AUTHOR	Hogben, Leslie
TITLE	Student Produced Advanced Mathematical Software.
INSTITUTION	Iowa State Univ. of Science and Technology, Ames.
SPONS AGENCY	Fund for the Improvement of Postsecondary Education (ED), Washington, DC.
PUB DATE	88
GRANT	G008440404
NOTE	42p.
PUB TYPE	Reports - Descriptive (141)
EDRS PRICE	MF01/PC02 Plus Postage.
DESCRIPTORS	*College Mathematics; *Computer Graphics; Course Descriptions; *Courseware; *Curriculum Development; Higher Education; Mathematics Curriculum; *Mathematics Instruction
IDENTIFIERS	*Differential Equations

ABSTRACT

The intent of this project was to develop a course for mathematics graduate students at Iowa State University. They would design and write computer programs for use by undergraduate mathematics students, and then offer the course and actually produce the software. Phase plane graphics for ordinary differential equations was selected as the topic. Prior to the course, the faculty coordinators designed the modular structure of the program and wrote some input/output routines. The course was held, but was plagued by a shortage of students. This caused delays, as the program was not completed during the course. The software was finally completed through a variety of methods, including adapting existing numerical programs, using graduate students to write parts of the program as master's degree projects, and hiring graduate students to write parts of the program. The computer program, Phase Plane Graphics for Ordinary Differential Equations, is listed in an appendix; it will be commerically distributed. (Author/MNS)

STUDENT PRODUCED ADVANCED MATHEMATICAL SOFTWARE

Leslie Hogben, Project Director

Department of Mathematics
Iowa State University
Ames, IA 50011

May 31, 1987

Grant Organization:

Iowa State University
Ames, IA 50011

Grant No

G008440404

Project Dates.

Starting Date August 1, 1984
Ending Date May 31, 1987 (originally July 31, 1986)
Number of months. 34 (originally 24)

Project Director:

Leslie Hogben
Department of Mathematics
Carver Hall
Iowa State University
Ames, Ia 50011

Fund Program Officer:

Jay Donahue (originally Susan Forman)

Grant Award.

Year 1.	$19,570	
Year 2:	$12,509	(carried over to 3rd year)
Total	$32,079	

Student Produced Advanced Mathematical Software

Summary

The intent of this project was to develop a course for mathematics graduate students to design and write computer programs for use of undergraduate mathematics students, and to offer the course and actually produce software The particular software project selected was a graphics program for use in an undergraduate differential equations course. The course was held, but suffered from a shortage of students. Students were recruited later to complete the software, Phase Plane Graphics for Ordinary Differential Equations, which will be commercially distributed.

Project Director: Leslie Hogben, Department of Mathematics,
Iowa State University, Ames, Ia 50011
Tel 515-294-8168 or 515-294-1752

The following materials developed by this project may be obtained by calling or writing the address above Telephone requests are recommended for computer diskettes.

PHASE PLANE GRAPHICS FOR ORDINARY DIFFERENTIAL EQUATIONS

Input/Output and Graphics Subprograms

Materials Developed for a Graduate Course In Mathematics Software Development

Project Title Student Produced Advanced Mathematical Software
Grantee Organization: Iowa State University, Ames, Ia 50011
Project Director: Leslie Hogben, Department of Mathematics,
Iowa State University, Ames, Ia 50011
Tel. 515-294-8168 or 515-294-1752

Executive Summary

Project Overview

The intent of this project was to develop a course for mathematics graduate students to design and write computer programs for use of undergraduate mathematics students, and to offer the course and actually produce software. The particular software project selected was phase plane graphics for ordinary differential equations, suitable for use in an unde graduate differential equations course. Prior to the course, the faculty coordinators, Leslie Hogben, Richard Tondra, and Roger Alexander, designed the modular structure of the program and wrote some input/output routines. The course was held, but was plagued by a shortage of students. This caused delays in the project, as the program was not completed during the course. The software was finally completed through a variety of methods, including adapting existing numerical programs, using graduate students to write parts of the program as Master's degree projects, and hiring graduate students to write parts of the program The computer program, Phase Plane Graphics for Ordinary Differential Equations, will be commercially distributed

Purpose

There is a serious lack of mathematical software at the advanced undergraduate (postcalculus) level, despite the many useful applications of computers to mathematics. There is also a lack of integration of computer use into graduate mathematics courses.

This project attempted to solve both these problems by developing a graduate course in which the students would write software for use in advanced undergraduate mathematics courses

Background and Origins

Iowa State University is a large (25,000 students) public university offering both undergraduate and graduate degrees The Department of Mathematics has more than 60 permanent faculty members and approximately 90 graduate students Bachelor of Science, Master of Science, and Doctor of Philosophy degrees are offered by the department The Iowa State Computation Center operates mainframe, minicomputer, and microcomputers, and provides consulting services to the university.

Project Description

The fall semester of 1984 was spent designing the graduate course, Math 517x, and making preparations for the software. We researched software design and review techniques and decided on a format for the course. We developed standard input/output subprograms for use in the software. We selected phase plane graphics for ordinary differential equations as the program we would develop in Spring 1985, and devised a plan of modular units for the program.

Given a differential equation of the form

$dx/dt = F(x,y) \qquad dy/dt = G(x,y)$

and an initial point, the solution is a path in the x,y-plane There are many applications of such differential equations. Because of their importance, equations of this type are frequently studied qualitatively in undergraduate differential equations courses, although quantitative solution of such equations is beyond the scope of an undergraduate course.

The program was to contain the following modules: main program (MAIN); to control the region of the x,y-plane that is graphed (GRAPH), to enter differential equations (PARSE); to graph a single trajectory, i.e , solution, (TRAJECTORY); to graph a series of trajectories, i.e. phase plane (PHASE), to graph a direction field of vectors (ARROWS); to graph null clines (NULLCLN), to find and classify critical points (CRITPT). There was also a module to handle menu input/output (MENUS), and for technical reasons two small machine-specific modules pertaining to graphics and menus

During the spring semester of 1985, Hogben and Tondra taught Math 517x, a graduate course covering numerical methods for studying phase planes and developing phase plane software. Serious difficulties were encountered with both the quantity and quality of students. Only four students registered for the course and two subsequently dropped it During Math 517x, two modules were completed, MAIN by James Coyle, who subsequently dropped the course, and PARSE by Hogben. One student was assigned to adapt public domain code in Shampine and Gordan, Computer Solution of Ordinary Differential Equations, but did an inadequate job. Another was assigned the CRITPT module but did not do it

During the summer of 1985, we hired Coyle to design and write the ARROWS module, which he did skillfully and efficiently. At the end of Fall semester 1985, Linda Ten Hoeve was given the CRITPT module as a creative project required for her Master's degree. She completed this project successfully in the summer of 1986, but did not interface it to the main program. During the spring of 1987, Hogben adapted the Ten Hoeve code and Alexander adapted the Shampine/Gordan code. Practical considerations involving the use of the program led to the elimination of the PHASE module Thus the program Phase Plane Graphics for Ordinary Differential Equations was completed in May 1987 (Z-100 version)

