prec),
    XtRImmediate,
    (XtPointer)3
},
{
    PlNxmmn,
    PlCXmn,
    XtRFloat,
    sizeof(float),
    offset(plot.xmn),
    XtRFloat,
    (XtPointer)mnDef
},
{
    PlNxmmx,
    PlCXmx,
    XtRFloat,
    sizeof(float),
    offset(plot.xmx),
    XtRFloat,
    (XtPointer)mxDef
},
{
    PlNymn,
    PlCYmn,
    XtRFloat,
    sizeof(float),
    offset(plot.ymn),
    XtRFloat,
    (XtPointer)mnDef
},
{
    PlNymx,
    PlCYmx,
    XtRFloat,
    sizeof(float),
    offset(plot.ymx),
    XtRFloat,
    (XtPointer)mxDef
}

```

```

    },

    /* Relative locations of the plot boundaries*/
    {
        PlNlb,
        PlCLb,
        XtRFloat,
        sizeof(float),
        offset(plot.lb),
        XtRFloat,
        (XtPointer) lbDef
    },
    {
        PlNrb,
        PlCRb,
        XtRFloat,
        sizeof(float),
        offset(plot.rb),
        XtRFloat,
        (XtPointer) rbDef
    },
    {
        PlNtb,
        PlCTb,
        XtRFloat,
        sizeof(float),
        offset(plot.tb),
        XtRFloat,
        (XtPointer) tbDef
    },
    {
        PlNbb,
        PlCBb,
        XtRFloat,
        sizeof(float),
        offset(plot.bb),
        XtRFloat,
        (XtPointer) bbDef
    },

    /* The depth of the color map
    {
        PlNdepth,
        PlCDepth,
        XtRInt,
        sizeof(int),
        offset(plot.prec),
        XtRImmediate,
        (XtPointer) 3
    },*/

    /* The default colors*/
    {
        PlNcolor1,
        PlCColor1,
        XtRString,
        sizeof(XtPointer),
        offset(plot.color[1]),
        XtRImmediate,
        (XtPointer) defColor1
    },
    {
        PlNcolor2,
        PlCColor2,
        XtRString,
        sizeof(XtPointer),

```

```

    offset(plot.color[2]),
    XtRImmediate,
    (XtPointer)defColor2
},
{
    PlNcolor3,
    PlCColor3,
    XtRString,
    sizeof(XtPointer),
    offset(plot.color[3]),
    XtRImmediate,
    (XtPointer)defColor3
},
{
    PlNcolor4,
    PlCColor4,
    XtRString,
    sizeof(XtPointer),
    offset(plot.color[4]),
    XtRImmediate,
    (XtPointer)defColor4
},
{
    PlNcolor5,
    PlCColor5,
    XtRString,
    sizeof(XtPointer),
    offset(plot.color[5]),
    XtRImmediate,
    (XtPointer)defColor5
},
{
    PlNcolor6,
    PlCColor6,
    XtRString,
    sizeof(XtPointer),
    offset(plot.color[6]),
    XtRImmediate,
    (XtPointer)defColor6
},
},

/* Callbacks for clicks in plot area */
{
    PlNclickHandler,
    PlCclickHandler,
    XtRcallback,
    sizeof(XtPointer),
    offset(plot.cHandler),
    XtRcallback,
    NULL
},

};

/* The basic window functions */
static void Initialize (
    PlotWidget request, /* The requested data*/
    PlotWidget newWdg, /* The new data */
    ArgList args, /* The arguments */
    Cardinal *numArgs); /* The number of arguments to initialize*/

static void Realize(/* Realize the widget. I call the superclass,
                    then add the drawing contexts and pixel map.*/
    PlotWidget wdg,
    XtValueMask valueMask,
    XSetWindowAttributes *attributes);

```

```

static Boolean SetValues ( /* Set values of widget */
    PlotWidget current, /* Current values */
    PlotWidget request, /* Requested values */
    PlotWidget newWdg, /* New values */
    ArgList args, /* From call */
    Cardinal numArgs);

static void Redisplay( /* Draw the window */
    PlotWidget wWdg, /* of this widget */
    XExposeEvent *event); /* for this event */

