Engineering Design Optimization using Calculus Level Methods: A Casebook Approach

By Phil B Brubaker

Welcome to Calculus-level Problem Solving!

Engineers in industry wanted to 'tweak' their parameters. So this textbook was written to show the simplicity of 'tweaking' parameters in algebraic through differential equation problems when using a Calculus-level language like PROSE or FortranCalculus. FortranCalculus (FC) is available on the web.

Automatic Differentiation (AD) and *Operator overloading* were key technologies that allowed numerical methods, now called solvers, to be stored in a FC library. A user will use a solver by stating a solver name in a 'find' statement using the 'by' clause. Want to switch solvers? Just change the solver name (e.g. from 'Ajax' to 'Jupiter') and you are ready to try a different numerical method! It is that easy to code. (See the FortranCalculus manual for suggestions on what solver to use for a given problem.)

Help spread the word about Calculus-level thinking and problem solving. Do you know any engineering or science professors that might have a problem that could be solved and shown to their future students?

This textbook tries to move today's thinking from solving one problem at a time, to solving all of their project's problems at once while tweaking parameters in order to achieve an optimum solution. This requires Calculus-level thinking. An analogy might be thinking in terms of Machine code, one bit at a time. Today, computer simulations have people thinking in terms of Algebraic code, one problem at a time. We are trying to move people to Calculus-level code, solving entire projects at a time. This will reduce development time and improve accuracy of their math models. (Future CEOs should study the Oil Refinery Production problem in order to see future possibilities with Calculus-level thinking.)

Mission Statement:
Get the FortranCalculus compiler operational and in use via the internet. It's a free compiler that simplifies solving math problems by minimizing code necessary to state & solve a problem. Some new thinking is necessary for those wanting to get the most for their buck; convert from simulation to optimization thinking.

What's the different between simulation and optimization? Picture a saw horse construction project. A Simulation would yield A saw horse where Optimization would yield an Optimal saw horse. If the objective (function) was good and proper then the Optimal saw horse would be the best solution, right? For example, the objective might be lightest & strongest saw horse. A wrong objective might be just the strongest saw horse. This might yield a strong horse but a very heavy one!

If you were a manager or CEO and had the choice of a simulation design versus an optimization design, which would you pick?

Modeling & Simulation's next step is (Mathematical) Optimizations. Optimizations require an Objective (function). Today's Engineers & Scientists solve problems with a "Find X" mind-set. With some Operational Research training they could expand their thinking to a "Find X to Optimize Y" mind-set. Then they would be ready for Optimizations, Calculus-level programming and software. (This would drop today's design times that require months even man years to one or two days! Manufacturing processes could be optimized to the days demand and thus maximize their profits.)

"Find X to Optimize Y" thinking among professors will cause most Engineering & Science textbooks to be rewritten with optimization examples and discussions. This will be great stuff for industries and government; applied engineering and/or science not just theories.

Table of Content

Welcome to Calculus-level Problem Solving! 1
About 5
Introduction 5
1 General Algebraic Equations 9
Background of TFH Math Model for a Readback Pulse from Magnetic Recording 9
A *Typical* Readback Pulse from Magnetic Recording 12
An *Unusual* Readback Pulse from Magnetic Recording 15
A *Typical* Readback Pulse from Magnetic Recording with Improved Model 17
An *Unusual* Readback Pulse from Magnetic Recording with Improved Model 19
Curve fitting: A Sinusoidal Signal 21
Curve fitting: A Damped Sinusoidal Signal 23
1.4 Conclusion on Curve Fitting 25
Pharmacokinetics 26
Slack Variable Techniques 29
Paper Bicycle Design 31
Chapter 1 Exercises 33
2 La Place Transforms 34
Optimum Matched Filter (Transfer Function) 34
Chapter 2 Exercises 44
3 Ordinary Differential Equations 46
Second Order Non-Linear ODE 47
A Third Order Non-Linear ODE 50
A Bang-Bang Control Problem 53
Non-Linear Equations of Motion 62
4 System of Differential Equations 65
The Lorentz Equations, a System of ODEs 66
The Convection Reaction Equations, a System of PDEs 69
Body Plasma Chemistry 71
Modeling a Nanostructured Solar Cell 76
Chapter 4 Exercises 82
5 Partial Differential Equations 83
PDEs: Stock Market to Biology 84
Burgers' Equation 86
Telegrapher's Equation 89
6 Inverse Problems 92
Custom Thermistor Design 93
Drug Development 95
Heat Transfer over 1D Slab 96
Robot Arm Movement 99
Plane Crash Locator 102
7 Implicit Equations 104
System of Implicit Algebraic Equations 105
2nd Order Implicit Differential Equation 108
8 Nesting Solvers 110
Nesting … Matched Filter 111
Oil Refinery Production 113
9 Miscellaneous 118
Monte Carlo Simulation OR Total Derivative? 118
Stiff Equations & Trouble Shooting 119
10 Conclusions 121
10.1 Future: Thinking outside the box 121
11 Appendix 125
Picking the right Solver 125
'aplot' source code 125
Spectral Estimation (freeware) Software 126
'readrit1.100' File Listing 126
'readrit2.200' File Listing 127
Arbitrary Equalization with Simple LC Structures 130
Incomplete Problems: can you help complete one or more? 133
Index 134

Preface

How to teach new problem solving technology to engineers and scientists? Problem solving requires a broad based knowledge in math and science as well as discernment and flexibility to challenge the way it has always been done in the past. Generally, an objective driven design will yield the best design in the least amount of time. Companies need engineers trained in setting objectives before they begin the time-consuming process of formulating and testing new concepts and designs.

This textbook considers design from the pragmatic concerns of industry. It utilizes casebook studies of math problems with their solutions in real life situations. Because it encourages students to view themselves as part of the design team, this text is the next best thing to an on-the-job training. It shows how setting objectives to problem solving assignments can help students complete work quickly and efficiently, but it also stresses that while every situation is different, the approach remains the same: objective-driven engineers state a math model and an objective function for a given problem while leaving the solving to a calculus-level computer language/compiler.

The text attempts to fill a gap in educational material in the mathematical problem solving arena. Traditional texts leave students in a simulation thinking mode. Simulations require many computer runs causing delays in solution and little gain, if any, in problem understanding. Simulations require a numerical algorithm to be meshed with their math model. In such form, math models are hard to recognize and discuss. Besides slowing their understanding, users lose confidence in program solutions.

This textbook tries to move today's thinking from solving one problem at a time, to solving all of their project's problems at once while tweaking parameters in order to achieve an optimum solution. This requires Calculus-level thinking. An analogy might be thinking in terms of Machine code, one bit at a time. Today, computer simulations have people thinking in terms of Algebraic code, one problem at a time. We are trying to move people to Calculus-level code, solving entire projects at a time. This will reduce development time and improve accuracy of their math models.

NASA funded the development of the first Calculus-level language through TRW called Prose. Prose became available to the public in 1974 through a national computer time-sharing network. Prose ran on large Control Data Corporation (CDC) 6600 computers. *Automatic differentiation* and *operator overloading* were key technologies for this project. I taught the Prose language to Engineers & Scientists in the San Francisco Bay Area from 1975 through 1979. Most national time-sharing computer networks died in the 1980s and thus went Prose. FortranCalculus is the next Calculus language on the horizon. It is in testing mode now and will soon be released on the web.

Things to learn from this textbook include:

- How Calculus-level programming simplifies problem solving;
- Use of Lorentzian (function) series for curve fitting;
- How to find frequency parameters when curve fitting sine series to data;
- Manage by Objective; and,
- How to 'tweak' hundreds of parameters at once.

Features that link concepts to the real world

Approach:

- Practical and elementary procedures which rely on true understanding. It is expected that students should understand Integral and Differential Equation notation before using this text.

- Casebook studies which involve students in the real life drama of the design engineer or applied mathematician.

- Exercises which state a math problem and its objective(s) in a simple and precise format. This format allows peers and associates to discuss a problem's definition in great detail. A problem definition normally requires just a few lines plus the math model. These compact definitions allow associates worldwide to receive faxed copies on which they may immediately respond.

- The opportunity to learn from other people's successes and failures sets this book apart from others. Just glancing at the table of contents reveals examples from many industries— Aerospace, Chemical, Computer, Pharmaceutical, Structual Engineering — to name a few.

Timeliness:

- During a time of company downsizing increasing engineering and science productivity would help keep U. S. jobs. For example, one objective-driven problem/solution used the calculus based language PROSE to reduce a matched filter design time from 12 weeks to one. The filter was used in a Memorex disc drive and normal turnaround design time would have required a 3-month cycle. With the proper objective function, math model and a calculus based computer language, an optimal filter was designed and tested in less than one week. This filter's objective function and math model originally came from many older engineering textbooks that are still in use today. Memorex employees required two years of testing and listening to find the true objective for this filter.

New Technologies:

- A calculus-based computer language for PC usage will be available in the near future. (The struggling economy which reduced venture capital has slowed the release date by several years. With or without software, students who learn the objective-driven solution methodology will increase productivity from their increased understanding, even if their solution is incorrect.) This computer language requires an objective function as well as a math model to determine an optimal solution.

- Computer usage is reduced by several orders of magnitude.

- Problem definition teaching that consists of a math model and an objective function.

Features:

- Objective-driven problem solving provides several user benefits:
  - Clear Problem Definition
  - Accelerates Problem "Understanding"
  - Decouples Models from Algorithms ... i.e., removes "noise" from the picture
- Other benefits when solutions come by way of a calculus based language:
  - Allows Rapid Model Prototyping
  - Allows Interchangeable Algorithms
  - Enabled by **Automatic Differentiation**
  - Allows Structured Nesting of Algorithms ... a first!

About

Notes: These example problems in this textbook were solved on a DOS version of *FortranCalculus* that the Software Architect, Joe Thames, provided for a few of us for testing in early 1990s. Joe was the main force behind these Calculus-level Compilers since the 1960s. See his MetaCalculus (MC) website for a history of Calculus compilers.

Author: Phil B Brubaker Accomplishments

- Reducing circuit development time from 12 weeks to less than 1 week was another increased productivity example at Memorex Corp. Plus the solution was optimal. This circuit was called a Matched Filter where minimizing inter-symbol interference (ISI) was the goal. This was accomplished by using the Calculus-level Problem-Solving computer language, Prose.
- Increased productivity resulting from optimizing a software program that required 20 to 30 days per execution to less than 10 hours at Lockheed Missiles & Space Co. Received Presidents award. Saved Lockheed $10 Million!

Software Architect: Joe Thames Background

Professional Mission: Meta Science

> Pioneering a means of hyper-simplification in very advanced mathematical software design and development by end-user scientists to address ad-hoc systems optimization problems, and suitable for learning science and mathematics in primary education.
> Applying automated intelligence (search engines) through metaphoric software construction methods (e.g. spreadsheets) from Apollo, now being adapted for web use via service-oriented architecture.

Specialties:

- Nonlinear Systems Optimization Modeling
- Mathematical Modeling Software Design, Development and Marketing
- Service Oriented Architecture (SOA) Design & Development
- Grid & Cluster Computer Software Architecture
- Web Site Generation for Rapid Comprehension Software Documentation
- Robotic Financial Options Trading Software Design
- Open source software engineering expertise in Linux, Perl, Apache, Mason, JavaScript

Introduction

Parameter estimation can tweak one's parameters to optimum values but if one's company has no Statistical Process Control (SPC) or equivalent procedure for monitoring one's manufacturing process, those optimum parameter values will do you little good. So here are some suggested requirements to using parameter estimation:

- In-house Statistical Process Control monitoring implemented; and,
- Models must be continuously differentiable, i.e. no statistical models;

Parameter estimation with a Calculus-level compiler, can help tweak math model parameters in algebraic and differential equations. The equations may be linear, non-linear, explicit, implicit, constrained, etc. Differential equation problems include initial value problems (IVP), boundary value problems (BVP), or just curve fitting. The number of parameters that one can vary at any one time is constrained only by the size of computer used. Say you want to solve for 'n' parameters, then you would need a computer that can handle arrays of the size n^2. So even a Personal Computer (PC) can solve for 100s or even 1,000s of parameters.

A 'find' statement is the work horse of a Calculus language. It is used in parameter estimation, boundary value problems, implicit equation problems, inverse problems, etc.. The find statement's solver varies parameters in ones model until the stated goal is achieved. Different solvers use either the jacobian or Hessian matrix to estimate where to jump next with ones parameter values. The partials are calculated using 'automatic differentiation' (AD) and thus are as exact as one's computer.

Models must be continuously differentiable for AD to work right and calculate the right step size in parameter values.

Topics covered in this textbook include:

- **Lorentzian series Curve fitting**: if you need a model for some data, try a couple of Lorentz functions (see Applications 1.1.1 through 1.1.4).

  The normal power series does little for fitting real data well. Suggest dropping this series and replace it with a Lorentzian series for more practicality.

- **Sine series Curve fitting**: is hard to fit to real data with the normal solvers available today. Use of a spectral estimation program, e.g. spectrumSolvers, to find good starting frequency values makes it possible for a solver to converge to excellent frequency values and a good fit to data. Why not create a solver that does both estimate initial frequency values and fit other parameters to data? Application 1.2 & 1.3 have some cases to prove the point that it is possible.

- **Initial value problems** (IVPs) for ordinary differential equations (ODEs) only require an integrate statement; i.e. no find statement. IVP require the least amount of time to write and solve.

- **Boundary value problems** (BVPs) for ordinary differential equations (ODEs) have a find statement wrapped around the integrate statement. The find statement varies initial condition variables, e.g. ydot0, y2dot0, etc. and parameter fitting variables. By varying ydot & y2dot at t=0 you can find the best solution for ones ODEs given the boundary conditions.

- **Partial Differential Equations** (PDEs) will be solved by converting them into ODEs using method of lines or other method. PDEs may be non-linear, implicit, constrained, etc. Time to write a program will be in the hours thus saving many man hours of time. Start thinking of nesting ODE & PDE problems so you can simulate not just one part of a project but the whole project as will be discussed in chapter 10.

- **Nesting of find statements** is possible and thus one should think of solving many math models in one run. For example, an oil refinery has many distillation units each requiring a different PDE math model. All can be combined into one program where key parameters are tweaked until production goals are achieved.

The examples in this book are also included in the FC-Compiler application. Our Curvfit demo application is highly recommended to be installed on your PC for more Curve Fitting examples.

The key questions that you will be faced with when doing computer simulations are:

- **How good is your math model** for your data set at hand?

- **How well does your solver converge?**

- **How well are the parameters related to your problem?**

We will be discussing these issues as we go through the following examples. 'How good is your math model?' is always number one question.

How good is your math model?

Are you sure that all effects are accounted for? I have found that people comment more on 'bad' math models than on 'good' models. For example, what is the 'worst condition' versus 'best condition' for a forest fire? Asking about the 'worst' got more comments. People seemed to have more to say or were willing to say something regardless of their background.

Lacking a parameter?

Application Problems 1.1.1 & 1.1.2 math model lacked one parameter that was added for Application Problem 1.1.3. & 1.1.4 Finding lacking parameters in one's math model is not always easy. Those who know your field are often satisfied with the present working math model, so it's often hard to get them to think outside the box! If you feel that some parameter is missing, keep asking, keep searching … don't give up!

Any errors in model?

Oh those little minus signs in a math model, how easy it is to miss type one or two. Do you have any of those little things floating around in your work? When deriving a math model for the application titled 'Optimum Matched Filter (Transfer Function)', see Application Problem 2.1 below. A twenty-two hand written page document was used to derive the desired math model with all its parameters. Unfortunately a minus sign or two were dropped in the development. Fortunately another employee found the errors and corrected them. The smaller the computer code for a problem, the easier it is to find those little but important errors.

Another example of model errors was found in the SPICE computer simulation program from a University. The program had been in use for around ten years when six out of nine equations were found to be wrong or outdated. Too many hands working on it and not enough control on inserting modified equations.

Many of the examples in this book come from the disc drive industry of the 1980s. So the problems are real and the math models have meaning to those developing disc drives.

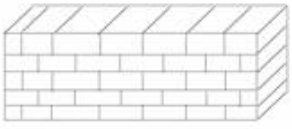
How to Improve Solver Convergence?

Normalize your equations so that your unknown parameter (absolute) values will be between .1 & 10. Removing large/small power calculations will help solvers converge to a solution; e.g. 10^6 & 10^{-6}. Initial values are thus either one or zero on the first run; future runs hopefully will have values between .1 and 10.

The Chapter 1 examples will be discussed in order to show there model's strengths. Where possible, the graphs shown are the data vs. model and error results. The best error plot can be seem in Application Problem 1.1, Figure 1.6b. The error is highly sinusoidal and has relatively low amplitude. Also, the Calculus-level code used to solve for their parameters will be shown and discussed.

One's Vision

Mr. Arithmetic



Mr. Algebra

Mr. Calculus

Before Computers	*With Computer, Gained some vision*	*Optimize the Whole Show in One Run*
***Process Methodology**:*		
One Step at a Time	Simulate Problem on Computer	Find Optimal Solution. Must 'See' Entire Problem & Objectives

We will be attempting to enlarge one's vision, especially in the last chapter, on future math models.

1 General Algebraic Equations

Application Problem 1.1

Introduction to Magnetic Recording

Curve fitting is the first type of parameter estimation problem that we will be discussing. We have two data sets retrieved from Magnetic Recording during the 1980s for what are called an Isolated (Readback) Pulse. Both these data sets had an 'okay' fit with model one, a Lorentzian series. Thus we tried to find a better math model, a modified model. This new model converged faster than the first model. Does this mean we should always use the new modified model?

Relating Model and Design Parameters

Assuming the digitized data fits a math model with quadratic convergence, how do the model parameters ($\bar{\mathbf{a}}$) relate to the design dimensions? For example, this present Thin-Film-Head (TFH) for disc drives example has the model parameters $\mathbf{v_i}$, $\mathbf{pw_50_i}$, and $\mathbf{t_i}$ (for i= 1 to 3) while the design parameters as shown in the following diagram are **A, B, C, D, E, & F**.

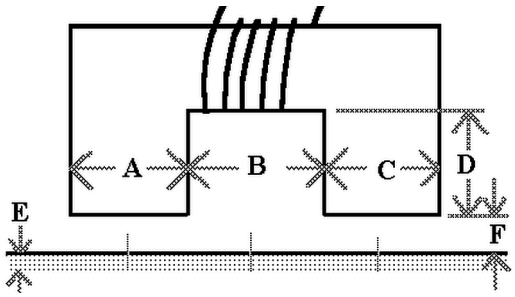


Figure 1.1 **A TFH at flying height F above a disc drive's media**

The governing equations may not be known for sure but someone with an understanding of the magnetic effects on a TFH could at least determine whether the parameters are proportional or inversely proportional. This would help as one starts building an understanding of what a math model might be in order to find the optimum design parameters to produce a symmetric and "narrow" (readback) pulse with no or minimal undershoots as represented in the curve shown below.

Optimum pulse shape?

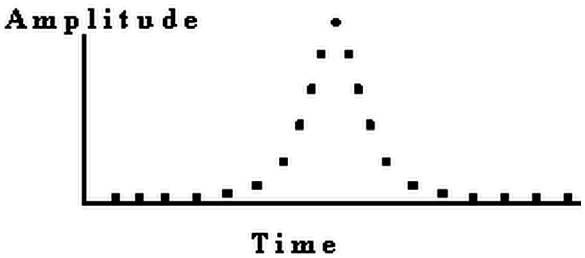


Figure 1.2 An "ideal" Readback Pulse from a disc drive

Through acquisition of many digitized pulses, with varying pulse model parameters will eventually provide the necessary design parameters for an optimum pulse. This would require many man-hours of time.

Background of TFH Math Model for a Readback Pulse from Magnetic Recording

Magnetic recording of transitions written onto a computer disc drive may produce an isolated pulse as shown below. This pulse comes from a disc drive's read-write channel. Each transition will cause such a signal to occur.

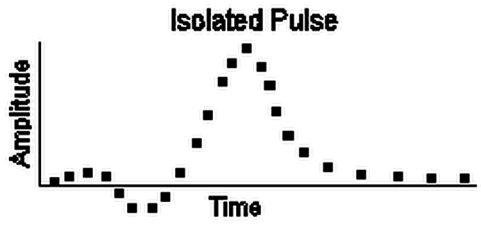


Figure 1.3 An isolated Readback Pulse from a 1980s disc drive

The signal's shape is very important to the electrical engineering development groups of disc drives. An isolated (readback) pulse should be symmetric and have a relatively fast rise time (i.e. sharp slope) for improved peak detection capability. A math model for the pulse can help gain insight into what electronic sub-system/components are causing the pulse to be asymmetric or have a slow rise time.

A Memorex physicist suggested that the longitudinal magnetic force was assumed the main contributing factor in determining a readback pulse shape, before the early 1980's. This force component was modeled by a series of three Lorentz functions. These functions have varying independent parameters that are dependent upon the drive's Thin-Film-Head (TFH) composition, size and shape. The values for these parameters were helpful in understanding a design and pinpointing any manufacturing flaws.

A Lorentz[1] function has represented/modeled an isolated readback pulse for some time. The basic Lorentz function is defined as $y=\frac{1}{1+x^2}$. The isolated pulse model is a composite of three Lorentz functions, called a **Lorentzian** series, as shown here:

$$signal_1(t)=\sum_{i=1}^{3}\frac{v_i}{1+\left(\frac{t-t0_i}{pw_50_i/2}\right)^2}$$

Equation 1.1 **Lorentzian Series**

where v_i = Amplitude of i^{th} Lorentzian pulse;

pw_50_i = Lorentzian pulse width, measured at 50% height of v_i; and,

$t0_i$ = Origin of the i^{th} Lorentzian.

In the early 1980s, this model was found to be inadequate when Thin-Film-Heads (TFH) were starting to be used in disc drives. An examination of the math model versus actual data plots (see Figure 1.5 & 1.6 below) showed that the 1970s model (Figure 1.4) was no longer sufficient. The longitudinal force, coupled with the increased vertical force, were used to provide an excellent model for TFH readback pulses (see Figures 1.6a below) in the mid 1980s. This math model we called a **Modified Lorentzian** series, $signal_2$, as shown here:

---

[1] **Hendrik Antoon Lorentz** (July 18, 1853 –February 4, 1928) was one of the greatest Dutch theoretical physicists. He was the second Nobel laureate in physics, together with Pieter Zeeman. They received the prize in 1902 for the discovery (by Zeeman) and the explanation (by Lorentz) of the Zeeman effect, the splitting of spectral lines in a magnetic field. Lorentz's main contribution to physics was in the theory of electromagnetism in which he continued and extended the work of the Scotsman James Clerk Maxwell.

$$signal_2(t)=\sum_{i=1}^{3}\frac{v_i+vc_i\left(\frac{t-t0_i}{pw_50_i/2}\right)}{1+\left(\frac{t-t0_i}{pw_50_i/2}\right)^2}$$

Equation 1.2 Mod. Lorentzian Series

where v_i = Amplitude of i^{th} longitudinal magnetic force;

vc_i = Amplitude of i^{th} vertical force component;

pw_50_i = Lorentzian pulse width, measured at 50% height of v_i; and,

$t0_i$ = Origin of the i^{th} Lorentz function.

Plots showing fit to a 1980s Thin-Film-Head data using Lorentzian & Modified Lorentzian series:

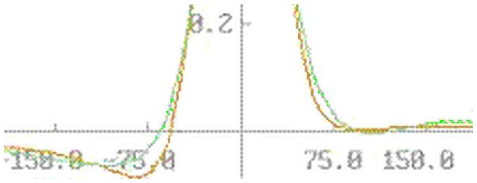


Figure 1.4a Lorentzian Series

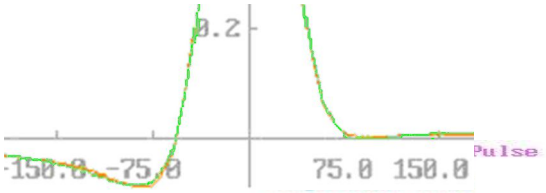


Figure 1.4b Mod. Lorentzian Series

Right math model?

How does one know when they have the right math model?

During the next 'head' generation development, Memorex acquired 200 isolated pulse datasets, each from a different head, and fit each one to see if our Modified Lorentzian series was good for all 200 heads. All fit well including the one in Figure 1.4c that shows a 'step' on its right side. This 'step' was due to the magnetic bars / logs at the tip of the head not being parallel to each other. Thus the Modified Lorentzian series became a new tool for finding defective heads. Note: Only a few orange spots (i.e. data points) are not hidden by green model points on this plot; a great fit!

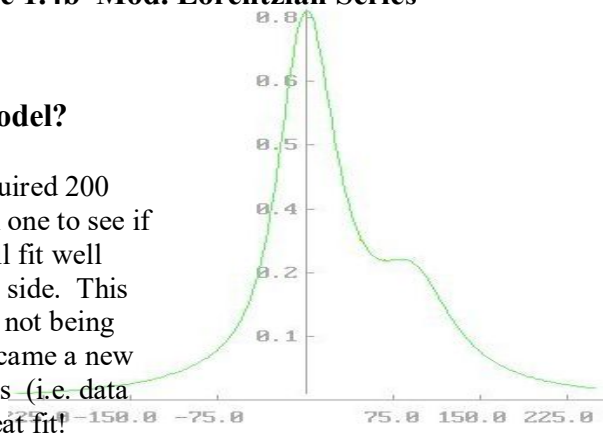


Figure.1.3c A perfect model to Odd data

Future TFH models

In **Figure 3.2 Solution to Lorentz ODE** it will be shown that model $y_2=\frac{1+vc*x}{1+x^2}$ is approximately equal to $y_1+vc_2*\frac{dy_1}{dx}$. Knowing that the signal derivative is part of the TFH signal may help give someone an understanding of the magnetic property that causes this and hopefully eventually get it removed from the TFH performance. Removing this effect would improve peak signal detection and thus save lost data on computers.

Parameter Estimation Problem

Find the parameter sets (v_i, pw_50_i and $t0_i$) and (v_i, vc_i, pw_50_i and $t0_i$) values necessary to fit the $Signal_1(t)$ and $Signal_2(t)$ models to a digitized isolated readback pulse. The Readrit?.?00 data files to curve fit are included in our FC-Compiler application, download at https://goal-driven.net/apps/fc-compiler.html.

Download: There is a freeware app available to try other CurvFit Math models.

Application Problem 1.1.1

A *Typical* Readback Pulse from Magnetic Recording

Problem Description

Magnetic recording of transitions written onto a computer disc drive may produce Figure 1.3. This pulse comes from a disc drive's read-write channel. Each transition will cause such a signal to occur.

A Lorentz function has represented/modeled an isolated readback pulse for some time. The 1970s isolated pulse model was a composite of three Lorentz functions, called a **Lorentzian** series, as shown here:

$$signal_1(t) = \sum_{i=1}^{3} \frac{v_i}{1 + \left(\frac{t - t0_i}{pw_50_i / 2} \right)^2}$$

Equation 1.3 Lorentzian Series

where v_i = Amplitude of i^{th} Lorentzian pulse;

pw_50_i = Lorentzian pulse width, measured at 50% height of v_i; and,

$t0_i$ = Origin of the i^{th} Lorentzian.

Computer Code

The FIND statement is the work horse of a Calculus-level compiler. It calls ones math model as many times as necessary in order to converge on a solution. It varies your parameters, in this case v, pw50 & t0, as it calls your math model. The '**in**' phase tells the name of math model routine to be used; the '**by**' phase tells what solver to use, Ajax here; and the '**to**' phase tells what the objective function is; '**match**' means all following variables must equal zero, 'sum' in this case.

FIND v, pw50, t0; IN pulse; BY AJAX; TO MATCH sum

```
graphics screen   ! real pulse from 1980s.
Global All
problem readrite
  call setup
  call pulse   ! Calc. array 'error' for plotting initial 'fit'
C   plot signal & data vs. time
  @aplot('rr-AJAX')

  Find v, pw50, t0; in pulse; by AJAX; to match sum
C   plot signal & data vs. time
  @aplot('rr-AJAX')
end
model pulse
  sum= 0
  do 20 j= 1, npoints
    call aLorentz( time(j), ampl)
    error(j)=ampl-data(j): sum=sum+error(j)**2
  20   continue
end
model aLorentz( t, ampl)     ! a Lorentz function
  ampl= 0
  do 10 i= 1, 3
    x= (t - t0(i)) / (pw50(i)/2)
    anum= v(i) – vc(i):     den= 1 + x**2
    ampl= ampl + anum / den
  10     continue
end
procedure setup
  real v(3), vc(3), t0(3), pw50(3)
  real data(100), time(100), error(100)
  npoints=100
  open(33, file= 'readrit1.100', status='old', err=99)
  do 20 j=1, npoints
    read(33,*) time(j), data(j)
20 continue
C initial values
  t0(1)=-40:  t0(2)=0:     t0(3)=100
  v(1)=-.05:  v(2)= .6:    v(3)= .1
  vc(1)= 0:  vc(2)= 0:    vc(3)= 0
  pw50(1)=70: pw50(2)=80:  pw50(3)=60
  return
99 write( 1, *) ' ---- Error ... Check Readrit1 data file --
--'
  stop
end
```

o o o (Download FC-Compiler 4 'readrite.fc' code)

Computer Plots

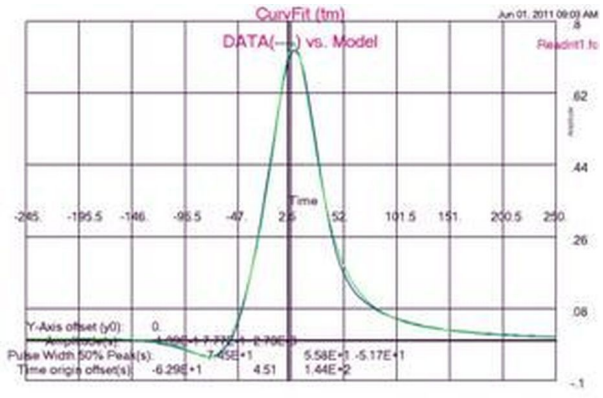


Figure 1.5a Data vs. Lorentzian Series

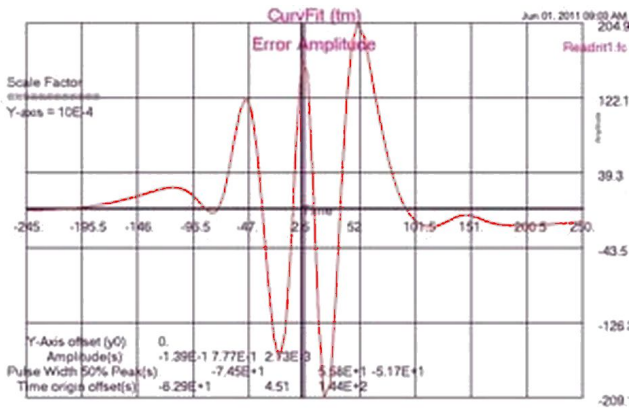


Figure 1.5b Lorentzian Fit Error Plot

Computer Output for AJAX Solver:

--- **AJAX** SUMMARY, INVOKED AT CURVEFIT[5] FOR MODEL PULSE ----

CONVERGENCE CONDITION AFTER 20 ITERATIONS
UNKNOWNS NOT CONVERGED
CONSTRAINTS UNSATISFIED
MAXIMUM ITERATIONS PERFORMED
SPECIFIED CRITERIA UNSATISFIED

LOOP NUMBER ...	[INITIAL]	1	2
UNKNOWNS			
V(1)	-5.000000E-02	3.731559E-02	-8.707396E-02
V(2)	6.000000E-01	5.912983E-01	7.650853E-01
V(3)	1.000000E-01	6.235979E-02	3.031518E-03
PW50(1)	7.000000E+01	4.953821E+01	-8.098887E+01
PW50(2)	8.000000E+01	5.518276E+01	5.464559E+01
PW50(3)	6.000000E+01	6.705979E+01	5.944982E+01
T0(1)	-4.000000E+01	-8.684141E+01	-1.332593E+02
T0(2)	0.000000E+00	7.892900E+00	4.476611E+00
T0(3)	1.000000E+02	1.005717E+02	1.065703E+02
OBJECTIVE			
||G|	8.040932E-01	6.043591E-01	3.843734E-01

o o o

LOOP NUMBER ...	[INITIAL]	19	20
UNKNOWNS			
V(1)	-5.000000E-02	-1.385877E-01	-1.385877E-01
V(2)	6.000000E-01	7.768310E-01	7.768310E-01
V(3)	1.000000E-01	2.725261E-03	2.725263E-03
PW50(1)	7.000000E+01	-7.446407E+01	-7.446407E+01
PW50(2)	8.000000E+01	5.583736E+01	5.583736E+01
PW50(3)	6.000000E+01	-5.171349E+01	-5.171346E+01
T0(1)	-4.000000E+01	-6.294041E+01	-6.294041E+01
T0(2)	0.000000E+00	4.507752E+00	4.507751E+00
T0(3)	1.000000E+02	1.437744E+02	1.437744E+02
OBJECTIVE			
||G||	8.040932E-01	6.970203E-02	**6.970202E-02**

---END OF LOOP SUMMARY

========== ErrSum= **0.069702** ==========

ELAPSED TIME= 22.69 SECONDS

Findings

The Lorentzian Series fit the 1980s Thin-Film-Head (TFH) data well except in the pre & post under shoots. Convergence was slow. An improved math model was achieved by adding another term to the model, i.e. $y_2= \frac{1+vc\cdot x}{1+a\cdot x^2}$. This improved model was called a Modified Lorentzian Series. It will be used to fit this data in the Application Problem 1.1 below.

Application Problem 1.1.2

An *Unusual* Readback Pulse from Magnetic Recording

Problem Description

This isolated (readback) pulse data set is unusual but helped solve a manufacturing problem when the model converged rapidly. The '$t0_3$' parameter was found far to the right of where it normally was found. Same math model used here as previous example.

Using Statistical Process Control one should be able to find these unusual TFHs and reduce the TFH standard deviation to insure readability of computer media regardless of the TFH doing the writing being different from the TFH reading the media.

Computer Code

Same as previous example except input data file changed to 'readrit2.200'.

Computer Plots

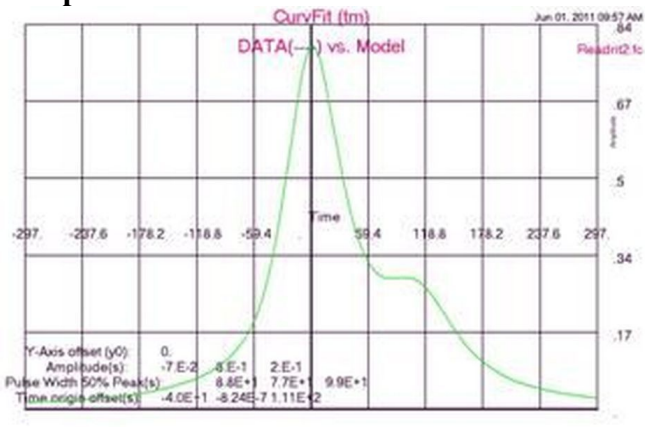


Figure 1.6a Data2 vs. Lorentzian Series

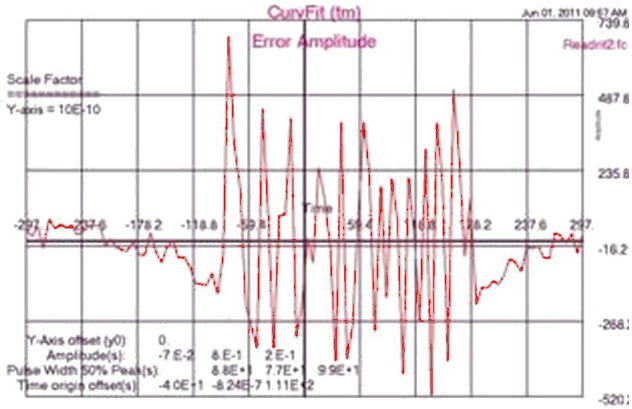


Figure 1.6b Lorentzian2 Fit Error Plot

Computer Output for AJAX Solver:

--- **AJAX** SUMMARY, INVOKED AT CURVEFIT[5] FOR MODEL PULSE ----

CONVERGENCE CONDITION AFTER 6 ITERATIONS
UNKNOWNS NOT CONVERGED
CONSTRAINTS SATISFIED
ALL SPECIFIED CRITERIA SATISFIED

LOOP NUMBER ...	[INITIAL]	1	2
UNKNOWNS			
V(1)	-5.000000E-02	-5.000000E-02	-3.974689E-02
V(2)	6.000000E-01	7.830934E-01	7.726742E-01
V(3)	1.000000E-01	1.804079E-01	1.971702E-01
PW50(1)	7.000000E+01	7.580762E+01	7.148666E+01
PW50(2)	8.000000E+01	7.313709E+01	7.429310E+01
PW50(3)	6.000000E+01	1.456453E+02	9.739544E+01
T0(1)	-4.000000E+01	-4.348545E+01	-4.912990E+01
T0(2)	0.000000E+00	1.839443E-01	5.267177E-01
T0(3)	1.000000E+02	1.136435E+02	1.081967E+02
OBJECTIVE			
||G||	8.178860E-01	1.300461E-01	4.984210E-02

ooo

LOOP NUMBER ...	[INITIAL]	5	6
UNKNOWNS			
V(1)	-5.000000E-02	-7.001261E-02	-7.000000E-02
V(2)	6.000000E-01	8.000095E-01	7.999999E-01
V(3)	1.000000E-01	1.999977E-01	2.000000E-01
PW50(1)	7.000000E+01	8.800782E+01	8.799999E+01
PW50(2)	8.000000E+01	7.700205E+01	7.700000E+01
PW50(3)	6.000000E+01	9.899785E+01	9.900000E+01
T0(1)	-4.000000E+01	-3.999659E+01	-4.000002E+01
T0(2)	0.000000E+00	-2.703632E-04	-8.238145E-07
T0(3)	1.000000E+02	1.110008E+02	1.110000E+02
OBJECTIVE			
||G|	8.178860E-01	1.550908E-05	**2.081722E-07**

---END OF LOOP SUMMARY

========== ErrSum= **0.000000** ==========

ELAPSED TIME= 20.93 SECONDS

Findings

Excellent rate of convergence! Nice error plot, Figure 1.6b, with relatively small amplitudes and sinusoidal! Parameter values seem reasonable. These results suggest a good math model.