Because of the difficulties encountered during Math517x, the students did not write the user manual as they went along, as originally intended Alexander is currently writing a detailed user manual for the program When this is completed, we intend to test the software in an undergraduate differential equations course In the meantime, brief instructions on how to use the program are available

In order to facilitate dissemination of the software, we intend to produce versions for other MS-DOS machines, in particular the IBM-PC. Technical difficulties have caused delays, but the IBM-PC version should be completed during 1987

Project Results

The principal product of this project is the computer program, PHASE PLANE GRAPHICS FOR ORDINARY DIFFERENTIAL EQUATIONS. This program is currently available for the Zenith Z-100 and is being adapted for the IBM-PC and similar MS-DOS machines In order to obtain widespread distribution, we are planning to distribute it commercially. It has been submitted to CONDUIT This program is included in Appendix B

In the process of developing this program, we developed menu format input/output and graphics control procedures callable from MS-DOS Pascal or FORTRAN. These procedures may be useful to other software developers They are included, along with a small demonstration program, in Appendix B

In the process of planning the course, a variety of materials were developed These include program design notes, class plan, and a brief software development bibliography These materials are included in Appendix C.

Copies of the above materials may be obtained from
Leslie Hogben, Department of Mathematics
Iowa State University, Ames, IA 50011
Tel 515-294-8168 or 515-294-1752
Because information about equipment is necessary to provide computer diskettes, such requests are more easily handled by telephone.

Summary and Conclusions

This project successfully completed the development of planned software, Phase Plane Graphics for Ordinary Differential Equations. However, the intended method of development by a class of graduate students was not completely successful. In overcoming the student shortage difficulty, two promising methods for developing software were utilized: Modifying currently available research programs, and hiring graduate students.

Well designed and thouroughly tested programs utilizing sophisticated numerical methods are widely available and are frequently in the public domain. Although such programs frequently employ a high level of numerical sophistication, they range from difficult to impossible for an inexperienced person to use. However, if combined with a user-friendly program to handle input/output and a well written manual, existing numerical programs can provide the basis for valuable software.

The other strategy employed to successfully complete the software involved using graduate students to design and write parts of the program, but not as part of the class. One student was hired to write a module and another wrote a module as a creative degree project Both these approaches provide software development experience, albeit without the team approach and design review experience originally planned

The results of this project suggest possible methods of developing undergraduate software by incorporating existing research subroutines and using graduate students, either by hiring them or by awarding academic credit for individual projects This project also developed a sophisticated and easy to use program for use in an undergraduate differential equations course, Phase Plane Graphics for Ordinary Differential Equations.

Student Produced Advanced Mathematical Software

Final Report

Project Overview

The intent of this project was to develop a course for mathematics graduate students to design and write computer programs for use of undergraduate mathematics students, and to offer the course and actually produce software The particular software project selected was phase plane graphics for ordinary differential equations, suitable for use in an undergraduate differential equations course. Prior to the start of the course, the faculty coordinators, Leslie Hogben, Richard Tondra, and Roger Alexander, designed the modular structure of the program and wrote input/output routines The course was held, but was plagued by a shortage of students This caused delays in the project, as the program was not completed during the course The software was finally completed through a variety of methods, including adapting existing numerical programs, using graduate students to write parts of the program as Master's degree projects, and hiring graduate students to write parts of the program. The computer program, Phase Plane Graphics for Ordinary Differential Equations, will be commercially distributed

Purpose

There is a serious lack of mathematical software at the advanced undergraduate (postcalculus) level, despite the many useful applications of computers to mathematics There is also a lack of integration of computer use into graduate mathematics courses.

A substantial body of software exists for elementary mathematics courses (calculus and precalculus) There are also a large number of sophisticated research level programs available, but they are unsuited for use in undergraduate courses Very little software relevant to advanced undergraduate (postcalculus) courses is currently available This is primarily a consequence of market conditions, rather than of the appropriateness of software to advanced courses While there is perhaps a greater range of interesting applications in advanced mathematics, major textbook publishers currently do not believe the development of advanced software is economically viable. Once the software is developed, there are means available to distribute it.

The use of programs written by others can make many aspects of advanced mathematics more concrete by allowing the user to see a variety of of examples, much as spreadsheet software can enable a businessman to obtain a better understanding of the effects of different decisions However, to thoroughly understand the mathematics one needs to understand the underlying computational procedures (numerical algorithms). This is learned by doing the programming oneself, but is beyond the scope of most undergraduate mathematics courses It is entirely appropriate at the graduate level.

There is a need to incorporate computer programming of numerical algorithms into graduate mathematics courses Few graduate mathematics courses use computers, despite the fact that many graduate students will need to use computers in future employment.

This project attempted to solve both these problems by developing a graduate course in which the students would write software for use in advanced undergraduate mathematics courses

Background and Origins

Iowa State University is a large (25,000 students) public university offering both undergraduate and graduate degrees The Department of Mathematics has more than 60 permanent faculty members and approximately 90 graduate students Bachelor of Science, Master of Science, and Doctor of Philosophy degrees are offered by the department The Iowa State Computation Center provides academic and research computer services to the university. It offers mainframe, minicomputers and microcumputers, and provides consulting services.

The Computation Center was helpful and cooperative throughout the project They provided consulting services on machine specifics and provided us with some public domain graphics software they had written. We had no problems getting access to the university's Zenith Z-100 micro computers

Project Description

This project was carried out by three faculty members in the Department of Mathematics at Iowa State University, Leslie Hogben, Roger Alexander, and Richard Tondra, with the assistance of several graduate students in the Mathematics Department

The fall semester of 1984 was spent designing the graduate course, Math 517x, and making preparations for the software We researched software design and review techniques and decided on a format for the course. We developed standard input/output subprograms for use in the software We selected phase plane graphics for ordinary differential equations as the program we would develop in Spring 1985, and devised a plan of modular units for the program

It was decided that the course would begin with lectures on the underlying mathematics that the programmers (graduate students) would need for the project. Then each student would be assigned a piece of the program to design The whole class would analyse the work in design reviews. A design is written in a mixture of English and block-structured programming language (such as Pascal) Ideally, a design should be readily intelligible to a nonprogrammer who is familiar with the mathematics, and yet should be mechanically translatable into programmed code In practice, a design usually starts in English and and progresses closer to programming language as the review process proceeds The author of the design distributes it to the review group (class) and the review group meets at a later time to discuss the design The reviewers ask questions to clarify the design, catch errors, ask how steps will be translated into programming language, etc Materials prepared for classroom use in Math 517x, including a sample design, are include in Appendix C.