/* static XtActionProc RecordClick; */
static void RecordClick( /*Record the button click and send it
    to the appropriate function in user
    coordinates */

    PlotWidget,
    XButtonPressedEvent*,
    String*,
    Cardinal*);

static void Resize(
    PlotWidget wdg
);

static void Destroy(
    PlotWidget wdg
);

/* The public functions are declared in Plot.h*/

/* Translations for mouse events */
static char defaultTranslations[] =
    "<Btn1Down>: RecordClick() \n\
    <Btn2Down>: RecordClick() \n\
    <Btn3Down>: RecordClick()";

/* Routines called for given translated event */
static XtActionsRec actions[] = {
    {"RecordClick", (XtActionProc)RecordClick},
};

PlotClassRec plotClassRec = {
    /* CoreClassPart */
    {
        (WidgetClass) &coreClassRec, /* superclass */
        "Plot", /* class_name */
        sizeof(PlotRec), /* widget_size */
        (XtProc) NULL, /* class_initialize */
        (XtWidgetClassProc) NULL, /* class_part_initialize */
        (XtEnum) FALSE, /* class_inited */
        (XtInitProc) Initialize, /* initialize */
        (XtArgsProc) NULL, /* initialize_hook */
        (XtRealizeProc) Realize, /* realize */
        actions, /* actions */
        XtNumber(actions), /* num_actions */
        resources, /* resources */
        XtNumber(resources), /* num_resources */
        NULLQUARK, /* xrm_class */
        TRUE, /* compress_motion */
        (XtEnum) TRUE, /* compress_exposure */
        TRUE, /* compress_enterleave */
        TRUE, /* visible_interest */
        Destroy, /* destroy */
        Resize, /* resize */
        Redisplay, /* expose */
    }
}

```

```

SetValues,                /* set_values          */
(XtArgsFunc) NULL,        /* set_values_hook     */
XtInheritSetValuesAlmost, /* set_values_almost   */
(XtArgsProc) NULL,        /* get_values_hook      */
(XtAcceptFocusProc) NULL, /* accept_focus         */
XtVersion,                /* version              */
(XtPointer) NULL,         /* callback private     */
defaultTranslations,      /* tm_table             */
(XtGeometryHandler) NULL, /* query_geometry        */
(XtStringProc) NULL,      /* display_accelerator   */
(XtPointer) NULL,         /* extension             */
},
/* Plot class fields */
{
0,                        /* ignore              */
}
};

WidgetClass plotWidgetClass = (WidgetClass) &plotClassRec;

static void Initialize (
    PlotWidget request, /* The requested data*/
    PlotWidget newWdg,  /* The new data */
    ArgList args,       /* The arguments */
    Cardinal *numArgs)  /* The number of arguments to initialize*/
{
    int i, screenNum=0, defDepth;

#ifdef DEBUG
    fprintf(stderr, "Initializing the plot widget\n");
#endif

    /* Get the screen number*/
    screenNum = DefaultScreen(XtDisplay(newWdg));

    /* Get the default visual and depth*/
    defDepth = DefaultDepth(XtDisplay(newWdg), screenNum);

    /* Diagnostic*/
#ifdef DEBUG
    fprintf(stderr, "Initialization of Plot is on screen %d\n", screenNum);
    fprintf(stderr, "Default depth is %d\n", defDepth);
    fprintf(stderr, "newWdg->plot.depth = %d\n", newWdg->plot.depth);
#endif

    /* Set the locations of the boundary pixels */
    /* Check for size */
    if((newWdg->plot.lb < 0) || (newWdg->plot.lb > 1.)) {
        fprintf(stderr, "lb=%e. Out of range. Set to default\n", newWdg->plot.lb);
        newWdg->plot.lb = lbDef[0];
    }
    if((newWdg->plot.rb < 0) || (newWdg->plot.rb > 1.)) {
        fprintf(stderr, "rb=%e. Out of range. Set to default\n", newWdg->plot.rb);
        newWdg->plot.rb = rbDef[0];
    }
    if((newWdg->plot.tb < 0) || (newWdg->plot.tb > 1.)) {
        fprintf(stderr, "tb=%e. Out of range. Set to default\n", newWdg->plot.tb);
        newWdg->plot.tb = tbDef[0];
    }
    if((newWdg->plot.bb < 0) || (newWdg->plot.bb > 1.)) {
        fprintf(stderr, "bb=%e. Out of range. Set to default\n", newWdg->plot.bb);
        newWdg->plot.bb = bbDef[0];
    }
    if( (newWdg->plot.lb + newWdg->plot.rb) > 1. ) {
        fprintf(stderr, "lb+rb > 1. Set to default values");
        newWdg->plot.lb = lbDef[0]; newWdg->plot.rb = rbDef[0];
    }
}

```