Application Problem 1.1.3

A *Typical* Readback Pulse from Magnetic Recording with Improved Model

Problem Description

The longitudinal force, coupled with the increased vertical force, were used to provide an excellent model for TFH readback pulses in the mid 1980s. This math model is called a **Modified Lorentzian** series, $signal_2$, as shown here:

$$signal_2(t) = \sum_{i=1}^{3} \frac{v_i + vc_i\left(\dfrac{t - t0_i}{pw_50_i / 2}\right)}{1 + \left(\dfrac{t - t0_i}{pw_50_i / 2}\right)^2}$$

Equation 1.4 Modified Lorentzian Series

where v_i = Amplitude of i^{th} longitudinal magnetic force;

vc_i = Amplitude of i^{th} vertical force component;

pw_50_i = Lorentzian pulse width, measured at 50% height of v_i; and,

$t0_i$ = Origin of the i^{th} Lorentz function.

Computer Code

Same code here as last two examples with the addition of the 'vc' parameter; used 'readrit1.100' input file.

FIND **v**, **vc**, **pw50**, **t0**; IN pulse; BY AJAX; TO MATCH sum

Computer Plots

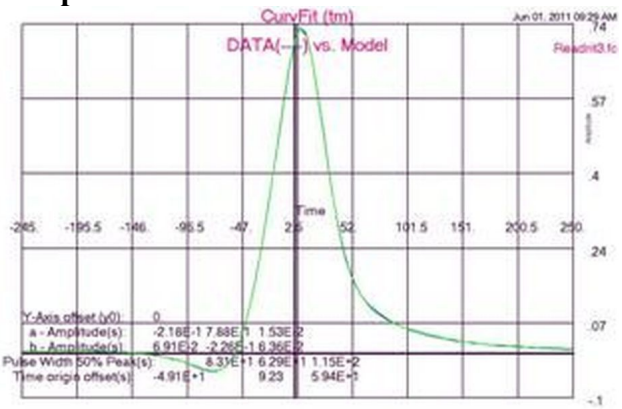


Figure 1.7a Data vs. Mod. Lorentzian Series

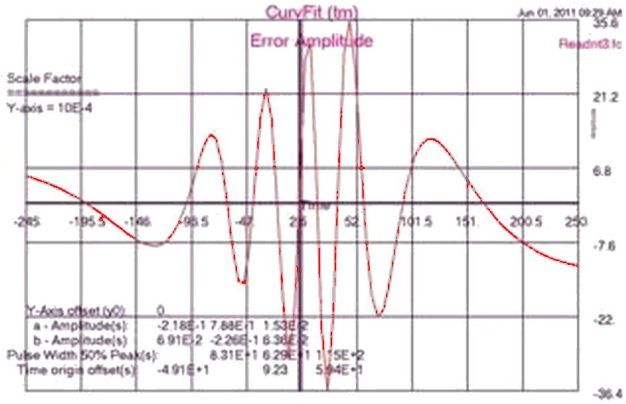


Figure 1.7b Mod. Lorentzian Fit Error Plot

Computer Output for AJAX Solver:

--- **AJAX** SUMMARY, INVOKED AT CURVEFIT[8] FOR MODEL PULSE ----

CONVERGENCE CONDITION AFTER 5 ITERATIONS
UNKNOWNS CONVERGED
CONSTRAINTS UNSATISFIED
ALL SPECIFIED CRITERIA SATISFIED

LOOP NUMBER ...	[INITIAL]	1	2
UNKNOWNS			
V(1)	-5.000000E-02	-1.381962E-01	-1.607859E-01
V(2)	6.000000E-01	7.551470E-01	7.206904E-01
V(3)	1.000000E-01	8.377075E-02	1.799287E-02
VC(1)	1.000000E-01	2.001194E-01	1.045891E-01
VC(2)	3.000000E-01	-1.549095E-01	-1.549095E-01
VC(3)	-1.000000E-01	6.449508E-02	2.308462E-02
PW50(1)	7.000000E+01	7.539570E+01	6.622947E+01
PW50(2)	8.000000E+01	4.315916E+01	5.626274E+01
PW50(3)	6.000000E+01	4.891829E+01	6.349559E+01
T0(1)	-4.000000E+01	-4.718956E+01	-4.813002E+01
T0(2)	0.000000E+00	9.886958E+00	9.161184E+00
T0(3)	1.000000E+02	8.846421E+01	8.549749E+01
OBJECTIVE			
‖G‖	1.262701E+00	4.839664E-01	7.244569E-02

O o o

LOOP NUMBER ...	[INITIAL]	5
UNKNOWNS		
V(1)	-5.000000E-02	-2.182126E-01
V(2)	6.000000E-01	7.879078E-01
V(3)	1.000000E-01	1.525311E-02
VC(1)	1.000000E-01	6.914184E-02
VC(2)	3.000000E-01	-2.256690E-01
VC(3)	-1.000000E-01	6.358915E-02
PW50(1)	7.000000E+01	8.310151E+01
PW50(2)	8.000000E+01	6.285273E+01
PW50(3)	6.000000E+01	1.150560E+02
T0(1)	-4.000000E+01	-4.910729E+01
T0(2)	0.000000E+00	9.232972E+00
T0(3)	1.000000E+02	5.936766E+01
OBJECTIVE		
‖G‖	1.262701E+00	**1.260813E-02**

---END OF LOOP SUMMARY

========== ErrSum= **0.012608** ==========

Findings

This Modified Lorentzian Series Model is definitely a better model than the Lorentzian Series shown in Application 1.1.1. Excellent rate of convergence! Figure 1.7b shows a nice error plot with relatively small amplitude and sinusoidal! Parameter values seem reasonable. These results suggest a good math model.

Application Problem 1.1.4

An *Unusual* Readback Pulse from Magnetic Recording with Improved Model

Problem Description

Both Lorentzian & Modified Lorentzian models worked for this data set. This isolated (readback) pulse data set is unusual but helped solve a manufacturing problem when the model converged rapidly. Thus saying the model was good. The $t0_3$ parameter was found far to the right of where it normally was found. A parameter out of normal range suggests a manufacturing error.

Using Statistical Process Control one should be able to find these unusual TFHs in manufacturing and reduce the TFH standard deviation to insure readability of computer media regardless of TFH writing being different from TFH reading media.

Computer Code

Same code used here as in last except added 'vc' parameter to 'find' statement as shown below; used 'readrit2.200' input file.

FIND v, vc, pw50, t0; IN pulse; BY AJAX; TO MATCH sum

Computer Plots

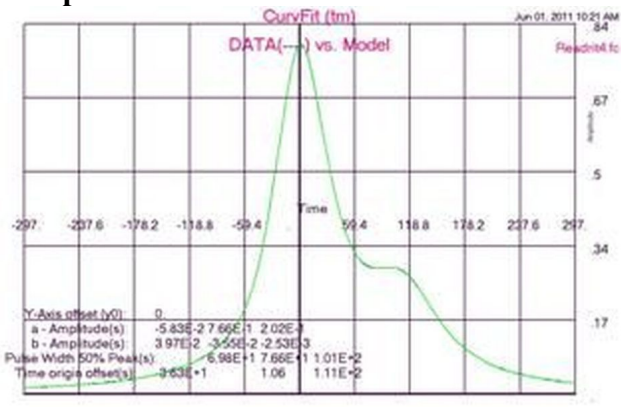


Figure 1.8a Data2 vs. Mod. Lorentzian Series

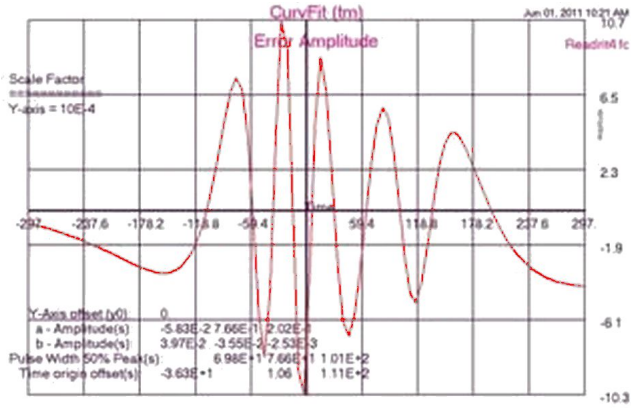


Figure 1.8b Mod. Lorentzian Fit2 Error Plot

Computer Output for AJAX Solver:

--- **AJAX** SUMMARY, INVOKED AT CURVEFIT[8] FOR MODEL PULSE ----

CONVERGENCE CONDITION AFTER 7 ITERATIONS
UNKNOWNS CONVERGED
CONSTRAINTS UNSATISFIED
ALL SPECIFIED CRITERIA SATISFIED

LOOP NUMBER ...	[INITIAL]	1	2
UNKNOWNS			
V(1)	-5.000000E-02	-9.348058E-02	2.624094E-02
V(2)	6.000000E-01	8.244893E-01	6.920350E-01
V(3)	1.000000E-01	2.412491E-01	2.337939E-01
VC(1)	1.000000E-01	2.178845E-03	-2.479759E-02
VC(2)	3.000000E-01	-2.658682E-02	5.868151E-02
VC(3)	-1.000000E-01	1.195240E-01	-7.941819E-02
PW50(1)	7.000000E+01	7.348288E+01	6.460946E+01
PW50(2)	8.000000E+01	7.830094E+01	6.563130E+01
PW50(3)	6.000000E+01	9.186359E+01	1.236225E+02
T0(1)	-4.000000E+01	-4.407704E+01	-5.687044E+01
T0(2)	0.000000E+00	8.737324E-01	-3.926429E-01
T0(3)	1.000000E+02	8.268578E+01	1.061227E+02
OBJECTIVE			
‖G‖	1.016135E+00	2.766644E-01	2.134743E-01

ooo **Good conversion … ‖G‖ dropping nicely each iteration!**

LOOP NUMBER ...	[INITIAL]	7
UNKNOWNS		
V(1)	-5.000000E-02	-5.832160E-02
V(2)	6.000000E-01	7.661447E-01
V(3)	1.000000E-01	2.022556E-01
VC(1)	1.000000E-01	3.971641E-02
VC(2)	3.000000E-01	-3.554289E-02
VC(3)	-1.000000E-01	-2.529758E-03
PW50(1)	7.000000E+01	6.979697E+01
PW50(2)	8.000000E+01	7.661058E+01
PW50(3)	6.000000E+01	1.011313E+02
T0(1)	-4.000000E+01	-3.629622E+01
T0(2)	0.000000E+00	1.058879E+00
T0(3)	1.000000E+02	1.109715E+02
OBJECTIVE		
‖G‖	1.016135E+00	**4.074845E-03**

========== ErrSum= **0.0040785** ==========

---END OF LOOP SUMMARY

Findings

This Modified Lorentzian Series model is as good a model as the Lorentzian Series shown in Application 1.1.2. Excellent rate of convergence! Nice error plot with relatively small amplitude and sinusoidal! Parameter values seem reasonable. These results suggest a good math model.

Based on this odd data set and 199 ‘normal’ TFH data sets, the Modified Lorentzian Model was accepted as the model for 1980s TFH data. Prior to this odd data set, manufacturing had an unknown error. It was believed the problem was due to the TFH magnets called logs. Every now and then these logs were thought to placed or grown where the logs were not parallel. This odd data set, with a great fit to the TFH math model, were key to pin-pointing this manufacturing problem.

Application Problem 1.2

Curve fitting: A Sinusoidal Signal

Problem Description

Fitting a sinusoidal series to data is a common problem. We will attempt to due this and discuss our findings. There are several numerical problems that one may incur. One problem in such a model is a term like $\mathbf{a_i}$ **sin(2 pi** $\mathbf{f_i}$ **t + theta)** that may cause a solver numerical difficulty when trying to find the next value for $\mathbf{f_i}$ parameter. If the parameter $\mathbf{a_i}$ is too small, any change in the $\mathbf{f_i}$ parameter will provide no change in the sin term. Thus, no change will occur in ones frequency parameter, $\mathbf{f_i}$. How does one get around this problem?

- Choose a relatively large values for sine amplitudes, $\mathbf{a_i}$, before your search starts; and,
- Use a spectral estimation program to find excellent starting frequencies value.

Computer Code

Added new parameters to find statement, ie. Amplitude, frequency, & theta, and changed solver to Jupiter. The objective function changed to MINIMIZE errsum

FIND ampl, freq, theta; **IN** pulse; **BY JUPITER**; **TO MINIMIZE** errsum

Computer Plots

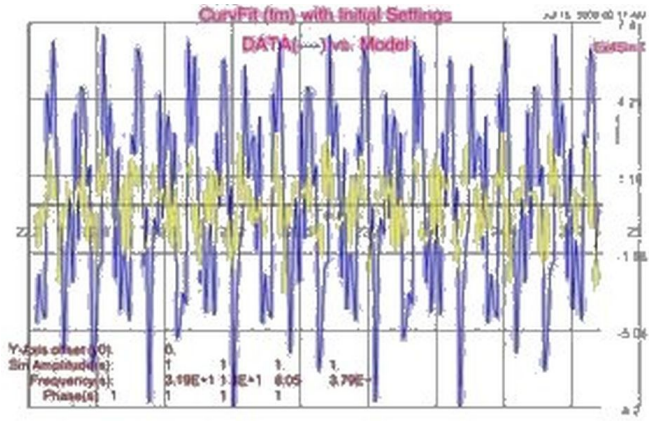


Figure. 1.9a Data3 vs. Initial Sine Series

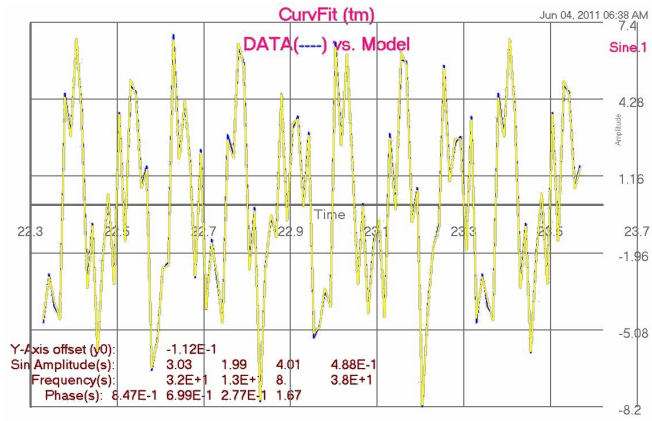


Figure 1.9b Data3 vs. Sine Series

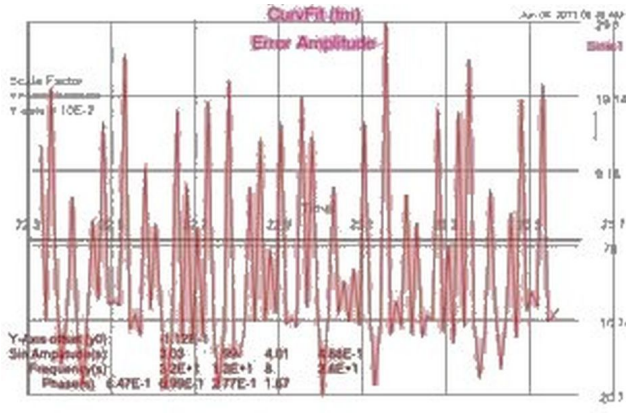


Figure 1.9c Sine Fit Error Plot

Computer Output for JUPITER Solver:

------ Plotting **initial parameters** --------
AS IS to give you a feeling for starting point

Y-Axis offset (y0):	0.0000			
Sine Amplitude (a):	1.000	1.000	1.000	1.000
Frequency (b):	32.00	13.00	8.000	38.00
Theta (c):	1.000	1.000	1.000	1.000

ErrSum= **118.8**

--- **JUPITER** SUMMARY, INVOKED AT FIT[21] FOR MODEL CURVE ----

CONVERGENCE CONDITION AFTER 1 ITERATIONS
OBJECTIVE CRITERION SATISFIED
ALL SPECIFIED CRITERIA SATISFIED

LOOP NUMBER ...	[INITIAL]	1
UNKNOWNS		
Y0	0.000000E+00	-1.118512E-01
A(1)	1.000000E+00	3.030360E+00
A(2)	1.000000E+00	1.991858E+00
A(3)	1.000000E+00	4.011098E+00
A(4)	1.000000E+00	4.883314E-01
Freq(1)	3.200000E+01	3.199882E+01
Freq(2)	1.300000E+01	1.299809E+01
Freq(3)	8.000000E+00	7.999652E+00
Freq(4)	3.800000E+01	3.799466E+01
Theta(1)	1.000000E+00	8.465198E-01
Theta(2)	1.000000E+00	6.991173E-01
Theta(3)	1.000000E+00	2.765086E-01
Theta(4)	1.000000E+00	1.671551E+00
OBJECTIVE		
ERRSUM	1.188445E+02	**2.210174E-01**

---END OF LOOP SUMMARY

========== ErrSum= **0.221017** ==========

ELAPSED TIME= 0.93 SECONDS

Findings

Sinusoidal curve fitting is hard to due. Initial values for the frequencies are key to finding a good fit. The first plot, Figure. 1.9a, was provided to show what one's starting conditions look like. In order to get good starting frequencies values, we executed the SpectrumSolvers program against our data set. SpectrumSolvers shows where key frequencies peak. We entered these peak locations in as starting frequency values and still had problems converging. (This frequency problem is due to the program calculating the next values by using derivatives. When changing a parameter value, a relatively small derivative value will suggest to the solver that no change will do any good. So it moves on to the next parameter. Frequency & Amplitude parameters are tied together. If frequency is ok but amplitude is too small, this too may cause frequency changes to be too small.)

Next, the solver was changed to Jupiter. This finally got good convergence as shown in the plots above.

Application Problem 1.3

Curve fitting: A Damped Sinusoidal Signal

Problem Description

A damped sinusoidal series is an extension of the last application. Here an exponential is added to each sin term like a_i sin(2 pi f_i t + theta) **exp(d_i t).** Now the d_i parameter must be found along with the others. Starting value of zero for all d_i is recommend. That tells a solver that you don't want this parameter unless necessary.

Computer Code

A new parameter, d, was added to others used without damping, i.e. Amplitude, frequency, & theta, and switched solver to Jupiter. The objective function remained the same. (see sinusoid.fc file in FC-Compiler application for code).

FIND ampl, freq, theta, d; **IN** pulse; **BY JUPITER; TO MATCH** error **And MINIMIZE** errsum

Computer Plots

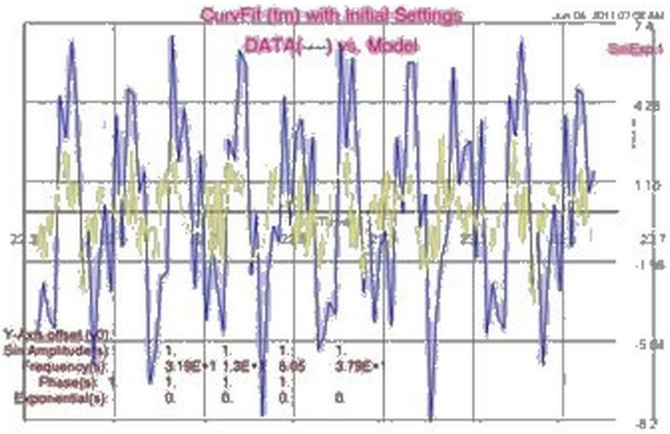


Figure 1.10a Data vs. Initial Damped Sine Series

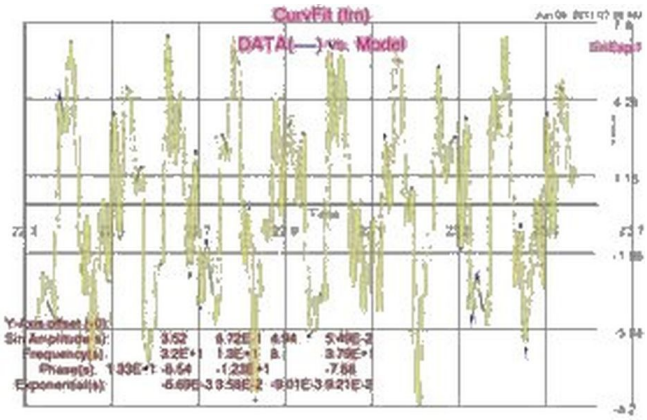


Figure 1.10b Data vs. Damped Sine Series

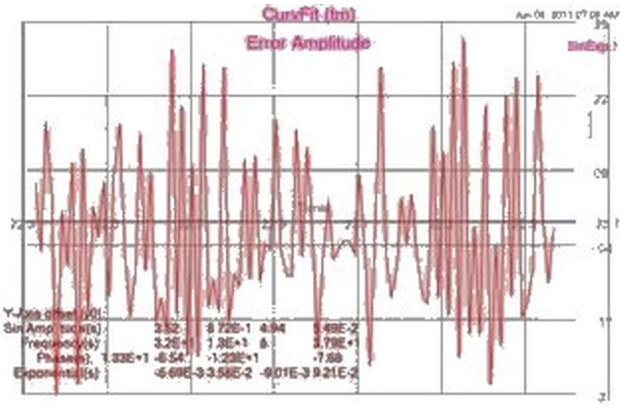


Figure 1.10c Damped Sine Fit Error Plot

Computer Output for JUPITER Solver:

~~~ AT EQU[72] OPERATION: Calculus Mode Assignment
*** OUT-OF-RANGE ARGUMENT TO EXP (I.E.-0.44E+04) IS REPLACED BY THE LIMIT (-0.10E+03)
~~~ AT EQU[72] OPERATION: Calculus Mode Assignment
*** OUT-OF-RANGE ARGUMENT TO EXP (I.E. 0.49E+04) IS REPLACED BY THE LIMIT (0.10E+03)

o o o **(many Out-of-Range) error stmts!!!**

~~~ AT EQU[72] OPERATION: Calculus Mode Assignment
*** OUT-OF-RANGE ARGUMENT TO EXP (I.E.-0.13E+03) IS REPLACED BY THE LIMIT (-0.10E+03)

--- **JUPITER** SUMMARY, INVOKED AT FIT[23] FOR MODEL CURVE ----

CONVERGENCE CONDITION AFTER 2 ITERATIONS
OBJECTIVE CRITERION SATISFIED
ALL SPECIFIED CRITERIA SATISFIED

LOOP NUMBER ...	[INITIAL]	1	2
UNKNOWNS			
Y0	0.000000E+00	-9.572788E-02	-9.572788E-02
A(1)	1.000000E+00	3.523554E+00	3.523554E+00
A(2)	1.000000E+00	8.715959E-01	8.715959E-01
A(3)	1.000000E+00	4.938423E+00	4.938423E+00
A(4)	1.000000E+00	5.492916E-02	5.492916E-02
Freq(1)	3.194670E+01	3.199966E+01	3.199966E+01
Freq(2)	1.304430E+01	1.300479E+01	1.300479E+01
Freq(3)	8.045260E+00	7.999537E+00	7.999537E+00
Freq(4)	3.788300E+01	3.788534E+01	3.788534E+01
Theta(1)	1.000000E+00	1.329229E+01	1.329229E+01
Theta(2)	1.000000E+00	-6.539949E+00	-6.539949E+00
Theta(3)	1.000000E+00	-1.227504E+01	-1.227504E+01
Theta(4)	1.000000E+00	-7.677276E+00	-7.677276E+00
D(1)	0.000000E+00	-6.694871E-03	-6.694871E-03
D(2)	0.000000E+00	3.577547E-02	3.577547E-02
D(3)	0.000000E+00	-9.009829E-03	-9.009829E-03
D(4)	0.000000E+00	9.207383E-02	9.207383E-02
OBJECTIVE			
ERRSUM	1.747659E+02	3.487218E-01	**3.487218E-01**

---END OF LOOP SUMMARY

========== ErrSum= **0.348721** ==========

ELAPSED TIME= 6.82 SECONDS

Findings

Excellent rate of convergence! Figure 1.10c shows a nice error plot with relatively small amplitude and somewhat sinusoidal! Parameter values are reasonable. This example's results suggest a good math model.

Do you have a better approach to this problem? Try it out with the FortranCalculus compiler.
~~~

1.4 Conclusion on Curve Fitting

Why curve fit data? It has the effect of

- Reducing number of data points to a few (hopefully) meaningful data points; i.e. parameter estimation values. These estimated values can then be compared with other estimated values. If some parameter values seem to stand out, then one might investigate as to why. This led a Memorex department to find a manufacturing flaw that had gone unsolved for a long time. (A major reason in forcing Memorex to fold.);
- Reducing noise in datasets; and,
- Helps compare datasets.

For example, say you have 100 channels that you want to compare. Take each channel and curve fit 20 to 100 points as a dataset every t_{step} units of time finding say n-parameter values for each dataset. Now plot these n-parameters on separate plots with parameter versus time. Are these plots a function of time and smooth curves? See any points that stick out? If so, are these points in error for some known reason? Work your way over all these plots and validate them by eye.

Next, compare the i^{th} and j^{th} channels. Do they seem to flow together as you would expect? Sometimes plotting the difference of channels may be helpful; e.g. the k^{th} parameter value of i^{th} minus j^{th} channels.

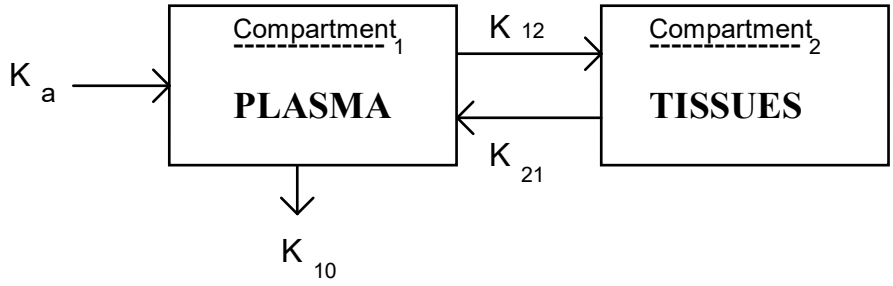
Their may be a lot of work here but sometimes you have no other option. Comparing parameters gives one an understanding of their system that others may understand; i.e. it improves communication.

Application Problem 1.5

Pharmacokinetics[2]

Problem Description

A Pharmacokinetics open-two-compartment model with first order absorption into elimination from central compartment (blood cleared of drug through the liver and/or kidneys) is presented here. The body tissues utilize the drug and therefore an amount is removed by the body's filtering system, i.e. the liver and/or kidneys. Given a dozen data points, find the parameter values to minimize 'sum' variable while limiting parameter values; i.e. a curve fitting application.



Rate of change in compartments is stated by the following differential equations:

$$\frac{dA_1}{dt} = K_a A_0 + K_{21} A_2 - K_{12} A_1 - K_{10} A_1 \qquad \text{Plasma compartment}$$

$$\frac{dA_2}{dt} = K_{12} A_1 - K_{21} A_2 \qquad \text{Tissue compartment}$$

where K_y represent Rate constants; y = a, 10, 12, and 21;
A_i = Amount of drug at the i^{th} site: 0. Absorption site; 1. Compartment 1; and 2. Compartment 2.

This system of differential equations can be solved analytically using La Place transforms. These solutions are usually expressed in terms of drug concentrations (i.e., parameters A, B, & C). The model equation for compartment 1 is

$$Cp(t) = A\, e^{-\alpha t} + B\, e^{-\beta t} + C\, e^{-K_a t}$$
$$C = -(A + B)$$

where Cp(t) is the plasma concentration at time t;
and α & β are hybrid parameters derived from K_{12}, K_{21}, K_{10}, and K_a.

The half-life of β is constrained to the range of three to nine years, and thus, adds the two constraints: Half_life $\geq$ 3 and $\leq$ 9 years where the Half_life = ln(2) / β

Relative error in this curve fitting problem was chosen due to the huge swing in amplitude over time.

Note: this can also be classified as an inverse problem: you know what you want, just don't know how to get there.

---

[2] Combs, D.

Computer Code

```
Global All
Problem Pharmaco  ! Kinetic parameters for open-two-compartment model
  dimension Time( 12), Plasma( 12), Error( 12), Half(2), aLows(5)
! Observed plasma concentrations ... Oral tablet of 10 mg
  data Time/0, .333, .5, .667, 1, 2, 4, 6, 8, 12, 24, 32/    ! X-Data
  data Plasma/1.e-4, .657, .727, .763, .695, .51, .307, .161,  & ! Y-Data
    .135, .046, .021, .008/ ! X-Units=Hr. & Y-Units=Mcg/Ml
  data aLows/ 5*0.D0/, Half/ 2*0.D0/
  Npoints = 12:   x = 1
!  Write(1,*) ' Enter Initial Starting Value ... '
 ! Read *, X
  A=X:  B=X:  aKa=X:  Alpha=X:  Beta=X     ! Initial Values
  Find A,B,aKa,Alpha,Beta; In Concentr; By Jupiter;  &
     With Lowers aLows; Holding Half; To Minimize Sum
End
Model Concentr   ! Concentration In Compartment 1
  Sum=0
  Do 10 i=1, Npoints
    T=Time(i)
    C1=A * Exp( - Alpha * T)
    C2=B * Exp( - Beta * T)
    C=-(A + B)
    C3=C * Exp( - aKa * T)
    Cpt=C1 + C2 + C3
    Error(i)=(Plasma(i) - Cpt) / Plasma(i)
    Sum=Sum + Error(i)**2
10  Continue
  Halflife=Log( 2) / Beta
  Half(1)=Halflife - 3    ! 3 Years Minimum
  Half(2)=9 - Halflife    ! 9 Years Maximum
End
```

Computer Output for JOVE Solver:

```
   Ooo

~~~ AT CONCENTR[22]  OPERATION: Calculus Mode Assignment
*** OUT-OF-RANGE ARGUMENT TO  EXP  (I.E.-0.10E+03) IS REPLACED
BY THE LIMIT (-0.10E+03)
~~~ AT CONCENTR[22]  OPERATION: Calculus Mode Assignment
*** OUT-OF-RANGE ARGUMENT TO  EXP  (I.E.-0.14E+03) IS REPLACED
BY THE LIMIT (-0.10E+03)


--- JUPITER SUMMARY, INVOKED AT PHARMACO[14] FOR MODEL CONCENTR ----


   CONVERGENCE CONDITION AFTER  1 ITERATIONS
     OBJECTIVE CRITERION SATISFIED
     ALL SPECIFIED CRITERIA SATISFIED


 LOOP NUMBER ...        [INITIAL]              1
 UNKNOWNS
   A                  1.000000E+00       8.955288E-01
   B                  1.000000E+00       1.071583E-01
   AKA                1.000000E+00       4.269202E+00
```

ALPHA	1.000000E+00	3.601325E-01
BETA	1.000000E+00	7.863350E-02
OBJECTIVE		
SUM	1.200000E+01	1.132438E+00
INEQUALITY CONSTRAINTS		
HALF(1)	-2.306853E+00	5.814910E+00
HALF(2)	8.306853E+00	1.850905E-01

---END OF LOOP SUMMARY

ELAPSED TIME = 0.11 SECONDS

Application Problem 1.6

Slack Variable Techniques

(From a PROSE manual)

Finding Feasible Solutions

Slack variable techniques can also be used to achieve initial feasibility (or maintain it) in inequality constrained nonlinear programs. Such applications generally lead to an underdetermined system of equations. It can be pointed out that underdetermined systems have no unique solution. Thus, the solution to a set of inequalities is merely the nearest of a set of points in a *feasible* subspace of $(x_1, ..., x_n)$ bounded by the inequality constraints. However, finding any point in this subspace, i.e., any feasible point, is an important initial step in constrained optimization, or may be employed as a substep of an unconstrained optimization search in order to solve a constrained optimization problem.

The following FC code can be used to solve the (slack) system of inequalities.

```
Problem Constraints
  n=???:  m=???:  allot x(n), z(m), g(m)
  for j = 1 to m,  z(j) = 1   ! Initial values
  Find x, z;  In Slack;  By Ajax;  To Match g
end
Model Slack
  execute Ineqls
  for j = 1 to m,  g(j) = g(j) - z(j)**2
end
Model Ineqls
  g(1) = G1(x1, x2, ..., xn) ! system of inequalities; e.g. x1 – 3*x2 > 0
  o
  o
  o
  g(m) = Gm(x1, x2, ..., xn)
end
```

In this code, the slack variables (z) are initialized to unity. This is not necessary in general, but for certain functions, and particular choices of initial guesses for the original n variables, the resulting Jacobian matrix may be exactly the zero matrix unless the slack variables are given non-zero initial values.

For example, consider the following system of inequality constraints:

$$\begin{aligned} (x_1 - 1)^2 + (x_2 - 1)^2 - 4 &\geq 0 \\ 4(x_1 - 2)^2 + 25(x_2 + 1)^2 - 100 &\geq 0 \\ x_1 - x_2 - .4 &\geq 0 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

Given an infeasible starting point, $x_1 = -100$, $x_2 = -150$, it is desired to find the nearest feasible point satisfying the inequalities. The FC program for this problem, a modification of the previous code, is shown below.

```
Global All
Problem Constraints
  execute .setup
  for j = 1 to m,  z(j) = 1
  Find x, z;  In Slack;  By Ajax( Knobs);  To Match
g
end
Model Slack
```

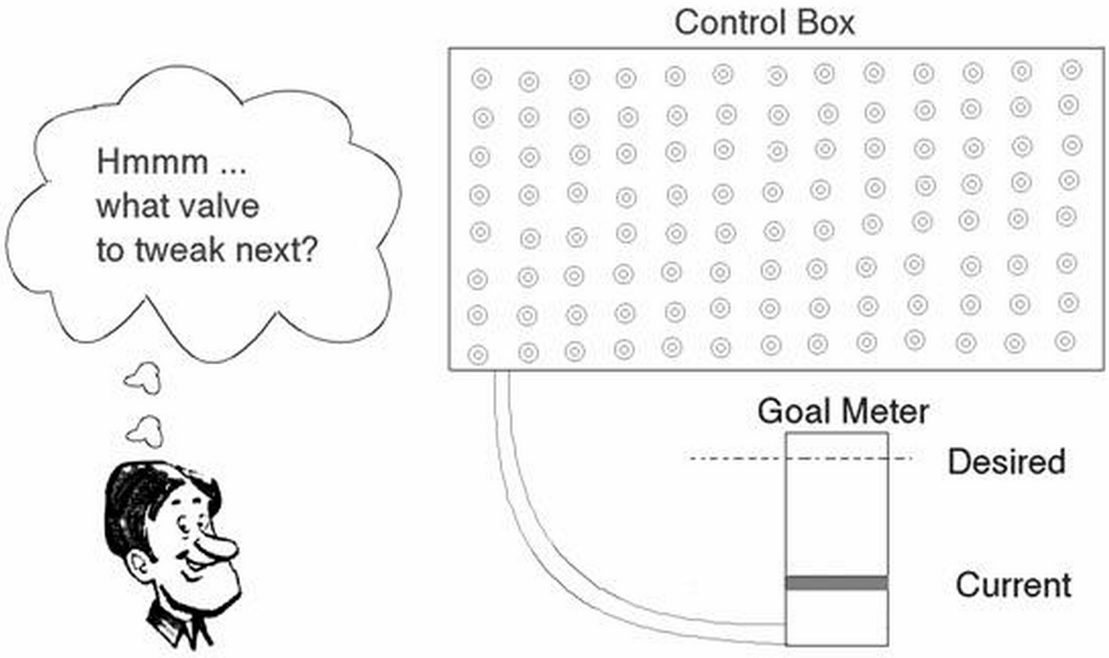
```
  execute .ineqls
  for j = 1 to m   g(j) = g(j) - z(j)**2
end
Model ineqls
  g(1) = (x(1) - 1)**2 + (x(2) - 1)**2 - 4
  g(2) = 4*(x(1) - 2)**2 + 25*(x(2) + 1)**2 - 100
  g(3) = x(1) - x(2) - .4
  g(4) = x(1)
  g(5) = x(2)
End
Controller Knobs   For AJAX
  converge = 2      maxit = 40
end
Procedure Setup
  m = 5
  allot x(2), z( m), g( m)
  x = .data( -100, -150)
End
```

Application Problem 1.7

Paper Bicycle Design

(Objective-Driven Design Example)

What's your present project's goal/objective?

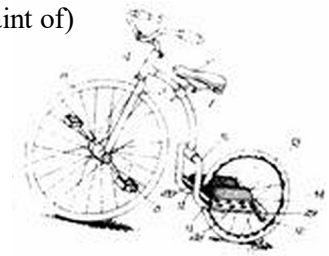


Are you sure? Why?

What is the ideal or desired objective/goal? Are you sure? Why?

A class in Mechanical Engineering at Stanford requires its students to design a Paper Bicycle. A student's grade is dependent on their bicycles lap times, total weight, and a penalty when their non-paper parts weight is more than 10% of their total weight. The lower the score the better for ones grade!