Phase plane graphics for ordinary differential equations was chosen as the software development project Given a differential equation of the form

$$dx/dt = F(x,y) \qquad dy/dt = G(x,y)$$

and an initial point, the solution is a path in the x,y-plane. There are many applications of such differential equations. Such equations are studied in various ways A trajectory (solution path) might be needed Critical points (both derivatives zero) are studied and classified Because of their importance, equations of this type are frequently studied qualitatively in undergraduate differential equations courses, although quantitative solution of such equations is beyond the scope of an undergraduate course

The availability of high speed computers has made numerical methods (carried out by computer) the best choice for dealing with such equations. These methods are well developed and accessible to graduate students Recent developments in microcomputers have provided the necessary speed and memory capabilities. User convenience combined with superior graphics made microcomputers the machines of choice. We had access to the University's Z-100 microcomputers.

During the fall of 1984, the modular structure of the phase plane graphics program was designed, and modules to handle the input/output via menus and the graphics window were written These modules may be useful to others who wish to design graphics software and are included in Appendix B

The primary considerations in designing the program were ease of use, flexibility, and accuracy Accuracy was a major concern in the implementation of the numerical methods In order to provide flexibility, the user should be able to enter any data used by the program For example, for graphing a trajectory, the initial point, time interval, and error tolerances for the numerical routines may be supplied by the user In addition to controlling the data for the various graphing actions, the user should be able to change the region of the x,y-plane that is graphed and the differential equations

Ease of use was a major consideration in the design of the program as a whole, and the design of input/output. Because menus provide t easiest form of user control, it was decided that the program should oe menu driven. That is, a list of possible choices appears on the screen The user selects an action from the menu of choices by moving the cursor to the desired choice and pressing the RETURN key In the phase plane program, the graphics display occupies most of the screen and the menu occupies the lower quarter of the screen. To avoid having a confusing menu with too many items, there are a number of submenus which are reached from the main menu Each subsidiary menu contains all the options and input data necessary for a single action, such as drawing a trajectory.

Various other features were incorporated into the design to facilitate ease of use. Defaults were to be supplied for all data, so that a novice user could have the program graph things without having to enter data Online helpfiles were to be provided. The program was to be crashproof

The modular structure of the program was designed to parallel the menu structure. The program was to contain the following modules: main program (MAIN), control of the region of the x,y-plane that is graphed (GRAPH), entery of differential equations (PARSE); graphing a single trajectory, i e., solution, (TRAJECTORY), graphing a series of trajectories, i.e. phase plane (PHASE); graphing a direction field of vectors (ARROWS); graphing null clines (NULLCLN), finding and classifying critical points

(CRITPT). There was also a module to handle menu input/output (MENUS), and for technical reasons two small machine-specific modules pertaining to graphics and menus.

One other design consideration was machine portability. The machines we had available were Zenith Z-100 microcomputers. Although loosely described as IBM-compatible, the graphics and screen control on the Z-100 are totally different from the IBM-PC. In order to make the program available on the IBM-PC and similar machines as well as the Z-100, we decided to isolate the machine-specific part of the program in small modules (separate from the major graphics and menus modules). Since the program was to be written in MS Pascal and FORTRAN, it could then be run on any IBM-class machine running the MS-DOS operating system, provided appropriate machine-specific graphic pixil and screen control modules were written. The success (and lack thereof) of this strategy is discussed below.

During the fall of 1984, a Z-100 specific graphic pixel control routine was obtained from the Iowa State University Computation Center. Using this, Hogben wrote the module GRAPH, which sets up the graphics display and has procedures to graph points, lines, etc. Hogben wrote the module MENUS to display menus and interact with the user, and with advice from the Computation Center, wrote a brief Z-100 screen control module As an illustration of designing a module and translating the module to programming language, Alexander wrote the module NULLCLN. In the testing of these modules, we discovered an error in the way double precision arithmetic was carried out in Pascal and Fortran in the version of MS-DOS we were using (MS-DOS v 1.25, Pascal and FORTRAN77 v.3.10). As there is no way to correct this error, and double precision is essential for the numerical methods, it was necessary to obtain updated versions (MS-DOS v.2.13, Pascal and FORTRAN v.3.20).

During the spring semester of 1985, Hogben and Tondra taught Math 517x, a graduate course covering numerical methods for studying phase planes and developing phase plane software. Serious difficulties were encountered with both the quantity and quality of students. Only four students registered for the course (under a variety of credit arrangements) and two subsequently dropped it. This created a serious shortage of manpower for writing the modules, and impaired the design reviews (too few reviewers).

Although the ablest student, James Coyle, dropped the class due to pressures of other commitments, he completed the main program before leaving, and participated in some of the later design reviews. There were difficulties with both the students who remained in the class.

One of the students who remained in the class was receiving credit for a creative project required for his Master's degree, rather than course credit Although this arrangement worked well with a different student later, it led to delays in the project. When the student didn't complete his project (CRITPT) by the end of the course, he asked for more time (repeatedly). Although the underlying problem was that he lacked the ability to do the work, it wasn't until six months after the end of the course that he was finally relieved of his module and it was given to another student (Linda Ten Hoeve) as a creative Master's project.

The other student who remained in the course lacked essential programming skills. He was given the job of adapting existing (public domain) numerical code for finding trajectories (Shampine and Gordan, Computer Solution of Ordinary Differential Equations) for use in our software. Although he completed his work, it was so flawed that it had to be essentially redone by Alexander later.

Beacuse of the shortage of students, Hogben wrote the module for entering differential equations, PARSE.

At the conclusion to Math 517x in May 1985 the following modules had been completed: MAIN, PARSE, GRAPH, MENUS. CRITPT was still assigned to the student who failed to complete it. During the summer of 1985, we hired Coyle to design and write the ARROWS module, which he did skillfully and efficiently.

At the end of Fall semester 1985, the student assigned to CRITPT was removed from the project. Ten Hoeve was given the CRITPT module as a creative project required for her Master's degree. She completed this project successfully in the summer of 1986. However, because of difficulties in working with the program in its entirity, she did not incorporate her procedures into the phase plane program, but wrote a small driver program.

By the end of the summer of 1986, the program was complete except for the module PHASE and interfacing the Shampine/Gordan and Ten Hoeve subprograms into the main program. During the spring of 1987, Hogben adapted the Ten Hoeve code and Alexander adapted the Shampine/Gordan code.

The only remaining part of the original design left undone was the PHASE module, to draw several trajectories. Since each trajectory takes up to several minutes to draw, since the user can produce a phase plane of trajectories by repeating the trajectory command several times, and because the location of the trajectories desired depends on the differential equation, it was decided to omit this feature. Thus the program itself was complete (Z-100 version).

With the exception of the PHASE module, the final program follows the design developed during the fall of 1984. For technical reasons some modules are split between disk files and some disk files contain more than one module.

Because of the difficulties encountered during Math517x, the students did not write the user manual as they went along, as originally intended. Alexander is currently writing a detailed user manual for the program. When this is completed, we intend to test the software in an undergraduate differential equations course. In the meantime, brief instructions on how to use the program are available.

In order to facilitate dissemination of the software, we intend to produce versions for other MS-DOS machines, in particular the IBM-PC. The program design isolated machine specific features in two small modules, graphic pixel control and screen position control. During the summer of 1987, Hogben began work on the IBM-PC version of these modules. This was designed to be a minor job, but technical difficulties were encountered.