```

    }
    if( (newWdg->plot.tb + newWdg->plot.bb) > 1.) {
        fprintf(stderr,"tb+bb > 1. Set to default values");
        newWdg->plot.tb = lbDef[0]; newWdg->plot.bb = rbDef[0];
    }
}

/* Now set the pixel locations */
newWdg->plot.lbp = (int)(newWdg->plot.lb*newWdg->core.width);
newWdg->plot.rbp = (int)((1. - newWdg->plot.rb)*newWdg->core.width);
newWdg->plot.storeWidth = newWdg->plot.rbp - newWdg->plot.lbp;
newWdg->plot.tbp = (int)(newWdg->plot.tb*newWdg->core.height);
newWdg->plot.bbp = (int)((1. - newWdg->plot.bb)*newWdg->core.height);
newWdg->plot.storeHeight = newWdg->plot.bbp - newWdg->plot.tbp;

/* Set the max precision */
newWdg->plot.prec = MIN(newWdg->plot.prec,NCHLIM - 7);
newWdg->plot.prec = MAX(newWdg->plot.prec,0);

/* Start with the color being black */
newWdg->plot.curColor = 0;

/* Diagnostic: print out the initial resource values */
#ifdef DEBUG
    fprintf(stderr,"xmn = %e\n",newWdg->plot.xmn);
    fprintf(stderr,"lb = %e, width = %d, lbp = %d\n",newWdg->plot.lb,
        newWdg->core.width,newWdg->plot.lbp);
    fprintf(stderr,"xmx = %e\n",newWdg->plot.xmx);
    fprintf(stderr,"rb = %e, rbp = %d\n",newWdg->plot.rb, newWdg->plot.rbp);
    fprintf(stderr,"ymn = %e\n",newWdg->plot.ymn);
    fprintf(stderr,"bb = %e, height = %d, bbp = %d\n",newWdg->plot.bb,
        newWdg->core.height,newWdg->plot.bbp);
    fprintf(stderr,"ymx = %e\n",newWdg->plot.ymx);
    fprintf(stderr,"tb = %e, tbp = %d\n",newWdg->plot.tb, newWdg->plot.tbp);
#endif
}

static int GetColorGCs(/* Get the requested colors*/
    PlotWidget wdg
)
{
    int i, screenNum, aColor=0;
    XtValueMask vMask; XGCValues vals;
    /* For colors*/
    Colormap defColormap;
    XColor thisColor;

    /* Get the screen number*/
    screenNum = DefaultScreen(XtDisplay(wdg));
    defColormap = DefaultColormap(XtDisplay(wdg),screenNum);

    /* Run through the named colors */
    wdg->plot.numColors = 1;
    for(i=1;i<MAXCOLORS;i++){

        /* See if color is in database*/
        #ifdef DEBUG
            fprintf(stderr,"Parsing color %s\n",wdg->plot.color[i]);
        #endif
        if( XParseColor(XtDisplay(wdg),defColormap,wdg->plot.color[i],
            &thisColor) ){

            /* Allocate the color*/
            #ifdef DEBUG
                fprintf(stderr,"Color %s in database.\n",wdg->plot.color[i]);
                fprintf(stderr,"Allocating color %s\n",wdg->plot.color[i]);
            #endif
        }
    }
}

```