What would be the design objective given each 3 member team has (a time constraint of) 2 weeks to build a team, design, test, and race their paper "bicycle"?



1. Minimize the design weight;
2. Maximize each individual's class grade;
3. Minimize the total lap times, i.e., win the race!
4. Minimize cost of design project;
5. Minimize time required for design
6. _____

Assume a design team chose the "win the race" objective. A math model or analysis for their design might be as follows:

TotalTime < 5.4321 minutes, since this was last years champions time.

TotalDistance = 900 Meters

RPM = 60, a "reasonable" average rate for cyclist on given course

DistPerRev = Circumference of drive wheel = π Diameter

Therefore, a Diameter great than TotalDistance/(π RPM TotalTime) will ensure a time faster than last year's champions, assuming it holds together throughout the race. For this case, the

Diameter = 900/(π * 60 * 5.4321)
= .88 Meters or larger

What type of paper "bicycle" will perform with the desired drive wheel? A two wheeler is out due to cost and time constraints. A wheel chair option is out due to the required drive wheel must average 60 rpm, thus, leaving a three wheel cycle or "Hot Wheeler" similar to what young kids ride, today.

Given the **Win** objective, what parameters or reasoning will determine the following:

1. Maximum drive wheel size;
2. Maximum surface friction between the drive wheel and race track;
3. Remaining geometry and size of bike components.

Race results: They won! The **Win** team had a well stated objective target, not to shoot at, but calculate and achieve the required design through math modeling, analysis, and good reasoning. Generally, an objective-driven design will yield the best design in the least amount of development/manufacturing time and expense.

Chapter 1 Exercises

1. What can a user do to increase accuracy of a solution when entering data for a problem? See 'readrit1.100' File Listing for ideas.
2. Constant values, e.g. π, should be calculated instead of entered whenever possible in order to keep overall calculations as accurate as possible. What equation could be used to calculate π?
3. Curve fit data file 'readrit1.100' with below series and same parameters as in applications 1.1.1. Do your problems converge to a solution? Is the error plot sinusoidal? Try another solver or two to see if things improve.
    a. Expand the Lorentz function to $1/(1 + x^2 + x^4)$ and curve fit it to the data. Thus, $signal_1$ becomes

$$signal_1(t) = \sum_{i=1}^{3} \frac{v_i}{1 + \left(\frac{t - t0_i}{pw_50_i / 2} \right)^2 + \left(\frac{t - t0_i}{pw_50_i / 2} \right)^4}$$

    See Equation 1.3 Lorentzian Series for simplified version and parameter definitions.

    b. Use a series of exponentials, e^{-x^2} and same parameters as above.
    c. Other functions: your turn to find a function that would fit the data; e.g. sine series, or power series or ???.
4. Curve fit data file 'readrit2.200' with below series and same parameters as in applications 1.1.2. Do your problems converge to a solution? Is the error plot sinusoidal? Try another solver or two to see if things improve.
    a. Use an expanded Lorentz function to $(1+x)/(1 + x^2 + x^4)$. Thus, $signal_2$ becomes

$$signal_2(t) = \sum_{i=1}^{3} \frac{v_i + vc_i \left(\frac{t - t0_i}{pw_50_i / 2} \right)}{1 + \left(\frac{t - t0_i}{pw_50_i / 2} \right)^2 + \left(\frac{t - t0_i}{pw_50_i / 2} \right)^4}$$

See Equation 1.4 Modified Lorentzian Series for simplified version and parameter definitions.

    b. Use a series of exponentials, $(1+x)e^{-x^2}$ and same parameters as above.
    c. Other functions: your turn to find a function that would fit the data; e.g. sine series, or power series or ???.
5. Assume you are responsible for determining what thin-film-heads (TFHs) are good in a production line. Over one hundred TFHs are produced a day. Using 'readrit1.100' results as good, develop a logic function for a pass/fail test. The function should be dependent of some or all of the parameters found in the above problems. Assume a 10% variance of 'readrit1.100' parameters is still okay but more than that should fail your test. Test 'readrit2.200' parameters to be sure it fails your test.

2 La Place Transforms

A Parameter Estimation for LaPlace Transforms in a Calculus-level **'Find' statement** is shown here:

FIND a ooo To Match Error

Where 'a' may be a vector with 'n' parts, a_1, a_2, a_3,…a_n;
'error' is the objective function.

The 'a' parameter(s) are varied to fit one's 'm' data points that make up the objective function, error. This technique can vary as many parameters as you want; e.g. 5 or 50 or 50,000. If there are less equations than parameters m < n, this would be classified as an under-determined system of equations. If there are more equations than parameters, m > n, this would be an over-determined system. Under- or Over-determined systems might force one to switch solvers to do the job.

Application Problem 2.1

Optimum Matched Filter (Transfer Function)

(Nested Processes ... Each Process controlled by a Solver)

Problem Description

The transfer function H(s) is the Laplace transform of the output signal $Y_{out}(s)$* divided by the Laplace transform of the input signal $Y_{in}(s)$*: that is $H(s)=\frac{Y_{out}(s)}{Y_{in}(s)}$ where each signal's transform is assumed to be a ratio of polynomials. Thus, H(s) can likewise be stated in the form:

$$H(s) = \frac{a_0 + a_1 s + a_2 s^2 + \ldots + a_m s^m}{b_0 + b_1 s + b_2 s^2 + \ldots + b_n s^n}$$

Equation 2.1 Generalized H(s)

Assuming the numerator and denominator can be factored, yields H(s) in the general form

$$H(s) = \frac{a_m (s - Z_1)(s - Z_2)\ldots(s - Z_m)}{b_n (s - P_1)(s - P_2)\ldots(s - P_n)}$$

Equation 2.2 Factored Transfer Function

where each Z_i is known as a "zero" and the P_i as a "pole" of the transfer function. Z_i and P_i are complex points in the Laplace domain.

A realizable transfer function must have poles and zeros with their conjugate points. That is, poles and zeros come in pairs. If a pole or zero is located at the complex point $\sigma_i + j\omega_i$, then its conjugate is located at $\sigma_i - j\omega_i$. Thus, a generalized transfer function is stated as

$$H(s) = gain \frac{(s - Z_\sigma)(s + Z_\sigma) \prod_{p=1}^{Z_Pairs} \{(s - Z_p)(s - Z_p^*)\} \prod_{q=1}^{Z_Quads} \{(s - Z_q)(s - Z_q^*)(s + Z_q)(s + Z_q^*)\}}{(s - P_\sigma) \prod_{i=1}^{P_Pairs} \{(s - \sigma_i)^2 + \omega_i^2\}}$$

Equation 2.3 Generalized Transfer Function H(s)

Given n-data points from a Bode plot (see Figure 2.1 below) that define the mainlobe of the desired transfer function, find the optimal Pole/Zero constellation such that H(s) has equal sidelobe peak amplitudes in a Bode plot and curve fits the given data in the mainlobe.

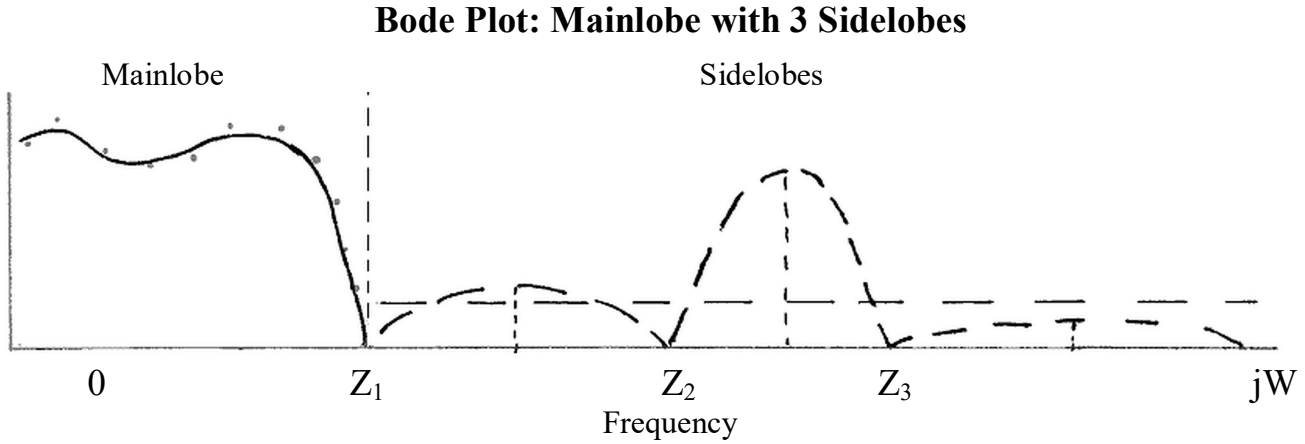


Figure 2.1 H(s) Mainlobe & Sidelobe Plot

To help view what's going on, think of the LaPlace domain covered with a rubber matt and pinned on its corners. From the under side, in the 'mainlobe' area, place one's poles (i.e. push up the rubber matt at these locations). This should give the impression of a hill or mountain. Place your zeros on the frequency axis by pressing down from the top at your zero locations (i.e. z_1, z_2, &z_3 points as shown above on the bode plot). Now if you cross cut the rubber matt on the frequency axis and you should have the bode plot above.

The objective is to keep the mainlobe 'mountain' while moving your zeros in order to get equal peak heights in your sidelobes. Sounds simple but a slight movement in those zeros changes the peak heights radically. See Figure 2.2b and note how far down these peaks are in amplitude: farther down, less noise in system.

Computer Plots

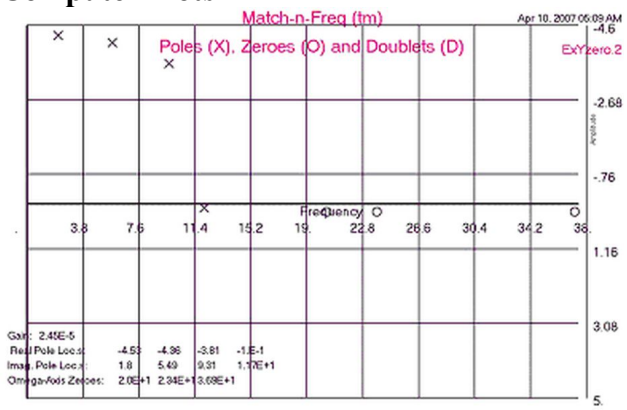


Figure 2.2a H(s) Pole/Zero Locations

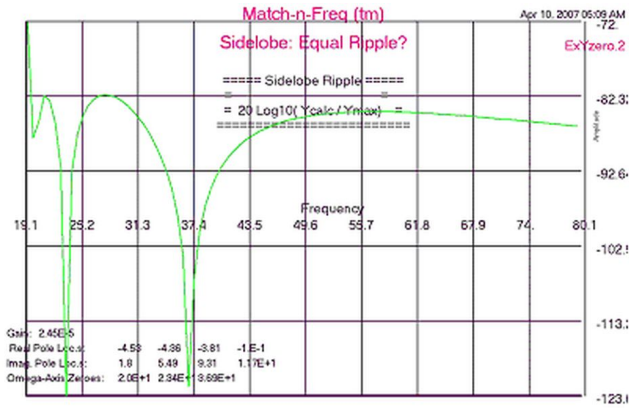


Figure 2.2b Equal Peaks in Sidelobes

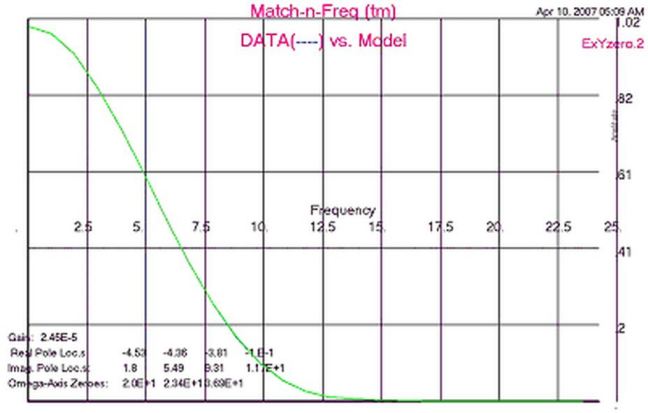


Figure 2.2c Data vs. H(s) Model Curves

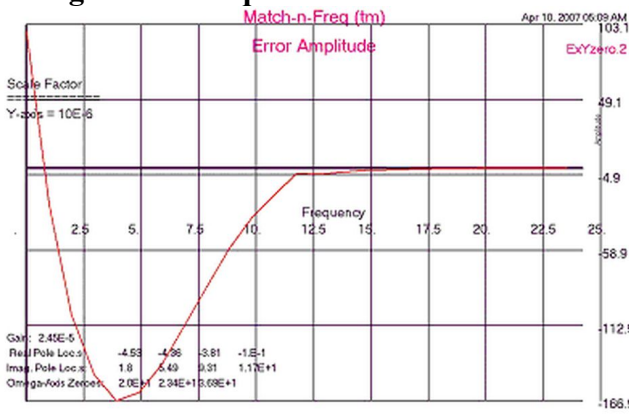


Figure 2.2d H(s) Fit Error Plot

Computer Code

The first FIND statement is used to find good values for mainlobe parameters **gain, p.real, p.imag**

FIND gain, p.real, p.imag; IN Laplace.Domain; BY AJAX; TO MATCH error

With good mainlobe parameters, then this above find statement executes two nested Find statements to find the sidelobe parameters.

FIND x.zeros **IN** .Stopband **BY HERA** **WITH BOUNDS** side.limits **TO MINIMIZE** peak.diff

```
Global All
 PROBLEM FILTER(40000, 5000, 5000)  ! Match-n-
Freq (tm)
C ---------------------------------------------
C --- FORTRANCALCULUS Application: Find
Pole/Zero Constellation of a ---
C    Matched Filter Transfer Function  ---
C ---------------------------------------------
 call input
C ---- Find Pole-Zero Locations ----
 nYzeros=0:  call fit   ! Don't vary yZeros yet.
 Yzeros8n= 1.6 * ylmax
 nYzeros= 3:  call fit   ! Now vary yZeros
 call output
end
model fit  ! Minimize Magnitude fit Error
C - Varying Gain & Pole/Zero locations -
 n= 1 + (nXpole+2*nPpairs) + nXzero +
2*(nZpairs+nZquads)
 allot h8low( n):   <h8low>= xlmin:  h8low(1)= 0
 allot h8hi( n):   <h8hi>= ylmax:  h8hi(1)= -1

 FIND gain,Xzero, Preal, Pimag;  &
  in Transfer;  by JOVE(contrl1);  &
  with lower h8low;  and uppers h8hi;  &
  MATCHING error; TO MINIMIZE errsum

end
```

```
model Transfer
  errsum= 0.D0
  do 50 ii= 1, npoints  !  --- CALCULATE TRANSFER
FUNCTION ----
    Y2=freq(ii)**2:  hw=xfunct(Y2)
    error( ii)= gain * hw * y8in( ii) - y8out( ii)   !
Absolute Error
    error(ii)= error(ii)/(y8out( ii)**ierrtyp)   ! Relative
    errsum= errsum + error( ii)**2
50  continue
  if( nYzeros .gt.2) call sideArea
end
Fmodel xfunct( Y2)
  real*8 num
  num= 1:    den= 1
  do 20 ij= 1, nPpairs
    den= den*factor(Y2,-Preal(ij), Pimag( ij))
20  continue
  if( nYzeros .gt. 0) then
    num= 100 * num
    do 40 ij= 1, nYzeros
      num= num * factor( Y2, 0., Yzeros(ij))
40  continue
  endif
  q= num / den
  if( q .gt. 1.D20) q= 1.D20
  xfunct= q
end
Fmodel factor( y2, sigma, omega)
  r2= sigma**2
  if( omega .eq. 0.D0) then
    factor= 1:        if( sigma .eq. 0.D0) return  ! not sure
on value
    factor= (y2 + r2) / r2  ! R2 normalizing factor
    return
  endif
  o2= omega**2:  sum=(r2+o2+y2)/10
  temp= sum*sum - 4*y2*o2/100:  factor= 0
  if( temp .eq. 0.D0) return
  temp= sqrt( temp)
  factor= temp / (r2 + o2)  ! this R2+O2 is 4
normalizing pole values
end  ! and adjusts Gain value for system.

model sideArea
  n1= nYzeros-1
  do 20 ij= 1, n1
    if(Yzeros(ij).ge.Yzeros(ij+1)) then
      tmp=Yzeros(ij):  Yzeros(ij)= Yzeros( ij+1)
      Yzeros( ij+1)= tmp
    endif
20 continue
  do 30 ij= 2, n1
    sidelims(ij-1)=Yzeros(ij+1)- Yzeros( ij)
    peakloc(ij)=(Yzeros( ij+1) + Yzeros( ij))/2  *.95
30  continue
  peakloc( 1)= (Yzeros( 2) + Yzeros( 1))/2  *.95
  peakloc( nYzeros)= (ylmax + Yzeros( nYzeros))/ 1.5
  sidelims( n1)= ABS( ylmax - Yzeros( nYzeros))
  do 40 ij= 1, n1
    sidelims( ij)= sidelims( ij)*ij/(nYzeros * 5)
    Yzeros2( ij)= Yzeros( ij+1)
40 continue

  FIND Yzeros2;  in stopband;  by Hera( contrl2); &
    with BOUNDS sidelims; & ! BANDLIM;
    TO MINIMIZE diff

  do 50 ij= 1, n1
    Yzeros( ij+1)= Yzeros2( ij)
    if( Yzeros(ij) .ge. Yzeros( ij+1)) then
      tmp= Yzeros( ij):  Yzeros( ij)= Yzeros( ij+1)
      Yzeros( ij+1)= tmp
    endif
50  continue
end
model stopband
  do 50 jj= 2, nYzeros
    Yzeros( jj)= Yzeros2( jj-1)
50 continue
  diff= 0:   sidelim= .02
  do 60 jj= 1, nYzeros
    ipeak= jj
    if( jj .gt. 1) then
      sidelim= sidelims( jj-1)
    endif

   FIND peakloc( jj);  in sidelobe;  by hera( contrl3); &
     with BOUNDS sidelim; &
     TO MAXIMIZE peakampl( jj)

    diff= diff + slope( jj)**2
    if( jj .gt. 1) then
      anorm= peakampl( jj)**2 + peakampl( jj-1)**2
      diff= diff + (peakampl( jj) - peakampl( jj-1))**2 /
anorm
    endif
60 continue
  anorm= peakampl( 1)**2 + peakampl( nYzeros)**2
  diff=diff+(peakampl(1)-peakampl( nYzeros))**2 /
anorm
  diff= diff * 1.d6
  peakave= 0:   errsumpk= 0
  do 70 jj= 1, nYzeros
    peakave=peakave+dabs(peakampl( jj))
 70 continue
  do 80 jj= 1, nYzeros
    peakerr(jj)=peakave - peakampl( jj)
    errsumpk=errsumpk+(peakerr(jj) / anorm)**2
80 continue
end
model sidelobe
  peakampl(ipeak)=sideAmpl( peakloc( ipeak))
  ampl1=sideAmpl(.9999*peakloc(ipeak))
  ampl2=sideAmpl(1.0001*peakloc( ipeak))
```

```
  slope(ipeak)=1.D6*(ampl1-ampl2)/ .0002   ! Slope
approx.
end
Fmodel sideAmpl( Y)
  Y2= Y * Y:  sideAmpl= xfunct( Y2)
end
controller contrl1( JOVE)
  remax=maxit(1):  detail=ireport(1):  zero=ch8tol(1)
  stepslim=limsteps:  stepout=stepout2:
evalmax=maxeval
  accuracy=accurcy
end
controller contrl2( Hera)
  remax=maxit(2):  detail=ireport(2):
progress=ch8tol(2)
  adjust= 1:  summary= 1
end
```

Computer Output for JOVE & HERA Solvers:

--- **JOVE** SUMMARY, INVOKED AT FIT[63] FOR MODEL FITBOTH ----

```
  CONVERGENCE CONDITION AFTER  5 ITERATIONS
    MODEL EVALUATION LIMIT EXCEEDED
    OBJECTIVE CRITERION UNSATISFIED
    MAXIMUM ITERATIONS PERFORMED
    SPECIFIED CRITERIA UNSATISFIED

 LOOP NUMBER ...       [INITIAL]            1                2
 UNKNOWNS
  GAIN                1.230000E-02      2.426167E-05     2.452567E-05
  PREAL( 1)           2.000000E-01      7.549617E-01     7.562047E-01
  PREAL( 2)           2.000000E-01      7.577069E-01     7.562061E-01
  PREAL( 3)           2.000000E-01      7.559795E-01     7.562061E-01
  PREAL( 4)           2.000000E-01      5.001370E-03     5.017036E-03
  PIMAG( 1)           1.000000E-02      5.002709E-03     5.033684E-03
  PIMAG( 2)           1.000000E-01      5.000923E-03     5.011472E-03
  PIMAG( 3)           2.000000E-01      5.000739E-03     5.009188E-03
  PIMAG( 4)           3.000000E-01      3.000000E+00     2.999494E+00
 OBJECTIVE
  ERRSUM              2.246747E+04      1.386396E+01     1.386382E+01
 EQUALITY CONSTRAINTS
  ERROR( 1)           4.664578E-03      1.812689E-06     1.912965E-06
  ERROR( 2)           3.036344E-03     -5.624263E-07    -4.925001E-07
  ooo
  ERROR(25)          -1.157844E-18     -1.084009E-18    -1.083120E-18

  ooo

 LOOP NUMBER ...       [INITIAL]            5
 UNKNOWNS
  GAIN                1.230000E-02      2.451293E-05
  PREAL( 1)           2.000000E-01      3.134831E-01
  PREAL( 2)           2.000000E-01      6.712762E-01
  PREAL( 3)           2.000000E-01      4.565040E-01
  PREAL( 4)           2.000000E-01      5.250445E-02
  PIMAG( 1)           1.000000E-02      3.674824E-01
  PIMAG( 2)           1.000000E-01      4.785724E-02
  PIMAG( 3)           2.000000E-01      5.398629E-03
  PIMAG( 4)           3.000000E-01      7.671278E-01
 OBJECTIVE
  ERRSUM              2.246747E+04      7.207860E-01
 EQUALITY CONSTRAINTS
  ERROR( 1)           4.664578E-03      1.908127E-06
  ERROR( 2)           3.036344E-03     -4.879579E-07
  ooo
```

```
  ERROR(25)          -1.157844E-18      -1.154290E-18

---END OF LOOP SUMMARY

  ooo

---- HERA SUMMARY, INVOKED AT SIDEAREA[180] FOR MODEL STOPBAND ----

  CONVERGENCE CONDITION AFTER  2 ITERATIONS
    UNKNOWNS CONVERGED
    OBJECTIVE CRITERION UNSATISFIED
    ALL SPECIFIED CRITERIA SATISFIED

 LOOP NUMBER ...      [INITIAL]          1                  2
 UNKNOWNS
  YZEROS2(1)        1.500000E+00     1.525001E+00     1.525000E+00
  YZEROS2(2)        2.200000E+00     2.060394E+00     2.060395E+00
 OBJECTIVE
  DIFF              5.476185E+18     8.586592E+06     1.890314E+06

---END OF LOOP SUMMARY

---- HERA SUMMARY, INVOKED AT SIDEAREA[180] FOR MODEL STOPBAND ----

  CONVERGENCE CONDITION AFTER  3 ITERATIONS
    UNKNOWNS CONVERGED
    OBJECTIVE CRITERION UNSATISFIED
    ALL SPECIFIED CRITERIA SATISFIED

 LOOP NUMBER ...      [INITIAL]          1                  2
 UNKNOWNS
  YZEROS2(1)        1.525000E+00     1.483725E+00     1.483717E+00
  YZEROS2(2)        2.060395E+00     1.958406E+00     1.958428E+00
 OBJECTIVE
  DIFF              2.453228E+16     2.538071E+06     1.887044E+06

 LOOP NUMBER ...      [INITIAL]          3
 UNKNOWNS
  YZEROS2(1)        1.525000E+00     1.483717E+00
  YZEROS2(2)        2.060395E+00     1.958429E+00
 OBJECTIVE
  DIFF              2.453228E+16     1.885848E+06

---END OF LOOP SUMMARY

  O o o

--- JOVE SUMMARY, INVOKED AT FIT[63] FOR MODEL FITBOTH ----

  CONVERGENCE CONDITION AFTER  5 ITERATIONS
    OBJECTIVE CRITERION UNSATISFIED
    MAXIMUM ITERATIONS PERFORMED
    SPECIFIED CRITERIA UNSATISFIED

 LOOP NUMBER ...      [INITIAL]          1                  2
 UNKNOWNS
  GAIN              2.451838E-05     2.451843E-05     2.451824E-05
  PREAL( 1)         1.903761E-01     1.900577E-01     1.898417E-01
```

PREAL(2)	2.174091E-01	2.170316E-01	2.167918E-01
PREAL(3)	2.261904E-01	2.258044E-01	2.255600E-01
PREAL(4)	6.035538E-03	6.026419E-03	5.945594E-03
PIMAG(1)	4.643045E-01	4.642716E-01	4.643402E-01
PIMAG(2)	2.736201E-01	2.736099E-01	2.736497E-01
PIMAG(3)	8.978034E-02	8.977338E-02	8.978452E-02
PIMAG(4)	5.907978E-01	5.906571E-01	5.902591E-01
OBJECTIVE			
ERRSUM	1.604998E-01	1.604787E-01	1.604562E-01
EQUALITY CONSTRAINTS			
ERROR(1)	1.910196E-06	1.910216E-06	1.910143E-06
ERROR(2)	-4.901276E-07	-4.901482E-07	-4.901328E-07
ooo			
ERROR(25)	-1.165042E-18	-1.165045E-18	-1.165047E-18

ooo

LOOP NUMBER ...	[INITIAL]	5
UNKNOWNS		
GAIN	2.451838E-05	2.451764E-05
PREAL(1)	1.903761E-01	1.904776E-01
PREAL(2)	2.174091E-01	2.177901E-01
PREAL(3)	2.261904E-01	2.266766E-01
PREAL(4)	6.035538E-03	5.000000E-03
PIMAG(1)	4.643045E-01	4.656372E-01
PIMAG(2)	2.736201E-01	2.742334E-01
PIMAG(3)	8.978034E-02	8.995914E-02
PIMAG(4)	5.907978E-01	5.862881E-01
OBJECTIVE		
ERRSUM	1.604998E-01	1.596411E-01
EQUALITY CONSTRAINTS		
ERROR(1)	1.910196E-06	1.909913E-06
ERROR(2)	-4.901276E-07	-4.890354E-07
ooo		
ERROR(25)	-1.165042E-18	-1.165045E-18

---END OF LOOP SUMMARY

---------- ExYzero.1

Resulting Parameters in De-normalized form:

Pole-pairs

	PREAL	PIMAG
1.	-3.809552E+00	9.312744E+00
2.	-4.355801E+00	5.484667E+00
3.	-4.533533E+00	1.799183E+00
4.	-1.000000E-01	1.172576E+01

Zeros

Zeros on Omega-axis: 0 +- 20.00 29.67 39.17

ELAPSED TIME= 5.99 SECONDS

Findings

The resulting plots told the story according to the theory & practical application of the time. The results suggested a good to excellent solution but we wanted better results. Converting the transfer function, H(s), to the time domain, h(t), is were this project ended. h(t) was achieved but had some problem finding the y.out(t) output signal peaks. The following equations will define this problem in the time domain.

Future

Next, find a good math model for $y_{out}(t)$ using a series of Lorentzian curves as done in **Application Problem 1.1**.1. Calculate a Pattern-Induces-Bit (PIB) shift given a bit stream of zeroes and ones with some time spacing. For example, a pattern of 3T-8T-3T-8T-3T etc. where T = 20.8 (ns) with a series of bits. The bits alternate their polarity for each '1' bit. This will generate a sinusoidal wave. Once satisfied that the PIB shifts are accurate, put this PIB shift program around the matched filter program above. The new program should have a FIND statement to find the number of poles (i.e. 'nXpole' & 'nPpairs') necessary to minimize the PIB shift. (After working some on the time domain approach, it seems that the zeroes do not need to be requested. If they are necessary, ones input & output function, $y_{in}(t)$ & $y_{out}(t)$, will bring them into play. See **Equation 2.4** or **Equation 2.5** for more on this.)

Minimizing the PIB shift should be the overall design objective for a read-write channel for a disc drive. If the PIB shift is to high, the data or bit pattern written will not be able to be retrieved.

A project objective is very important to say the least. Get your team to agree on one and keep it short; just a few words e.g. minimize this or maximize that. Here is where Programming Calculus really shines. If an objective changes over time, just change it in your model and rerun the problem. Without Programming Calculus one may have been playing with a numerical method. A change in objective could force one to practical start over; loosing months of time.

Download: There is a freeware app available for making your own Matched Filters.

***Note** Error in Yin & Yout calculations when digital:

The best way to determine Yin & Yout functions is to find their desired functions in the time-domain (if digital data, then approximate function with a good Curve fit routine) and then calculate their (analog) Fourier transforms.

Yin is an 'isolated readback pulse' created with some type of disc drive head. Capture this signal digitally and then curve fit it using CurvFit with a series of Lorentzian pulses.

Yout is a desired signal with some desired features; e.g. 'thin' pulse, no pre- nor post-undershoots, etc. A Lorentzian pulse is what we choose (I think).

Application Problem 2.2

Inverse Problem: Optimum Matched Filter

This above matched filter design is an algebraic problem but it also is an inverse problem where one knows what they want, just need to find a way to get there. The following plot shows what we wanted, in time the domain, now we need to find the right h(t) to produce this signal.

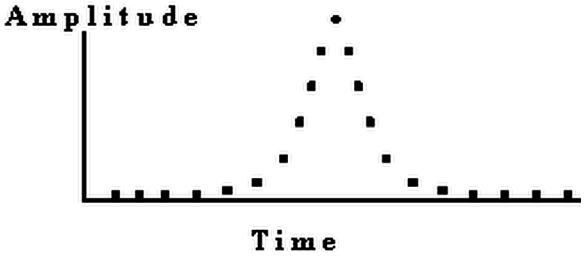


Figure 2.3 An "ideal" Readback Pulse for disc drives

Here is the development of h(t). After partial fraction expansion, H(s) from **Equation 2.3** can be written as

$$H(s) = \frac{A_o}{(s - P_\sigma)} + \sum_{i=1}^{P_Pairs} \frac{A_i(s - \sigma_i) + B_i\omega_i}{\{(s - \sigma_i)^2 + \omega_i^2\}}$$

(For a causal system) the impulse response, h(t), of the transfer function, H(s), is the inverse LaPlace transform of H(s). Thus, h(t) is stated as a sum of damped sine & cosine functions:

$$h(t) = A_0 e^{P_{\sigma 0} t} + \sum_{i=1}^{P_Pairs} e^{\sigma_i t}\left[A_i \cos(\omega_i t) + B_i \sin(\omega_i t)\right]$$

$$= A_0 e^{P_{\sigma 0} t} + \sum_{i=1}^{P_Pairs} e^{\sigma_i t}\left[C_i \sin(\omega_i t + \theta_i)\right]$$

Equation 2.4 Impulse Response, h(t)

This gives us $y_{out}(t) = y_{in}(t) * h(t)$ where '*' means convolvution operation and assume $y_{in}(t)$ as the 'readrit1.100' data file from Equation 1.1 **Lorentzian Series** with npoints = 100. Using discrete convolvution we obtain $y_{out}(t)$ as the following:

$$y_{out}(i) = \sum_{k=1}^{npoints} y_{in}(k)\ h(i - k) \quad \text{for i = 1 to npoints.}$$

Equation 2.5 Discrete output function

Filter Summary

Thus, h(t), requires a complex sinusoidal curve fitting procedure as shown in Application Problem 1.3. Things to note between h(t) & H(s) findings include:

1. For h(t) only the number of Poles are required by user … no zeroes, the curve fitting procedure will determine if any zeroes are necessary.
2. h(t) has no errors due to Fourier Transforms of $y_{in}(t)$ & $y_{out}(t)$; and,
3. Only one Find statement … no nesting as required for finding H(s) zeroes.

The frequency domain approach was used and the resulting pole-zero constellation put into an electrical circuit. The summary was published and presented at a conference; see *Arbitrary Equalization with Simple LC Structures* in appendix. We used a pole removal software program to determine the electrical circuit

components. The order of pole removal was key to building the circuit. There was more than one order of pole removal that worked, thus, more than one circuit to test. Somewhere along the line, the frequency domain approach became undesirable. The time domain approach we never got too (company went under!). On paper it does look the best.

Chapter 2 Exercises

1. Lets determine which method is more accurate the Frequency or Time calculations?

   Frequency Calculations:

   a. Lets make sure we are comparing apples to apples, so Calculate the Fourier Transform of our $y_{in}(t)$ math model, see **Equation 1.4 Modified Lorentzian Series** for math model and parameters found in **Application Problem 1.1**.

   b. Make a new updated 'readrit1.100' file by copying it to a file named 'readrit3.100'. Next replace the frequency data values with correct values from your Fourier Transform calculated above. Rerun **Application Problem 2.1** using this new 'readrit3.100' file. Save the output as 'readrit3.*' files.

   c. Calculate $y_{out}(t)$ by performing an inverse Fourier Transform on Yout(f) array.

   Time Calculations:

   d. Desired Objective? Finding σ_i, ω_i, & θ_i parameters in

$$= A_0 e^{P_{\sigma 0} t} + \sum_{i=1}^{P_Pairs} e^{\sigma_i t}\left[C_i \sin\left(\omega_i t + \theta_i\right)\right]$$

   is alot simpler than solving them in **Equation 2.3** as stated in **Application Problem 2.1** if you have a good 'error' definition. What would be your objective function (i.e. 'error' definition) in order to find the various poles & zeroes for h(t)?

   Hint: The desired output signal, $y_{out}(t)$ (see **Figure 2.3**), is defined in the time domain as a symmetric and slimmed pulse with minimum undershoots.

   e. Find pole (σ_i, ω_i, & θ_i) parameters in

$$= A_0 e^{P_{\sigma 0} t} + \sum_{i=1}^{P_Pairs} e^{\sigma_i t}\left[C_i \sin\left(\omega_i t + \theta_i\right)\right]$$

   f. by writing code similar to the code used in **Application Problem 1.3**.

   Compare Results:

   g. Compare $y_{out}(t)$ from Frequency & Time calculations by calculating the difference between them in the time domain and plot the difference. Is this difference plot sinusoidal in nature?

   h. Compare the pole locations by sight and using the Time pole locations as starting values for another run in **Application Problem 2.1**. Does it converge to same old pole locations, stay right where the start, or what? What does this say about the two methods; i.e. are they equal? If not, what may be the problem, e.g.truncation?

2. If the desired output signal, $y_{out}(t)$ (see **Figure 2.3**), in the time domain is to be a symmetric and a slimmed pulse, what might the h(t) function shape be or equivalent to?

3. How will the h(t) function in the convolvution $y_{out}(t) = y_{in}(t) * h(t)$ guarantee symmetry?

4. What $y_{in}(t)$ amplitudes will guarantee a slimmed $y_{out}(t)$ pulse?

5. Curve fit h(t) in

$$= A_0 e^{P_{\sigma_0} t} + \sum_{i=1}^{P_Pairs} e^{\sigma_i t}\left[C_i \sin\left(\omega_i t + \theta_i\right)\right]$$

to math model achieved in **Application Problem 1.1** with V, Vc, PW50, & T0 parameters. Find h(t) parameters C_i, σ_i, ω_i, & θ_i with P_Pairs = 4 for a good fit.

6. If your objective function consists of matching 'npoints' to $y_{out}(t)$, and h(t) has 'p_pairs' of poles (& no P_σ pole), how large must 'npoints' be in order to be classified as an over-determined system of equations?
7. The Fourier Transform enters several errors into calculating $y_{out}(t)$ / $y_{in}(t)$. The first error is due to computer truncating each value to n-digits; e.g. 10 digits per word or value. They should be infinite in length (in theory). What other errors do Fourier Transform enter into the calculations? How might these errors be minimized?

3 Ordinary Differential Equations

A Parameter Estimation for an Ordinary Differential Equations (ODEs) in an Initial Value Problem (IVP) or Boundary Value Problem (BVP) is solved using the Calculus-level **'Find' statement** shown here:

IVP: **Find a** ooo To Match Error

BVP: **Find a, ydot0, y2dot0** ooo To Match Error

Where 'a' is a vector with 'n' parts, a_1, a_2, a_3,…a_n;
ydot0, y2dot0,etc. are derivatives at independent variable = 0; and,
'error' is the objective function.

The 'find' statement is wrapped around an integrate and integration statements in order to solve the ODE while finding the best 'a' parameter(s) for the given problem.

The 'a' parameter(s) are varied to fit one's 'm' data points that make up the objective function, error. This technique can vary as many parameters as you want; e.g. 5 or 50 or 50,000. If there are less equations than parameters, m < n, this would be classified as an under-determined system of equations. If there are more equations than parameters, m > n, this would be an over-determined system. Under- or Over-determined systems might force one to switch solvers to do the job.

The 'integration' statement sets up the integrate statement. Its 'equations' phase shows the order of equation variables; i.e. y3d/y2d, y2d/y1d, etc is saying that 'y3d' = the derivative of 'y2d' and 'y2d' = the derivative of 'y1d', etc.

Application Problem 3.1

Second Order Non-Linear ODE

(Under- or Over-determined Systems)

Problem Description

An n^{th} order non-linear ODE in an Curve fitting Problem may be solved as shown in this application.