The screen control module controls the position of the cursor on the screen, normal or reverse video, beeps, etc. All these features are simple to do on the Z-100. If the implementation of the IBM-PC had been similarly straightforward, the IBM-PC version would have been quickly completed. However, we discovered that screen control on the IBM-PC is not automatically available. Although it can be arranged, it does not work as well as the Z-100, and some features, e.g., reverse video, are not available on the graphics screen. We had been using reverse video to highlight the cursor position and error messages. The cursor position can be boxed with graphics and the error messages can appear normally, but separating out these two cases is requiring changes in other parts of the program. The IBM-PC version of the software should be completed this fall.

The graphic pixel module for the Z-100 was obtained from the Iowa State University Computation Center. Fortunately, they also had available an IBM-PC version. This was assembled, tested, and is working well.

Project Results

The principal product of this project is the computer program, PHASE PLANE GRAPHICS FOR ORDINARY DIFFERENTIAL EQUATIONS. This program is currently available for the Zenith Z-100 and is being adapted for the IBM-PC and similar MS-DOS machines In order to obtain widespread distribution, we are planning to distribute it commercially. It has been submitted to CONDUIT. This program is included in Appendix B

In the process of developing this program, we developed menu format input/output and graphics control procedures callable from MS-DOS Pascal or FORTRAN These procedures may be useful to other software developers. They are included, along with a small demonstration program, in Appendix B

In the process of planning the course, a variety of materials were developed. These include program design notes, class plan, and a brief software development bibliography. These materials are included in Appendix C.

Copies of the above materials may be obtained from

Leslie Hogben
Department of Mathematics
Iowa State University
Ames, IA 50011
Tel. 515-294-8168 or 515-294-1752

Because information about equipment is necessary to provide computer diskettes, such requests are more easily handled by telephone.

Summary and Conclusions

This project successfully completed the development of planned software, Phase Plane Graphics for Ordinary Differential Equations. However, the intended method of development by a class of graduate students was not completely successful. The class did not have enough students to write the program or provide effective design reviews. There were also technical difficulties encountered, with computer arithmetic and machine specificity. These difficulties were less serious.

There a number of possible ways the student shortage could be overcome. Although information about the course was distributed to both the mathematics department and other departments, more personal recruiting through faculty advisors could have been done. Our best source of academic credit/unpaid student help was students who obtained credit for a Master's degree creative project, rather than actual course credit

There should have been more lead time before the course, to allow it to be listed in the schedule of classes, and allow students more time to plan to include it in their schedules.

It is possible that with changes such as these that more students could have been obtained. However, we seriously underestimated the rigidity of the graduate student program. Many students told us that they would like very much to take the course, but they had so many required courses that they couldn't fit it into their programs.

In overcoming the student shortage difficulty, several promising methods for developing software were utilized: Modifying currently available research programs, having graduate students develop software as a creative project associated with a Master's degree, and hiring graduate students.

Well designed and thouroughly tested programs utilizing sophisticated numerical methods are widely available on mainframe computers, and are frequently in the public domain. Microcomputers have now become so powerful that many of these programs are accessible to them Although such programs frequently employ a high level of numerical sophistication, they range from difficult to impossible for an inexperienced person to use. Such programs are often in the form of subroutines that must be called from another program, thus rendering them inaccessible to the nonprogrammer. However, if combined with a user-friendly program to handle input/output and a well written manual, existing numerical programs can provide the basis for valuable software. We successfully used the Shampine-Gordan code for the finding of trajectories in Phase Plane Graphics for

Ordinary Differntial Equations. Another example of a research level numerical analysis program combined with a user-friendly input/output program is the MATLAB program for matrix operations using LINPACK, developed by Eugene Johnson and others at the University of Iowa.

The other strategy employed to successfully complete the software involved using graduate students to design and write parts of the program, but not as part of the class. One student, Linda Ten Hoeve, wrote the module for finding and classifying critical points as the creative project for her Master's degree. Another student, James Coyle, who had written the main program in the class, was hired to write the ARROW module. Both these parts were done well and on time, because the students had the necessary ability, programming skill, and sense of responsibility. Both these approaches provide good software development experience, albeit without the team approach and design review experience originally planned. Given the rigidity of graduate student schedules, it is unlikely that enough students could be recuited to work on Master's projects simultaneously to provide an effective group experience. It is possible that a group of students could be hired during the summer, although it is not clear that enough high caliber students with the requisite skills are available. This approach has been used by David Sharp and Colin Prowse, Southern Illinois University, in a FIPSE project involving computer environmental simulation.

The results of this project suggest possible methods of developing undergraduate software by incorporating existing research subroutines and using graduate students, either by hiring them or by awarding academic credit for individual projects. This project also developed a sophisticated and easy to use program for use in an undergraduate differential equations course, Phase PLane Graphics for Ordinary Differential Equations

Student Produced Advanced Mathematical Software

Appendix A - Insights for FIPSE

Although there was substantial contact during the first year with FIPSE staff concerning FIPSE matters and the Technology Study Group, there was little contact concerning our specific project. There was also a lack of continuity of Program Officers. After Susan Forman left, we had contact with a variety of Program Officers.

I'm not sure more contact about the project would have been particularly useful, however. I do have a little concern that serious participation in the Technology Study Group could divert time and energy from projects. Although it is developing interesting ideas, our participation in FTSG was limited by lack of time available.

The intrusion of other interests of the project coordinators was somewhat of a problem in this project All of us have a variety of interests and other projects, which sometimes interfered with getting things done in timely fashion During the duration of this project, one of us wrote a book, one was assistant department chair, one was overloaded with graduate students, and two had leaves of absence

One other remark about difficulties encountered. We found "people problems" such as lack of students much harder to deal with than technical problems. This may be due to partly to the backgrounds of the coordinators, who have more technical experience.

Student Produced Advanced Mathematical Software

Appendix B

Phase Plane Graphics for Ordinary Differential Equations

Contents

The source files (PAS, . FOR, . ASM) can be read on any IBM-PC compatible microcomputer running MS-DOS The program itself can be run on a Zenith Z-100 running MS-DOS v 2 13 Place Disk 1 in the default drive and give the command

```
PHASEPLN
```

PHASE PLANE GRAPHICS FOR ORDINARY DIFFERENTIAL EQUATIONS

L. Hogben, R Alexander, R Tondra, J Coyle, L. Ten Hoeve, and
Iowa State Computation Center.
This program was developed with support from the Fund for the Improvement of Postsecondary Education, U S. Department of Education, and the Iowa State University Sciences and Humanities Research Institute All rights reserved, Iowa State University Research Foundation.

SUMMARY OF INSTRUCTIONS FOR USE

This program graphically displays solutions to the system of differential equations

$x'(t) = F(x,y) \qquad y'(t) = G(x,y)$.