```

#endif
    if( XAllocColor(XtDisplay(wdg),defColormap,&thisColor) ){
#ifdef DEBUG
        fprintf(stderr,"Color %s allocated.\n",wdg->plot.color[i]);
#endif
        vals.foreground = thisColor.pixel;
        vals.background = WhitePixel(XtDisplay(wdg),screenNum);
        vals.font = wdg->plot.fontID;
        vMask = GCForeground | GCBackground | GCFont;
#ifdef DEBUG
        fprintf(stderr,"Getting graphics context for color number %d\n",
            wdg->plot.numColors);
#endif
        wdg->plot.colorGC[wdg->plot.numColors] =
            XtGetGC((Widget)wdg, vMask, &vals);
        wdg->plot.numColors++;
        if( aColor == 0 ) aColor = i;
    }
    /* Errors for above two steps */
    else(
        fprintf(stderr,"Unable to allocate color %s.\n");
    )
    else {
        fprintf(stderr,"Color %s not in database.\n");
    }
}
#ifdef DEBUG
    fprintf(stderr,"%d colors allocated.\n",wdg->plot.numColors);
#endif
return(aColor);
}

static void Realize(/* Realize the widget. I call the superclass,
                    then add the drawing contexts and pixel map.*/
    PlotWidget      wdg,
    XtValueMask      valueMask,
    XSetWindowAttributes *attributes)
{
    int i, screenNum, defDepth;
    XtValueMask vMask; XGCValues vals;
    Cursor myCursor;
    /* For colors*/
    Visual *defVisual;
    XVisualInfo visInfo;

#ifdef DEBUG
    fprintf(stderr,"Entering Realize\n");
#endif

    /* Use realize method from superclass */
    (*superclass->core_class.realize) ((Widget)wdg,valueMask,attributes);
#ifdef DEBUG
    fprintf(stderr,"Superclass realization done\n");
#endif

    /* Diagnostic*/
#ifdef DEBUG
    fprintf(stderr,"wdg->plot.depth = %d\n",wdg->plot.depth);
#endif

    /* Get the screen number*/
    screenNum = DefaultScreen(XtDisplay(wdg));
    defDepth = DefaultDepth(XtDisplay(wdg),screenNum);

    /* Set the cursor to a cross */

```



```

/*myCursor = XCreateFontCursor(XtDisplay(wdg),XC_crosshair);*/
myCursor = XCreateFontCursor(XtDisplay(wdg),XC_tcross);
XDefineCursor(XtDisplay(wdg),XtWindow(wdg),myCursor);

/* Get the font for the drawing context*/
wdg->plot.fontID = XLoadFont(XtDisplay(wdg),wdg->plot.font);
#ifdef DEBUG
    fprintf(stderr,"Font loaded\n");
#endif
if( wdg->plot.fontID == 0 )
    fprintf(stderr,"Unable to load font %s\n",wdg->plot.font);

/* We use XtGetGC for the eraseGC (white) because it is fixed */
vals.foreground = WhitePixel(XtDisplay(wdg),screenNum);
vals.background = BlackPixel(XtDisplay(wdg),screenNum);
vMask = GCForeground | GCBackground;
wdg->plot.eraseGC = XtGetGC((Widget)wdg, vMask, &vals);
#ifdef DEBUG
    fprintf(stderr,"Erase context made\n");
#endif

/* The black graphics context is the first color context */
vals.foreground = BlackPixel(XtDisplay(wdg),screenNum);
vals.background = WhitePixel(XtDisplay(wdg),screenNum);
vals.font = wdg->plot.fontID;
vMask = GCForeground | GCBackground | GCFont;
wdg->plot.colorGC[0] = XtGetGC((Widget)wdg, vMask, &vals);

#ifdef DEBUG
    fprintf(stderr,"Black context made\n");
#endif

/* Get the other colors */
if(defDepth > 1) {
    i = 5;
    while( !XMatchVisualInfo(XtDisplay(wdg),screenNum,defDepth,
        /*visual class*/ i--, &visInfo) );
#ifdef DEBUG
    fprintf(stderr,"Visual of number %d found\n",++i);
#endif
    if( i < StaticColor ) {
        fprintf(stderr,"No color visual found. Will use black and white.\n");
        goto EndColor;
    }
#ifdef DEBUG
    fprintf(stderr,"Allocating colors\n");
#endif
    wdg->plot.curColor = GetColorGCs(wdg) - 1;
}
else wdg->plot.numColors = 1;