Computer Code

The 'Find' statement is the work horse of a Calculus-level compiler. It calls ones math model as many times as necessary in order to converge on a solution. It varies your parameters (a, y0, & ydot0) as it calls your math model. The '**in**' phase tells the name of math model routine. The '**by**' phase tells what solver to use, Jove here. And the '**to**' phase tells what the objective function is; '**minimize**' means all following variables must be as small as possible, error variable in this case.

FIND a, y0, ydot0; IN ODE-xCos; BY JOVE; TO MINIMIZE error

```
  graphics screen   ! A second order ODE
  global reals
  problem ODE-xCos(6000, 3000, 3000)
    dimension xp(100), yp(100), ydotp(100),
y2dotp(100)
    xp(1)=.5:  xp(2)=1:  xp(3)=1.5:  xp(4)=2
    xp(5)= 2.5:   xp(6)= 3
    yp(1)=.4390:  yp(2)=.5400:  yp(3)=.1060
    yp(4)=-.8320: yp(5)=-2.0: yp(6)=-2.97
    a= 1           ! initial values
    x0=xp(1):  y0=1:  ydot0=1   ! initial conditions
    points= 6:   npoints= points
    print *, ' '
    print *, 'Starting search for parameters to minimize
|error|'
    print *, ' '
    FIND a,y0,ydot0; IN someODE; BY JOVE; &
      TO MINIMIZE error
C   plot y solution vs. x
    x0= xp(1):   xfinal= 10
    points=100:   npoints= points
    deltaX=(xfinal-x0)/npoints
    do 10 i= 1, npoints
      xp(i)= x0 + (i-1)*deltaX
10  continue
    call someODE
    @aplot('rr-AJAX')
  end
  model someODE
    y= y0:   ydot= ydot0:  x= x0:  dx= .01
    initiate ISIS;  for diffeqs; &
      equations y2dot/ydot, ydot/y; &
      of x;  step dx;  to xfinal
    npoints= points:   error= 0
    do 10 i= 1, npoints
      xfinal= xp(i)
      integrate diffeqs; by isis
      error= error + (yp(i) - y)**2
      if( npoints .eq. 100) then
        yp(i)=y:  ydotp(i)=ydot:  y2dotp(i)= y2dot
      end if
10  continue
    terminate diffeqs
  end
  model diffeqs
    y2dot=2*ydot/x-(1+ a/x**2)*y  ! 2nd order non-
linear ODE
  end
  procedure aplot( plot77)

ooo  ! (See 'aplot' code in appendix.)
```

Computer Plots

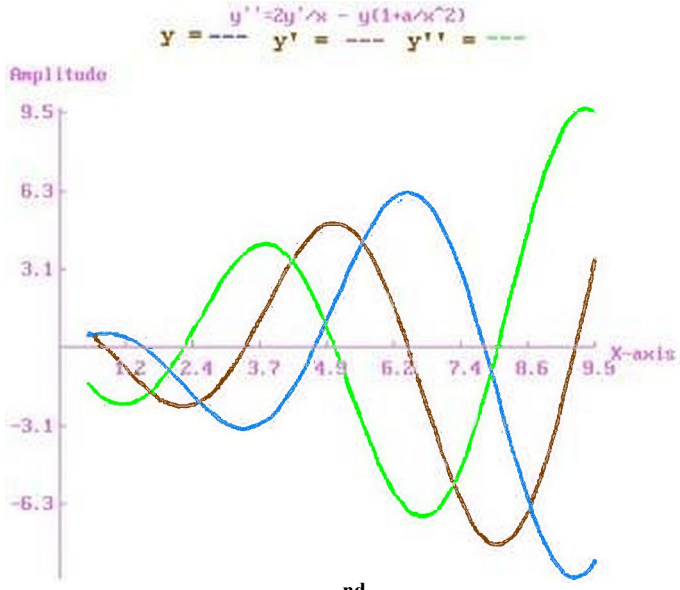


Figure 3.1 Solution Plot for 2nd order differential equation

Computer Output for JOVE Solver:

Starting search for parameters to minimize |error|

JOVE STEP 5 OF ITERATION 1
OBJECTIVE= 2.910987E-02 PENALIZED OBJECTIVE= 2.910987E-02
AFTER 122 CUMULATIVE EVALUATIONS OF SOMEODE
INDEPENDENT VARIABLES
4.113128E-01 5.603355E-01 1.728102E-01

JOVE STEP 10 OF ITERATION 1
OBJECTIVE= 2.007301E-03 PENALIZED OBJECTIVE= 2.007301E-03
AFTER 235 CUMULATIVE EVALUATIONS OF SOMEODE
INDEPENDENT VARIABLES
1.759555E+00 4.657767E-01 5.927012E-01

--- **JOVE** SUMMARY, INVOKED AT ODE[26] FOR MODEL SOMEODE ----

CONVERGENCE CONDITION AFTER 1 ITERATIONS
OBJECTIVE CRITERION SATISFIED
ALL SPECIFIED CRITERIA SATISFIED

LOOP NUMBER ...	[INITIAL]	1	
UNKNOWNS			
A	1.000000E+00	1.978276E+00	
Y0	1.000000E+00	4.454773E-01	
YDOT0	1.000000E+00	6.399512E-01	
OBJECTIVE			
ERROR	3.388487E+00	**8.211294E-05**	! Excellent convergence

---END OF LOOP SUMMARY

x	y	y'	y''
0.50000	0.44548	0.63995	-1.4108
0.59500	0.49907	0.49577	-1.6214
0.69000	0.53797	0.33263	-1.8092
0.78500	0.56051	0.15288	-1.9704
0.88000	0.56524	-4.07979E-02	-2.1019
0.97500	0.55099	-0.24545	-2.2011
1.0700	0.51686	-0.45790	-2.2658

ooo

9.7150	-9.1746	1.7705	9.7313
9.8100	-8.9592	2.6938	9.6925
9.9050	-8.6564	3.6090	9.5597

ELAPSED TIME= 18.73 SECONDS

Findings

Nice and quick convergence for parameter 'a'.

Application Problem 3.2

A Third Order Non-Linear ODE

Problem Description

The non-linear ordinary differential equation

$$d^3y/dx^3 = 3 * (dy/dx * d^2y/dx^2 + dy/dx **2 / x) / y$$

was found knowing that the solution is a Lorentz function; i.e. worked backwards from solution to ODE. The Lorentz function has small y values on its left & right side that make it a stiff ODE to solve numerically. In order to help get pass this stiffness problem the integration was limited from -50 to +50 thus truncating its side tails or ramps (see Lorentz figure below).

Computer Code

The 'Find' statement is the work horse of a Calculus-level compiler. It calls ones math model as many times as necessary in order to converge on a solution. It varies your parameters, in this case y0, ydot0, & y2dot0, as it calls your math model. The '**in**' phase tells the name of math model routine. The '**by**' phase tells what solver to use, ajax here. And the '**to**' phase tells what the objective function is; '**match**' means all following variables must equal zero, error variable in this case.

FIND Alpha; IN Lorentz; BY AJAX; TO MATCH error

```
  graphics screen   ! A 3rd order ODE
  global reals
  problem Lorentz(6000, 3000, 3000)
    real xp(100),yp(100),ydotp(100), y2dotp(100),
y3dotp(100)
    points= 1:  npoints= points:  xp(1)= 0:  yp(1)= 1
    y3dot= 1e-5   ! initial values
    x0= -50:  y0= .03:  ydot0= .001:  y2dot0= 7e-5  !
initial conditions
    print *, ' '
    print *, 'Starting search for parameters to minimize
|error|'
    print *, ' '

    FIND y0,ydot0,y2dot0; IN someODE;  BY AJAX(
           cntl); TO MATCH error
C   plot y solution vs. x
    points= 100:    npoints= points:   deltaX= (50 -
x0)/npoints
    do 10 i= 1, npoints
      xp(i)= x0 + (i-1)*deltaX
10  continue
    call someODE
    @aplot('rr-AJAX')
  end
  model someODE
    npoints= points  ! initial conditions
    print *, 'npoints =', npoints
    y=y0:  ydot=ydot0:  y2dot= y2dot0:  x= x0:  dx=
.002
    initiate ISIS;  for diffeqs; &
      equations y3dot/y2dot, & y2dot/ydot, ydot/y; &
      of x;  step dx;  to xfinal
    error= 0
    do 10 i= 1, npoints
      xfinal= xp(i)
      integrate diffeqs;  by ISIS
      error= error + (yp(i) - y)**2   !/ (.001+y**2)
      if( npoints .eq. 100) then
        yp(i)= y:   ydotp(i)= ydot:   y2dotp(i)= y2dot
        y3dotp(i)= y3dot
      end if
10  continue
  end
  model diffeqs
    if( x .le. 1e-13 .and. x .ge. 0) then
      y3dot= 3 * (ydot * y2dot + ydot**2 / 1e-13) / y
    elseif(x.ge.-1e-13 .and. x.le.0) then
      y3dot=-3*(ydot*y2dot+ydot**2 / 1e-13) / y
    else
      y3dot=3*(ydot*y2dot+ydot**2 / x) / y
    end if
  end
  controller cntl( Ajax)
    converge=2: zero=1e-13: remax =30
  end
  procedure aplot( plot77)

   ooo  ! (See 'aplot' code in appendix.)
```

Computer Plots

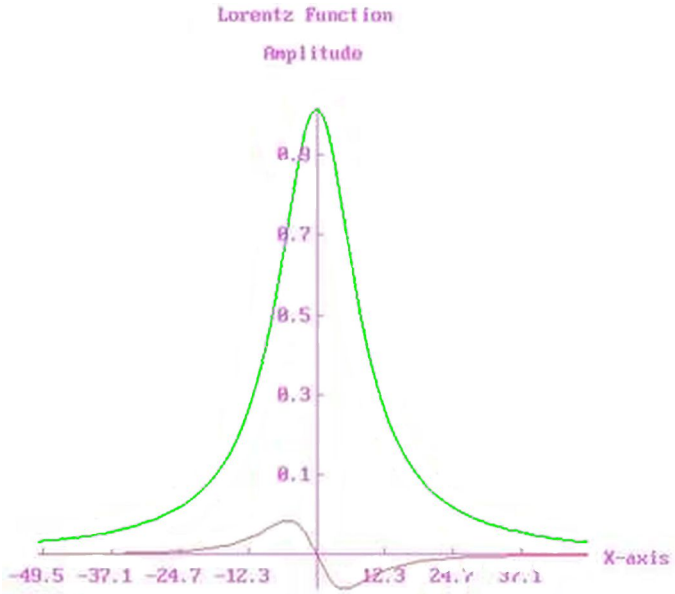


Figure 3.2 Solution to Lorentz ODE

Computer Output for AJAX Solver:

Starting search for parameters to minimize |error|

--- **AJAX** SUMMARY, INVOKED AT LORENTZ[20] FOR MODEL SOMEODE ----

CONVERGENCE CONDITION AFTER 23 ITERATIONS
UNKNOWNS CONVERGED
CONSTRAINTS SATISFIED
ALL SPECIFIED CRITERIA SATISFIED

LOOP NUMBER ...	[INITIAL]	1	2
UNKNOWNS			
Y0	3.000000E-02	3.000000E-02	3.000000E-02
YDOT0	1.000000E-03	9.999979E-04	9.999959E-04
Y2DOT0	7.000000E-05	6.936494E-05	6.874562E-05
OBJECTIVE			
‖G‖ @MIN ‖X‖	1.403871E+00	5.153686E-01	1.767260E-01

ooo

LOOP NUMBER ...	[INITIAL]	**23**	
UNKNOWNS			
Y0	3.000000E-02	3.000001E-02	
YDOT0	1.000000E-03	9.999914E-04	
Y2DOT0	7.000000E-05	6.725475E-05	
OBJECTIVE			
‖G‖ @MIN ‖X‖	1.403871E+00	**7.576720E-14**	! slow convergence

---END OF LOOP SUMMARY

x y	y’	y’’
-50.000	3.00000E-02	9.99991E-04	6.72547E-05

-49.000	3.10345E-02	1.06969E-03	7.22181E-05
-48.000	3.21412E-02	1.14461E-03	7.76945E-05
-47.000	3.33257E-02	1.22528E-03	8.37460E-05
-46.000	3.45940E-02	1.31232E-03	9.04432E-05

ooo

47.000	3.03687E-02	-1.39562E-03	7.69468E-05
48.000	2.90105E-02	-1.32183E-03	7.07426E-05
49.000	2.77232E-02	-1.25395E-03	6.51134E-05

ELAPSED TIME= 133.14 SECONDS

Findings

Limiting x-axis got past the stiff ODE problem but convergence was slow. Objective dropped by a factor of 10^{-14}. Nice! The x-axis limit of +/- 50 avoided this stiff problem. Extend the limits to +/- 200 or more and you'll have a stiff ODE problem again. More digits may remove your stiff problem for a time but given time and you'll need more digits again. We are always pushing the envelope!

Application Problem 3-3

A Bang-Bang Control Problem

Problem Description

Bang-Bang Control is used in this Voice Coil Motor[3]actuator system Design. A Voice Coil Motor (VCM) is basically an electromagnetic transducer in which a coil placed in a magnetic pole gap experiences a force proportional to the current passing through the coil. VCM can be generally classified as a dc brushless motor if the coil is wound as the starter and the permanent magnet with the attached payload is allowed to move or rotate. For rotary applications, a number of types of limited travel & brushless motors are available on the market, possessing flat torque output over a region of ±20°. By providing the necessary encoding and electronics, these motors are also used as constant speed motors.

Assuming the force generated is independent of position, the governing equation can be written as

$$L \frac{di}{dt} + R\, i = E - \phi\, v$$

$$\frac{dv}{dt} = \phi\, i / J$$

$$v = dx / dt$$

where

L	inductance of the motor coil (henry);
i	current intensity through the motor coil (ampere);
R	resistance of the motor coil (ohm);
E	applied voltage to the coil (volt);
ϕ	transducer constant; for linear motion, its units are either Newton/ampere or volt/meter/second, and for rotary motion, the units are N-m/A or volt/radians/second;
v	velocity of motion (radians/second or meter/second);
J	total inertia of the motor and access mechanism (kg - m^2 or kg);
x	response (radians or meter).

The VCM design objective is to move from point A to B in the shortest time possible.

A physical constraint may exist to limit R / ϕ**2 to a constant ± a tolerance. H1 & H2 in attached code show how such a constraint may be handled. R and ϕ will be varied in order to find a minimal seek time, Total_t, while meeting this physical constraint.

A dc brushless motor is to be used whose coil has an inductance of .015 H in the presence of the permanent magnet. The supply voltage is 24 V, and there is a drop of about 1 V in the electronics before the motor winding. To reduce the thermal gradient, the maximum current should be limited to 0.5 A. Obtain the voltage transition times (Tau_i), plus the torque sensitivity (ϕ) of the dc motor and the resistance (R) of the coil to meet the design objectives.

---

[3] Ananthanarayanan, K. S., **Third-Order Theory and Bang-Bang Control of Voice Coil Actuators**, IEEE Trans. on Magnetics, Vol. MAG-18, No. 3, May 1982, pp. 888-892.

Computer Code
1. Theoretical Problem/Solution ... Zero time for change in polarity

```
global all
Problem Seektime    ! Voice Coil Motor For An Actuator
! Objective = Move from Point A to B in desired time (Tf)
  call setup
  Call Actuator
End
procedure setup
  dimension xTau(3)
  Tf= .072    ! Time In Seconds
  R= 10:  Emax= 23:  aL= .015    ! Ohms, Volts, Henry
  Phi= .4:  aJ= 5e-4    ! Volts * Sec./Rad. & Kg * M * M *
Ohm
  xTau(1)= .3*Tf:  xTau(2)=.4*Tf:  xTau(3)= .3*Tf
  Tmech= .12:  Tm8tol= .05  ! Mechanical Time constant
& % tolerance
  Pi= 4*Atan(1.):  xFinal= Pi / 6     ! (180/6) 30 degrees of
travel
End
Model Actuator
  H1=(Tmech*(1+Tm8tol))-R*aJ/Phi**2  ! H1,H2 & H3
must be >= 0
  H2=R*aJ/Phi**2-Tmech*(1-Tm8tol)
H3=.5-C     ! upper limit for Current

  Find xTau; In Bangbang; By Ajax( Cntl1); &
    To Match errpos    !, errvel

  Print *, "Solution: "
  Print *, xTau(1), xTau(2), xTau(3)
End
Controller Cntl1( Ajax)
  damp = .0002
End
Model Bangbang
  Tp=0:  T= 0:  C= 0:  V= 0:  X= 0     ! Initial Values
  do 10 I= 1, 3
    Total8t= Tp + abs( xTau( i))
    Dt= xTau( i)/50:  Dp= 10 * Dt:  Tp= Tp + Dp
    Initiate Isis;   For VCMotor; &
      Equations Cdot/C, Vdot/V, Xdot/X;  of T;  Step
Dt;  To Tp
    Polarity= -(-1)**I:  E= Emax * Polarity
    Do While (Tp .lt. Total8t) then
      Tp= Tp + Dp
      Integrate VCMotor;   By Isis
      Print 79, T, V, Vdot  ,C,CDOT,X,XDOT
    End Do
10 continue
  errpos= (xFinal - X)**2:  errvel= V**2
  Print *, " "
  Print 79, T, H1, H2, H3
79 format( 1x, f7.5, 1x, 6(1pg13.5, 1x))
End
Model VCMotor
C C=Current=i
  Xdot= V
  Vdot= Phi * C / aJ
  Cdot= (E - Phi * V - R * C) / aL
End
```

2. Theoretical Problem/Solution ... Zero time for change in polarity
New objective: Minimize Seek Time

```
global all
Problem Seektime(200000,25000, 25000) ! Voice Coil
Motor For An Actuator
! Objective = Minimize Seek Time
  call setup
  Find Phi, R;   In Actuator;  By Jupiter( Cntl2);
~   Holding H1, H2, H3;  To Minimize tTotal
End
procedure setup
  dimension xTau(3)
  Tf= .072    ! Time In Seconds
  R= 10:   Emax= 23:  aL= .015    ! Ohms, Volts, Henry
  Phi= .4:  aJ= 5e-4    ! Volts * Sec./Rad. & Kg * M *
M * Ohm
  xTau(1)= .3*Tf:  xTau(2)=.4*Tf:  xTau(3)= .3*Tf
  Tmech= .12:  Tm8tol= .05  ! Mechanical Time
constant & % tolerance
  Pi= 4*Atan(1.):  xFinal= Pi / 6     ! (180/6) 30 degrees
of travel
End
Model Actuator
  H1=(Tmech*(1+Tm8tol))-R*aJ/Phi**2  ! H1,H2 &
H3 must be >= 0
  H2=R*aJ/Phi**2-Tmech*(1-Tm8tol)
  H3=.5-C     ! upper limit for Current
  Find xTau;  In Bangbang;  By Ajax( Cntl1);
~   To Match errpos    !, errvel
End
Controller Cntl1( Ajax)
  damp = .0002     !:    summary=0
End
Controller Cntl2( Jupiter)
  maxeval= 4000000
End
   ooo same as previous example
```

3. Practical Problem/Solution ... ArcTan() used to change polarity

```
Problem Seektime              ! Voice Coil Motor For An
Actuator
  ! Objective = Minimize Seek Time with constraint on
  ! rise/fall times; ie. |Edot| < Emax_slope]

 ooo   (same code as in previous example)

 Pi = 4*Atan(1.):  Xfinal = Pi / 6:  !radians = 30 degrees
of travel
 Ypeak = .985 * Pi/2:  Xmax = Tan( Ypeak)
 Trise = 5*Tf/100:  Pw50 = Trise:  Tends = .6*Trise
 Find Phi, R;  In Actuator;  By Jupiter( Cntl1)  &
   Holding H1, H2, Hc:  To Minimize Total_t
End

 ooo   (same code as in previous example)

Model VCMotor
 Vdot = Phi * C / J:  Xdot = V
 Call Risetime    ! E = constant except during transition
times
 Cdot = (E - Phi * V - R * C) / L:  Hc = .5 - C          ! Hc
must be >= 0
End

Model Risetime
 If( I .Eq. 1) then ! Calculate E during transition time
   If( T .lt. Tends ) then     ! ie. E = f(t) during rise/fall
times
     E = (Atan((4*T/Pw50 - 1.)*Xmax) / Ypeak + 1.)/2.
   Endif
 Else If( I .eq. 2) then
   If( T-Xtau(1) .lt. Trise) then        !Switch voltage
polarity
     Tt = T - Xtau(1)
     E = - Atan((2*Tt/Pw50 - 1.)*Xmax) / Ypeak
   Endif
 Else If( I .eq. 3) then
   If( T-Xtau(1)-Xtau(2) .lt. Trise) then        ! Switch
voltage polarity
     Tt = T-Xtau(1)-Xtau(2)
     E = Atan((2*Tt/Pw50 - 1.)*Xmax) / Ypeak
 Else If( Time-T .lt. Tends) then     ! Drop voltage to
zero
   Tt = T-Time+Tends
   E = - (Atan((4*Tt/Pw50 - 1.)*Xmax) / Ypeak-1.)/2
 Endif
End
```

Computer Output for AJAX Solver:

1. Theoretical Problem/Solution ... Zero time for change in polarity

T	V	Vdot	C	CDOT	X	XDOT
0.00926	13.056	1492.4	1.8655	-58.504	5.44372E-02	13.056
0.01389	19.469	1281.5	1.6019	-53.756	0.13010	19.469
0.01852	24.963	1096.6	1.3708	-46.187	0.23328	24.963
0.02315	29.664	938.19	1.1727	-39.525	0.36002	29.664
0.03392	7.8787	-2199.0	-2.7488	89.092	0.57893	7.8787
0.03930	-2.9663	-1837.9	-2.2973	77.311	0.59128	-2.9663
0.04469	-12.017	-1532.9	-1.9162	64.579	0.55021	-12.017
0.05007	-19.566	-1278.5	-1.5982	53.864	0.46456	-19.566
0.05882	-4.8919	2086.9	2.6086	-75.313	0.34306	-4.8919
0.06320	3.6420	1814.2	2.2678	-75.643	0.34076	3.6420
0.06757	11.026	1566.3	1.9579	-65.939	0.37325	11.026
0.07195	17.398	1351.6	1.6895	-56.939	0.43578	17.398
0.07633	22.897	1166.3	1.4578	-49.134	0.52423	22.897

0.07633 -62.374 62.386 -0.95783

---- AJAX SUMMARY, INVOKED AT ACTUATOR[28] FOR MODEL BANGBANG ----

CONVERGENCE CONDITION AFTER 12 ITERATIONS
UNKNOWNS NOT CONVERGED
CONSTRAINTS SATISFIED
ALL SPECIFIED CRITERIA SATISFIED

LOOP NUMBER ...	[INITIAL]	1	2
UNKNOWNS			
XTAU(1)	2.160000E-02	2.391058E-02	2.269326E-02
XTAU(2)	2.880000E-02	2.653013E-02	2.725812E-02
XTAU(3)	2.160000E-02	2.214367E-02	2.178550E-02
OBJECTIVE			
‖G‖ @MIN ‖X‖	2.439001E-01	5.005191E-02	1.591911E-02

ooo

LOOP NUMBER ...	[INITIAL]	11	12
UNKNOWNS			
XTAU(1)	2.160000E-02	2.315396E-02	2.314758E-02
XTAU(2)	2.880000E-02	2.691873E-02	2.692375E-02
XTAU(3)	2.160000E-02	2.188192E-02	2.187862E-02
OBJECTIVE			
‖G‖ @MIN ‖X‖	2.439001E-01	1.616726E-06	**4.042006E-07**

---END OF LOOP SUMMARY

Solution:
0.02314757710280506 0.02692374777334727 0.021878623739922533
ELAPSED TIME = 0.88 SECONDS

2. Theoretical Problem/Solution ... Zero time for change in polarity
New objective: Minimize Seek Time

--- **AJAX** SUMMARY, INVOKED AT ACTUATOR[28] FOR MODEL BANGBANG ----

CONVERGENCE CONDITION AFTER 14 ITERATIONS
UNKNOWNS NOT CONVERGED
CONSTRAINTS SATISFIED
ALL SPECIFIED CRITERIA SATISFIED

LOOP NUMBER ...	[INITIAL]	1	2
UNKNOWNS			
XTAU(1)	2.160000E-02	2.391058E-02	2.460828E-02
XTAU(2)	2.880000E-02	2.653013E-02	2.584422E-02
XTAU(3)	2.160000E-02	2.214367E-02	2.246816E-02
OBJECTIVE			
||G|| @MIN ||X||	2.439001E-01	2.676381E-02	1.676711E-03

ooo

LOOP NUMBER ...	[INITIAL]	13	14
UNKNOWNS			
XTAU(1)	2.160000E-02	2.415900E-02	2.414942E-02
XTAU(2)	2.880000E-02	2.617551E-02	2.618261E-02
XTAU(3)	2.160000E-02	2.230714E-02	2.230376E-02
OBJECTIVE			
||G|| @MIN ||X|	2.439001E-01	2.807086E-06	**7.017860E-07**

---END OF LOOP SUMMARY

ooo

--- **AJAX** SUMMARY, INVOKED AT ACTUATOR[28] FOR MODEL BANGBANG ----

CONVERGENCE CONDITION AFTER 12 ITERATIONS
UNKNOWNS CONVERGED
CONSTRAINTS UNSATISFIED
ALL SPECIFIED CRITERIA SATISFIED

LOOP NUMBER ...	[INITIAL]	1	2
UNKNOWNS			
XTAU(1)	1.521291E-02	1.542049E-02	1.542090E-02
XTAU(2)	3.567217E-02	3.548515E-02	3.548479E-02
XTAU(3)	1.377865E-02	1.373695E-02	1.373687E-02
OBJECTIVE			
||G|| @MIN ||X||	1.836747E-03	4.606212E-04	4.588237E-04

ooo

LOOP NUMBER ...	[INITIAL]	11	12
UNKNOWNS			
XTAU(1)	1.521291E-02	1.560927E-02	1.560927E-02
XTAU(2)	3.567217E-02	3.531630E-02	3.531630E-02
XTAU(3)	1.377865E-02	1.369971E-02	1.369971E-02
OBJECTIVE			
||G|| @MIN ||X||	1.836747E-03	4.541167E-06	**4.540613E-06**

---END OF LOOP SUMMARY

3. Practical Problem/Solution ... Lorentz function used to minimize Jerk

--- **AJAX** SUMMARY, INVOKED AT ACTUATOR[33] FOR MODEL BANGBANG ----

CONVERGENCE CONDITION AFTER 20 ITERATIONS
UNKNOWNS NOT CONVERGED
CONSTRAINTS UNSATISFIED
MAXIMUM ITERATIONS PERFORMED
SPECIFIED CRITERIA UNSATISFIED

LOOP NUMBER	[INITIAL]	1	2
UNKNOWNS			
XTAU (1)	2.160000E-02	2.350391E-02	2.396152E-02
XTAU (2)	2.880000E-02	2.153747E-02	1.997667E-02
XTAU (3)	2.160000E-02	2.751439E-02	2.804971E-02
OBJECTIVE			
||G|| @MIN ||X||	2.171108E+02	2.169100E+01	1.439846E+01

ooo

LOOP NUMBER	[INITIAL]	19	20
UNKNOWNS			
XTAU (1)	2.160000E-02	2.485609E-02	2.485575E-02
XTAU (2)	2.880000E-02	1.510293E-02	1.509968E-02
XTAU (3)	2.160000E-02	2.901244E-02	2.901266E-02
OBJECTIVE			
||G|| @MIN ||X||	2.171108E+02	3.521112E-02	3.520294E-02

---END OF LOOP SUMMARY

--- **AJAX** SUMMARY, INVOKED AT ACTUATOR[33] FOR MODEL BANGBANG ----

```
   CONVERGENCE CONDITION AFTER 20 ITERATIONS
      UNKNOWNS NOT CONVERGED
      CONSTRAINTS UNSATISFIED
      MAXIMUM ITERATIONS PERFORMED
      SPECIFIED CRITERIA UNSATISFIED


 LOOP NUMBER .........   [INITIAL]         1              2
 UNKNOWNS
   XTAU    (     1)    2.485575E-02  2.163943E-02  2.128472E-02
   XTAU    (     2)    1.509968E-02  2.543858E-02  2.664468E-02
   XTAU    (     3)    2.901266E-02  1.611228E-02  1.347697E-02
 OBJECTIVE
   ||G|| @MIN ||X||    9.500061E+02  1.120377E+02  1.114299E+01

ooo

--- AJAX SUMMARY, INVOKED AT ACTUATOR[33] FOR MODEL BANGBANG ----


   CONVERGENCE CONDITION AFTER  1 ITERATIONS
      UNKNOWNS CONVERGED
      CONSTRAINTS UNSATISFIED
      ALL SPECIFIED CRITERIA SATISFIED


 LOOP NUMBER .........   [INITIAL]         1
 UNKNOWNS
   XTAU    (     1)    2.109537E-02  2.109537E-02
   XTAU    (     2)    1.363358E-02  1.363358E-02
   XTAU    (     3)    3.235079E-03  3.235079E-03
 OBJECTIVE
   ||G|| @MIN ||X||    2.318342E-03  1.540415E+00


---END OF LOOP SUMMARY


--- JUPITER SUMMARY, INVOKED AT SEEKTIME[4] FOR MODEL ACTUATOR ---


   CONVERGENCE CONDITION AFTER  1 ITERATIONS
      OBJECTIVE CRITERION SATISFIED
      ALL SPECIFIED CRITERIA SATISFIED
```

```
 LOOP NUMBER .........    [INITIAL]          1
 UNKNOWNS
   PHI                  4.000000E-01  4.137417E-01
   R                    1.000000E+01  9.999781E+00
   TRISE                0.000000E+00  0.000000E+00
   PW50                 3.600000E-03 -8.430538E-03
   TENDS                0.000000E+00  0.000000E+00
 OBJECTIVE
   TTOTAL               6.896810E-02  3.796403E-02
 INEQUALITY CONSTRAINTS
   H1                   9.475000E-02  9.679199E-02
   H3                   5.000000E-01  2.178381E+00


---END OF LOOP SUMMARY
ELAPSED TIME =  266.45 SECONDS
```

Findings

Fair rate of convergence. Solvers seemed to favor negative values for parameters being solved. This forced more 'holding' parameters to be added to problem; e.g. H1 & H3. PW50 parameter is negative but gets squared in model so doesn't matter. A positive value will hold same weight in model.

The main objective for this example was to show nesting of solvers. The 1st example showed just the most inner solver solving a boundary value problem (BVP). The 2nd example varied two parameters in the BVP, R & Phi, using the Jupiter solver. Jupiter required a huge number for its control parameter 'maxEval', i.e. 4 million! That will allow Jupiter to execute the inner BVP up to 4 million times. Normally 5 thousand executions will be enough, why so many here?

The 3rd example is the same as the 2nd example with 3 more parameters being varied. It took time to find the right solver, Jupiter in this case, and what constraints to add in order to help guide solver to a useful solution.

Applied Voltage waveforms to Coil

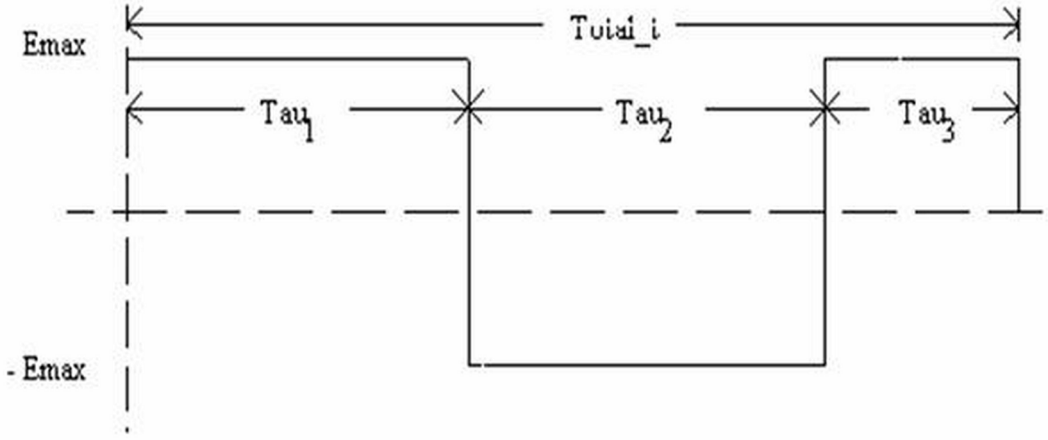


Figure **3**.3 **Theoretical Problem/Solution ... Zero time for change in polarity**
Objective: Specific time (Tf) for movement; i.e. Total_t must equal Tf.

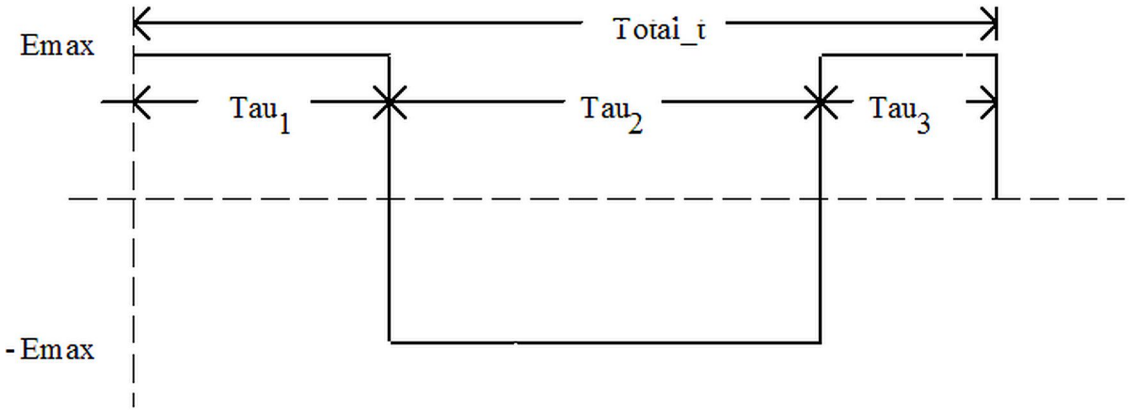


Figure **3**.4 **Theoretical Problem/Solution ... Zero time for change in polarity**
Objective: Minimize Seek Time (Total_t)

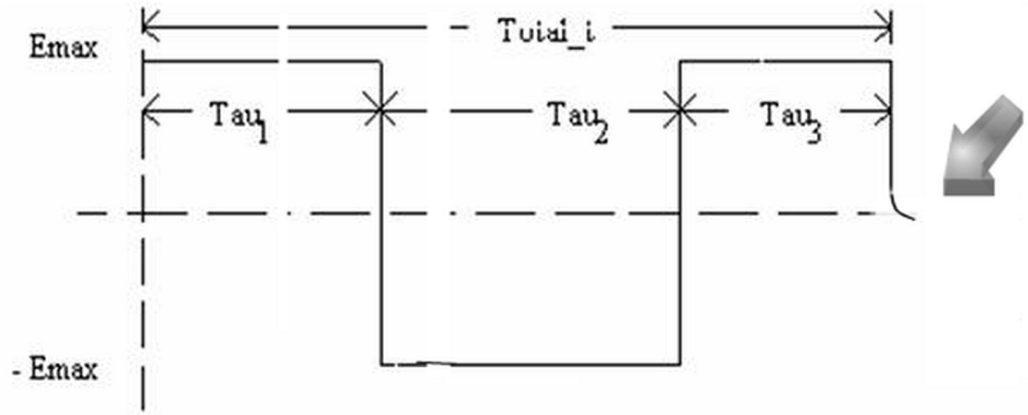


Figure **3**.5 **Practical Problem/Solution ... Lorentz function used to minimize Jerk**
Objective: Minimize Seek Time while constrainting rise/fall time.

Findings

At switch points where 'E' changes polarity, no derivative exists and this must be causing the solvers problems. On a simple test without nesting, the problems did converge to a reasonable solution but took a revelatively long time getting to it.

When nesting, trying to shorten Tau_1 and Tau_2, the solvers seem to have gotten lost on their search. I assume it's the lack of derivatives that is the problem, but maybe I'm the problem for missing something!

Application Problem 3-4

Non-Linear Equations of Motion

Problem Description

"Why should one be interested in nonlinear differential equations? The basic reason is that many physical systems - and the equations that describes them - are simply nonlinear from the outset. The usual linearization approximating devices that are partly confessions of defeat in the face of the original nonlinear problems and partly expressions of the practical view that half a loaf is better than none. It should be added at once that there are many physical situations in which a linear approximation is valuable and adequate for most purposes. This does not alter the fact that in many other situations linearization is unjustified* ." [4]

A nonlinear problem that is the equation of motion for an undamped pendulum of length A whose bob has a mass M is

$$\frac{d^2x}{dt^2} + \frac{g}{A}\sin x = 0 \qquad (1)$$

and if there is present a damping force proportional to the velocity of the bob, then the equation becomes

$$\frac{d^2x}{dt^2} + \frac{c}{M}\frac{dx}{dt} + \frac{g}{A}\sin x = 0 \qquad (2)$$

In the usual linearization we replace sin x by x, which is reasonable for small oscillations but amounts to a gross distortion when x is large.