INITIAL RUN

If necessary, boot the system (MS-DOS v. 2 13). Insert the PHASE PLANE disk 1 in the default drive. Type

```
PHASEPLN
```

A title screen will appear, with the instruction (at the bottom) to press RETURN. Do so, and the graphics screen and main menu will appear. The top two-thirds of the screen is a region of the phase plane where things will be graphed The lower part of the screen is a menu that allows you to carry out various actions The last line supplies information, such as what you can do at this point or error messages

You can move around the menu by using the arrow keys. To select an option, press the ENTER (or RETURN) key when the cursor is on the desired menu item (as indicated by the fact that the this menu item appears in reverse video). You can obtain information about what various menu options mean by pressing the HELP key.

To get the feel of the program, you should begin by moving the cursor to GRAPHICS MENU (press the down arrow key once), and select this option (press ENTER). This menu provides you with 5 choices: ARROWS, SINGLE TRAJECTORY, NULL CLINES, CRITICAL POINT and CLEAR GRAPHICS. Each of the first four provides another menu. Select SINGLE TRAJECTORY (by pressing the right arrow key once and then pressing ENTER) This menu allows you to enter the starting point of the trajectory, X(O), Y(O), DRAW or ERASE the trajectory, enter the error parameters for the numerical routines, ABS ERROR, REL ERROR, or the duration of the trajectory, TIME T.

Defaults are supplied for all data The default differential equation is

$x'(t) = (-x-y+4)/2 \qquad y'(t) = (x-y)/2$

Begin by selecting the DRAW TRAJECTORY option and watch the program draw the trajectory Note that while the program is graphing, the line at the bottom of the screen has changed It tells you it is graphing and allows you to stop the graphing by pressing the ESC key (in case it takes too long and you decide you don't want to wait for it to finish).

Now draw another trajectory by changing the initial point To do this, you will have to select menu item X(O), enter a new value (type in the number, ending with RETURN; see DATA ENTRY below for more information), select Y(O), enter a new value, and select DRAW again. If you want. you may also try to ERASE a trajectory, or change its length by changing TIME T

Next, return to the graphics menu by pressing HOME This key always returns you to the previous menu Then experiment with the other graphics options: ARROWS, which allows you to draw the direction field of derivative vectors, NULL CLINES, which allows you to draw the null clines (x' = O or y' = O), CRITICAL POINT, which allows you to find and classify a critical point (x' = O and y' = O) and mark it on the screen

Next, clear the graphics screen, go back to the main menu, select SET DIFFERENTIAL EQUATION, and enter your own equations, by selecting X' = and typing in a new equation. (N.B multiplication must be represented by *. See below for a complete list). Do the same for Y'. Now go back and try the various graphics options on your own equations.

Finally, return to the main menu and select SET WINDOW. This provides a menu that allows you to modify the region of the phase plane that is graphed (as well as alter other features, such as whether axes appear). Note that when you change the minimum or maxixmum x or y values, the graphics screen does not change automatically. Once you have entered your new bounds, you must select REDRAW WINDOW to adjust the area graphed. This will also clear the graphics screen.

When you have finished using the program, either turn off the machine or return to the main program, select QUIT PROGRAM, and respond Y for yes when prompted

Further information about the various menus is provided below, as is detailed information on entering data.

DATA ENTRY

You enter a piece of information (integer, real number, diffewrential equation) by selecting a menu option that calls for the information and typing it in During data entry you cannot move around the menu or select other options.

To enter a real number or integer, type in the number in the usual form (not scientific notation), ending with RETURN (or ENTER) If you make a mistake while typing (e g , a letter rather than a digit), the program will detect this and send you an error message to remove the error by pressing BASCKSPACE. If (before pressing RETURN) you decide that you do not want the new entry, you can press ESC and the program will revert to the its previous information (just prior to beginning entry).

While entering a differential equation, you can move around in the equation by using the left or right arrows, delete the character left of the cursor with BACKSPACE (or DELETE), delete the whole equation with DEL LINE, and insert characters by pressing the the appropriate keys. To end entry, press RETURN to accept your new equation or ESC to revert to the previous one A differential equation may include numbers (real or integer), the variables x and y, operands +,-,*,/,', functions, parentheses, and the parameters P1, P2, P3, P4, P5, P6 The table below lists symbols and their meanings The equation must also be in a form that the program can understand. It will detect anything it cannot understand as an error and supply an appropriate message

Symbol	Meaning
+	addition
-	subtraction or negation
*	multiplication
/	division
^	exponentiation
EXP()	exponential function
LN()	natural logarithm
LOG()	base 10 logarithm
SIN()	sine
COS()	cosine
TAN()	tangent
ATAN()	arctangent
SQRT()	square root
ABS()	absolute value
( )	parentheses (must be matched)
1 2 3 4 5 6 7 8 9 0	digits for numbers
.	decimal point
	blank (ignored except within number)
x, y	variables
P1, P2, P3 P4, P5, P6	parameters (values assigned by selecting SET PARAMETERS)

LIST OF MENUS

MAIN MENU

SET WINDOW. Menu for options controlling portion of phase plane displayed

SET DIFFERENTIAL EQUATION: menu for entering differential equation

GRAPHICS MENU. menu for graphics options

CLEAR GRAPHICS. clears graphics screen

QUIT PROGRAM quit program; requests confirmation Y (yes) or N (no), default is no

SET WINDOW Menu for options controlling portion of phase plane displayed

MIN X =. minimum x value graphed (left edge)

MAX X =. maximum x value graphed (right edge)

X GRID =. number of grid points in the x direction

MIN Y =: minimum y value graphed (bottom edge)

MAX Y =: maximum y value graphed (top edge)

Y GRID = number of grid points in the y direction

DRAW/ERASE AXES. draw axes if absent, erase axes if present

REDRAW GRID: draw grid with X GRID and Y GRID points; does not clear grahics screen