EndColor:

/* Create and erase the pixel map for the drawing*/
#ifdef DEBUG
    fprintf(stderr,"Creating %d x %d pixmap",wdg->plot.storeWidth,
        wdg->plot.storeHeight);
    fprintf(stderr," of depth %d\n",defDepth);
#endif
wdg->plot.pixmap = XCreatePixmap(
    XtDisplay(wdg), /*the display for this pixmap */
    XtWindow(wdg), /*The window of this pixmap */
    wdg->plot.storeWidth, /*the width of the pixmap */
    wdg->plot.storeHeight, /*the height of the pixmap */
    defDepth); /* Depth of pixmap */
#ifdef DEBUG

```

```

    fprintf(stderr, "Pixmap created\n");
#endif

/* Erase the pixmap */
XFillRectangle (
    XtDisplay(wdg),
    wdg->plot.pixmap,
    wdg->plot.eraseGC,
    0, 0, /* From upper left corner */
    wdg->plot.storeWidth, /* the width of the pixmap */
    wdg->plot.storeHeight); /* the height of the pixmap */

}

static void Redisplay( /* Draw the window */
    PlotWidget wdg, /* of this widget */
    XExposeEvent *event) /* for this event */
{
    /* The storage for the strings */
    XFontStruct *fontInfo;
    char limChar[NCHLIM+1];
    int nchars, direction, ascent, descent;
    int h, v, len, hsep, vsep, dvFont, vCentFont;
    XCharStruct textStruct;

#ifdef DEBUG
    fprintf(stderr, "Entering Redisplay\n");
#endif

    /* Make sure the widget exists
       .If( !XtIsRealized(wdg) ) return; */

    /* Draw a rectangle around the plotting region */
    /* Note that one must diminish the location by 1 and increase
       the width and height by one to prevent overwriting */
#ifdef DEBUG
    fprintf(stderr, "About to draw the rectangle\n");
#endif
    XDrawRectangle(XtDisplay(wdg), XtWindow(wdg),
        wdg->plot.colorGC[0],
        wdg->plot.lbp - 1, wdg->plot.tbp - 1,
        wdg->plot.storeWidth + 1, wdg->plot.storeHeight + 1);

    /* Draw the limits of the plot */
    wdg->plot.prec = NCHLIM - 6; /* The displayed precision */
    fontInfo = XQueryFont(XtDisplay(wdg), wdg->plot.fontID);
    dvFont = fontInfo->ascent + fontInfo->descent;
    vCentFont = (fontInfo->ascent - fontInfo->descent)/2;
    vsep = fontInfo->ascent/2 + fontInfo->descent;
    hsep = fontInfo->ascent/2 + fontInfo->descent;

    /* Each limit */
    sprintf(limChar, "%.3g", wdg->plot.xmn);
    len = strlen(limChar);
    h = wdg->plot.lbp - XTextWidth(fontInfo, limChar, len)/2;
    v = wdg->plot.bbp + vsep + dvFont;
    XDrawString(XtDisplay(wdg), XtWindow(wdg), wdg->plot.colorGC[0],
        h, v, limChar, len);
    sprintf(limChar, "%.3g", wdg->plot.xmx);
    len = strlen(limChar);
    h = wdg->plot.rbp - XTextWidth(fontInfo, limChar, len)/2;
    XDrawString(XtDisplay(wdg), XtWindow(wdg), wdg->plot.colorGC[0],
        h, v, limChar, len);
    sprintf(limChar, "%.3g", wdg->plot.ymn);
    len = strlen(limChar);
    h = wdg->plot.lbp - XTextWidth(fontInfo, limChar, len) - hsep;
    v = wdg->plot.bbp + (fontInfo->ascent - fontInfo->descent)/2;

```

```

XDrawString(XtDisplay(wdg), XtWindow(wdg), wdg->plot.colorGC[0],
    h, v, limChar, len);
sprintf(limChar, "%.3g", wdg->plot.ymx);
len = strlen(limChar);
h = wdg->plot.lbp - XTextWidth(fontInfo, limChar, len) - hsep;
v = wdg->plot.tbp + (fontInfo->ascent - fontInfo->descent)/2;
XDrawString(XtDisplay(wdg), XtWindow(wdg), wdg->plot.colorGC[0],
    h, v, limChar, len);
#ifdef DEBUG
    fprintf(stderr, "Limits drawn\n");
#endif