The following is just a rough sketch of necessary code to solve such a problem

Computer Code

This is an example of an Initial Value Problem (IVP) . The code integrates from the initial value until the final condition is met.

```
Problem motion
 C=…  m=…   g=…    a=…  x0=…      xfinal=…        dx=(xfinal-xo)/100
 initiate ISIS;  for diffeqs; &
   equations x2dot/xdot, xdot/x;   of t;  step dt;  to tfinal
 npoints= 100
 do 10 i= 1, npoints
   tfinal= dt * i
   integrate diffeqs; by isis
   print *, tfinal, x
10 continue
 end
 model diffeqs
   x2dot= -c/m*xdot-g/a*sin(x)  ! 2nd order non-linear ODE
 end
```

* "It has even been suggested by Einstein that since the basic equations of physics are nonlinear, all of the mathematical physics will have to be done over again."

4 Simmons, G.F., **Differential Equations with Applications and Historical Notes**, McGraw-Hill, p.291, 1972.

Chapter 3 Exercises

1. Pendulum Problem (continued)

For the following problems assume a pendulum length A = 1 m, bob mass m = 1000 g, and gravitational acceleration g = 9.8 m/s^2.

Assume 'n' boundary value points for x_1, x_2, ... x_n and change the above IVP code to a BVP code that could solve for a 'c' value.

2. Harmonic Oscillator

The Schrödinger wave equation for a classical harmonic oscillator [5] is

$$\frac{d^2\psi}{dx^2} + \frac{8\pi^2 m}{h^2}(E - 2\pi^2 m v^2 x^2)\,\psi = 0$$

where m = mass
v = vibration frequency
h = Planck's constant = $6.6\text{x}10^{-34}$ J·s
x = position (i.e., independent variable)
E = Total energy
$\psi(x)$ = Schrödinger wave function

What value of vibration frequency, v, is sufficient to satisfy the boundary conditions $\psi(0)$ = **??,** $\psi(2)$ = **??** and $\psi(5)$ = **??** given values for parameters m and E?

3. Nuclear Reaction[6]

Neutrons are created (by a nuclear reaction) inside a hollow sphere of radius R. The newly created neutrons are uniformly distributed over the spherical volume. Assuming that all directions are equally probable (isotropy), what is the average distance ($\bar{r}$) a neutron will travel before striking the sphere's surface? Assume straight line motion, no collisions and following math model

$$\bar{r} = \frac{3}{2} R \int_0^1 \int_0^{\pi} k^2 \sqrt{1 - k^2 \sin^2\theta}\, dk \sin\theta\, d\theta$$

4. Boundary Layer Matching Example:

Find ε for the following differential equation [7]

$$\varepsilon\, y'' + (1 + x)\, y' + y = 0$$
where $y(0) = y(1) = 1$
subject to the constraint $0 < \varepsilon << 1$

---

[5] Simmons, G.F., **Differential Equations with Applications and Historical Notes**, McGraw - Hill, p. 194, 1972.
[6] Arfken, G., **Mathematical Methods for Physicists**, Academic Press, p. 205, 1968.
[7] Bender, C.M. and Orszag, S.A., **Advanced Mathematical Methods for Scientists and Engineers**, McGraw-Hill, 1978.

5. Heat Transfer

Heat transfer through a laminar boundary layer was modeled by Lighthill [8] as an implicit nonlinear Volterra integral equation as shown here:

$$F(z)=1-\frac{3\sqrt{3}}{2\pi}\int_0^z \frac{u\left[F(u)\right]^4}{\left(z^{3/2}-u^{3/2}\right)^{2/3}}du$$

For .1 ≤ z ≤ **???**, plot F(z) over this range

6. Painleve transcendent ODE

Given the (first Painleve transcendent [9]) ODE defined as

$$\frac{d^2y}{dt^2}=6y^2+t$$

with boundary conditions of y(0)=-.678, y(.5)=-1.012,& y(1)=.0313. Solve for y and plot y vs. t over the range 0 ≤ t ≤ 1.5.

---

[8] Lighthill, M.J., **Contributions to the theory of Heat Transfer through a Laminar Boundary Layer**, Proc. Roy. Soc. 202A, pp. 359-377, 1950.

[9] Chang, Y.F., **The ATOMCC Toolbox**, Byte Magazine, pp. 215-224, April 1986.

4 System of Differential Equations

A Parameter Estimation for a system of Ordinary Differential Equations (ODEs) in an Initial Value Problem (IVP) or Boundary Value Problem (BVP) is solved using the Calculus-level **'Find' statement** shown here:

IVP: **Find a** ooo To Match Error

BVP: **Find a, ydot0, y2dot0** ooo To Match Error

Where 'a' may be a vector with 'n' parts, a_1, a_2, a_3,…a_n;
ydot0, y2dot0,etc. are derivatives at independent variable = 0; and,
'error' is the objective function.

The 'find' statement is wrapped around an integrate and integration statement in order to solve the ODE while finding the best 'a' parameter(s) for the given problem.

The 'a' parameter(s) are varied to fit one's 'm' data points that make up the objective function, error. This technique can vary as many parameters as you want; e.g. 5 or 50 or 50,000. If there are less equations than parameters $m < n$, this would be classified as an under-determined system of equations. If there are more equations than parameters, $m > n$, this would be an over-determined system. Under- or Over-determined systems might force one to switch solvers to do the job.

Application Problem 4.1

The Lorentz Equations, a System of ODEs

Problem Description

Lorentz system of differential equations is found in many fields, e.g. electro-magnetics, hydrodynamics, & mechanical systems. Here we will find the parameter 'σ' that best curve fits the given data in order to show parameter estimation for systems of ordinary differential equations (ODEs).

Lorentz wrote his non-linear equations in the form:

$$dx/dt = \sigma(y - x)$$

$$dy/dt = rx - y - xz$$

$$dz/dt = xy - bz$$

t is the dimensionless time.

where σ, r and b are real, positive parameters. Initial values were the following:
σ= 1, r= 36, b= 1

Computer Code

The 'Find' statement is the work horse of a Calculus-level compiler. It calls ones math model as many times as necessary in order to converge on a solution. It varies your parameters, in this case σ, as it calls your math model. The '**in**' phase tells the name of math model routine. The '**by**' phase tells what solver to use, hera here. And the '**to**' phase tells what the objective function is; '**minimize'** means converge objective function to a minimum value, 'error' variable in this case.

FIND Alpha; IN Lorentz; BY HERA; TO MINIMIZE error

```
 graphics screen   ! Lorentz system of ODES, 1900(?)
 problem ODEsys(6000, 3000, 3000)
  common /vars/ Alpha, r, b, t, x0, y0, z0, error,
npoints, &
    tp(100), xp(100), yp(100), zp(100)
  common /eqs/ dxdt,dydt,dzdt,x,y,z
  tp(1)= .04:  tp(2)= 1.28:  tp(3)= 3.28
  xp(1)= 15.941:  yp(1)= 5.9966:   zp(1)= 57.498
  xp(2)= 13.770:  yp(2)= 16.314:   zp(2)= 37.115
  xp(3)= -0.6357:  yp(3)= -1.0121:  zp(3)= 18.994
  Alpha= 11:  r= 36:   b= 1   ! initial values
  x0= 19:  y0= 20:  z0= 50     ! initial conditions
  npoints= 3
  print *, ' '
  print *, 'Starting search for parameters to minimize
|error|'
  print *, ' '
  Find Alpha; in lorentz; by HERA; to minimize error
C   plot x,y,z solution vs. time
  npoints= 100
  do 10 i= 1, npoints
    tp(i)= i / 25.
 10   continue
  call lorentz
  @aplot('rr-AJAX')
 end
 model lorentz
  common /vars/ Alpha, r, b, t, x0, y0, z0, error,
npoints, &
    tp(100), xp(100), yp(100), zp(100)
  common /eqs/ dxdt,dydt,dzdt,x,y,z
  x=x0: y=y0: z=z0: t=0:  dt= .01
  initiate ISIS; for diffeqs; &
    equations dxdt/x, dydt/y,  dzdt/z; &
    of t;  step dt;  to tfinal
  error= 0
  do 10 i= 1, npoints
    tfinal= tp(i)
    integrate diffeqs; by isis
    error= error + (xp(i) - x)**2 + (yp(i)-y)**2 +
(zp(i)-z)**2
    if( npoints .eq. 100) then
      xp(i)=x:  yp(i)=y:  zp(i)= z
    end if
 10   continue
  terminate diffeqs
 end
 model diffeqs
  common /vars/ Alpha, r, b, t, x0, y0, z0, error,
npoints
  common /eqs/ dxdt,dydt,dzdt,x,y,z
  dxdt= Alpha * (y - x)
```

```
  dydt= (r-z) * x - y
  dzdt= x * y - b * z
end
```

```
procedure aplot( plot77)

o o o     basically the same as appended 'aplot' routine
```

Computer Plots

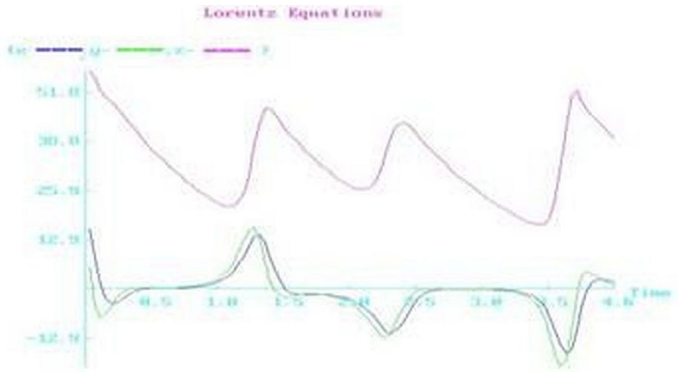


Figure 4.1 Solution to Lorentz Equations

Computer Output for HERA Solver:

Starting search for parameters to minimize |error|

---- **HERA** SUMMARY, INVOKED AT ODESYS[28] FOR MODEL LORENTZ ----

```
  CONVERGENCE CONDITION AFTER  6 ITERATIONS
    UNKNOWNS CONVERGED
    OBJECTIVE CRITERION UNSATISFIED
    ALL SPECIFIED CRITERIA SATISFIED

 LOOP NUMBER ...      [INITIAL]          1                  2
 UNKNOWNS
   ALPHA            1.100000E+01    1.650000E+01    1.509860E+01
 OBJECTIVE
   ERROR            6.883523E+02    6.725837E+00    1.028192E+01

o o o

 LOOP NUMBER ...      [INITIAL]          5                  6
 UNKNOWNS
   ALPHA            1.100000E+01    1.576978E+01    1.577000E+01
 OBJECTIVE
   ERROR            6.883523E+02    8.826836E-07    6.944056E-08

---END OF LOOP SUMMARY
```

Findings

Excellent rate of convergence! Alpha parameter value is very reasonable. This example's results suggest a good math model.

While Lorentz studied this system of ODEs, he probably needed a math model to simulate the solutions to his ODEs. Just like a pulse train simulated at Memorex in the 1980s. The Lorentz function is defined as $y=\frac{1}{1+x^2}$. This allows for an excellent math model of an isolated pulse for the

disc drive industry. Memorex put many isolated pulse models together with the models separated by some offset times to simulate a pulse train that is similar to the z(t) plot in above plot.

Application Problem 4.2

The Convection Reaction Equations, a System of PDEs

(An Initial Value Problem)

Problem Description

The equations are in the form:

$$\frac{\partial U_1}{\partial t} = P_1 \frac{\partial U_1}{\partial x} - K \bullet U_1 \bullet U_2$$

$$\frac{\partial U_2}{\partial t} = P_2 \frac{\partial U_2}{\partial x} - K \bullet U_1 \bullet U_2$$

$$\frac{\partial U_3}{\partial t} = K \bullet U_1 \bullet U_2$$

where initial values of parameters were assumed to be the following:
$P_1 = 1.23$, $P_2 = 9.87$, & $K = .3$

Boundary conditions for $0 \le x \le 100$ are: $U_1 = \exp(-(x-10)^2/15)$,

$U_2(x,0) = U_1(100\text{-}x,0)$ & $U_3(x, 0) = 1$

Computer Code

The 'Find' statement is the work horse of a Calculus-level compiler. It calls ones math model as many times as necessary in order to converge on a solution. It varies your parameters, in this case Ua & aK, as it calls your math model. The '**in**' phase tells the name of math model routine. The '**by**' phase tells what solver to use, Jupiter here. And the '**to**' phase tells what the objective function is; '**minimize'** means converge objective function to a minimum value, 'error' array in this case.

FIND Ua, aK; IN tAxis; BY JUPITER; TO MINIMIZE error

```
global all
problem Convection  ! Reaction
C ----------------------------------------
C --- Calculus Programming example: 1D Equation;
PDE Initial Value Problem
C --- Method of Lines
C ----------------------------------------
  dynamic U1t, U1, U2t, U2, U3t, U3
C
C User parameters ...
  p1 = 1.23:  p2 = 9.87:  aK = .3
  ipoints = 20  ! grid pts. over x-axis
  tFinal =  1   ! not sure when to stop
C
C x-parameter initial settings: x ==> i
  xFinal=100:  dx= xFinal/ipoints:  ip=ipoints:  yesno=
0

C t-parameter initial settings: t ==> j
  tPrint = tFinal/20
  allot U1(ip), U1t(ip), U2(ip), U2t(ip), U3(ip), U3t(ip)

  find Ua,aK;  in tAxis;  by jupiter;  to minimize error
   yesno=1:   call Axis ! print results
end
model tAxis
C ... Integrate over t-axis
C settings at t = 0
  do 1 ii = 1, ipoints
    U1(ii)=exp(-((ii-1)*dx-10)**2 / 15):   U3(ii)=1
 1  continue
  do 2 ii = 1, ipoints
    U2(ii)=U1(ip-ii+1)
2 continue

  t=0:  tPrt=tPrint:  dt= tPrt / 10
  Initiate ISIS;  for PDE; &
    equations U1t/U1,U2t/U2,U3t/U3;  of t;  step dt;  to
        tPrt
  do while (t .lt. tFinal)
    Integrate PDE;  by ISIS
    if( t*yesno .ge. tPrt) then
      print 79, t, (U1(ii), ii=1,ip)
      print 79, t, (U2(ii), ii=1,ip)
      print 79, t, (U3(ii), ii=1,ip)
    end if
```

```
   tPrt = tPrt + tPrint
  end do
79 format(1x,f8.4,1x,20(g14.5, 1x))
end
model PDE          ! Partial Diff. Equ.
C          ! Method of Lines
  if( t .ge. tFinal/2 .and. &
    t .lt. tFinal/2+dt) error= (U2(16)-.7654)**2   ! BC1:
U2(16)=.7654 @ t=tFinal/2
  do 20 ii=1,ipoints-1  ! System of ODEs
    U1t(ii)=p1*(U1(ii+1)-U1(ii))/dx - aK * U1(ii) *
U2(ii)
    U2t(ii)=p2*(U2(ii+1)-U2(ii))/dx - aK * U1(ii) *
U2(ii)
    U3t(ii) = aK * U1(ii) * U2(ii)
 20  continue
  U1t(ip)=p1*(U1(ip)-U1(ip-1))/dx - aK * U1(ip) *
U2(ip)
  U2t(ip)=p2*(U2(ip)-U2(ip-1))/dx - aK * U1(ip) *
U2(ip)
  U3t(ip)=aK * U1(ip) * U2(ip)
end
```

Computer Output for Jupiter Solver:

```
   ooo

~~~ AT TAXIS[34]  OPERATION: Calculus Mode Assignment
*** OUT-OF-RANGE ARGUMENT TO  EXP  (I.E.-0.43E+03) IS REPLACED
BY THE LIMIT (-0.10E+03)
~~~ AT TAXIS[34]  OPERATION: Calculus Mode Assignment
*** OUT-OF-RANGE ARGUMENT TO  EXP  (I.E.-0.48E+03) IS REPLACED
BY THE LIMIT (-0.10E+03)

--- JUPITER SUMMARY, INVOKED AT CONVECTI[15] FOR MODEL TAXIS ----

   CONVERGENCE CONDITION AFTER  1 ITERATIONS
     OBJECTIVE CRITERION SATISFIED
     ALL SPECIFIED CRITERIA SATISFIED


 LOOP NUMBER ...        [INITIAL]              1
 UNKNOWNS
   UA                  0.000000E+00       1.910618E+00
   AK                  1.000000E+00       1.000000E+00
 OBJECTIVE
   ERROR               5.838906E-01       1.715911E-01


---END OF LOOP SUMMARY


~~~ AT TAXIS[34]  OPERATION: Calculus Mode Assignment
*** OUT-OF-RANGE ARGUMENT TO  EXP  (I.E.-0.11E+03) IS REPLACED
BY THE LIMIT (-0.10E+03)
~~~ AT TAXIS[34]  OPERATION: Calculus Mode Assignment
*** OUT-OF-RANGE ARGUMENT TO  EXP  (I.E.-0.14E+03) IS REPLACED
BY THE LIMIT (-0.10E+03)
```

Findings

Application Problem 4.3

Body Plasma Chemistry[10]

Problem Description

Determine the concentration of a Therapeutic treatment drug - and for that matter any drug - that is in the body over a period of time by finding:

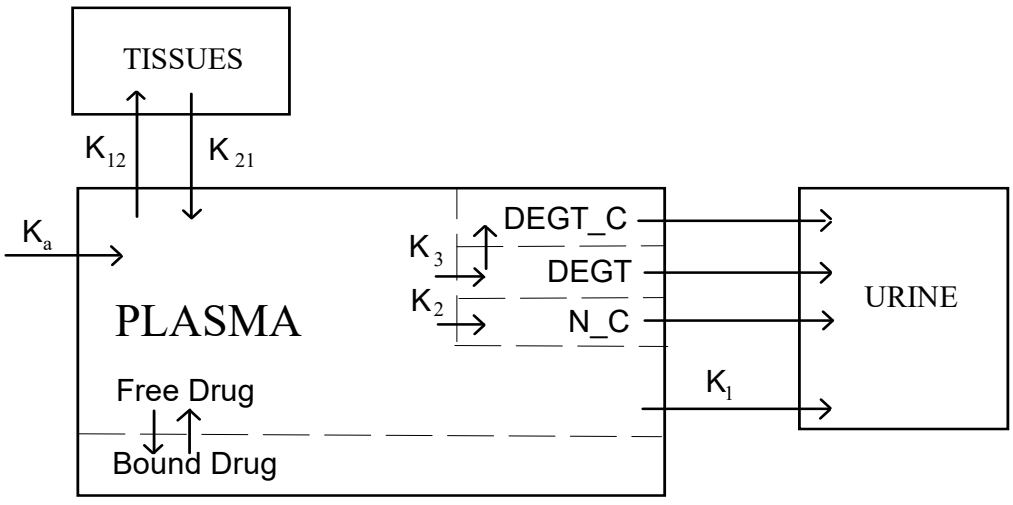
1. The rate constant (K_a) that determines the diffusion of therapeutic treatment drug from the stomach into the blood-stream (plasma);
2. The rate at which the drug enters and leaves the tissues, K_{12} and K_{21};
3. The loss of therapeutic treatment drug into the urine, K_1;
4. The break-down of therapeutic treatment drug into conjugated form and DEGT, $K_2 + K_3$;
5. The volume of blood, V.

?? 6. The binding or non-binding of the drug with free proteins in the plasma, $K_?$

The body tissues utilize the drug and therefore an amount is removed by the body's filtering system, i.e. the Kidneys and urine. As with most compounds, some binding with proteins can occur, as well as conjugation and degradation of the drug. This will also provide information as to how often the treatment drug needs to be administered to keep the concentration high enough to allow for the required treatment to occur.

Given: Observed values (concentrations) for the plasma levels of the therapeutic treatment drug; observed accumulated values (amounts) of the therapeutic treatment drug, conjugated therapeutic treatment drug and DEGT (degraded therapeutic treatment drug); and the dose of therapeutic treatment drug.

Goal/Objective: Unknown to author. If you know what the goal/objective should be, please contact us so future versions of this textbook will be able to show a complete problem with answers.



[10] Wolski, D. and Petersen, D.M., about 1977

NP_t = Total therapeutic treatment drug in plasma
NP_f = Free therapeutic treatment drug in plasma
NT = therapeutic treatment drug in tissues
N_u = therapeutic treatment drug excreted
N_C_u = Conjugated therapeutic treatment drug excreted
$DEGT_u$ = Total (free and conjugated) DEGT excreted
D = Dose of therapeutic treatment drug
V = Volume of distribution
P_n = Protein binding constants

Constraints:

$$[NP_f] = f([NP_t], P_1, P_2, P_3, P_4)$$

$$\frac{d[NP_t]}{dt} = K_a * D/V * e^{(-K_a * t)} - K_{12}\,[NP_f] + K_{21}\,[NT] - (K_1 + K_2 + K_3)\,[NP_f]$$

$$\frac{d[NT]}{dt} = K_{12}\,[NP_f] - K_{21}\,[NT]$$

$$\frac{d[N_u]}{dt} = K_1\,V\,[NP_f]$$

$$\frac{d[N_C_u]}{dt} = K_2\,V\,[NP_f]$$

$$\frac{d[DEGT_u]}{dt} = K_3\,V\,[NP_f]$$

where "[X]" implies Concentration of X
and K_y represent Rate constants, y = a, 12, 21, 1, 2 & 3

"BOUND THERAPEUTIC TREATMENT DRUG" is that portion of Therapeutic treatment drug in the plasma that is bound to protein and thus unable to participate in other reactions. In analyzing blood samples, no distinction can be made between free $[NP_f]$ and protein-bound therapeutic treatment drug and thus the observed values are total therapeutic treatment drug $[NP_t]$. The function that relates NP_f to NP_t is shown in the 1st equation. It involves finding a root of a 3rd order polynomial, the coefficients of which are functions of NP_t and the four protein binding constants.

The four reactions involving N_C, DEGT and DEGT_C that are shown without a rate constant are non-limiting reactions and assumed to be instantaneous. No distinction is made between free DEGT and conjugated DEGT (DEGT_C), they are simply totaled as DEGT in the model.

Computer Code
The following code is in the PROSE computer language. It is close to the FortranCalculus language but needs some editing. If you are interested in completing this example and understand it, please contact us.

```
Problem DrugConcentration
    execute Initialize
    b = Data( .05, 1, .005, .1, .5, .5, 50)
    Find k, v; in DrugModel;   by Hera (Optcont);   with bounds b;   to minimize error
End
```

```
Model DrugModel
    local i, j
    plasma = 0:          tissue = 0:          excret = 0:          error = 0
    bsamp = 1:           usamp = 1:           time = 0:            dt = dtstart
    Initiate Gemini( StepSize);   for Kinetics;  &
      equations dplasma / plasma, dtissue / tissue, dexcret / excret;  &
      of time;  step dt;    to tnext
    execute Tprint
    for i = 1 to nb + nu   do
        if samptype(i) is blood then
            tnext = bloodtime( bsamp)
            integrate Kinetics
            error.b( bsamp) = plasma / obs.plasma( bsamp) - 1
            es = error.b( bsamp)**2:              w = dplasma**2 + 1
            we = es / w:                          error = error + es
            if( bloodtime( bsamp) .ne. urinetime( usamp))          execute Bprint
            bsamp = bsamp + 1
        else
            tnext = urinetime( usamp)
            integrate Kinetics
             for j = 1 to 3   do
                error.u( usamp, j) = excret(j) / obs.excret( usamp, j) - 1
                es = ( error.u( usamp, j)**2 ) / weight
                w = dexcret(j)**2 + 1:            we = es / w:          error = error + es
             repeat
             if urinetime( usamp) .ne. bloodtime( bsamp - 1)
                then execute Uprint
                else execute Buprint
             close
             usamp = usamp + 1
        close
    repeat
  oldk = k:                                       oldv = v
    execute Eprint:                               olderror = error
end  [ DrugModel]
Model Kinetics
    execute ProBind
    dtissue = k(2) * free - k(3) * tissue
    dplasma = k(1) * dose / v * Exp(-k(1)*time) - (k(4)+k(5)+k(6)) * free - dtissue
    dexcret(1) = k(4) * free * v:                 dexcret(2) = k(5) * free * v
    dexcret(3) = k(6) * free * v
end
Model ProBind
    a = ap - plasma:            b = bp + bp2 * plasma:          c = cp * plasma
    free = plasma / 10
    for ii = 1 to 20   do
        xu = ((free + a) * free + b) * free + c
        xl = (3 * free + 2 * a) * free + b
        xu = xu / xl:                         free = free - xu
        if( Abs( xu) .lt. .005 * free)   exit
  repeat
    free = free + xu / 2
end
Controller Optcont for Hera
    detail=1:  detout=0:  maxit=7:  adjust=2:  improve=5.e-4
End
Controller StepSize for Gemini
    maxerr = .0001
```

```
end
Procedure Tprint
    eject 'Calculated values for this simulation.' Page
    vector print k, v
    dk = Sub( oldk, k)                    rdk = Div( dk, k)rdk = Mul( rdk, 100)
    dv = oldv - v:rdv = dv / v * 100
    skip 4 lines
  display (for i = 1 to 6, k(i)), v, (for i = 1 to 6, dk(i)), dv,  &
      (for i = 1 to 6, rdk(i)), rdv   in  &
      '           k(1)    k(2)       k(3)       k(4)       k(5)       k(6)       v'  &
      '         Value:   **.***    ***.**     **.****    **.***     ***.***    **.***   ****',  &
      '        Change:   **.***    ***.**     **.****    **.***     ***.***    **.***   ****',  &
      '  PrCnt Chg:     **.**      **.**      **.**      **.**      **.**      **.**     **.**'
    skip 4 lines
    text print  'Time ---Therapeutic treatment drug, Micrograms/Milliliter------- cum.  &
      amounts excreted, Milligrams TTD equiv----'
    text print  ' Hrs ------Plasma------ -Tissue- --Free-- -----Therapeutic treatment drug-----[  &
      ] ----Conjugated---- -----Total DEGT----'
end [.Tprint]
Procedure Bprint
    local i
    display time, plasma*1000, error.b( bsamp)*100, tissue*1000, free*1000  &
            in ' **  ****.*** (****.**) ****.*** **.***** [  &
            ]    ---      ---     ---      ---     ---      ---'
end
Procedure Uprint
    local i, erru:          Allot erru(3)
    for i = 1 to 3 erru(i) = error.u( usamp, i) *100
    display time,excret(1), erru(1), excret(2), erru(2), excret(3), erru(3)  &
            in ' **       ---       ---            ---          --- [  &
            ] ****.***.(**** **) ****.***.(**** **) ****.***.(**** **)'
end
Procedure BUprint
    local i, erru:          Allot erru(3)
    for i = 1 to 3 erru(i) = error.u( usamp, i) *100
    display time, plasma*1000, error.b( bsamp-1)*100, tissue*1000, free*1000  &
            excret(1), erru(1), excret(2), erru(2), excret(3), erru(3)  &
            in ' **  ****.*** (****.**) ****.*** **.***** [  &
            ] ****.***.(**** **) ****.***.(**** **) ****.***.(**** **)'
end
Procedure Eprint
    de = error -olderror:                 rde = de * 100 / error
    skip 4 lines
    display error, de, rde,  &
        in 'Error:  ***.******;  Change:  ***.*****;  % Change:  ***.*****'
end
Procedure Initialize
    blood = 1:              urine = 2:              weight = 1
    read data
    allot bloodtime(nb), obs.plasma(nb), error(nb)
    allot urinetime(nu), obs.excret(nu,3), error.u(nu,3)
    allot excret(3), dexcret(3), samptype(nb+nu)
    allot p(4), k(6), oldk(6), dk(6), rdk(6)
    read data
    oldk = k:               oldv = v
    ap = p(1) + p(2) + p(3) + p(4)
    bp = p(1) * p(4) + p(2) * p(3) + p(2) * p(4)
    bp2 = -(p(2) + p(4)):                       cp = - p(2) * p(4)
```

```
    display dose, dtstart, nb, nu, blood, urine, ap, bp, bp2, cp   in  &
      '****  *.***  * * * *  *.***E***  *.***E***  **.***E***  **.***E***'
    vector print p, bloodtime, obs.plasma, urinetime, obs.excret, samptype
End
```

Application Problem 4.4

Modeling a Nanostructured Solar Cell[11]

Goal/Objective: Unknown to author. If you know what the goal/objective should be, please contact us so future versions of this textbook will be able to show a complete problem with answers.

Problem Description
Problem: How to develop solar cells with a new (higher) efficiency; grätzel cells.

There are many things said about what's most important for the solar cell. So what they need is a model to know what's the rate is limits for the whole system. By then they can choose what combination of parameters will give the best solar cell. The model shown is a one-dimensional non-steady state model; a start to compare it with the Laser experiments. The laser experiments are one of the things they use to predict the efficiency.

But without a model, does experiments really tell one anything? This model is only for one excitation from a laser beam and to analyze how the decay of all species are. There is a model done for steady state, but its not really working very good in practice. Simulating the non-steady state model for some time should converge to the steady state solution when there is equilibrium in the system. This means when the change of all species are zero over the film. This could be interesting to compare with other steady-state models.

When we are talking about efficiency, it should be for simulation of the whole system. Then we have to add certain things. There are continuous excitations of electrons which is the starting conditions in this model for the electrons and the excited dye. There are a few more reactions and we have to consider the other part of the solar cell which isn't contained by nanostuctrured TiO2.

The main thing about the efficiency is that we want as many electrons leaving the back contact which is at x=0. In the reality the electrons will go out in an outer circuit to make a full circuit. But in the Laser experiments this does not happen because the outer circuit is open. In reality we will get out a current dependent on the incident light. There are many ways to measure the efficiency. IPCE(\lambda) incident-photon-to-current efficiency says how much of the incident light was converted to external current.

Title: Modeling a Nanostructured Solar Cell

Short review of the system:

We have a dye sensitizer attached to nano-structured Titanium dioxide (TiO2) film. The nano-structured particles are in a dye which transports the electrons from the electrode to the dye sensitizer. Incident light at a certain wavelength excites electrons in the dye sensitizer. So what happens to this electron after the excitation? A very fast process in nanosecond scale injects the electron to the TiO2 and its making a random walk (that's what most people think its doing) to the back contact. A new electron from the dye is put in the place of the injected electron. The electron's goes through the nano-structured film to a back contact to the outer circuit and we have a total circuit.

But there are other reactions involved in the process. The excited electron can travel other ways then to the back contact like reacting with the dye or dye sensitizer. These reactions are limiting the efficiency of the cell.

---

[11] Jarl xxx, Stanford University, (Mechanical?) Engineering, 1994.

Thus I thought it would be a good idea for the model to set up rate constants for all these reactions. Make a discretization along x which is the distance to the back contact. And the step through time and see how the kinetics, diffusion and the electric field is changing the concentration of the species along x for different times.

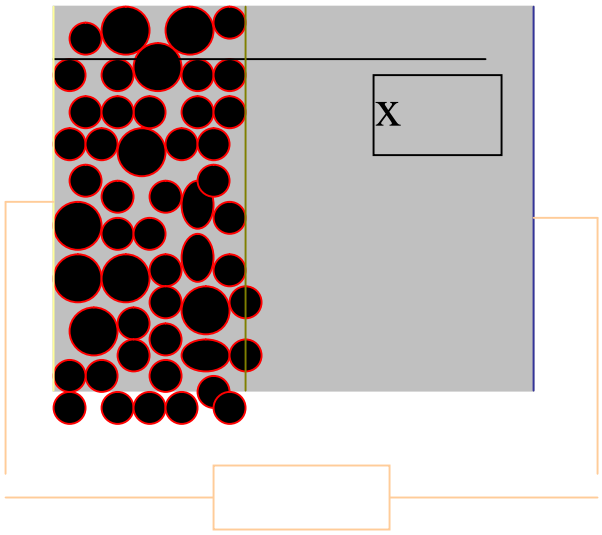
A macroscopic model for the concentration of s (the dye sensitizer) could look something like this:

kinetics:
ds(x,t) / dt = -k_3*s(x,t)*e(x,t)-k_4*s(x,t)*i(x,t)
(k_3 and k_4 rate constants e = electron concentration, i = iodine conc.)

diffusion
ds(x,t) / dt = D*d^2(s(x,t))/dx^2
(D = diffusion constant)

electric field E(x,t):
ds(x,t) / dt = my*ds(x,t) /dt*dE(x,t)/dt
(my = mobility for the species)

The electric field we get from integrating concentrations of all the charged species along x.



Explanations of each colour:

= the dye which is the charge carrier, giving new electrons to the dyes and get new one at the anode. It is a redox couple of Iodine. It can also react with the excited electrons which gives a less good efficiency. There are also other leakage's that contribute to decline

= Dye molecules, the electrons of those are excited at incident light of certain wavelengths

= The nanostructured semi conductor, most used is TiO2, the electrons diffuse in this medium towards the back contact.

= back contact (x=0), where the electrons go to get to outer circuit, anode.

= end of the nanostructured film, x=8*10^(-6)

= the "entrance" for the electrons from outer circuit, the cathode.

We start with the species in the solar cell

S+ = excited dye
S = dye
I- = Iodine
I3- = three iodine
I0 = iodine radical
I02- = di iodine radical

There are some reactions between the species during simulation with reaction rates k1..k6

{S+} + {e-} -> {S} k_1
{S+} + {I-} -> {I0} k_2
{I0} + {I-} -> {I02-} k_3
2{I02-} -> {I3-} + {I-} k_4
{I02-} + {e-} -> 2{I-} k_5
{I3-} + 2{e-} -> 3{I-} k_6

The concentrations of each species is defined as

s(x,t) = {s+} i(x,t) = {I-}
e(x,t) = {e-} w(x,t) = {I02-}
q(x,t) = {I3-} z(x,t) = {I0}

The starting conditions are (after a laser pulse there is excitation of the dyes s(x,0) and we look at the relaxation of all species after that)

s(x,0) = 360*10^(-9)*0.34*10^6*0.1*exp(-0.34*10^6*x)
i(x,t) = 0.5 e(x,t) = s(x,0)
w(x,t) = 0.0 q(x,t) = 0.05
z(x,t) = 0.0

The partial differential equations with electric field diffusion and reactions with the diffusion constants: Di, De, Dw, Dq, Dz and mobility constants: my_s, my_i, my_e, my_w, my_q we set the constants

diffusion:

Ds, Di, Dq, Dw = 1.5*10^(-9)
De = 200*10^(-9)

mobility:

my_s, my_i, my_w, my_q = 1.5*10^(-9)*1.602*10^(-19)/(1.38*10^(-23)*273)
my_e = 200*10^(-9)*1.602*10^(-19)/(1.38*10^(-23)*273)

rate constants:

k_1 = 1.0 * 10^(-6)
k_2 = 3.0 * k_1 k_3 = 4.0 * k_1
k_4 = 5.0 * k_1 k_5 = 6.0 * k_1
k_6 = 9.0 * k_1

movement from diffusion:

ds(x,t)/dt = 0.0 (stationary)
di(x,t)/dt = Di*d^2(i(x,t))/dx^2
de(x,t)/dt = De*d^2(e(x,t))/dx^2
dw(x,t)/dt = Dw*d^2(w(x,t))/dx^2
dq(x,t)/dt = Dq*d^2(q(x,t))/dx^2
dz(x,t)/dt = Dz*d^2(z(x,t))/dx^2

movement from electric force:

ds(x,t)/dt = my_s*s(x,t)*dE(x,t)/dx+my_s*E(x,t)*ds(x,t)/dx
di(x,t)/dt = my_i*i(x,t)*dE(x,t)/dx+my_i*E(x,t)*di(x,t)/dx
de(x,t)/dt = my_e*e(x,t)*dE(x,t)/dx+my_e*E(x,t)*de(x,t)/dx
dw(x,t)/dt =
my_w*w(x,t)*dE(x,t)/dx+my_w*E(x,t)*dw(x,t)/dx
dq(x,t)/dt =
my_q*q(x,t)*dE(x,t)/dx+my_q*E(x,t)*dq(x,t)/dx
dz(x,t)/dt = 0 (not charged)
! next comes from *Poisson's* equation
dE(x,t)/dx = 26.19925089*(s(x,t)-i(x,t)-e(x,t)-w(x,t)-q(x,t))

That makes the Electric field E=0 over the whole film at time t=0, (sum of all charges in the simulation cell will always be zero).

kinetics:

ds(x,t)/dt = - k_1*s(x,t)*e(x,t) - k_2*s(x,t)*i(x,t)
di(x,t)/dt = - k_1*s(x,t)*e(x,t) - k_3*i(x,t)*z(x,t)
de(x,t)/dt = - k_1*s(x,t)*e(x,t) - k_5*w(x,t)*e(x,t) - k_6*q(x,t)*e(x,t)
dw(x,t)/dt = - k_5*w(x,t)*e(x,t) + k_3*i(x,t)*z(x,t) - 2*k_4*w(x,t)^2
dq(x,t)/dt = k_4*w(x,t)^2 - k_6*q(x,t)*e(x,t)
dz(x,t)/dt = - k_3*i(x,t)*z(x,t) + k_2*s(x,t)*i(x,t)

x will be between 0 and 8*10^(-6) which is the thickness of the film where those reactions are.

At the boundaries we need to approximate the derivatives each time step for the diffusion and the electric field. There is no flow of particles out of these boundaries so we could set the concentrations change to zero at the boundaries; i.e. for the boundaries:
di(x,t)/dx=0 de(x,t)/dx=0
dw(x,t)/dx=0 dq(x,t)/dx=0
dz(x,t)/dx=0

So far, this is a *initial value problem* with six coupled nonlinear partial differential equations.

Future:
Making a full scaled 3 dimensional model and optimize the parameters for a optimal solar cell. Most of the parameters are adjustable, they are all dependent of the materials used. There are many different things said about what the cell efficiency really depends on. Some even say that the important thing is to have as good a cathode as possible and others say its something completely different. A model is needed to guide the research for a better solar cell.