REDRAW WINDOW. draw window with indicated values, clears graphics screen

```
SET DIFFERENTIAL EQUATION  menu for entering differential
                           equation
     SET PARAMETERS: menu for enter parmateters for
                     differential equation
     X' =:  enter equation for the derivative of x with
            respect to time
     Y' =:  enter equation for the derivative of y with
            respect to time

SET PARAMETERS  menu for entering parmateters for
                differential equation
     P1 =:  enter parameter P1 (real number)
     P2 =:  enter parameter P2 (real number)
     P3 =:  enter parameter P3 (real number)
     P4 =   enter parameter P4 (real number)
     P5 =:  enter parameter P5 (real number)
     P6 =:  enter parameter P6 (real number)

GRAPHICS MENU. menu for graphics options
     ARROWS: menu for drawing direction field of derivative vectors
     SINGLE TRAJECTORY: menu for drawing single trajectory
     NULL CLINES: menu for drawing null clines
     CRITICAL POINT: menu for finding and drawing critical point
     CLEAR GRAPHICS: clears graphics screen

ARROWS. menu for drawing direction field of derivative vectors
     ARROW DENSITY =: enter integer to control arrow density;
                      larger means more smaller arrows
     DRAW ARROWS: finds and draws direction field
     ERASE ARROWS. finds and erases direction field

SINGLE TRAJECTORY: menu for drawing single trajectory
     X(O) =: enter the x-coordinate of the initial point (real number)
     Y(O) =. enter the y-coordinate of the initial point (real number)
     DRAW TRAJECTORY  find & draw path from t=O to t=TIME.
     ABS ERROR =  error tolerance for trajectory finder (real
                  number; at least one error tolerance must be >O).
     REL ERROR =: error tolerance for trajectory finder (real
                  number, at least one error tolerance must be >O).
     ERASE TRAJECTORY. find & erase path from t=O to t=TIME
     TIME T =: end time for trajectory

NULL CLINES: menu for drawing null clines
     NUMERICAL SEARCH SPACING =. enter space between points tested for
            sign change in the derivative (integer, larger = more
            space between marks, fewer, less accurate marks)
     DRAW NULL CLINES. finds and marks null clines.
     ERASE NULL CLINES. finds and erases null clines.
```

```
CRITICAL POINT. menu for finding and drawing critical point
     NEARBY X =. enter x value of point near critical point sought,
               "NEARBY" changes to "FOUND" and value changes after
               the critical point is found
     NEARBY Y =: enter y value of point near critical point sought,
               "NEARBY" changes to "FOUND" and value changes after
               the critical point is found
     DRAW CRITICAL PT:  draw critical point (must be found first)
     ERASE CRITICAL PT.  erase critical point (must be found first)
     FIND CRITICAL PT. find and classify critical point; changes
               "NEARBY" to "FOUND".
```

Student Produced Advanced Mathematical Software

Appendix C

Materials Developed for Classroom Use

Contents

General Requirements for Math 517X (FIPSE Project) Software

L. Hogben January 1985

This document was developed with support from the Fund for the Improvement of Postsecondary Education, U. S. Department of Education, and the Iowa State University Sciences and Humanities Research Institute. All rights reserved, Iowa State University Research Foundation.

Specifications

Specifications will be provided by the instructors, detailing the modular structure of the project. For each module, those procedures intended to be available to other modules or programs will have purposes, declarations, calling syntax, and parameters specified

Designs

Design will be done during the course. Designs will b adequately documented, following the style of the procedure NULCLN. Prior to coding the design, a design review will be conducted by the class After the design is accepted it will be coded.

Code

All code will be written in MS-PASCAL or MS-FORTRAN and will be capable of being run successfully on any MS-DOS machine with 192K or more memory. Should it be necessary to provide machine specific information, this will be isolated in a datafile (e.g machine constants) or a minimal machine-specific code module (e.g. ZSCREEN), commonly called a device handler.

All code will conform to the layout and comment style exhibited in modules MENUS and GRAPH. Specifically, most comments will be one line explanations of a particular action interspersed in the code, except that an initial explanation of all global module variables will be given. Each level of dependence will be indicated by indenting 2 spaces. For example, under an IF-THEN statement, indent an additional 2 spaces. Separate statemnts will be on separate lines.

After the code is written, grammar will be tested by compiling it and any grammatical flaws will be corrected. Compilable code will then be presented presented to the class for a code review. During the code review, structural and functional testing will be designed.

Testing

After the code is accepted it will be tested. Any errors found during testing will be corrected and the module retested On completion, the software will be tested as a whole.

User Interface

Material will be presented to the user by a series of menus. Each menu will appear at the bottom of the screen with a command line. The upper portion of the screen will be a graphics area.

The software must be crashproof. In case of error it should provide helpful diagnostic messages Program accessible helpfiles will be be part of the software package. Any action the user takes should be reversible. An exception to this requirement (e.g. terminating the program) should reuire user confirmation. All I/O must be done in such a manner as to prevent screen scrolling.

A clear, well-written user manual will be written. It will explain to the user the function of each menu and command, illustrating these functions with examples.

Modular Structure

The main program will be responsible for initializing the menus and machine constants. It will also be responsible for controlling the transition between menus. The authors of the main program will be responsible for providing an appropriate helpfile for each menu. The main program must be written in Pascal. It will have access to modules to carry out the graphics, menu layout, and necessary mathematics.

Each module will be responsible for its own error handling and diagnostic messages.

REVIEW CHECKLIST

Note anything you do not understand.

Note anything that is not sufficiently well defined to enable you to carry out the next stage or to check with the previous stage

Does it conform to the general standards?

Does it conform to previous stages?

Are there any unstated assumptions or restrictions?

Is it well structured?

Is it testable?

Is it maintainable?

I/O:
- Is it crash proof?
- Are useful messages supplied to the user to assist in recovery from errors?
- Is it flexible?

Arithmetic Errors:
- Is it crashproof?

Numerical Algorithms:
- Is it crashproof (i.e guaranteed correct to within some tolerance)?
- Are useful messages supplied to the user to assist in recovery from errors?

Is it hardware compatible?

Is it software compatible?

QA76.6 J66	Jones - Software development, a rigorous approach
QA297 M527 1984	Miller - The engineering of numerical software
QA76.6 D845 1984	Dunn - Software defect removal
QA76.6 F86	Wulf - Fundamental structures of computer science
QA372 S416	Shampine - Computer solution of ordinary differential equations The initial value problem

Draft design for NULCLN procedure to plot nullclines

Design Review Comments

R. K. Alexander January 1985

(Hogben)

This document was developed with support from the Fund for the Improvement of Postsecondary Education, U. S. Department of Education, and the Iowa State University Sciences and Humanities Research Institute. All rights reserved, Iowa State University Research Foundation.

on all
real -> int conv
(including int/int) specify
rounding

OVERALL STRATEGY

Search for sign changes between the points of a square grid placed on the window. When a sign change is found in either derivative, localize it further by a single secant step and mark it by three pixels in a vertical column if dx/dt = 0 or three pixels in a horizontal row if dy/dt = 0.

what does secant
step mean --
cf. seekz lin interp.

IMPORTS

```
    XO, YO, X1,Y1:  REAL;  {Coorinates of window [XO,X1] x [YO,Y1]     }
    SPACE: INTEGER;        {Size in pixels of grid on which changes    }
                           {of sign are sought; user selects one of    }
                           {4,6,12,16,20 from menu.                    }
```

does not
conform to
main specs

EXTERNAL and FUNCTIONAL REFERENCES:

```
    DERIV  {Subroutine to evaluate deriva.ives}
    GRNON {Procedure to turn on individual pixels}
    SGN    {Signum function}
    MIN    {INTEGER minimum of INTEGER arguments}
```