/* Draw the title and the axis labels */
#ifdef DEBUG
    fprintf(stderr, "Title is %s\n", wdg->plot.title);
#endif
len = strlen(wdg->plot.title);
h = (wdg->plot.lbp + wdg->plot.rbp -
    XTextWidth(fontInfo, wdg->plot.title, len))/2;
v = wdg->plot.tbp - vsep - fontInfo->descent - dvFont;
XDrawString(XtDisplay(wdg), XtWindow(wdg), wdg->plot.colorGC[0],
    h, v, wdg->plot.title, len);
len = strlen(wdg->plot.xLabel);
h = (wdg->plot.lbp + wdg->plot.rbp -
    XTextWidth(fontInfo, wdg->plot.xLabel, len))/2;
v = wdg->plot.bbp + 2*(dvFont + vsep);
XDrawString(XtDisplay(wdg), XtWindow(wdg), wdg->plot.colorGC[0],
    h, v, wdg->plot.xLabel, len);
len = strlen(wdg->plot.yLabel);
h = wdg->plot.lbp - XTextWidth(fontInfo, wdg->plot.yLabel, len)
    - hsep;
v = (5*wdg->plot.bbp + 8*wdg->plot.tbp)/13 - vCentFont;
XDrawString(XtDisplay(wdg), XtWindow(wdg), wdg->plot.colorGC[0],
    h, v, wdg->plot.yLabel, len);

/* Just erase the window, then copy the pixmap */
#ifdef DEBUG
    fprintf(stderr, "Recopying pixmap\n");
#endif
XCopyArea(XtDisplay(wdg), wdg->plot.pixmap, XtWindow(wdg),
    wdg->plot.colorGC[0],
    0, 0, /* Upper left of source rectangle */
    wdg->plot.storeWidth,
    wdg->plot.storeHeight,
    wdg->plot.lbp, wdg->plot.tbp); /*Upper left of destination rectangle */

}

static Boolean SetValues (      /* Set values of widget */
    PlotWidget    current,      /* Current values */
    PlotWidget    request,      /* Requested values */
    PlotWidget    newWdg,       /* New values */
    ArgList       args,         /* From call */
    Cardinal       numArgs)
{
    fprintf(stderr, "Entering SetValues\n");
}

static void Resize ( /* Draw the new region of the window */
    PlotWidget    wdg
)
{
}

```

```

static void Destroy ( /* Release the memory for this widget*/
    PlotWidget wdg
)
{
    if( wdg->plot.pixmap )
        XFreePixmap(XtDisplay(wdg), wdg->plot.pixmap);
    if( wdg->plot.colorGC[wdg->plot.curColor] )
        XtReleaseGC ((Widget)wdg, wdg->plot.colorGC[wdg->plot.curColor]);
    if( wdg->plot.eraseGC )
        XtReleaseGC ((Widget)wdg, wdg->plot.eraseGC);
}

static void RecordClick( /* Record the button click and send it
                        to the appropriate function in user
                        coordinates */
    PlotWidget          wdg,
    XButtonPressedEvent *event,
    String               *params,
    Cardinal             *numParams)
{
#ifdef DEBUG
    fprintf(stderr, "Clicked in RecordClick\n");
#endif
    /* Bump the color if there are colors */
    if( wdg->plot.numColors > 1 ){
        wdg->plot.curColor++;
        wdg->plot.curColor %= wdg->plot.numColors ;
        wdg->plot.curColor = MAX(1, wdg->plot.curColor);
    }