Computer Code

```
Problem SolarCel
  include 'SolarCel.inc'
C  Ok we start with the species in the solar cell:
C ----------------
C  S+  = excited dye            S   = dye
C  I-    = Ioidine              I3-  = three iodine     I0   = iodine radical       I02- = di idodine radical
C  we have some reaktions between the species during simulation  with reaction rates k1...k6:
C ----------------
C  {S+} + {e-}   ->  {S}              k1
C  {S+} + {I-}   ->  {I0}             k2
C  {I0} + {I-}   ->  {I02-}           k3
C  2{I02-}       ->  {I3-} + {I-}     k4
C  {I02-} + {e-}  ->  2{I-}           k5
C  {I3-} + 2{e-}  ->  3{I-}           k6
C  the concentrations of each species is defined as:
C ----------------
C  s(x,t)  =  {s+}      i(x,t)  =  {I-}     e(x,t)  =  {e-}
C  w(x,t)  =  {I02-}    q(x,t)  =  {I3-}    z(x,t)  =  {I0}
```

```
C  The partial differential equations with electric field diffusion and reactions with the diffusion constants:
C  Di, De, Dw, Dq, Dz and mobility constants: mys, myi, mye, myw, myq we set the constants:
C  diffusion:
  constDs = 1.5*1.e-9  :   constDe = 200*1.e-9
  constDi = constDs    :   constDq = constDs  :  constDw = constDs
  print *,'Const.',constDs,constDi,constDe,constDw,constDq,constDz
C  mobility:
  mys = 1.5*1.e-9*1.602*1.e-19/(1.38*1.e-23*273)     myi = mys     :    myw = mys    :    myq = mys
  mye = 200*1.e-9*1.602*1.e-19/(1.38*1.e-23*273)
  print *, 'My.', mys, myi, mye, myw, myq, myz
C  rate constants:
  k1 = 1.0 * 1.e-6:    k2 = 3 * k1:     k3 = 4 * k1:   k4 = 5 * k1:    k5 = 6 * k1:   k6 = 9 * k1
  print *, 'Ks.', k1, k2, k3, k4, k5, k6
C  x will be between 0 and 8*1.e-6 which is the thickness of the
C  film where those reactions are:
   xfinal = 8*1.e-6:             xprint = xfinal / 100
  tfinal = 1.e2:      tprint = tfinal / 100:           dt = tprint / 10:     dx = xprint / 10
C  At the boundaries I suppose we need to approximate the derivatives
C  each time step for the diffusion and the electric field. There is
C  no flow of particles out of these boundaries so we could set the
C  concentrations change to zero at the boundaries; i.e for the boundaries.
   didx=0:             dedx=0: dwdx=0:dqdx=0: dzdx=0
C  the starting conditions are ( after a laser pulse there is
C  excitation of the dyes s(x,0) and we look at the relaxation
C  of all species after that):

  initiate JANUS; for distance; equations  &
      dsdx/s, d2idx/didx, didx/i, d2edx/dedx, dedx/e, d2wdx/dwdx,  &
      dwdx/w, d2qdx/dqdx, dqdx/q, d2zdx/dzdx, dzdx/z, dEsumdx/Esum;  &
      of x;  step dx; to xf;
  print *,'       TIME          DSDT          S          DIDT             I'
  xf=xprint
  do while (xf .le. xfinal)
    integrate distance; by JANUS
    print '(7(1pg13.5))', x, s, i, e, w, q, z
C         @curves('plot')
    xf=xf+xprint
  end do
C       @show('plot')
end
model distance
  include 'SolarCel.inc'
  s  = 360*1.e-9*0.34*10**6*0.1*exp(-0.34*10**6*x)
  i= 0.5:  e = s:  w = 0.:  q = 0.05:  z = 0.
C  movement from diffusion:
  dsdt = 0.             ! stationary
  didt = constDi * d2idx:  dedt = constDe * d2edx
  dwdt = constDw * d2wdx:  dqdt = constDq * d2qdx
  dzdt = constDz * d2zdx

  initiate ATHENA; for ide; equations  &
      dsdt/s, didt/i, dedt/e, dwdt/w, dqdt/q, dzdt/z; of t;  step dt; to tf;
  print *,'       X             TIME          DSDT          S          DIDT             I'
  tf=tp
  do while (tf .le. tfinal)
    integrate ide; by ATHENA
    print '(6(1pg13.5))', x, t, dsdt, s, didt, i
    tf=tf+tp
  end do
```

```
  find dsdx, didx, dedx, dwdx, dqdx, dzdx, dEsumdx;  &
      in eForce; by AJAX( cntrl1);  to match xs, xi, xe, xw, xq, xz, xEsum
C       @show('plot')
end
model ide   ! Implicit Partial Differential Equations
  include 'SolarCel.inc'
  find dsdt, didt, dedt, dwdt, dqdt, dzdt;  &
      in kinetics; by AJAX( cntrl1);  to match ts, ti, te, tw, tq, tz
end
model kinetics
  include 'SolarCel.inc'
C  kinetics:
  ts = dsdt - (- k1 * s * e  - k2 * s * I):          ti = didt - (- k1 * s * e  - k3 * i * z)
  te = dedt - (- k1 * s * e  - k5 * w * e -k6 * q * e)
  tw = dwdt - (- k5 * w * e + k3 * i * z - 2 * k4 * w**2)
  tq = dqdt - (k4 * w**2  - k6 * q * e):          tz = dzdt - (- k3 * i * z + k2 * s * i)
end
model eForce
  include 'SolarCel.inc'
C  movement from electric force:
  xs = dsdt - (mys * s * dEsumdx + mys * Esum * dsdx)
  xi = didt - (myi * i * dEsumdx + myi * Esum * didx)
  xe = dedt - (mye * e * dEsumdx + mye * Esum * dedx)
  xw = dwdt - (myw * w * dEsumdx + myw * Esum * dwdx)
  xq = dqdt - (myq * q * dEsumdx + myq * Esum * dqdx)
  xz = dzdt - 0                ! not charged
    ! next comes from poissons equation
  xEsum = dEsumdx - (s + i + e + w + q)       ! Objective is xEsum = 0 ???
C  I guess this is a initial value problem with six coupled nonlinear
C  differential equations.        /Jarl
end
controller cntrl1( AJAX)
  summary=0
end
```

Chapter 4 Exercises

1. What computer code statement in the above 'Body Plasma Chemistry' (see Application Problem 4.3) makes this a parameter estimation problem instead of an initial value problem?

2. Assume you are designing a new 'Nanostructured Solar Cell' (see Application Problem 4.4). What parameters might you be tweaking for a better Solar Cell design?

   What objective might you have (e.g. maximize energy gain OR minimize weight)? State objective in the form minimize/maximize _____ .

   After what statement would you put a FIND statement in computer code in order to find the optimal parameter values? This FIND statement would be followed by two more additional statements; an END and then a MODEL. These statements would be like the following:

   FIND p1, p2, p3, etc. in MyThing ooo to Minimize/Maximize ____

   End

   Model MyThing

5 Partial Differential Equations

A Parameter Estimation for Partial Differential Equations (PDEs) in an Initial Value Problem (IVP) or Boundary Value Problem (BVP) is solved using *Method of lines* (MOL) and the Calculus-level **'Find' statement** shown here:

IVP: **Find a** ooo To Match Error

BVP: **Find a, ydot0, y2dot0** ooo To Match Error

Where 'a' may be a vector with 'n' parts, a_1, a_2, a_3,…a_n;
ydot0, y2dot0,etc. are derivatives at independent variable = 0; and,
'error' is the objective function.

The 'find' statement is wrapped around an integrate and integration statement in order to solve the ODE. while finding the best 'a' parameter(s) for the given problem.

Ydot0, y2dot0, etc. variables may be arrays that are necessary to solve MOL problems.

The 'a' parameter(s) are varied to fit one's 'm' data points that make up the objective function, error. This technique can vary as many parameters as you want; e.g. 5 or 50 or 50,000. If there are less equations than parameters m < n, this would be classified as an under-determined system of equations. If there are more equations than parameters, m > n, this would be an over-determined system. Under- or Over-determined systems might force one to switch solvers to do the job.

Application Problem 5.1

PDEs: Stock Market to Biology

The following article was found on the http://www.brucelilly.com/particles.html website in 2012.

The following piece appeared in the Spring, 2002 issue of *Research and Creative Activity* magazine, a publication from Indiana University.

Jacob Rubinstein, Professor of Mathematics

"You give me anything, any area, from the stock market to biology, and I'll show you where partial differential equations appear."

The voice of Jacob "Koby" Rubinstein, professor of mathematics at IU Bloomington, bursts with enthusiasm as he points at the door to his office and launches into an explanation of how math figures in door manufacturing. And that's just the beginning. Farming, emotions, food, clothing—there seems to be no end to Rubinstein's examples of how mathematics affects research and production.

After explaining how math has helped makers of garage doors understand why a certain bar tended to break in the same place over and over again, he moves on to economics. "In the stock market, the main tool for the options market is partial differential equations," he says. "Now, every main brokerage firm is employing mathematicians and physicists to solve partial differential equations arising in the stock market."

Rubinstein, who came to Bloomington from Technion Israeli Institute of Technology in Haifa, Israel, has made a career out of connecting the ethereal world of higher math to the concrete world in which we live. An applied mathematician, Rubinstein has analyzed problems ranging from the behavior of superconductors at extremely low temperatures to the behavior of human beings in highly complex situations. (One example of his work in the latter area concerns auction theory, a subset of a field known as game theory). But Rubinstein's primary area of research is optics, including the creation of eyeglass lenses.

Many PDE problems require systems of PDEs to model them. Thus we are including a rough draft of solving a system of PDEs here.

Problem Description
We are converting the above Telegrapher's equations code (see **Application Problem** 5.3) from a PDE to a system of PDEs. The key change is converting variables into array variables; e.g. U becomes U(n).

Computer Code
The 'Find' statement is the work horse of a Calculus-level compiler. It calls ones math model as many times as necessary in order to converge on a solution. It varies your parameters, in this case (a, U0, & Ut0), as it calls your math model. The '**in**' phase tells the name of math model routine. The '**by**' phase tells what solver to use, Zeus here. And the '**to**' phase tells what the objective function is; '**minimize**' means converge objective function to a minimum value, 'errsum' variable in this case.

FIND a, U0. Ut0; IN tAxis; BY ZEUS; TO Minimize errsum

The following is just a rough sketch of necessary code to solve such a problem

```
global all
problem Telegraf(880000,15000,15000)
C -------------------------------------
C --- Calculus Programming example: Telegraph
Equation; a PDE Initial
C --- Value Problem solved.
C -------------------------------------
  real l
  dynamic U, Ut, Utt, Ut0, U0, Uend, UtEnd, U0Start
  nEqu= 3  ! # of PDEs
C
  ooo

  allot U(nEqu,ip), Ut(nEqu,ip), Utt(nEqu,ip),
Ut0(nEqu,ip)
  allot U0(nEqu), Uend(nEqu), UtEnd(nEqu),
U0Start(nEqu)
  ooo

model PDE  ! System of Partial Differential Equ.
C              ! Method of Lines
  do 20 ii=2,ipoints-1 !System of ODEs
    Utt(1,ii) = … equ. 1
    Utt(2,ii) = … equ. 2
      ooo
    Utt(nEqu,ii) = … equ. N
20  continue
end
  ooo
```

Application Problem 5.2

Burgers' Equation

(A non-linear Partial Differential Equation)

Problem Description

Burgers' Equation[12], a PDE, occurs as a model for a number of physical problems (e.g. Fluids, Heat, Traffic, Shock Waves, etc.). The equation is Ut + Ux U = vis Uxx, where U = U(x,t), Ux = Partial of U w.r.t. X, & vis = viscosity. Burgers' Equation is a nonlinear partial differential equation similar in structure to the Navier-Stokes Equation.

An example optimization problem with **Burgers' Equation** is found in Optimal Control for fluid flow. The problem is to determine the most inexpensive control that will produce a flow to match a given target. Solution: Add 1) the user parameters that can be varied in your model, 2) objective function, & 3) outer find statement. Then you are ready to solve your optimization problem. Now tweak, tweak, tweak until experience corrects your math model and objective function for your problem. (I'm speaking from experience; one job/problem took some two years to solve! Our math model and objective function had to be modified and modified and ... modified.)

Computer Code

The 'Find' statement is the work horse of a Calculus-level compiler. It calls ones math model as many times as necessary in order to converge on a solution. It varies your parameters, in this case 'a', as it calls your math model. The '**in**' phase tells the name of math model routine. The '**by**' phase tells what solver to use, ajax here. And the '**to**' phase tells what the objective function is; '**match**' means all following variables must equal zero, 'error' variable in this case.

FIND a; IN xAxis; BY AJAX; TO MATCH error

[12] The physicist Johannes Martinus Burgers (1895-1981) was a professor at the Technological University of Delft where he worked on turbulence. His simplified equation for turbulence is now what is called Burgers' Equation.

```
      global all
      problem BurgersPDE

C ----------- viscous Burgers' equation -----------
C
C     Ut = - U * Ux + viscosity * Uxx ... with Boundary
Conditions
C
C -----------------------------------------------------

C I'm new to PDE solving, so please check my work!
C Internet shows solutions to Burgers Equation that
differ greatly!!!
C Are the solutions different types; e.g. Steady State
vs. zzz?

C -----------------------------------------------------
C --- Calculus Programming example: Burgers' 1D
Equation; a PDE Initial
C --- Value Problem solved using Method of Lines.
C -----------------------------------------------------
C ... Warning ... a numerical problem exists when 'dt'
or 'viscosit' values
C ...   are too small.
C -----------------------------------------------------
      dynamic U, Ut, error
C
C User parameters ...
      viscosit = 1        ! viscosity between .1 & .001 are
of interest
      tFinal =  .5        ! not sure when odd numeric
problem surfaces
      jpoints = 10*tFinal  ! grid pts. over t-axis
C
C x-parameter initial settings: x ==> i
      xFinal= 1
      dx = .1
      ipoints = xFinal/dx + 1.99       ! grid pts. over x-
axis
      allot U( ipoints), Ut( ipoints), error( ipoints)

C t-parameter initial settings: t ==> j
      dt = .005:     tPrint= dt*jpoints:  pts = ipoints
      print 78, "viscosity, dt, dx, ipoints =", viscosit,
dt,dx,pts
 78     format( 1x, a, f5.3, 20(2x, f8.4))
C
      call xAxis
      end
      model xAxis
C ... Integrate over x-axis ... for a steady state solution
       last = 55       ! number of iterations for Steady
State solution
      do 10 i = 1, last
       t= 0:   tPrt = tPrint:  dt = tPrt / 10
       <error> = <U>
       Initiate ISIS;  for PDE;
    ~       equations Ut/U;  of t;  step dt;  to tPrt
       do while (t .lt. tFinal)
         Integrate PDE;  by ISIS
         if((t .ge. tPrt) .and. (i .eq. 1)) then
               print 79, t, (U( j), j = 1, ipoints)
          tPrt = tPrt + tPrint
         elseif((t .ge. tPrt) .and. (i .eq. last)) then
               print 79, t, (U( j), j = 1, ipoints)
          tPrt = tPrt + tPrint
         elseif(t .ge. tPrt) then
               print 78, "----------", i
          print 78, " "
          tPrt = tPrt + tPrint
         endif
       end do
 10        continue
       <error> = <U> - <error>
       print 78, " "
       print 78, "i & 'error' array follows = ", i
       print 79, t, (error( j), j = 1, ipoints)
       print 78, "----------"
       print 78, " "
 78     format( 1x, a, i8)
 79     format( 1x, f4.3, 20(1x, f8.5))
      end
      model PDE       ! Partial Differential Equation
C                     ! Method of Lines
C Boundary Conditions for Burgers' Equation
       U(1) = 1/(1 + Exp(-t/(4*viscosit))
       U(ipoints) = 1/(1 + Exp((2-t)/(4*viscosit))

            do 20 jj = 2, ipoints-1      ! System of ODEs
... Method of Lines
              Ut(jj) = -U(jj) * (U(jj+1)-U(jj-1)) / (2*dx)
              Ut(jj) = Ut(jj) + viscosit*(U(jj+1)-2*
U(jj)+U(jj-1))/(dx*dx)
 20     continue
      end
```

Computer Output for AJAX Solver:

viscosity, dt, dx, ipoints =1.000 0.0050 0.1000 11.0000

t	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10	U11
.025	0.50156	0.49998	0.49668	0.49047	0.48142	0.47014	0.45700	0.44174	0.42370	0.40252	0.37901
.025	0.50156	0.49998	0.49668	0.49047	0.48142	0.47014	0.45700	0.44174	0.42370	0.40252	0.37901
.050	0.50312	0.49811	0.49255	0.48567	0.47693	0.46604	0.45291	0.43753	0.42003	0.40080	0.38048
.075	0.50469	0.49786	0.49073	0.48271	0.47333	0.46227	0.44935	0.43458	0.41816	0.40044	0.38196
.100	0.50625	0.49815	0.48983	0.48085	0.47084	0.45953	0.44677	0.43255	0.41704	0.40052	0.38343
.125	0.50781	0.49874	0.48951	0.47977	0.46924	0.45771	0.44505	0.43126	0.41646	0.40090	0.38491
.150	0.50937	0.49956	0.48961	0.47928	0.46833	0.45661	0.44402	0.43056	0.41633	0.40152	0.38639
.175	0.51094	0.50054	0.49003	0.47923	0.46796	0.45609	0.44355	0.43034	0.41655	0.40232	0.38788
.200	0.51250	0.50164	0.49071	0.47954	0.46801	0.45602	0.44351	0.43050	0.41705	0.40328	0.38936
.225	0.51406	0.50285	0.49157	0.48012	0.46838	0.45630	0.44382	0.43096	0.41777	0.40436	0.39085
.250	0.51562	0.50413	0.49259	0.48091	0.46901	0.45685	0.44439	0.43165	0.41867	0.40553	0.39234
.275	0.51718	0.50548	0.49373	0.48187	0.46985	0.45762	0.44518	0.43252	0.41970	0.40677	0.39383
.300	0.51874	0.50687	0.49495	0.48296	0.47084	0.45856	0.44613	0.43354	0.42084	0.40807	0.39532
.325	0.52030	0.50830	0.49626	0.48415	0.47195	0.45964	0.44721	0.43467	0.42206	0.40942	0.39682
.350	0.52186	0.50975	0.49761	0.48542	0.47316	0.46081	0.44839	0.43589	0.42335	0.41081	0.39831
.375	0.52342	0.51123	0.49902	0.48676	0.47444	0.46207	0.44965	0.43718	0.42469	0.41222	0.39981
.400	0.52498	0.51273	0.50045	0.48814	0.47579	0.46340	0.45097	0.43852	0.42608	0.41366	0.40131
.425	0.52654	0.51424	0.50192	0.48956	0.47718	0.46477	0.45234	0.43991	0.42749	0.41512	0.40281
.450	0.52810	0.51576	0.50340	0.49102	0.47861	0.46619	0.45375	0.44133	0.42893	0.41659	0.40432
.475	0.52965	0.51729	0.50490	0.49249	0.48007	0.46763	0.45520	0.44278	0.43040	0.41807	0.40583
.500	0.53121	0.51883	0.50642	0.49399	0.48155	0.46910	0.45666	0.44425	0.43187	0.41956	0.40733

ELAPSED TIME = 19.28 SECONDS

Findings

It seems that Find statements with PDE models using *Method of Lines* require a lot of memory. To lower memory usage, use Find statement solvers that use only 1st order partials when possible; i.e. jacobian matrix or equivalent. Solvers using 2nd order partials as in the Hessian matrix use more memory.

Method of lines in *FortranCalculus* uses less memory than an algebraic language does. For example, a 2nd order ODE uses one dimensional arrays; a 3rd order ODE uses two dimensional arrays; etc. This is a hugh savings in required computer memory!

Application Problem 5.3

Telegrapher's Equation

Problem Description

The Telegrapher's Equation is of the following form: $c^2 \nabla^2 U = U_{tt} + (\alpha + \beta) U_t + \alpha\beta U$. We will solve it for the 1-dimensional case where there is no grid work necessary; i.e. method of lines is not used. The 2-dimensional case will require method of lines and thus a grid. The find statement will be used to find parameters a1, a2, a3 and initial condition U0 used to start integration process.

Computer Code

The 'Find' statement is the work horse of a Calculus-level compiler. It calls ones math model as many times as necessary in order to converge on a solution. It varies your parameters, in this case (a, U0, & Ut0), as it calls your math model. The '**in**' phase tells the name of math model routine. The '**by**' phase tells what solver to use, Zeus here. And the '**to**' phase tells what the objective function is; '**minimize**' means converge objective function to a minimum value, 'errsum' variable in this case.

FIND a, U0. Ut0; IN tAxis; BY ZEUS; TO Minimize errsum

```
global all
problem Telegraf(880000,15000,15000)
C -------------------------------------
C --- Calculus Programming example: Telegraph
Equation; a PDE Initial
C --- Value Problem solved.
C -------------------------------------
  real l
  dynamic U, Ut, Utt, Ut0
C
C User parameters:
  r=141.3:  aL=.1543:  c=15.72:  g=8.873
  alpha=g/c:  beta=r/aL:  c=sqrt(aL*c)
  ipoints=10  ! grid pts. over x-axis
  xFinal= 10  ! final x
  tFinal= 1   ! final time
C
C x-parameter init. settings: x ==> i
  ip=ipoints:  dx=xFinal/ipoints
C
C t-parameter initial settings: t ==> j
  tPrint=tFinal/ip:  yesno= 0
  allot U(ip), Ut(ip), Utt(ip), Ut0(ip)
C
  a=1    ! parameters to vary
  do 1 i = 1, ipoints
    Ut0(i)= 2
1 continue
  U0=-.2345:  Uend=3.4567:  UtEnd=9.8765
  Find a, U0, Ut0;  in tAxis;  by Zeus;  to minimize
errsum
  Yesno= 1:  Call tAxis ! print results
end
model tAxis
C settings at t = 0
  do 1 ii = 1, ipoints
    xPrt=ii * dx:  call fnctU0( xPrt)
    Ut(ii)=Ut0(ii):  U(ii)= U0start
 1 continue
  t=0:  tPrt=tPrint:  dt=tPrt/10
  Initiate ISIS;  for PDE; &
    equations Utt/Ut, Ut/U;  of t;  step dt;  to tPrt
  errsum=(0-U0)**2+(Ut(1)-Ut0(1))**2
  do while (t .lt. tFinal)
    Integrate PDE;  by ISIS
    if( t*yesno .ge. tPrt) print 79, t, (U(ii),ii=1,ip)
    tPrt=tPrt + tPrint
  end do
  errsum=errsum+(U(ip-1)-Uend)**2 + (Ut(ip-1)-UtEnd)**2
! BC at tFinal
  print 79, errsum, errsum
 79  format( 1x,f8.4,20(g14.5, 1x))
end
model PDE  ! Partial Differential Equ.
C          ! Method of Lines
  do 20 ii=2,ipoints-1 !System of ODEs
    Uxx=(U(ii+1)-2*U(ii)+U(ii-1))/(dx*dx)!4 2nd order in 'x'
    Utt(ii)= c**2 * Uxx - (alpha + beta)* Ut - alpha*beta*U
    ! add function with relation to a1,a2, & a3 parameters.
 20  continue
end
model fnctU0(xx)  ! Initial starting
  U0start = 0     ! values @ t = 0
  tmp = 1+(2*(xx-.75)/.157)**2
  if((xx.gt.0.5) .and. (xx.le.1.0)) &
    U0start=a*(x-.5)*(1-x)*(.5-1/tmp)
end
```

Computer Output for HERA Solver:

0.1000	-0.51264E-02	-9.2392	-9.2597	-9.2598	-9.2598
0.2000	0.19487	-36.540	-36.856	-36.857	-36.857
0.3000	0.39487	-82.261	-83.814	-83.826	-83.826
0.4000	0.59487	-146.34	-151.13	-151.20	-151.20
0.5000	0.79487	-228.32	-239.77	-240.01	-240.01
0.6000	0.99487	-327.39	-350.58	-351.29	-351.30
0.7000	1.1949	-442.47	-484.34	-486.05	-486.09
0.8000	1.3949	-572.24	-641.62	-645.32	-645.43
0.9000	1.5949	-715.26	-822.84	-830.06	-830.33
1.0000	1.7949	-870.01	-1028.2	-1041.2	-1041.8
1.1000	1.9949	-1035.0	-1257.6	-1279.7	-1281.0

ooo

0.1000	-0.22468	0.38391	0.38826	0.38827	0.38827
0.2000	-0.21550	1.0700	1.0943	1.0944	1.0944
0.3000	-0.20631	1.9948	2.0700	2.0710	2.0710
0.4000	-0.19713	3.0899	3.2668	3.2705	3.2706
0.5000	-0.18794	4.2841	4.6345	4.6454	4.6455
0.6000	-0.17876	5.5056	6.1216	6.1479	6.1485
0.7000	-0.16957	6.6841	7.6745	7.7303	7.7319
0.8000	-0.16038	7.7533	9.2371	9.3443	9.3484
0.9000	-0.15120	8.6522	10.751	10.941	10.950
1.0000	-0.14201	9.3272	12.157	12.472	12.490
1.1000	-0.13283	9.7327	13.391	13.885	13.920

Ran out of memory!!! Still not done converging to a solution.

Findings

Need more computer memory to complete this problem.

Chapter 5 Exercises

1. Have a PDE that you want solved? Give it a try with Method of Lines within *FortranCalculus*. How difficult was it? Easier using FortranCalculus than other languages? Estimate time savings over previous languages?
2. *Method of lines* is but one way to solve PDEs. If you know of another method, try solving **Burgers' Equation** using your method within *FortranCalculus*. Other methods are listed on Wikipedia's free encyclopedia at **Numerical partial differential equations**.

3. What variable(s) in above computer code statements for **Burgers' Equation** makes this a parameter estimation problem instead of a boundary value problem?

4. Assume you are designing a new black box that requires tweaking some parameters in the **Telegrapher's Equation**. Let's say parameters r, aL, c, & g might need some tweaking for a better black box design. Show the computer code statement(s) with changes that would be necessary for an improved solution.

6 Inverse Problems

Inverse Problems (IPs) are an important special set of problems that fit the statement "You know what you want, you just don't know how to get there." A proper directive can get an Inverse Problem solved in hours! Here are some examples of Inverse Problems:

a. Law enforcement: Shells found at scene where did it come from?
b. Airplane crash with wreckage all over the place. How did these parts get where they lay?
c. Missile target: Have target, how to get missile there?
d. Want a 'black box' to have an efficiency of 54.3%. How to design/build such a black box?
e. Car seat storage H x W x D slot, how to design seat so as it will fit into slot while maximizing seat comfort?

A Parameter Estimation for an Inverse Problem is solved using the Calculus-level **Find** statement shown here:

Find a ooo To Match Error

If a BVP then: **Find a, ydot0, y2dot0** ooo To Match Error

Where 'a' may be a vector with 'n' parts, a_1, a_2, a_3,…a_n;
ydot0, y2dot0,etc. are derivatives at independent variable = 0; and,
'error' is the objective function.

If the IP problem contains any differential equations, the 'find' statement is wrapped around an integrate and integration statement in order to solve the ODE or PDE while finding the best 'a' parameter(s) for the given problem.

The 'a' parameter(s) are varied to fit one's 'm' data points that make up the objective function, error. This technique can vary as many parameters as you want; e.g. 5 or 50 or 50,000. If there are less equations than parameters m < n, this would be classified as an under-determined system of equations. If there are more equations than parameters, m > n, this would be an over-determined system. Under- or Over-determined systems might force one to switch solvers to do the job.

Application Problem 6.1

Custom Thermistor Design

A **thermistor** is a type of resistor whose resistance varies with temperature, more so than in standard resistors. Several Silicon Valley companies (Aertech Industries of Sunnyvale, CA being first) in the 1970s needed to design a customized thermistor that would go through several custom data values. A customized circuit board design had up to four thermistors in series on the top branch and four more possible on the bottom branch. Thus, these two branches were in parallel. The Calculus-level code might be along the line shown here. The first pass has the solver summary turned off so user won't see all the execution going on. Once the best combination is discovered, the solver summary is turned on and the 'find' statement is re-calculated so the results will now be shown.

The following is just a rough sketch of necessary code to solve such a problem

Computer Code

```
Global all
Problem thermistor
  Yesno=0  ! summary report off
  R=.998    ! .2% Improvement factor

  call initialize
  call botOnly
  call parallel  ! comparing all combination of
thermistors;
! 0-4 on top level & 1-4 on bottom level

  Yesno=1:  ! summary report on
  If( jjtherm .eq. 0) then
    Allot abot(iitherm), alow(iitherm),
ahigh(iitherm)
    <abot>=1:  <alow>=1:  <ahigh>=ntypes

   Find abot; in therm; by thor(cntl) &;
     with lower alow and upper ahigh; &
     to minimize errsum  ! and thus Cost

  else
    ntherms=iitherm+jjtherm
    Allot abot(iitherm)), atop(jjtherm)),
alow(ntherms), ahigh(ntherms)
    <abot>=1:  <atop>=1:  <alow>=1:
<ahigh>=ntypes

    Find abot,atop; in therm; by thor( cntl); with lower
          alow and upper ahigh; to minimize errsum

  End if
  Errmin=errsum
End
Model botOnly
  If( yesno .eq. 0) then
    Errmin=9999:  iitherm=9999:  jjtherm=0
  End if
  Do 10, i=1,4
    Allot abot(i), alow(i), ahigh(i)
    <abot>=1:  <alow>=1:  <ahigh>=ntypes
    Find abot; in therm; by thor(cntl); &
      with lower alow and upper ahigh; &
      to minimize errsum  ! and thus Cost
    If(errsum .lt. r*errmin) then Errmin=errsum:
iitherm=i:
endif
10 continue
end
Model parallel     ! series
  jjtherm=9999
  Do 20, j=1,4
    Do 10, i=1,4
      ntherms=i+j
      Allot abot(i), atop(j), alow(ntherms),
ahigh(ntherms)
      <abot>=1:  <atop>=1:  <alow>=1:
<ahigh>=ntypes
      Find abot,atop; in therm; by thor(cntl); with
              lower alow and upper ahigh; to
              minimize errsum  ! and Cost
      If(errsum .lt. r*errmin) then
        Errmin=errsum:  iitherm=i:  jjtherm=j
      endif
10  continue
20 continue
end
controller cntl( thor)
  summary=yesno   ! solver only prints when
summary=1!
end
model therm
  <toptherm>=0:  <bottherm>=0
  if( jjtherm .eq. 0) then
```

```
   toptherm=1
   do 12,jj=1, j
    jbot=abot(jj)
    call thermMod( tdata, jbot)
    <bottherm>=<bottherm>+<rcalc>
12  continue
  else
   do 21,ii=1, i
    itop=atop(ii)
    call thermMod( tdata, itop)
    <toptherm>=<toptherm>+<rcalc>
21  continue
   do 22,jj=1, j
    jbot=abot(jj)
    call thermMod( tdata, jbot)
    <bottherm>=<bottherm>+<rcalc>
22  continue
  endif
  if( jjtherm .eq. 0) then
   <rcalc>=<bottherm>
  else
   do 33,ij=1, npoints
    rcalc(ij)=(toptherm(ij)
*bottherm(ij))/(toptherm(ij) +bottherm(ij))
33  continue
  endif
  err1=((rcalc(1)-rcalc(2))–(rdata(1)-
rdata(2)))**2  ! relative errors
  err2=((rcalc(1)-rcalc(3))–(rdata(1)-
rdata(3)))**2
  err3=((rcalc(3)-rcalc(2))–(rdata(3)-
rdata(2)))**2
  errsum=err1+err2+err3:  cost=(iitherm+jjtherm)
!*unit_prise
end
procedure initialize
  nytpes=123:   npoints=3
  allot tdata(npoints), rdata(npoints)  ! user data
  tdata=data(150, 181, 205)
  rdata=data(90, 76, 70)
  allot therms(ntypes, npoints), temps(npoints),
res(npoints)
  temps=data(150, 180, 200)
  therms=data(90, 74, 68,
              90, 75, 71,
               ooo  ('ntypes' datasets of 'npoints')
end
model thermMod(t, ijk)
  dimension t(*)
  res(1)=therms(ijk,1):  res(2)=therms(ijk,2):
res(3)=therms(ijk,3)
  y1=res(1)*(t(1)-temps(2))*(t(1)-temps(3))
  y2=(temps(1)-temps(2))*(temps(1)-temps(3))
  rcalc(1)=y1/y2
  y1=res(2)*(t(2)-temps(1))*(t(2)-temps(3))
  y2=(temps(2)-temps(1))*(temps(2)-temps(3))
  rcalc(2)=y1/y2
  y1=res(3)*(t(3)-temps(1))*(t(3)-temps(2))
  y2=(temps(3)-temps(1))*(temps(3)-temps(2))
  rcalc(3)=y1/y2
End
```

Findings

A program similar to this one above saved many man-hours in finding the best combination of in-house thermistors to build a customized thermistor as required by their customer. The customized circuit used the fewest number of in-house thermistors while also having the best fit to customer data. The design time was in seconds (actually overnight!) and saved up to two weeks on worst case problems.

This is an algebraic problem but it also is an inverse problem where one knows what they want, just need to find a way to get there.

Is this **circuit optimum**? While working through this calculus code, it seemed to me that a leading thermistor component would be very helpful. Unfortunately, no data for testing idea.

The ‘therms’ data in source code shows resistance at ‘npoints’ temperatures. Calculating a table showing the resistance difference between all ‘npoints’ could help one solve the tough custom thermistor problems. Do the same type table for your desired differences; i.e. ‘rdata’ values in source code.

Application Problem 6.2

Drug Development

Problem Description

A drug company is trying to develop a drug with a given/desired half-life. They know what they want, they just don't know how to get their, a true inverse problem (IP)! (Most IPs contain an implicit equation for solving.) They have several basic components that need to be combined to together to determine what combination will provide the best fit to their desire half-life. Here is the basic code for finding the best combination.

Note: The way the problem was first stated made it an inverse problem but it could also be classified as boundary value problem (BVP)

The following is just a rough sketch of necessary code to solve such a problem

Computer Code

The 'Find' statement is the work horse of a Calculus-level compiler. It calls ones math model as many times as necessary in order to converge on a solution. It varies your parameters, in this case (vol1, vol2, & vol3), as it calls your math model. The '**in**' phase tells the name of math model routine. The '**by**' phase tells what solver to use, Asolver here; e.g. AJAX. And the '**to**' phase tells what the objective function is; '**match**' means all following variables must equal zero, 'g' variable in this case.

FIND vol1,vol2,vol3**;** **IN** drugs**;** **BY Asolver;** **TO MATCH** g

```
Global all
Problem halflife
 prnt = 0:  desired = 1.234      ! desired half-life time
 R = 0.9998          ! .02% Improvement factor
 nQty = 2:  nDrugs = ???
 vol1 = 1:  vol2 = 0:  vol3 = 0
 Do 20 i=1, nDrugs
  Do 20 j = i+1, nDrugs

   Find vol1,vol2; in drugs;  by Ajax(cntl);  to Match g

   If(g .lt. R*errmin) then
    Errmin=g: idrug=i:  jdrug=j
   endif
20 continue
 nQty = 3:  vol1 = 1:  vol2 = 0:  vol3 = 0
 Do 30 i=1, nDrugs
  Do 30 j = i+1, nDrugs
   Do 30 k = j+1, nDrugs

   Find vol1,vol2,vol3; in drugs;  by Asolver(cntl);  to
Match g

   If(g .lt. R*errmin) then
    Errmin=g: idrug=i:  jdrug=j:  kdrug =k
   endif
30 continue
 nQty = 2:  prnt = 1   ! Print final results
 i=idrug:  j=jdrug:  k = kdrug:  vol1=1:  vol2=0:  vol3=0
 if( kdrug .lt. 1) then

   Find vol1,vol2; in drugs;  by Asolver(cntl);  to
Match g

 else
  nQty = 3

   Find vol1,vol2,vol3; in drugs;  by Ajax(cntl);  to
Match g

 endif
end
controller cntl for 'Asolver'
  summary = prnt   ! only prints when summary=1
end
model drugs
 dimension cl( 'nDrugs')  ! 'nDrugs' characteristic go
here
 data cl/ 0.1234, 987.54, etc. /  ! 'cl' = clearance factor

 p1 = vol1 / cl(i):  p2 = vol2 / cl(j):  p3 = 0
 if( nQty .eq. 3)  p3 = vol3 / cl(k)
 halfLif = .693 * (p1 + p2 + p3)
 g = (desired – halfLif)**2
 cost = ???:    sellPric = ???:    profit = sellPric - cost
end
```

Findings

This drug program would execute all 'nDrugs' models and determine which model was best. No output during the first pass. Once the best was determined, the 'prnt' switch would be turned on to allow for a summary output to be printed showing the resulting parameters vol1, vol2, & vol3.

Application Problem 6.3

Heat Transfer over 1D Slab

Problem Description

Heat Transfer over 1D slab surface; a PDE Inverse Problem solved using Method of Lines. Given data points at 600^{o} and 1200^{o} temperatures; i.e. U(x, 600) & U(x, 1200) at 'ipoints' over the x-axis.