LOCAL IDENTIFIERS:

```
    VECTOR means ARRAY[1..2] OF REAL8;
        X: VECTOR;        {Position of current point in grid}
        DX: VECTOR;       {Vector of distances in the coordinate directions }
                          {                                                 }
                          {DX[1] = ((X1-X0)/640) * SPACE                    }
                          {DX[2] = ((Y1-Y0)/180) * SPACE                    }
        XDOT: VECTOR;       {Derivatives evaluted at current point        }
        OLDRV: VECTOR       {Derivatives evaluated at 'north' neighbor  }
                            {of current point.              }
        SAVDRV: ARRAY[0..45] OF VECTOR  {Derivatives at grid points in the     }
                                        {next column to the left.  It is       }
                          {assumed tha  he window will occupy at most pixels 0 to }
                          {179 in the vertical direction so that all derivatives in a }
                          {column of grid points with the densest spacing (4) can be }
                          {saved in this array.                                        }
        RTEDGE: INTEGER CONSTANT; {Rightmost pixel column = 639}
        BMEDGE: INTEGER CONSTANT; {Bottom pixel row = 179}
        IX,IY:  INTEGER;      {Counters for pixel position in x,y directions      }
        INDY:   INTEGER;      {The ordinal number of the current grid point in    }
                              {its column, counting the top point as 0.  Used     }
                              {as an index into the array of saved derivatives    }
```

Machine constants
should be passed
this should be in body

Good note of restriction
other machines

Import this

Specify integer type

LOCAL PROCEDURE:

```
        SEEKZ(X,DX,IX,IY,DIR,XDOT,OLDRV);
        { Checks each component of the derivative for a sign change in the     }
        { direction DIR (=1 or 2). If one is found, localize it further by     }
        { a secant step, and turn on the appropriate pixels on the screen.     }
        { Details for SEEKZ are given after design of NUCLN.                   }

BEGIN { NULCLN }
    { Initialize DX     }                                                          cf previous page
    {                   }
    { Loop through the columns in the grid }
    IX := 0;  X[1] := X0;                                                          use separate lines
    WHILE  IX < RTEDGE  DO  { Loop through the grid points down this column   }   use For instead
        IY := 0;  X[2] := Y1;  INDY := 0;
        WHILE IY < BMEDGE  DO
            DERIV(X,PAR,XDOT); { Evaluate derivatives at new grid point  }
            {                                                            }
            IF  IY > 0  THEN {There is a grid pt to the north; check     }
                SEEKZ(X,DX,IX,IY,2,XDOT, OLDRV); { for sign changes.     }
            {                                                            }
            { Check for sign changes to the left of the current point    }
            {                                                            }
            IF  IX > 0  THEN SEEKZ(X,DX,IX,IY,1,XDOT, SAVDRV[INDY]);
            {                                                            }
            { Save the derivatives from this point:          }
            OLDRV <-- XDOT;
            SAVDRV[INDY] <-- XDOT;
            {                                                }
            { On to the next point in the column             }
            IY := IY + SPACE;
            INDY := INDY + 1;
            X[2] := X[2] - DX[2];
            END WHILE { IY < BMEDGE }
            {                                                }
            { On to the next column                          }
            IX := IX + SPACE;
            X[1] := X[1] + DX[1];
        END WHILE { IX < RTEDGE }
    END NULCLN
```

INTERNAL PROCEDURE SEEKZ:

```
    X,DX,XDOT,OLDRV,IX,IY  as in NULCLN
    DIR: INTEGER; { Index of coordinate search direction -- 1 or 2 }
```

LOCAL IDENTIFIERS:

```
    K: INTEGER; {Index on the components of the derivative                  }
    PIXEL: ARRAY[1..2] OF INTEGER: { Coordinates of pixel to light on screen }
```

```
BEGIN {SEEKZ}
    FOR K := 1 TO 2 DO { Check each component }
        IF SGN(XDOT[K]) < > SGN(OLDRV[K]) THEN { Sign change found }

            PIXEL[DIR] := PIXEL[DIR] - $\frac{(-1)^{(DIR-1)}\ XDOT[K]}{XDOT[K] + OLDRV[K]}$     # SPACE

            { The power of (-1) is because Y decreases as you go down the screen}
            { This formula amounts to a secant step                              }

            { The pixel in the other direction should be the common IX or IY     }
            { index of the grid points being checks.                            }

            {     .                                                              }
            { If x = 0, illumine 3 pixels in a vertical column centered here.    }
            {     .                                                              }
            { If y = 0, turn on 3 pixels in horizontal rown centered here.       }
            { "Here" means at location (PIXEL[1],PIXEL[2]) on the screen.        }
            {                                                                    }
```

incorrect

unclear; linear interp
of loc of 0
TRUNC (+0.5)

Summary of changes to first draft design of NULCLN plotter: (Alexander)

Imports: The pixel boundaries of the window have been added. I don't
assume any more that the window is [0,639] x [0,179].
The draw/erase flag is added. The former design drew only.
The vector PAR of user-set parameters in the derivative functions
is needed, of course.

Body: DX[1], DX[2] are now defined to be the spacing of single pixels
along the coordinate axes.

SEEKZ proc: Delete spectator parameters X, DX
Add grid spacing, pixel #s of window boundaries, draw/erase flag.

In addition to recognizing sign changes, the design now handles the
case of a derivative being zero at both grid points. The design
is rather crude -- this rare case is tested first; and interior grid
points at which a derivative is exactly zero will be marked twice.

The formula for the secant step was incorrect; it is now corrct.

MARK proc: This new, lower-level routine has been added to actually perform
the draw/erase on the screen. It uses MIN & MAX to ensure that
GRNON & GRNOFF are only called with pixel addresses inside the
window. Details are given which did not appear in the first draft.

January 13, 1985 Design Review: Comments

(Hogben)

Memo: Mathematics 517X

From: Roger Alexander

Re: Revised Design of Nullcline Procedure

OVERALL DESCRIPTION: The user has selected a grid spacing of 4,8,12,16 or 20 pixels from the menu. This routine searches for sign changes in both derivatives between points of a grid with the selected spacing. When a sign change is found, the root is further localized by one secant step, and then marked by three pixels in a vertical column if dx/dt = 0, or three pixels in horizontal row if dy/dt = 0. The pixels are turned on if the user has requested "DRAW", turned off if the user requested "ERASE".