    /* If the event was in the plot, send location */
    if( (event->x >= wdg->plot.lbp) && (event->x <= wdg->plot.rbp)
        && (event->y >= wdg->plot.tbp) && (event->y <= wdg->plot.bbp) ){
        wdg->plot.cDataRec.x = (event->x - wdg->plot.lbp) *
            (wdg->plot.xmx - wdg->plot.xmn)
            / wdg->plot.storeWidth + wdg->plot.xmn;
        wdg->plot.cDataRec.y = (wdg->plot.bbp - event->y) *
            (wdg->plot.ymx - wdg->plot.ymn)
            / wdg->plot.storeHeight + wdg->plot.ymn;
#ifdef DEBUG
        fprintf(stderr, "Clicked at x = %g and y = %g\n",
            wdg->plot.cDataRec.x, wdg->plot.cDataRec.y);
#endif
        XtCallCallbacks((Widget)wdg, PlNclickHandler,
            (XtPointer) (&wdg->plot.cDataRec));
    }
    else( /* If it is in the limits, change and send*/
    )

#ifdef DEBUG
    fprintf(stderr, "Color in RecordClick now number %d\n",
        wdg->plot.curColor);
#endif
}

/* From here follow the public functions */

void PlDrawPoints( /* Draw points into the plot*/
    PlotWidget wdg,
    int npts, /* The number of points */
    double *xarray, /* Array of x values */
    double *yarray /* Array of y values */
)

```

```

{
int i,h,v,h0,v0;

/* Translate each point and plot it*/
for(i=0;i<npts;i++){
    h = (int)((xarray[i] - wdg->plot.xmn)*(wdg->plot.rbp - wdg->plot.lbp)/
        (wdg->plot.xmx - wdg->plot.xmn));
    h0 = h + wdg->plot.lbp;
    v = (int)((wdg->plot.ymx - yarray[i])*(wdg->plot.bbp - wdg->plot.tbp)/
        (wdg->plot.ymx - wdg->plot.ymn));
    v0 = v + wdg->plot.tbp;

/* Do not draw if outside plot region */
if( (h0 < wdg->plot.rbp) && (h0 > wdg->plot.lbp)
    && (v0 < wdg->plot.bbp) && (v0 > wdg->plot.tbp) ) {

/* Draw into the pixmap */
    XDrawPoint(XtDisplay(wdg), /* the display for this pixmap */
        wdg->plot.pixmap, /* The pixmap */
        wdg->plot.colorGC[wdg->plot.curColor], /* Use draw pattern */
        h,v); /* The point */

/* Draw into the plot*/
    XDrawPoint(XtDisplay(wdg), /* the display for this widget*/
        XtWindow(wdg), /* The drawable*/
        wdg->plot.colorGC[wdg->plot.curColor], /*Use draw pattern*/
        h0,v0);
    }
}

void PlErasePlot( /* Erase the plot */
    PlotWidget pWdg /* of this widget */
)
{
/* Erase the pixmap */
    XFillRectangle (
        XtDisplay(pWdg),
        pWdg->plot.pixmap,
        pWdg->plot.eraseGC,
        0,0, /* From upper left corner */
        pWdg->plot.storeWidth, /* the width of the pixmap */
        pWdg->plot.storeHeight); /* the height of the pixmap */

/* Erase the screen */
    XFillRectangle (
        XtDisplay(pWdg),
        XtWindow(pWdg),
        pWdg->plot.eraseGC,
        pWdg->plot.lbp, /* From left */
        pWdg->plot.tbp, /* and upper corner */
        pWdg->plot.storeWidth, /* the width of the pixmap */
        pWdg->plot.storeHeight); /* the height of the pixmap */
}

```

```

/*****
 * Plot.h: Public header file for Plot Widget Class
 * by John R. Cary
 *****/

#ifndef _PLOT_H
#define _PLOT_H
#define FUNCPROTO

/* Define the widgets and structures */
extern WidgetClass plotWidgetClass;
typedef struct _PlotClassRec * PlotWidgetClass;
typedef struct _PlotRec      * PlotWidget;

/* These functions may be called from the outside */
extern void PlDrawPoints(      /* Translate points and send */
    PlotWidget, /* to this plot widget */
    int,        /* this many pairs */
    double*,    /* of x's */
    double*);   /* and y's */

extern void PlErasePlot(      /* Erase the plot */
    PlotWidget /* of this widget */
);

/* This is passed as the call data for a mouseDown event */
typedef struct _ClickData {
    double x;
    double y;
} *ClickData, ClickDataRec;