Computer Code

The 'Find' statement is the work horse of a Calculus-level compiler. It calls ones math model as many times as necessary in order to converge on a solution. It varies your parameters, in this case (U0, aK, & h), as it calls your math model. The '**in**' phase tells the name of math model routine. The '**by**' phase tells what solver to use, Mars here. And the '**to**' phase tells what the objective function is; '**minimize'** means all following variables must converge towards zero, 'fitErr' variable in this case.

FIND U0, aK, h; IN tAxis; BY JUPITER; TO MINIMIZE fitErr

```
global all
problem Heat-IP(30000, 5000, 5000)
C ------------------------------------------------------
----------------
C --- Calculus Programming example: Heat
Transfer over 1D slab surface;
C --- a PDE Inverse Problem solved using
Method of Lines.
C ---
C --- When the geometry, the initial condition,
the boundary condition,
C --- material properties and the heat source term
are known, the temper-
C --- ature distribution, U(x, t), can be
calculated. These problems
C --- are then called the direct problems. On the
other hand, when any of
C --- this information, or a combination of them,
is unknown, but the
C --- field U(x, t) is known somewhere in the
space-time domain an estima-
C --- tion of the unknown quantities may be
attempted. These are known as
C --- the inverse problems.
C ------------------------------------------------------
----------------
  dynamic U, Ut, U0
  dynamic U600, U1200
C
C User parameters ...
  alpha = 2.e-5      ! thermal diffusion coefficient
C ans.  aK = 10.5 ! W/mK ! thermal conductivity
(in the solid)
C ans.  h = 22.5  ! W/m2K ! heat convection at
slab surface
  ipoints = 11       ! grid pts. over x-axis
  jpoints = 100      ! grid pts. over t-axis
  tFinal =  1200     ! not sure when odd numeric
problem surfaces
C
C x-parameter initial settings: x ==> i
  ip = ipoints
  xFinal=1:  dx=xFinal / (ipoints-1)
  allot U( ip), Ut( ip), U0( ip)
C
C t-parameter initial settings: t ==> j
  jp = jpoints
  tPrint = tFinal / jpoints

  Ua=1:  h=22:  U00=1:  ak=11:  prnt = 0
  print *, h, U00, ak, dx, tPrint
  do 10 i = 1, ipoints          ! Initial values for U0
array
    U0(i) = U00
 10  continue

  <U600>=data( 66.935, 78.466, 84.376, 86.697,
87.419, 87.602, &
     87.641, 87.649, 87.650, 87.650, 87.650)
  <U1200>=data( 61.632, 72.887, 80.239,
84.365, 86.361, 87.200, &
     87.509, 87.610, 87.640, 87.648, 87.650)

  find U0,aK,h; in tAxis; by Jupiter; &
    to minimize fitErr
  prnt = 1.:     call tAxis
end
model tAxis
C settings at t = 0
! print *, 're-start integration'
```

```
  do 10 ii = 1, ipoints
    U(ii) = U0(ii)
10 continue
  tPrt = tPrint:  dt = tPrt / 20:  t = 0:  fitErr = 0
C ... Integrate over t-axis
  Initiate ISIS;  for PDE; &
    equations Ut/U;  of t;  step dt;  to tPrt
  do while (t .lt. tFinal)
    Integrate PDE;  by ISIS
    if( t .ge. tPrt) then
      if( t .eq. 600) then  ! fit data points
        do 12 ik = 1, ip
          fitErr = fitErr + (U(ik) - U600(ik))**2
12      continue
      elseif( t .eq. 1200) then  ! fit data points
        do 13 ik=1,ip
          fitErr=fitErr+(U(ik) - U1200(ik))**2
13      continue
      end if
      if(prnt.eq.1) print 79, t,fitErr,(U(ii),ii=1,5)
      tPrt = tPrt + tPrint
    end if
  end do
79  format(1x,g10.4,1x,10(g11.5, 1x))
end
model PDE   ! Partial Differential Equation
C           ! Method of Lines
  do 20 ij = 2, ipoints - 1       ! System of ODEs
    tmp=(U(ij+1)-2*U(ij) + U(ij-1))  ! using
central difference
    Ut(ij)= - alpha * tmp / dx**2
20 continue
end
```

Computer Output for JUPITER Solver:

--- **JUPITER** SUMMARY, INVOKED AT HEATIP[26] FOR MODEL TAXIS ----

CONVERGENCE CONDITION AFTER 1 ITERATIONS
OBJECTIVE CRITERION SATISFIED
ALL SPECIFIED CRITERIA SATISFIED

LOOP NUMBER ...	[INITIAL]	1	
UNKNOWNS			
U0(1)	1.000000E+00	6.613667E+01	
U0(2)	1.000000E+00	7.236318E+01	
U0(3)	1.000000E+00	7.783022E+01	
U0(4)	1.000000E+00	8.202572E+01	
U0(5)	1.000000E+00	8.482188E+01	
U0(6)	1.000000E+00	8.642815E+01	
U0(7)	1.000000E+00	8.721404E+01	
U0(8)	1.000000E+00	8.753322E+01	
U0(9)	1.000000E+00	8.763345E+01	
U0(10)	1.000000E+00	8.765071E+01	
U0(11)	1.000000E+00	8.764430E+01	
AK	1.100000E+01	1.100000E+01	! aK & H missing from math
H	2.200000E+01	2.200000E+01	! model, error on my side!
OBJECTIVE			
FITERR	1.514486E+05	**9.652942E+01**	! needs better convergence

---END OF LOOP SUMMARY

12.00	0.00000	66.137	72.381	77.861	82.059	84.850
24.00	0.00000	66.137	72.400	77.892	82.093	84.879
36.00	0.00000	66.137	72.419	77.923	82.127	84.908
48.00	0.00000	66.137	72.437	77.954	82.162	84.937
60.00	0.00000	66.137	72.456	77.986	82.196	84.966

ooo

1176.	55.795	66.137	74.814	82.807	86.653	87.860
1188.	55.795	66.137	74.829	82.908	86.716	87.887
1200.	96.529	66.137	74.843	83.012	86.779	87.913

ELAPSED TIME = 77.45 SECONDS

Findings

Poor convergence; missing two parameters in model; a tough problem when one retires without any textbooks!

Application Problem 6.4

Robot Arm Movement

Problem Description

Design a mechanical/robotic arm that can move to any point P_k specified in a three-dimensional array "Points". What is the maximum allowable standard deviation in each arm or limb component when the standard deviation requirement around any point P_k is the radius ⑨$_r$?

The mechanical arm must be able to pick up an item laying on the floor and move the item to a point on a wall. The robotic arm's base is in the X-Z plane, D_Z feet from the wall, and centered around the P_k points that are in the X-Y plane. The destination points P_k form a rectangle on the wall. The robotic hand and rectangle corners (c_1, c_2, c_3, & c_4) are located in XYZ ordinates at

Arm_base = (0, 0, 0)
c_1 = (- .5 Width, Bottom, D_Z)
c_2 = (.5 Width, Bottom, D_Z)
c_3 = (.5 Width, Bottom + Height, D_Z)
c_4 = (- .5 Width, Bottom + Height, D_Z)
where Width = 10, Height = 5, Bottom = 2, and D_Z = 5

Rotation:
Find the angle θ for rotating the arm about the Y-axis such that the Z-axis component of any P_k point is zero. That is transform P_k into P_j through rotation as stated here:

$$\overline{P}_j = \begin{bmatrix} u_j \\ v_j \\ 0 \end{bmatrix}, \overline{P}_k = \begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix}, and\ \overline{Directional_cosine(2)} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

Solve:
Thus, find a θ that will yield the objective function $\overline{G_k}$ = (0, 0, 0) for the following definition

$$\overline{G_k} \equiv \overline{P}_j - \overline{Directional_cosine(2)}\,\overline{P}_k$$

Now that the arm lays entirely in a new X-Y (prime) plane, the problem is reduced to a 2-dimensional space. A 2-dimensional matrix, rotate(ϕ_i), will now be used to build the necessary math model for this mechanical arm. The rotation matrix rotate(ϕ_i) is defined as follows:

$$\overline{rotate(\phi_i)} = \begin{bmatrix} \cos\phi_i & \sin\phi_i \\ -\sin\phi_i & \cos\phi_i \end{bmatrix}$$

Assume any destination point P_k in the original X-Y plane is now transformed into the point P_j in the new X-Y plane.

$$\overline{P}_j = \begin{bmatrix} x_j \\ y_j \end{bmatrix}$$

$$\overline{Arm_end} = \sum_{i=1}^{n} \overline{Arm_joint_i}$$

$$where\ \overline{Arm_joint_i} = \lim b_i \angle \phi_i = \overline{rotate(\phi_i)} \begin{bmatrix} \lim b_i \\ 0 \end{bmatrix}$$

A new angle α_j and objective function, $\overline{G_j}$, are defined as

$\alpha_{j+1} = \phi_{j+1} - \phi_j$ = an angle relative to the previous $limb_j$

$$\overline{G_j} \equiv \overline{P_j} - \overline{Arm_end}$$

The following code has not been tested to solve such a problem. But seems complete and ready for a test case.

Computer Code

Here we have a nested group of three find statements. The 'Find' statement is the work horse of a Calculus-level compiler. It calls ones math model as many times as necessary in order to converge on a solution. It varies your parameters, in this case (Alimb, Angle, & Phi), as it calls your math models at varies levels. The bottom solver must converge before the one above takes its next small delta step in changing its parameter(s). Here 'Phi' parameter must converge before 'Angle' can change its value. Then 'Angle' must converge before 'Alimb' can change. One must think through this process before deciding whether to merge parameters onto one find statement or not.

Nesting 'find' statements is a very powerful technique to solve company wide problems.

FIND Alimb; IN Design; TO MINIMIZE Errors
FIND Angle; IN Rotate; TO MATCH Xprime(3)
FIND Phi; IN Joints; WITH UPPER Hi AND LOWER Low; TO MATCH Gjx, Gjy

```
Global All
Problem RobitArm
  Dynamic Boundary, Alimb, Phi, Alpha, hi, low
  Call Setup
! Find Nlimbs  In ArmMembers  To Minimize
Errors & Cost ?
  Do 10 Nlimbs = 2, 3
    Call ArmMembers
    Print *, ' Nlimbs = ', Nlimbs, ' With Errors = ',
Errors
10 Continue
End
Model ArmMembers
  Allot Alimb( Nlimbs), Phi( Nlimbs), Alpha(
        Nlimbs), Hi( Nlimbs), Low( Nlimbs)
  Do 20 ij = 1, Nlimbs
    Hi( ij) = 180:  Low(ij) = -180
20 Continue
  Find Alimb  In Design  To Minimize Errors
End
Model Design
  Errors = 0
  Do 30 ijk = 1, Npoints
    X1 = Boundary( ijk, 1):  X2 = Boundary( ijk,
         2):  X3 = Boundary( ijk, 3)
    Angle = 20:  FixedAxis = 3        !"Theta" =
         Angle
    Find Angle  In Rotate  To Match Xprime(3)
    Pjx = Xprime(1):  Pjy = Xprime(2)
    Find Phi  In Joints  With Upper Hi  And
         Lower Low &
      To Match Gjx, Gjy
    Errors = Errors + Gjx**2 + Gjy**2
30 Continue
  Errors = Errors / Npoints
End
Model Joints
  X2 = 0:  X3 = 0:  Xpos = 0:  Ypos = 0
  Do 40 ijk = 1, Nlimbs
    X1 = Alimb( ijk):  Angle = Phi( ijk)
```

```
   call Rotate
   Xpos = Xpos + Xprime(1):  Ypos = Ypos +
          Xprime(2)
40    Continue
  Gjx = Pjx – Xpos:  Gjy = Pjy – Ypos
End
Model Rotate
  Do 50 ii = 1, 3
    Do 50 jj = 1, 3
     a(ii, jj) = 0
50 Continue
  If FixedAxis .eq. 1 Then
    ii = 2:  jj = 3
  elseIf FixedAxis .eq. 2 Then
    ii = 1:  jj = 3
  elseIf FixedAxis .eq. 3 Then
    ii = 1:  jj = 2
  Else
    Abort
  End if
  Sign = 1
  If FixedAxis .eq. 2 then Sign = -1
  a( ii, ii) = Cosd( Angle):  a( ii, jj) = -Sign *
          Sind( Angle)
  a( jj, ii) = Sign * Sind( Angle):  a( jj, jj) = Cosd(
          Angle)
  Do ii = 1 To 3,  Xprime(ii) = a(ii,1) * X1 +
          a(ii,2) * X2 + a(ii,3) * X3
End
Procedure Setup
  Npoints = 4
  Allot Xprime(3), a(3,3), Boundary( Npoints, 3)
  Width = 10:  Height = 5:  Bottom = 2:  Dz = 5
  Do 70 i = 1, 4    ! corners of surface that the
          arm must reach
    Boundary(i, 1) = .5 * Width
    Boundary(i, 2) = Bottom
    Boundary(i, 3) = Dz
70 Continue
  Boundary(1, 1) = -.5 * Width
  Boundary(4, 1) = -.5 * Width
  Boundary(3, 2) = Bottom + Height
  Boundary(4, 2) = Bottom + Height
End
```

Application Problem 6.5

Plane Crash Locator

Problem Description

Help! The equations for modeling flight and debris floating in the ocean, are in need. If you know the equations for either model, please contact us. When models are found, they will be added here in future releases.

An airplane disappears with parts scattered here and there. Little is known about some of the crash parts found and we want to locate the crash point.

Lets assume we have 'nparts' with location coordinates and time when found. With a debris model (e.g. see Gyre Current model) and a few parameter estimates to vary, e.g. wind velocity, current velocity, etc. The crash program could estimate where the debris traveled over the last 5, 10, or more days. This would give a time-line for the debris.

Next, find the possible path of the airplane, given its last known contact point; i.e time, speed, & coordinates.

Any constraints? Any estimate of amount of fuel remaining in the plane should be entered as a limit. The maximum speed a plane can fly may be entered as a limit. Any other limits?

Now, let the crash program find a point on the debris time-line that this plane could get to. The crask program would vary the parameters, debris wind velocity, debris current velocity, plane direction and speed, in order to find a time point where the two curves meet; the crash point. Alas, a solution point!

Unfortuneatly this is a common problem. Here we will show how to solve this inverse problem. The larger number of items found the better for reconstruction purposes. We will work at determining a time-line of advents.

Computer Code

The 'Find' statement is the work horse of a Calculus-level compiler. It calls one's math model as many times as necessary in order to converge on a solution. It varies your parameters, in this case (h2oVel, windVel, plane velocity, etc.), as it calls your math models.

The following code is NOT tested! It is here to get one started on how to solve such a problem. The 'setup' routine has several parameters that need to be set for a true crash case. With no debris math model, this can't be run. Need to add a debris math model and some good true settings in setup in order to starting finding my typos, missing equations, proper conversions (e.g. minutes to hours), etc. (The 'dimension' statements should be replaced with an 'allot' statement. But there is a bug in the allot stmt., so be careful. My demos directory has some examples how to use the allot statement correctly and bypass the bug.)

The key point in solving this problem involves the parameter 'theSpot'. This is the time estimate for the crash. With 'theSpot' parameter value, the program calculates the flight time and can estimate the planes velocity and direction.

If the crash is a result of an explosion, the program should provide a good solution. If the plane goes wandering around for more than say a minute or two, it will be harder for the program to find a soltuion

```
    global all
    problem plane
C Requirement: At lease one piece ('nparts') of debris found of crashed airplane
      Dimension h2oVel(1), windVel(1), directon(1), velocity(1)
      Dimension xDebris(1), yDebris(1), tDebris(1), tDays(1)
      nparts = 1

    call setup
```

```
      do 10 ij = 1, nparts
        Find h2oVel(ij), windVel(ij), velocity(ij);
     ~     with lowers 1, 1, 1, 0;
     ~     with uppers 0, 0, 0, velMax;                    ! only constrain velocity, for now.
     ~     in crash; by Ajax( cntrl); to minimize gPlane
        xAverage = xAverage + xLoc(ij)**2:  yAverage = yAverage + yLoc(ij)**2
 10     continue
        xAverage = sqrt( xAverage/ nparts):    yAverage = sqrt( yAverage/ nparts)
        print *, ' Average or center location of debris = ', xAverage, yAverage      end
      model crash
C  Find the crash spot ... time, xloc, and yloc
       call location
C      distance = (plane velocity - wind velocity) * time
C      plane velocity = wind velocity + distance / time
       fltTime = (theSpot - tPlane)/60            ! flight time in hours
       distance = sqrt( (xloc(ij) - xPlane)**2 + (yloc(ij) - yPlane)**2 )     ! where is earth radius and Altidude?
       gPlane = (velocity(ij))**2 - (windSped + distance / fltTime)**2
       temp = arctan( (yEnd - yloc(ij)) / (xEnd - xloc(ij)))
       gPlane = gPlane + direction(ij)**2 - temp**2
      end
      model location
       x0 = xDebris(ij):  y0 = yDebris(ij):  t0 = tDebris(ij)
       dt = - t0 / 100       ! integrating from found location back to start, thus negative dt.
       g1 = y - y1  :  x = x1
       initiate isis; for debris;
     *     equations y2dot/ydot, ydot/y;
     *     of t; step dt; to theSpot
       integrate debris;  by isis
       terminate debris
       xloc(ij) = x:     yloc(ij) = y
       direction(ij) = arctan( (yPlane - yloc(ij)) / (xPlane - xloc(ij)))
      end
      model debris     ! Equations for math model of how debris travels on ocean waves.
       ooo             ! Litature refers to this as a Reverse Drift
        Equations must be a function of h2oVel(ij) & windVel(ij) parameters
        and must be continuous & differentable.
       ooo
      end
      subroutine setup
            xEnd = 987.12              ! miles ... x-location for destination
            yEnd = 4321.12             ! miles ... y-location for destination
            h2oVel(1) = 7.89           ! miles/hour ... estimated water velocity where debris was found
            windVel(1) = 3.21          ! miles/hour ... estimated wind velocity where debris was found
            xDebris(1) = 1234.56       ! miles ... x-location where debris found
            yDebris(1) = 987.65
            tDays(1) = 5               ! Days, 24 hour periods, debris in water since crash
            tDebris(1) = 13:45         ! time of day (24-hour clock) when debris was found
            xPlane = 431.1             ! miles ... x-location where last contact
            yPlane = 567.89
            windSped = -200            ! miles/hour ... Head-on wind speed ... in line with flight
            planeVel = 543             ! miles/hour ... plane's velocity at last known location
            velMax = 600               ! miles/hour ... plane's Maximum velocity
            planeDir = 45.67           ! degrees ... direction at last known location
            altitude = 34567           ! feet ... altitude at last known location ... not sure this is needed
            tPlane = 21:12             ! time of day when last contact
            planeTim = 12.34           ! minutes ... estimated time before crash
            planeTim = planeTim + tConvert( tPlane )              ! convert time to minutes
           ooo
```

7 Implicit Equations

An implicit equation has the form y = f(x, y) or yndot = f(x, y, ydot, y2dot, , yndot) so both sides of the equal sign have the y or yndot. The problem is getting the y or yndot both on the left side of the equal sign or somehow determining what the right values for y or yndot are that will make these true identities. By setting your objective function g = y – f(x,y) or g = yndot – f(x, y, ydot, y2dot, ooo , yndot) then the find statement will find the appropriate values for y or yndot in order to have g equal to zero.

Find y ooo To Match g
or
Find yndot ooo To Match g

Where **y** = f(x, **y**) or **yndot** = f(x, y, ydot, y2dot, , **yndot**); and,
'g' is the objective function.

Application Problem 7.1

System of Implicit Algebraic Equations

Problem Description

This is an example of solving a system of implicit algebraic equations. The value of y is found for each value of x and the result is printed. Substitute your own implicit algebraic equation for the ones given and the code below should solve it too.

Computer Code

The 'Find' statement is the work horse of a Calculus-level compiler. It calls ones math model as many times as necessary in order to converge on a solution. It varies your parameters, in this case (a, b, c), as it calls your math model. The '**in**' phase tells the name of math model routine. The '**by**' phase tells what solver to use, 'ABCsolver' here. And the '**to**' phase tells what the objective function is; '**match**' means all following variables must equal zero, 'g' variable in this case.

The 'cntrl' argument is the name of a controller block that has control variables for various things. (Need a manual for more on these variables.) Here we change the 'summary' variable to the global variable 'yesno'. When 'summary' equals zero, no output will be printed or created in output file. Once the solver converges for the first value of x, the following find statements should converge rapidly and thus no need to show these summary reports.

FIND a,b,c; IN drug?; BY ABCsolver(cntrl)**; TO MATCH g**

```
Global Reals
Problem implicitAlg
  yesno = 1:  xstep= .25
  y1=2:  y2 = 2:  y3= 1       ! Starting values
  do i=1, 2
    x= (i-1)*xstep
    Find y1, y2, y3; in equation; by Ajax( cntrl);
          to match g1, g2, g3
    y = y1 + y2 + y3
    Print *, x, y
  end do
  yesno = 0          ! turn off controller output …
                     not necessary
  do i=3 50
    x= (i-1)*xstep
    Find y1, y2, y3; in equation; by Ajax( cntrl);
        to match g1, g2, g3
    y = y1 + y2 + y3
    Print *, x, y
  end do
end
model equation
    den= 1.1 + (x / 37)**2 – y1*y1
    g1= y1 – [ -.139 * (x + 63) / den]
    den= 1.5 + (x / 28)**2 – y2*y2
    g2= y2 – [.777 * (x – 4.5) / den]
    den= 1.1 + (x / 26)**2 – y3*y3
    g3= y3 – [.003 * (x - 144) / den]
end
controller cntrl ( Ajax)
  summary = yesno        ! solver only prints
                         when summary = 1!
end
```

Computer Output for AJAX Solver:

--- **AJAX** SUMMARY, INVOKED AT IMPALG[10] FOR MODEL EQUATONS ----

```
 CONVERGENCE CONDITION AFTER  8 ITERATIONS
   UNKNOWNS NOT CONVERGED
   CONSTRAINTS SATISFIED
   ALL SPECIFIED CRITERIA SATISFIED
```

LOOP NUMBER ...	[INITIAL]	1	2
UNKNOWNS			
Y1	2.000000E+00	2.197415E+00	2.237531E+00
Y2	2.000000E+00	1.814254E+00	1.842004E+00
Y3	1.000000E+00	9.391304E-01	7.774956E-01
CONSTRAINTS			
G1	-1.019655E+00	-1.511682E-01	-4.090080E-03
G2	6.014000E-01	-1.374418E-01	-5.085348E-03
G3	5.320000E+00	2.920473E+00	1.649341E+00

ooo

LOOP NUMBER ...	[INITIAL]	7	8
UNKNOWNS			
Y1	2.000000E+00	2.238679E+00	2.238679E+00
Y2	2.000000E+00	1.843112E+00	1.843112E+00
Y3	1.000000E+00	-5.218140E-01	-5.221294E-01
CONSTRAINTS			
G1	-1.019655E+00	4.440892E-16	**4.440892E-16**
G2	6.014000E-01	-4.440892E-16	**-4.440892E-16**
G3	5.320000E+00	1.078324E-04	**1.453169E-07**

---END OF LOOP SUMMARY

0.000000000000000 3.55966162824445

--- **AJAX** SUMMARY, INVOKED AT IMPALG[10] FOR MODEL EQUATONS ----

CONVERGENCE CONDITION AFTER 3 ITERATIONS
UNKNOWNS CONVERGED
CONSTRAINTS SATISFIED
ALL SPECIFIED CRITERIA SATISFIED

LOOP NUMBER …	[INITIAL]	1	2
UNKNOWNS			
Y1	2.238679E+00	2.241173E+00	2.241177E+00
Y2	1.843112E+00	1.819766E+00	1.820453E+00
Y3	-5.221294E-01	-5.193115E-01	-5.193444E-01
CONSTRAINTS			
G1	-8.909878E-03	-1.470945E-05	-4.018297E-11
G2	1.023220E-01	-3.204482E-03	-2.997031E-06
G3	-9.645671E-04	1.153337E-05	1.557209E-09

LOOP NUMBER …	[INITIAL]	3	
UNKNOWNS			
Y1	2.238679E+00	2.241177E+00	
Y2	1.843112E+00	1.820453E+00	
Y3	-5.221294E-01	-5.193444E-01	
CONSTRAINTS			
G1	-8.909878E-03	**0.000000E+00**	**! good convergence &**

```
 G2                  1.023220E-01      -2.625455E-12   ! excellent values
 G3                  -9.645671E-04     0.000000E+00    ! in 3 iterations!


---END OF LOOP SUMMARY


  0.250000000000000     3.54228585238770
  0.500000000000000     3.52450507072720
  0.750000000000000     3.50623793444296

  ooo

   12.2500000000000      -0.807167908292338D-001
ELAPSED TIME =    0.06 SECONDS
```

Findings

Here the objective function g1, g2, & g3 were defined in the form g = (y) – [f(x, y)] and problem converged quickly. If convergence is slow, try squaring both side of equations and take the difference as your objective function; i.e. g = (y)**2 – [f(x, y)]**2. This often helps rate of convergence.

Application Problem 7.2

2nd Order Implicit Differential Equation

Problem Description

Couldn't find a good implicit equation from industry so this one was fabricated from **Application Problem 3.1**. The ODE equation was squared on both sided of the equal sign and then the equal sign was replaced with a minus sign. Then 'g' variable was set equal to this difference. This is the form one needs for solving an implicit equation. Now a find statement will find the highest order derivative term; i.e. y2dot in this case.

Computer Code

Implicit equations require an extra find statement in order to solve for the highest order derivative term as shown here. This second find statement just insures that the objective function, 'g' in this case, approximately equals zero. If so, then you have a value for your derivative that balances the equation; a solution point!

FIND y2dot; IN ide; BY Ajax; TO MATCH g

Same code as in Application Problem 3.1 except the following change:

```
model diffeqs
  kkk = kkk + 1
  if(kkk.eq.10) yesno = 0   ! This stops AJAX summary reports … Too much output
  FIND y2dot;   in IDE;   by AJAX(cntl);   to match g
end
model IDE          ! Implicit Differential Equation (IDE)
  g = y2dot**2 - (2 * ydot / x - (1 + a/x**2) * y)**2
end
controller cntl( ajax)
  summary = yesno                                    ! This flag reports only when yesno = 1
end
procedure aplot( plot77)
```

Computer Output for AJAX Solver:

Starting search for parameters to minimize |error|

--- **AJAX** SUMMARY, INVOKED AT DIFFEQS[63] FOR MODEL IDE ----

```
   CONVERGENCE CONDITION AFTER  0 ITERATIONS
     UNKNOWNS CONVERGED
     CONSTRAINTS SATISFIED
     ALL SPECIFIED CRITERIA SATISFIED

 LOOP NUMBER ...        [INITIAL]
 UNKNOWNS
   Y2DOT                1.000000E+00
 CONSTRAINTS
   G                    0.000000E+00
```

---END OF LOOP SUMMARY

--- **AJAX** SUMMARY, INVOKED AT DIFFEQS[63] FOR MODEL IDE ----

```
  CONVERGENCE CONDITION AFTER  2 ITERATIONS
    UNKNOWNS NOT CONVERGED
    CONSTRAINTS SATISFIED
    ALL SPECIFIED CRITERIA SATISFIED

 LOOP NUMBER ...       [INITIAL]            1                  2
 UNKNOWNS
   Y2DOT              1.000000E+00      9.663029E-01       9.657153E-01
 CONSTRAINTS
   G                  6.739429E-02      1.135498E-03       3.452120E-07

---END OF LOOP SUMMARY


--- AJAX SUMMARY, INVOKED AT DIFFEQS[63] FOR MODEL IDE ----

  CONVERGENCE CONDITION AFTER  1 ITERATIONS
    UNKNOWNS CONVERGED
    CONSTRAINTS SATISFIED
    ALL SPECIFIED CRITERIA SATISFIED

 LOOP NUMBER ...       [INITIAL]            1
 UNKNOWNS
   Y2DOT              9.657153E-01      9.661115E-01
 CONSTRAINTS
   G                 -7.651330E-04      1.569335E-07

---END OF LOOP SUMMARY
```

ooo

Findings

Lots of output! Results are good.

8 Nesting Solvers

Nesting is a very important feature. For example, if one needs to tweak ten parameters, all parameters could be in one Find statement and thus the solver's hessian or jacobian matrix would require (10*10) 100 storage cells. Break the 10 parameters into 5 & 5, where one Find statement is nested within the other, would reduce the storage requirement to (2*5*5) 50 storage cells. Resulting in a 50% savings in storage.

Another reason to nest is when one's parameters may have some dependency amongst themselves. A series of Sine curves fitting to data as shown in an earlier example is such a problem. It's parameters *frequency* & *theta* are dependent on their *amplitude*.

Nesting also seems to make the coding flow more natural as in the previous Matched Filter Design problem. The filters BandPass had *pole* parameters in the outer Find statement and *zero* parameters for the StopBand were in the inner Find statement.

Application Problem 8.1

Nesting … Matched Filter

(Nested Processes … Each Process controlled by a Solver)

Given n-data points from a Bode plot (see Figure 2.1 below) that define the **mainlobe** of the desired transfer function, find the optimal Pole/Zero constellation such that H(s) has equal **sidelobe** peak amplitudes in a Bode plot and H(s) curve fits the given data in the mainlobe.

Bode Plot: Mainlobe with 3 Sidelobes

Problem Description

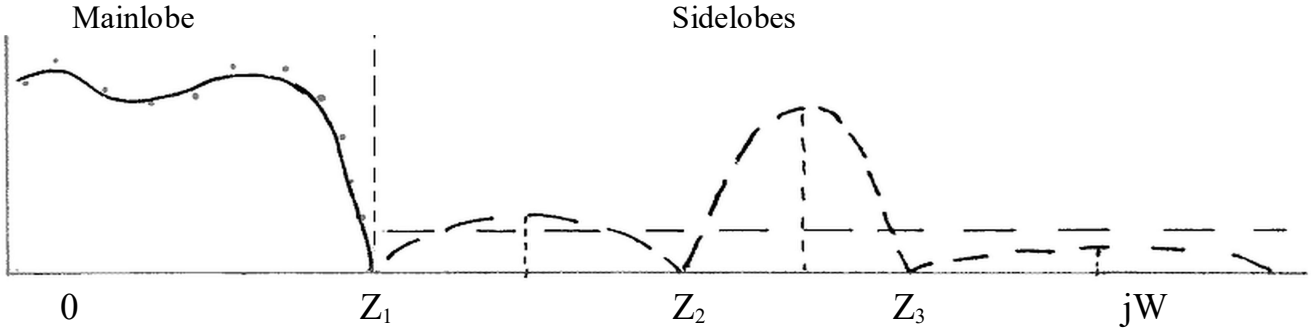


Figure 2.1 H(s) Mainlobe & Sidelobe Plot

Computer Plots

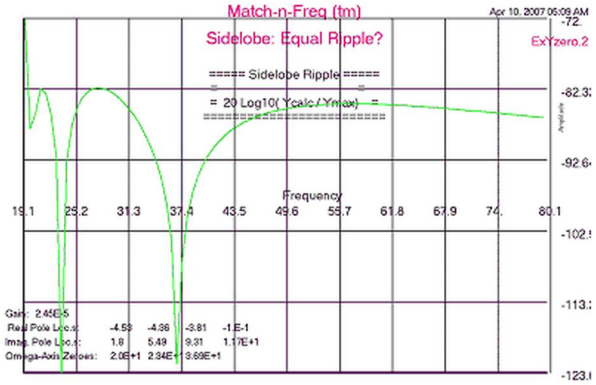


Figure 2.2b Equal Peaks in Sidelobes

Computer Code

The following find statement was used to determine the best Pole location for the transfer function H(s). Once the Pole locations are pretty well set, we nest another Find statement to determine where the Zeros, on the omega axis, will provide equal peaks in this stopband area of H(s). The following is the outer Find statement.

FIND gain, Preal, Pimag **IN Transfer** **BY** JOVE **TO MATCH** error

With good mainlobe parameters, this above find statement executes two nested Find statements to find the sidelobe parameters.

FIND xZeros **IN** Stopband **BY** HERA **WITH BOUNDS** sidelimits
TO MINIMIZE peak.diff

Computer Code

```
 PROBLEM FILTER
     ooo   ! Outer Find statement to determine Passband
 FIND gain, Preal, Pimag  in Transfer  by JOVE(contrl1)  &
  with lower h8low  and uppers h8hi  &
  MATCHING error  TO MINIMIZE errsum

     ooo   ! Middle Find statement to determine Stopband equal ripple
  FIND xZeros  in stopband  by Hera( contrl2) &
   with BOUNDS sidelims &
   TO MINIMIZE peak.diff

     ooo   ! Inner Find statement to locate peaks on omega axis.

    FIND peakloc( jj)  in sidelobe  by hera( contrl3) &
      with BOUNDS sidelim &
      TO MAXIMIZE peakampl( jj)
end
```

Computer Output for JOVE & HERA Solvers: (download freeware app for more output)

Findings

The results were great but tweaking the inner two find statements was a very delicate matter. A better objective would have simplified matters. For example, if equal ripple would be obtained when the area under the transfer function H(s) was minimized, that would have reduced the inner find statements to just one. But no one was such that minimizing the stopband area would guarantee that this would result in equal ripple.

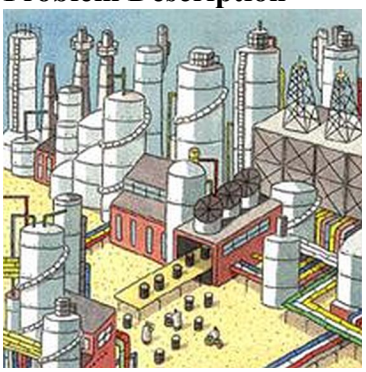
This problem was coded in a few days but required months of time to work through the math calculations behind the problem. The solution was optimal for the frequency domain. But truncation of numbers from converting time domain data to frequency data got us thinking that doing this problem in the time domain would be better. If for no other reason than small truncations in time domain didn't bother anyone. The time domain problem/solution is an exercise problem in chapter 2.

Solutions from this calculus problem were optimal and reduced development time from 3 months to less than a week.

Application Problem 8.2

Oil Refinery Production

Problem Description



An oil company may have many different distillation processes going on at each of its refineries. Each time new crude oil arrives at a refinery, or one distillation unit goes in or out of service, a new tweaking of all distillation parameters must be done in order to maximize the company's profit.

Here we will assume that the company has 'nDistUnits' at each refinery and 'nRefineries' for total number of refineries. The real issue will be agreeing on the company objective. Here we will use maximize profit while minimizing pollution.

Marketing effects: pollution is mainly seen as a cost item but it does not provide some profit to a company who's marketing group properly advertises the product as a 'green' product or has no detrimental chemicals in it? Check with your marketing group to create a math model showing some profit from products that have minimized pollution and add that model to this OilProduction program.

A monitor of present in stock inventories would be helpful for instant updating of ones model as used here.

Refinery Buy/Sell Options: Once one runs this OilProduction program a few times and is confident in its results, it would be time to start using it to play 'what if' games. For example, what if another 'B' refinery was up for sale and your company was trying to decide whether to buy it. Say this 'B' refinery is the same relative size with the same number of distillation units as your 'A' refinery, just a different location. Change your refinery 'A' location in this program and re-run it. The resulting increase or decrease in 'profit' will help your company decide to reject 'B' refinery or make an offer. Changing a refineries location will affect the distribution cost of your products. A new location might have a better grade of crude oil; another variable to test and see if the new value is worth it from a computer simulation view point.

Another use of this program would be deciding which refinery should get a new distillation unit first. Every one wants to have the latest gadgets in their backyard to show off but these can be very very expensive. So use this program to add a new unit to each refinery one at a time. This should give you some evidence to which site to start implementing new distillation units.

Computer Code

Our objective in the first level (or outer) find statement is to determine how much of the various products to produce at various refineries in order to maximize the companies profit.

FIND totProdPrct IN refineries BY JUPITER
MATCHING productErr TO MAXIMIZE profit

The goal of this 2nd level find statement is to determine product quantities to manufacture at each refinery and minimize pollution in the process. Limits are added to insure that the quantities are above some set values and below other values. Setting the upper limit to the lower limit will stop production of that one product; e.g. low(3) equals hi(3) then product(3) will be zero. There is another constraint that needs to be added to the find statement here. The total crude oil processed at each refinery must equal the total crude oil available; no more, no less. 'Matching crudeErr' addition to this 2nd 'find' statement will ensure the variable 'crudeErr' is (near) zero.