-- why only 4, 8, etc.

```
IMPORTS:
     X0,X1,Y0,Y1: REAL;  { Coordinates of window [X0,X1] x  }
                         { [Y0,Y1]                          }
     LFEDGE,RTEDGE,BMEDGE,TPEDGE: INTEGER:
                         { Pixel numbers corresponding to X0,X1  }     good
                         { and Y0,Y1 respectively.               }
     SPACE: INTEGER      { Grid spacing = 4,8,12,16 or 20.       }     use 4 ≤ SPACE ≤ 20
     DRAW: BOOLEAN       { True --> Draw; False --> Erase.       }
     PAR: VECTOR(6) of REAL8; {Parameters in derivatives.        }

EXTERNAL AND FUNCTION REFERENCES:
     EVAL    {Subroutine to evaluate derivaitives   }
     GRNON,GRNOFF        { Procedures to turn off {resp. on} single pixels          }   turn on (resp. off)
     SGN                 { Signum function                                          }
     MIN                 { INTEGER minimum of INTEGER arguments                     }
     DBLE                { INTEGER or REAL to DOUBLE PRECISION conversion           }

LOCAL IDENTIFIERS:
     VECTOR means ARRAY[1..2] of DOUBLE PRECISION
     X: VECTOR;          { Position of current grid point                           }
     DX: VECTOR;         { Vector of distances in the coordinate directions         }
                         { represented by one pixel                                 }
     XDOT: VECTOR;       { Derivatives evaluated at current point                   }
     OLDRV: VECTOR;      { Derivatives at 'north' neighbor of X                     }
     SAVDRV: ARRAY[0..45] of VECTOR;
                         { Saved derivatives at grid points in the previous         }
                         { column to the left. The size of this array is            }
                         { based on the assumption that the window                  }
                         { occupies at most pixels 0 to 179 in the vertical         }
                         { direction.                                               }
     IX,IY: INTEGER;     { Pixel position in x,y directions.                        }
     INDY: INTEGER;      { The ordinal number in its column of the current          }
                         { grid point, counting the top point as 0.                 }
                         { Used as an index into the array of saved                 }
                         { derivatives.                                             }
     MINPX, MAXPX: ARRAY[1..2] of INTEGER;
                         { Pixel boundaries of window     }
```

```
begin { NULCLN }
    { Compute x- and y- spacing of pixels }
    DX[1] := (X1 - X0) / (RTEDGE - LFEDGE);
    DX[2] := (Y1 - Y0) / (BMEDGE - TPEDGE);
    MINPX[1] := LFEDGE;
    MINPX[2] := TPEDGE;
    MAXPX[1] := RTEDGE;
    MAXPX[2] := BMEDGE;
    { Loop through columns of the grid  }
    for IX := LFEDGE to RTEDGE in steps of SPACE do
        { Compute x-coordinate of grid points in this column     }
        X[1] := X0 + DX[1] * (IX - LFEDGE) ;
        { Loop through grid points in this column       }
        for IY := TPEDGE to BMEDGE in steps of SPACE do
            { Compute index and y-coordinate of this point      }
            INDY := (IY - TPEDGE) / SPACE;
            X[2] := Y1 - DBLE(IY - TPEDGE) * DX[2];
            EVAL(X,PAR,XDOT); {Evaluate derivatives at this pt  }
            if IY > TPEDGE then { Check for sign-changes to north     }
                SEEKZ(IX,IY,SPACE,MINPX,MAXPX,2,XDOT,OLDRV,DRAW);
            fi;                                                          fi=endif
            if IX > LFEDGE then { Check for sign-changes to the left  }
                SEEKZ(IX,IY,SPACE,MINPX,MAXPX,1,XDOT,
                    SAVDRV[INDY],DRAW);
            fi;
            { Save derivatives from this point      }
            OLDRV <-- XDOT;  SAVDRV[INDY] <-- XDOT;
        od; { IY }
    od; { IX }
end; { NULCLN }


proc     SEEKZ (IX,IY: INTEGER; { Pixel coordinates of current pt }
              SPACE: INTEGER;        { Grid spacing     }
              MINPX, MAXPX: ARRAY[1..2] of INTEGER;
                                     { Pixel boundaries of window     }
              DIR: INTEGER;          { Coordinate direction in which  }
                                     { to search: 1 or 2              }
              XDOT,OLDRV: VECTOR       { Derivatives at current grid     }
                                       { point and previous point        }
              DRAW: BOOLEAN);        { Draw/erase request    }


local identifiers:
    K: INTEGER;     { Index for components of derivative }
    PIXEL: ARRAY[1..2] of INTEGER;
                      { Screen coordinates of zero of a derivative     }
```

```
begin { SEEKZ }
    for K := 1 to 2 do      { Check each component of derivative      }

        PIXEL[1] := IX; PIXEL[2] := IY;
        if XDOT[K]=0 and OLDRV[K]=0 then { Two zeros }
            MARK(PIXEL,MINPX,MAXPX,K,DRAW);
            PIXEL[DIR] := PIXEL[DIR] - SPACE;
            { Back up PSACE pixels in the DIR direction }
            MARK(PIXEL,MINPX,MAXPX,K,DRAW);                              why do it twice?
        else
            if SGN(XDOT[K] <> SGN(OLDRV[K] then
                { Sign change found in this component.  Back up    }
                { by a secant step parallel to the DIR axis.       }
                PIXEL[DIR] := PIXEL[DIR] - SPACE *
                          XDOT[K] / (XDOT[K] - OLDRV[K]);
                MARK(PIXEL,MINPX,MAXPX,K,DRAW);
            fi; { Sign change }
        fi; { Two zeros }
    od; { Check components }
end; { SEEKZ }


proc MARK(PIXEL: ARRAY[1..2] of INTEGER;
            { Screen location of zero of derivative    }
        MINPX,MAXPX: ARRAY[1..2] of INTEGER;
            { Boundaries of window on screen   }
        K: INTEGER; { Index of vanishing derivative component      }
        DRAW: BOOLEAN); { Draw/erase request    }


local identifiers:
    PFIX: ARRAY[1..2] of INTEGER; { Pixels to alter }
    L: INTEGER; { Loop index to alter 3 pixels    }
    NOTK: INTEGER;      { The index (either 1 or 2) which is not K.     }
                        { If XDOT[K] is zero, the vectorfield           }
                        { is perpendicular to the K coordinate axis.    }


begin { MARK }
    PFIX[K] := PIXEL[K];
    NOTK := 3 - K;
    for L := -1 to 1 do     { Fix 3 pix   centered at the zero         }
        PFIX[NOTK] := PIXEL[NO  K] + L;
        { But don't get out of the window }
        PFIX[NOTK] := MIN(MAXPX[NOTK],PFIX[NOTK]);
        PFIX[NOTK] := MAX(MINPX[NOTK],PFIX[NOTK]);
        if DRAW then GRNON(PFIX[1],PFIX[2]);
             else GRNOFF(PFIX[1],PFIX[2]);
        if; { DRAW }
    od; { L }
end; { MARK }
```

Remarks on 13 JAN 85 draft design of nulldine plotter. (Alexander)

1. There is no checking of arguments. Possibilities include:
   a) create a type, SPACETYP with only values 4,8,12,16, & 20.
      Then the compiler can check SPACE.

   b) type XPIXEL = [0..639]; YPIXEL = [0..224];
      have the <u>routine</u> actually check

```
LFEDGE < RTEDGE,
TPEDGE > DMEDGE
  X0 < X1
  Y0 < Y1
```

2. PASCAL-like declarations of external & function references have not been given. It is likely that this is desirable for checking argument types. Some possible problems with functioanlity, too:
   a) SGN must implement the mathematical, 3-valued signum function.
      Fortran's SIGN (which takes two arguments) may not do the trick.

   b) The design is completely sloppy about REAL4/REAL8 and INTEGER2/INTEGER4.
      What "INTEGER" means depends on the implementation language! the MACRO-assembler code for GRNON gives the calling sequence as

```
FORTRAN:    Call GRNON(IX,IY)
PASCAL:     GRNON(X,Y: INTEGER4);                This is wrong
```

   I would be very surprised if both worked! Conclusion: we need a language-independent way to specify argument types in "absolute" terms, a.:d we must pay careful attention to type conversion at the design stage, or the code stage will be a killer.