/* Resources defining the plot limits. These are my defs, so prefixed Pl */
#define PlNtitle      "title"
#define PlCTitle      "Title"
#define PlNxLabel     "xLabel"
#define PlCXLabel     "XLabel"
#define PlNyLabel     "yLabel"
#define PlCYLabel     "YLabel"

#define PlNfont        "font"
#define PlCFont        "Font"
#define PlNprec        "prec"
#define PlCPrec        "Prec"

#define PlNxmn         "xmn"
#define PlCXmn         "Xmn"
#define PlNxmx         "xmx"
#define PlCXmx         "Xmx"
#define PlNymn         "ymn"
#define PlCYmn         "Ymn"
#define PlNymx         "ymx"
#define PlCYmx         "Ymx"

#define PlNlb          "lb"
#define PlCLb          "Lb"
#define PlNrbb         "rb"
#define PlCRb          "Rb"
#define PlNtb          "tb"
#define PlCTb          "Tb"
#define PlNb          "bb"
#define PlCBb          "Bb"

#define PlNdepth       "depth"
#define PlCDepth       "Depth"

#define PlNcolor1      "color1"

```

```
#define PlCColor1      "Color1"
#define PlNcolor2      "color2"
#define PlCColor2      "Color2"
#define PlNcolor3      "color3"
#define PlCColor3      "Color3"
#define PlNcolor4      "color4"
#define PlCColor4      "Color4"
#define PlNcolor5      "color5"
#define PlCColor5      "Color5"
#define PlNcolor6      "color6"
#define PlCColor6      "Color6"

#define PlNclickHandler "clickHandler"
#define PlCclickHandler "ClickHandler"

#endif /* _DRAWRGN_H */
```

```

/*****
 * PlotP.h: Private header file for Drawing Widget Class
 * by John R. Cary, University of Colorado
 *
 * This code uses the X Window System
 *
 */

#ifndef _PLOT_P_H
#define _PLOT_P_H

/* This is based on only the core declarations */
#include <X11/CoreP.h>
#include "Plot.h"

typedef struct _PlotClassPart {
    int ignore;
} PlotClassPart;

typedef struct _PlotClassRec {
    CoreClassPart core_class;
    PlotClassPart plot_class;
} PlotClassRec;

extern PlotClassRec plotClassRec;

#define MAXCOLORS 7
typedef struct _PlotPart {
    /* resources */
    char* title; /* Title */
    char* xLabel; /* x-axis label */
    char* yLabel; /* y-axis label */
    char* font; /* The font for the limits */
    int prec; /* The display precision of the limits */
    float xmin; /* The minimum value of x plotted */
    float xmax; /* The maximum value of x plotted */
    float ymin; /* The minimum value of y plotted */
    float ymax; /* The maximum value of y plotted */
    float lb; /* Rel. location of the left boundary from the left */
    float rb; /* Relative location of the right boundary from right */
    float tb; /* Relative location of top boundary from the top */
    float bb; /* Relative location of bottom boundary from bottom */
    unsigned int depth; /* Depth of screen */
    int numColors; /* The number of colors,
                    including black but not white */
    int curColor; /* Current color index */
    char* color[MAXCOLORS]; /* The requested colors */
    XtCallbackList cHandler; /* Callback list for handling clicks */

    /* private state */
    Dimension storeWidth; /* The width of the stored pixel map */
    Dimension storeHeight; /* The height of the stored pixel map */
    Font fontID; /* The font for this */
    int lbp; /* The pixel of the left plot boundary */
    int rbp; /* The pixel of the right plot boundary */
    int tbp; /* The pixel of the top plot boundary */
    int bbp; /* The pixel of the bottom plot boundary */
    Pixmap pixmap; /* the main pixmap */
    GC eraseGC; /* for erasing the pixmap */
    GC colorGC[MAXCOLORS]; /* For drawing in color */
    ClickDataRec cDataRec; /* The data passed on a mouse click */
} PlotPart;

typedef struct _PlotRec {
    CorePart core;
    PlotPart plot;
}

```


PlotRec;

#endif /* _PLOT_H */

END

DATE
FILMED
3 12 7 1992

I