FIND qtyIn IN distillation BY JOVE;
with UPPER hi and LOWER low
MATCHING crudeErr TO MINIMIZE pollution

The following is just a rough sketch of necessary code to solve such a problem. Calculating the cost of manufacturing and distribution, pollution, profits of each product, etc. are left to the user. Each of these variables may require a math model of their own and knowledgeable people to be accurate enough for this OilProduction program.

```
global all
problem OilProduction
   nRefineries= 22:  nProducts= 33
   dynamic hi, low, totCrudeIn, ooo
   call setup    ! initial values
   call history  ! extrapolate 4 today's usage
   ! find product percentages for all Refineries in order to maximize profit.
   find totProdPrct     in refineries     by jupiter
      matching errsum      to maximize profit
End
Model refineries
   pollution=0:  profit=0:  cost=0:  errSum=0
   do i=1, nProducts
     do j=1, nRefineries
       sameProd(j) = totProdPrct(j, i) * totCrudeIn(i)
     end do
     crudeUsed = 0:  crudeErr=0
   ! finds qty production @ each refinery to minimize overall pollution

   ! to restrict prod., e.g. 3rd qty, set hi(3) = low(3)

      find sameProd    in processing    by Jove
           with upper hi    and lower low
           matching crudeErr      to minimize pollution

     do j=1, nRefineries
   errSum=errSum+ (sameProd(j) - totProdPrct(j,i) * totCrudeIn(i))**2
     end do
   end do
   ! find best routes to deliver products

     ooo

   find routes   in distribution   ooo   to minimize distPollution
   profit = profit - cost
end
Model distribution
  distPollution =0
  ! your (algebraic?) equations that model your distribution go here.

  ooo

  distPollution = distPollution + ???
End
Model processing! jth distillation unit @ refinery
! assume distillation requires solving a PDE or two.  So below is the bases for solving a PDE.
   t=0:  tPrt= tPrint
   do k = 1, nDistUnits( j)
      kDistModel = ???
```

```
      Initiate ISIS  for PDEquations   ooo
      do while (t .lt. tFinal)
        Integrate PDEquations    by ISIS
        if( t .ge. tPrt) print 79, t, (U(ii),ii=1,ip)
        tPrt=tPrt + tPrint
      end do
   end do
   crudeErr=crudeErr+(totCrudeIn(j)– crudeUsed)**2
79  format( 1x,f8.4,20(g14.5, 1x))
end
Model PDEquations
   if( kDistModel .eq. 1) then
        pde_1= pde equations with parameters
      ! assume # 3, 7, & 8 products are created
      qtyProd(3) = qtyProd(3) + ???
      qtyProd(7) = qtyProd(7) + ???
      qtyProd(8) = qtyProd(8) + ???
   elseif( kDistModel .eq. 2) then
        pde_2= pde equations with parameters
      ! assume # 2 & 8 products are created
      qtyProd(2) = qtyProd(2) + ???
      qtyProd(8) = qtyProd(8) + ???

      ooo

   elseif( kDistModel .eq. k) then
      pde_k= pde equations with parameters
      ! assume # 1, 2, & 8 products are created
      qtyProd(1) = qtyProd(1) + ???
      qtyProd(2) = qtyProd(2 )+ ???
      qtyProd(8) = qtyProd(8 )+ ???
   end if
   crudeUsed = crudeUsed + ???
   pollution= pollution + ???
   cost= cost + mfgCost + distCost + ???
   profit = profit + ???
end
procedure Setup
   allot totProdPrct(nRefineries, nProducts), hi(nProducts), low(nProducts), ooo

   ! today's available Crude Oil at different refineries
     <totCrudeIn>=data( …' available crude INPUT levels at each Refinery goes here' …>
   <totHi>= data( … storage limits for various products goes here …)
   <totLow>= data( … target amounts less inventory goes here …)
   <nDistUnits>=data( … # of distillation units @ each refinery goes here …)
End
procedure history
   ! here, use past history to estimate today's oil needs
     totQtyOut( 1,1)>=data( …' amount of crude oil to be targeted for today at the 1st Refinery goes here'
     totQtyOut ( 2,1)>=data( …' 2nd Refinery' …)
       ooo
     totQtyOut (nRefineries,1)>=data( …' nth Refinery' …)
end
```

This example shows nesting of find statements that will help maximize productivity. Getting agreement on what a companies objective is or should be may take some time. It is hoped that this example will aide you

on solving your problem with Calculus programming. Solve not just one equation but your entire problem/project in one program.

Influential Parameters
Grade of crude oil, air quality, types of distillation units available, number of distillation units available, etc. at each refinery are important parameters that will be necessary for this program to find the optimum solution for each day it is executed. These parameters need to be included in ones math model. Such parameters (and their derivatives) will aide the built-in solvers in finding the amounts, locations, & distillation units at each refinery in order to maximum company's profit.

Findings

A computer simulation may look good on paper but implementing the method may be a problem. In the 1960s or 70s, the Chevron refinery at Richmond, CA implemented a computerized monitoring system at each of their control rooms. It was found that the average employee started their eight hour shift by tweaking their controls to settings that they new were safe. For the rest of their shift they read books or did other things of self interest. Then the computer monitor was turned on along with the plant manager telling these controllers that they could earn gold or silver or red stars as rewards for doing a good job of improving oil production. The computer monitors would 'watch' their tweaking. If they went into an unsafe zone for any control, it would stop them. After a few weeks most controllers were tweaking their controls to maximize some oil production and thus were receiving some gold/silver/red stars.

This program built in-house competition that resulted in a huge increase in productivity. (I don't know prices of the 1960s or 70s. Let's us today's prices for this example.) Say that crude oil cost $50 / barrel and after refinement, sold for $100 / barrel. One element in the refining process was **Black Gold**. Say it increased one once per barrel. Today, Gold is selling for above $1,100 per ounce. Thus, this competition with computer monitoring, yielded a $1,000 / barrel interest in profit; a ten fold increase!

We are not talking peanuts here.

Computer Output: What if an output listing shows zero volume of oil should be produced by the k^{th} distillation unit at the j^{th} refinery. If it continually shows a zero volume for several weeks, then it might be saying its time to replace the unit. Or, maybe you need to add a new product such as the tire companies did when they added shoes to their production line. Time to think outside the box.

Future

Maintaining a program such as the one described here is relatively simple. If the number of products or refineries or distillation models changes then update the number in code. If the company objective changes then some more code may need to be added and/or deleted. It's pretty simple!

Keeping your distillation (math) models updated is essential. Each refinery must routinely verify that the models are correct for their refinery. Here is where you will spend most of your time for keeping this type program valid.

Feedback Request

How many companies are interested in solving their similar problem? At present, this program may not work due to amount of storage necessary. In order to fix this storage problem, we need your values for 'nRefineries' & 'nProducts'. The product of nRefineries * nProducts may be the problem. Assume each is 100 then their product is 10^4. Internal arrays (e.g. jacobian) is the square of this product, 10^8! Knowing that

the majority of problems would work with 10^n would provide a target value for future releases of Calculus compilers.

If interested in solving such production problems please contact FortranCalculus.

9 Miscellaneous

Application Problem 9.1

Monte Carlo Simulation OR Total Derivative?

Exact Derivative Calculations

Problem Description

For those wanting a *tolerant design or analysis* using the Monte Carlo Simulation method to estimate a derivative, why not try a Calculus-level compiler in order to do such calculations exactly to the number of digits your computer will allow.

The total derivative of a function is stated mathematically as $dF = \sum \frac{\partial F}{\partial c_i} dc_i$ where c_i equals the i^{th} component of the total project. The total variance would be $dF^2 = \sum \left(\frac{\partial F}{\partial c_i} \right)^2 dc_i^2$ where the partial derivatives, $\frac{\partial F}{\partial c_i}$, may be calculated for you.

The following code is just a rough sketch of necessary code to solve such a problem

Computer Code

```
global all
problem totalDerivative
   ooo
  invoke GRADIENTS on var1, var2, var3   in equat
   ooo
  print *, 'df, Dvar1, Dvar2, Dvar3=', df, Dvar1, Dvar2, Dvar3
end
model equat
   ooo
  f = function of ...,var1,var2,var3, ...
  Dvar1= 1.234:  Dvar2= 9.8765:  Dvar3= 543.21
  df=sqrt((#PARTIAL(f,var1) * Dvar1)**2 &
    + (#PARTIAL(f,var2) * Dvar2)**2 + (#PARTIAL(f,var3) * Dvar3)**2)
   ooo
end
```

Stiff Equations & Trouble Shooting
Application Problem 9.2
Stiff Equations

What is the definition of a Stiff Equation?

Wikipedia: definition: "In mathematics, a **stiff equation** is a differential equation for which certain numerical methods for solving the equation are numerically unstable, unless the step size is taken to be extremely small. It has been proven difficult to formulate a precise definition of stiffness, but the main idea is that the equation includes some terms that can lead to rapid variation in the solution."[13]

PROSE's definition: "An ODE system is termed **stiff** when it contains greatly differing time constants or oscillation frequencies. Conventional integration techniques are impractical for such systems because they require inordinately small step sizes to achieve acceptable accuracy and stability."[14]

Scholarpedia's **definition: "Stiff systems** of ordinary differential equations are a very important special case of the systems taken up in Initial Value Problems. There is no universally accepted definition of stiffness. Some attempts to understand stiffness examine the behavior of fixed step size solutions of systems of linear ordinary differential equations with constant coefficients. The eigen values of the Jacobian matrix completely characterize the stability of the system in this case. They also determine the behavior of explicit numerical methods applied to the system."[15]

Elsewhere on the Web: "**stiff differential equations** are those with two or more widely differing scales. For example, the solution could have a component that quickly becomes insignificant, and another component that changes much more slowly."

Since there is no clear definition of a stiff equation, we will enlarge this problem class to include all ODEs/PDEs that have difficult converging on a solution.

Problem Description

Flat Spots: Solvers have numerical difficulties whenever ones equation(s) have one or more flat spots, i.e. say five or more consecutive points that make a flat line. This situation will provide a solver with a derivative value of approximately zero and eventually lead to numerical problems when trying to converge on a solution.

---

[13] As stated on Wikipedia's webpage http://en.wikipedia.org/wiki/Stiff_equation , on June 14, 2011.
[14] "PROSE Calculus Manual" 5-11
[15] As stated on Scholarpedia's webpage http://www.scholarpedia.org/article/Stiff_systems, on June 14, 2011.

Trouble Shooting

Calculus-level programming minimizes the user's code necessary to describe ones math problem. When an execution problem arises here are some steps to locate ones errors:

1. Replace all 'find' statements with 'call' statements

   e.g. replace 'find a, b; in MyEqs; o o o'

   with 'call MyEqs'

2. Place a print statement in your math model procedure that prints some variables that should be changing upon each execution of your model.

3. Place a print statement at beginning of your 'MyEqs' routine called by your 'find' statement. This print should contain some text for use in search through your output; e.g. print "starting". After execution, search for your text and see if other values printed seem to jump; i.e. values are larger after your text printed. If so, this may suggest your 'dx' or 'dt' or ??? is too larger. Decrease by factor of ten and rerun problem. Now does a search show a jump in value size at your printed text? Is it improved? If so, continue dropping your delta step size until satisified.

Are the variables changing as expected? Common problem: variable(s) are not passed into ones math model due to missing in common block or input parameter list. Fix the link error(s) and retry. Problem gone? If not find other variables that are not getting into your model. (Note: Highly recommend using some form of a 'global' statement to pass variables.)

4. Try various solvers in each 'find' statement.

A common error message from solvers says something along the line of 'your model produces a zero jacobian matrix'. This error may mean some parameters, that you are varying, are missing a link and thus have values of zero or you have an underdetermined system this is ill-conditioned or unstable.

Underdetermined system may exist if the number of equations is less than the number of independent variables. For example, if you have the find statement 'find a, b, c; in MyEqs; ooo' try removing 'c' variable then 'b' variable to see if that gets your 'find' statement working like it should. If decreasing the number of (independent) variables fixes the problem, then consider:

a. Try other solvers;

b. Add some bound statements, i.e. Lower and/or Upper; or,

c. Try better initialization values for variables/parameters. Suggest trying values of zero, 1e-4, 1, 11, 111, etc. If one or more independent variable is a frequency then your initial value may need to be good to three significant digits. Setting the amplitude parameters high for initial values may help the solver find the frequency values. (Note: highly recommend showing a plot with initial value settings in order to view ones problem. A sinusoidal problem with zero amplitude will make it impossible for a solver to solve!

10 Conclusions

A 'find' statement is the work horse of a Calculus language. It is used in parameter estimation, boundary value problems, implicit equation problems, inverse problems, etc.. The find statement's solver varies parameters in ones model until the stated goal is achieved. Different solvers use either the jacobian or Hessian matrix to estimate where to jump next with ones parameter values. The partials are calculated using 'automatic differentiation' (AD) and thus are as exact as one's computer.

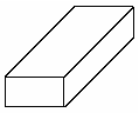
How good is your math model?

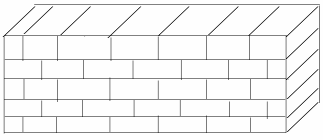
Are you sure that all effects are accounted for in your math model? People comment more on 'bad' math models than on 'good' models. For example, what is the 'worst condition' versus 'best condition' for a forest fire? Asking about the 'worst' got more comments. People seemed to have more to say or were willing to say something regardless of their background.

Keep your models up to date. Calculus Compilers make that easy to do. After all, a good math model is worth its weight in gold.

10.1 Future: Thinking outside the box

One's Vision

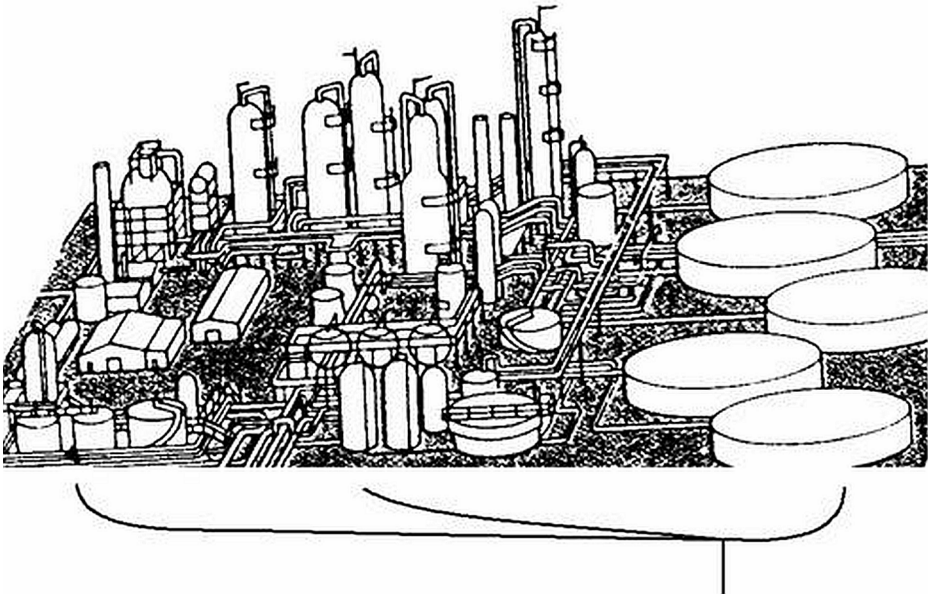




Mr. Arithmetic	Mr. Algebra	Mr. Calculus
Before Computers	*With Computer, Gained some vision*	*Optimize the Whole Show in One Run*
Process Methodology:		
One Step at a Time	Simulate Problem on Computer	Find Optimal Solution. Must 'See' Entire Problem & Objectives

Today, most individuals are working on getting their math model to provide an accurate description of one component of a project. We would suggest moving past that and on to considering all components of a project or site or company. For example, those working at an oil refinery, they may be modeling one distillation unit. Why not consider modeling the whole refinery? This is

now possible with a Calculus-level compiler. Finding the right objective may be an issue for total project simulation. This may require input from engineers to president of company.

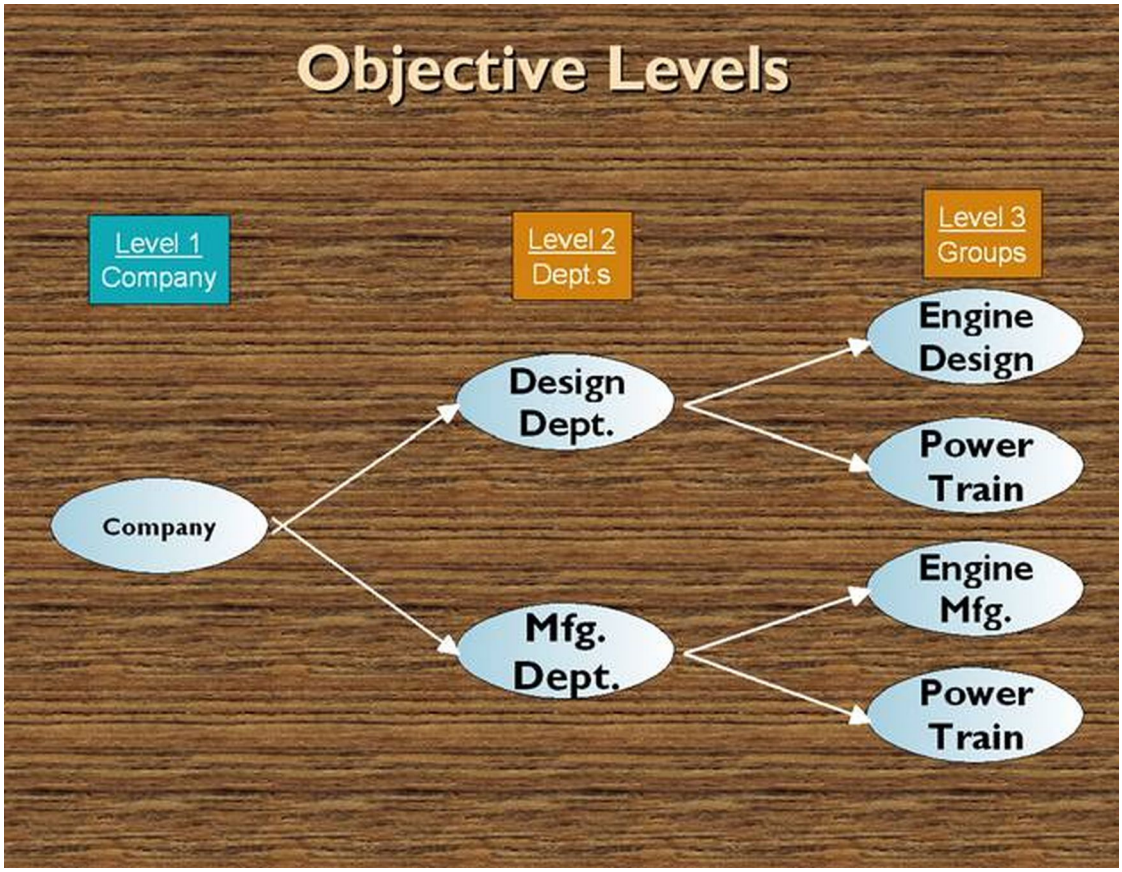


Where are you going?

What is your Company's Objective?

Most companies should have several levels of objectives as shown in the chart below. All Company level objectives must be known by all employees and clearly stated in order to achieve all goals/objectives. For example, a person sweeping the floor might change from a left-to-right movement to a right-to-left movement if they knew that some objective might be improved upon by such a change in their work. Get the word out, "Our company's goal is ooo ".

Keep it simple, like a slogan, and easy to remember. Limit your Company level objective to one or two key words; i.e. maximize **profit** while minimizing **pollution**. The more confining the objective, the less freedom for creative solutions.



In the 1970s, I asked one of my key leaders what our company goal was in the disc drive industry. He replied 'minimize cost'. In this world of computers, minimize cost would be equivalent to minimize cost regardless of profit, quality, sales, etc. On the other hand, maximize profit as a goal/objective would minimize cost whenever possible and did NOT affect profit. Sales, quality, etc. would be effected ONLY IF they did not reduce profit. When explained this way, the senior leader changed to 'maximize profit' as our goal. Pick your goal/objectives carefully! Get input from everyone to be sure you didn't overlook some important effect of your goal.

What is *Your* Project Objective(s)?

Minimize cost
Maximize Profit
Minimize Pollution
Maximize Productivity
Minimize Weight of Products
Maximize Green Energy Usage
Minimize Customer Complaints

All The Above!

Conclusions

Calculus-level languages offer an easy way to solve many parameter estimation problems. These quick solutions help one keep their math models up to date and debugged. But more accurate answers may be of little use if a Statistical Process Control (SPC) program or similar program is not used to monitor one's production process. How does one know that their process is doing the right job if they have no process monitoring system? Is the process producing goods that are good to the n^{th} degree; 'n' is your choice? If one's solution is to be reproduced then a Statistical Process Control program must be in use at one's company in order to achieve the solutions from Calculus-level software.

It is hoped that the examples in this textbook are helpful and prove the point that Calculus-level programming is simple. Now you need to prove it to yourself. Write your own Calculus code and execute it. Did it save you time? Was it easy to debug and get a reasonable answer? How many parameters did you vary in an execution?

Give FortranCalculus a try!

11 Appendix

Picking the right Solver

It sure is nice having solvers in a library where a user can pick from and not get involved into all their codes. There are two items to consider when picking a solver.

1. Are all your parameters independent of each other? If so, a 1st order (e.g. jacobian) solver should do the trick for you (e.g. Ajax). For a 2nd order solver example, curve fitting a sine series [a_i sin($freq_i$ t + $theta_i$)] to data has dependant parameters (a_i, & $freq_i$) and thus needs a solver using the 2nd order partials (e.g. Hessian matrix) to resolve this conflict.

2. Memory in short supply? If so, stay with 1st order solvers using the jacobian matrix or the like.

'aplot' source code

```
procedure aplot( plot77)
  character*(*) plot77
  @plots( 'error', 1)  ! error plot
  @plots( plot77, 0)  ! measured data vs. calc.
curve
end
procedure plots( plot77, ierror)
  common /rr/ v(3), vc(3), pw50(3), t0(3),
+  npoints, deltat, data(100), time(100),
error(100)
  character*(*) plot77
  real*8 signal( 100)
  @graph(plot77, '2dgraph')
  xmin= -300:    xmax= 300   ! time( npoints)
  ymin= 1.e10:   ymax= -ymax
  do 10 i= 1, npoints
    if( ierror .ne. 1) then
      signal(i)= error(i) + data(i)
      if( ymin .gt. data(i)) ymin= data(i)
      if( ymax .lt. data(i)) ymax= data(i)
    else
      signal(i)= 1000 * error(i)
    end if
    if( ymin .gt. signal(i)) ymin= signal(i)
    if( ymax .lt. signal(i)) ymax= signal(i)
  10      continue

@window(plot77,100,500,50,400,xmin,xmax,
ymin,ymax, 0,0,0,1,1.5)
  xstep= (xmax - xmin)/8
  @xaxis( plot77, xmin, xmax, xstep, 0, 1, 1)
  ystep= (ymax - ymin)/6
  @yaxis( plot77, ymin, ymax, ystep, 0, 1, 1)
  @xelabel( plot77, 9, 'Time (ns)', 11)
  if( ierror .ne. 1) then
    @setup( plot77, 'pp', 0, 14, -2, 0) ! profile
npoints (light yellow)
    @setup( plot77, 'cr', 0, 10, -2, 0) ! profile
curve (light green)
    @yelabel( plot77, 14, 'Amplitude (v)', 11)
  else
    @setup( plot77, 'er', 0, 12, -2, 0) ! error
curve (light red)
    @yelabel( plot77, 20, 'Error Amplitude
(Mv)', 11)
  end if
  @label( plot77, 25, 'Isolated (ReadBack)
Pulse', 13, 190, 460,0)
  if( vc(1)+vc(2)+vc(3) .eq. 0) then
    @label( plot77, 13, 'with out "vc"', 14, 245,
446, 0)
  else
    @label( plot77, 9, 'with "vc"', 14, 255, 446,
0)
  end if
  do 20 j=1, npoints
    if( ierror .ne. 1) then
      @curve( plot77, 'pp', time(j), data(j))
      @curve( plot77, 'cr', time(j), signal(j))
    else
      @curve( plot77, 'er', time(j), signal(j))
    end if
   20   continue
  @show( plot77)
end
```

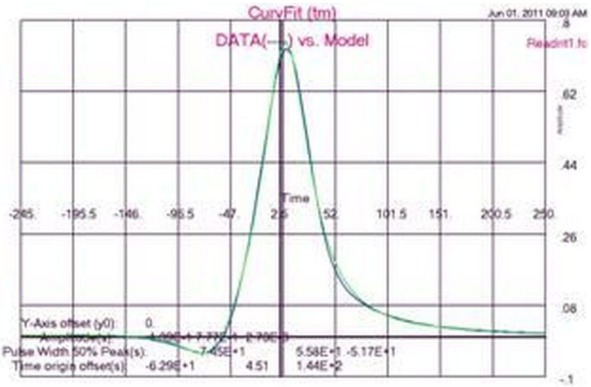
Spectral Estimation (freeware) Software

How to find key frequencies in y(t)? This question has been worked on by many over the years. The freeware SpectrumSolvers program, download at https://goal-driven.net/apps/spectrumsolvers.html, provides some 12 methods that give frequency estimates of your time series data.

SpectrumSolvers (tm) has a menu of Spectral estimators from Steve Kay's textbook, titled "Modern Spectral Estimation", 1988. The results differ dramatically from one estimator to another as shown by one test case where all methods were used and their results shown in Steve's book. The 'true' Power Spectral Density (PSD) plot is shown in the center of his page.

At the bottom of a run, SpectrumSolvers lists all key (i.e. peaked) frequencies. It is highly recommend that one uses these frequencies as starting values when trying to solve for a frequency parameter(s) in a math model. Start with the frequencies that are the strongest (i.e. highest amplitudes) in your output listing from SpectrumSolvers.

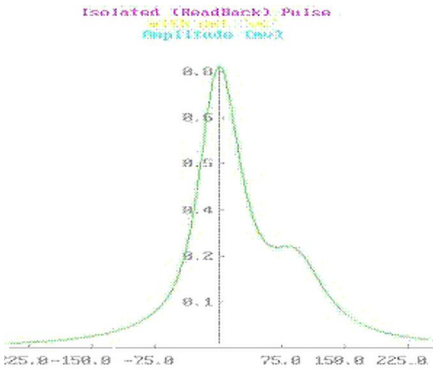
'readrit1.100' File Listing



------- Lorentzian Series ... test case --------- Optimal Designs Enterprise
Y-offset (Y0): 0.0000
Amplitude(s): -0.2000 0.9000 -0.1700
Pulse Width @50% Peak(s): 88.00 70.00 71.00
Time origin offset(s): -54.30 6.660 35.70
---> No. of data points: 100
========== Header lines = 7

-245	.0042924609	-75	-.039934312	95	.061273445
-240	.0043593047	-70	-.039400857	100	.056191676
-235	.0044219848	-65	-.034280079	105	.051671955
-230	.0044793115	-60	-.022768189	110	.047635172
-225	.0045298648	-55	-.0032151243	115	.044017227
-220	.0045719490	-50	.025564682	120	.040764939
-215	.0046035393	-45	.064135166	125	.037833452
-210	.0046222163	-40	.11251947	130	.035184494
-205	.0046250875	-35	.17043669	135	.032785148
-200	.0046086919	-30	.23748947	140	.030606944
-195	.0045688848	-25	.31309585	145	.028625155
-190	.0045006980	-20	.39603570	150	.026818228
-185	.0043981701	-15	.48359078	155	.025167328
-180	.0042541421	-10	.57045233	160	.023655954
-175	.0040600098	-5	.64796448	165	.022269617
-170	.0038054259	0	.70475713	170	.020995564
-165	.0034779442	5	.72970477	175	.019822551
-160	.0030625975	10	.71655238	180	.018740643
-155	.0025414051	15	.66731736	185	.017741042
-150	.0018928092	20	.59172363	190	.016815943
-145	.0010910543	25	.50319606	195	.015958408
-140	.00010554257	30	.41441986	200	.015162258
-135	-.0010997576	35	.33459370	205	.014421979
-130	-.0025667058	40	.26854898	210	.013732640
-125	-.0043429180	45	.21719202	215	.013089826
-120	-.0064807673	50	.17877824	220	.012489573
-115	-.0090346669	55	.15043056	225	.011928321
-110	-.012055358	60	.12930122	230	.011402861
-105	-.015579208	65	.11312956	235	.010910298
-100	-.019609577	70	.10033113	240	.010448019
-95	-.024086370	75	.089869354	245	.010013655
-90	-.028839435	80	.081085418	250	.0096050618
-85	-.033523058	85	.073559765		
-80	-.037534698	90	.067018123		

‘readrit2.200’ File Listing



---- Lorentzian Series ... test case ----- Optimal Designs Enterprise

Y-offset (Y0): 0.

Amplitude(s): -.070 .8000 .2000

Pulse Width @50% Peak(s): 88.00 77.00 99.00
Time origin offset(s): -40.00 .0000 111.0
---> No. of data points: 200
========== Header lines = 7

-297	.01412870	-129	.05984182	39	.4424565
-294	.01439184	-126	.06215114	42	.4175903
-291	.01466242	-123	.06459535	45	.3953065
-288	.01494072	-120	.06718525	48	.3755265
-285	.01522705	-117	.06993283	51	.3581384
-282	.01552171	-114	.07285141	54	.3430114
-279	.01582504	-111	.07595593	57	.3300068
-276	.01613736	-108	.07926313	60	.3189835
-273	.01645904	-105	.08279196	63	.3098015
-270	.01679047	-102	.08656392	66	.3023225
-267	.01713203	-99	.09060362	69	.2964089
-264	.01748415	-96	.09493937	72	.2919214
-261	.01784725	-93	.09960396	75	.2887158
-258	.01822181	-90	.1046356	78	.2866390
-255	.01860831	-87	.1100794	81	.2855254
-252	.01900726	-84	.1159883	84	.2851929
-249	.01941920	-81	.1224253	87	.2854408
-246	.01984470	-78	.1294658	90	.2860490
-243	.02028436	-75	.1371992	93	.2867806
-240	.02073881	-72	.1457328	96	.2873869
-237	.02120872	-69	.1551938	99	.2876170
-234	.02169480	-66	.1657326	102	.2872302
-231	.02219779	-63	.1775246	105	.2860109
-228	.02271849	-60	.1907714	108	.2837841
-225	.02325773	-57	.2056995	111	.2804288
-222	.02381640	-54	.2225562	114	.2758872
-219	.02439544	-51	.2416019	117	.2701674
-216	.02499585	-48	.2630972	120	.2633404
-213	.02561868	-45	.2872868	123	.2555296
-210	.02626505	-42	.3143780	126	.2468973
-207	.02693617	-39	.3445161	129	.2376283
-204	.02763330	-36	.3777576	132	.2279146
-201	.02835779	-33	.4140398	135	.2179418
-198	.02911108	-30	.4531474	138	.2078791
-195	.02989471	-27	.4946768	141	.1978733
-192	.03071031	-24	.5379979	144	.1880457
-189	.03155963	-21	.5822193	147	.1784914
-186	.03244453	-18	.6261670	150	.1692813
-183	.03336702	-15	.6683907	153	.1604648
-180	.03432923	-12	.7072152	156	.1520725
-177	.03533344	-9	.7408511	159	.1441205
-174	.03638211	-6	.7675654	162	.1366129
-171	.03747787	-3	.7858934	165	.1295446
-168	.03862355	0	.7948501	168	.1229042
-165	.03982218	3	.7940875	171	.1166757
-162	.04107706	6	.7839522	174	.1108398
-159	.04239169	9	.7654240	177	.1053757
-156	.04376989	12	.7399552	180	.1002616
-153	.04521577	15	.7092559	183	.09547578
-150	.04673377	18	.6750800	186	.09099647
-147	.04832872	21	.6390534	189	.08680289
-144	.05000585	24	.6025644	192	.08287509
-141	.05177082	27	.5667154	195	.07919419
-138	.05362984	30	.5323222	198	.07574241
-135	.05558965	33	.4999442	201	.07250314
-132	.05765762	36	.4699282	204	.06946092

207	.06660142	240	.04405223	273	.03132216
210	.06391134	243	.04259523	276	.03045188
213	.06137842	246	.04121010	279	.02961801
216	.05899135	249	.03989223	282	.02881854
219	.05673970	252	.03863739	285	.02805159
222	.05461388	255	.03744164	288	.02731542
225	.05260503	258	.03630135	291	.02660838
228	.05070503	261	.03521315	294	.02592894
231	.04890638	264	.03417392	297	.02527568
234	.04720218	267	.03318077	300	.02464726
237	.04558608	270	.03223102		

Arbitrary Equalization with Simple LC Structures

Robert Kost, MEMBER IEEE, and Philip Brubaker

Abstract-*Equalization for magnetic recording with LC filters was reported in 1963 [1], and since then many other approaches have been used to alter the readback signal to reduce error. These ideas have been extended to arbitrary input-arbitrary output fillers which are realized as LC structures without mutual inductance. An asymmetrical signal from an isolated pulse is equalized to become optimum in the linear Van der Maas sense [2]. The change in the signal to noise ratio as a result of equalization is computed as a function of pulse slimming.*

INTRODUCTION

Implicit in efficient utilization of a communication channel is proper signal design. This can be illustrated by noting that the Nyquist limit cannot be achieved for an arbitrary symbol (pulse) shape, but only for symbols that have the proper zero crossings. In general then, equalization will be required to effectively use the available bandwidth. If the readback signal can be viewed as coming from a linear system which has a restricted set of input signals, a linear filter can be used to remove intersymbol interference. The conditions under which this notion is valid were reported in 1969 and 1978 [3,4]. If the equalizer is viewed as a windowed inverse filter, it is clear, at least in principle, that the readback signal can be altered to more effectively utilize the bandwidth.

This is a report of a frequency domain design of an equalizer with the input frequency function derived directly from an isolated readback pulse. The output frequency function is the linear Van der Maas quasi-optimum approximation. The equalizer's pole-zero constellation is determined by using a nonlinear optimization routine available in the PROSE language [5]. The filter is realized so that mutual inductance is not possible [6]. Additionally, the realization can be accomplished with closed form expressions without recourse to insertion loss filter design.

Since the equalizer affects the signal to noise ratio, a discus-sion of the minimum signal to noise change is included.

Input Signal Acquisition

The Fourier transform (FT) of an isolated readback pulse is computed by taking the Fourier transform of signal samples (FT*) [7]. Since the time data are rectangularly windowed and band limited the FT* is a least-square fit to FT [8]. Because of this, FT* is least-square fitted to estimate FT. The time function, t(ω), is obtained by taking the negative derivative of the phase function. The input frequency function is described in the following way:

$$X(\omega) = |X(\omega)| \exp\left(-j \int_0^{\omega} t(x)\,dx\right) \quad (1)$$

The magnitude and the time functions are shown in Fig. 1. It is interesting to note that there appears to be a discontinuity at the origin in the phase function. No fundamental reason was found for this.

Output Signal Design

For systems that use peak detection, loosely stated requisites for a signal are that it be narrow and the sidelobe disturbance be low. These were the criterion that were used to design the pulse that Vakman refers to as quasi-optimal [2].

This pulse was designed to give the narrowest pulse for a specified bandwidth and sidelobe suppression. The width of the pulse (distance between zero crossings) for 60 dB sidelobe suppression is 15.48/WB where WB, is the bandwidth.

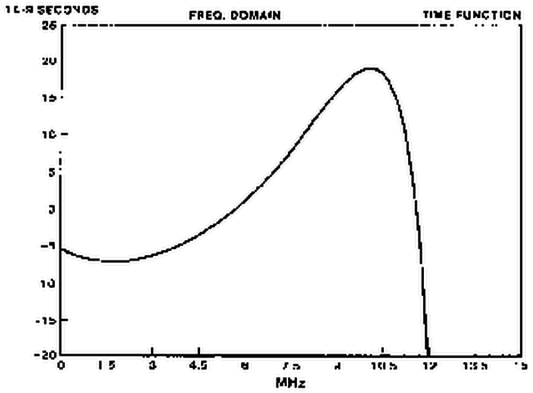


Fig. 1a. Time function of isolated readback signal

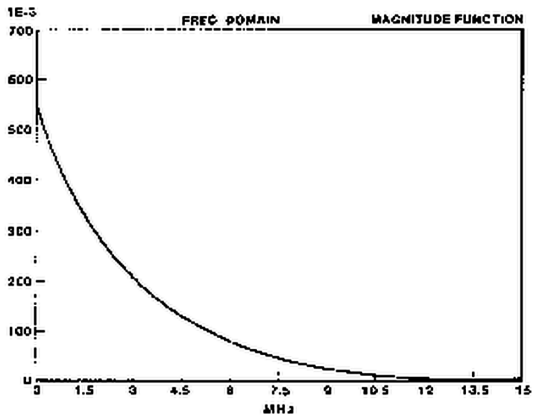


Fig. 1b. Magnitude function of isolated readback signal.

FILTER APPROXIMATION

It should be noted that since this is going to be an LC filter realization, the group delay of the filter is completely a function of the pole locations. The zeros do not contribute to the filter group delay. With this in mind, the design process is broken into three stages:

1) adjust the filter pole locations only until the output frequency function's time function, t(ω), is approximately constant.

2) adjust the zero locations (while holding the pole locations constant) until the magnitude of the output frequency function is satisfactory.

3) adjust all critical frequencies simultaneously while constraining the maximum group delay error.

Consider the following definitions:

X(ω) = input frequency function

Hj(ω) = equalizer transfer function at j'th iteration

Yj(ω) = equalizer transfer function at j'th iteration

Yd(ω) = equalizer transfer function at j'th iteration

C(ω) = equalizer transfer function at j'th iteration

where:

$$Yj(\omega) = Hj(\omega)\, X(\omega) \qquad (2)$$

The objective function, θj, to be minimized is defined to be:

$$\theta j = \sum_k E_j^2(\omega_k)$$

where

$$Ej(\omega) = \frac{(Yj(\omega) - Yd(\omega))\, C(\omega)}{Yd(\omega)} \qquad (3)$$

and ω_k are discrete values of ω.

The equalizer's pole-zero constellation is obtained by iteratively adjusting the root locations until the objective function is minimized. The calculation is done by a nonlinear optimization routine available on PROSE, while the filter is being driven by the frequency function of an isolated pulse.

FILTER REALIZATION

During the approximation portion of the design, terminated reactance two port realization conditions were carefully observed. This will guarantee that the equalizer can be built as an LC filter. These realizability conditions, however do not guarantee a filter without mutual inductance or negative element values. In addition to these problems, the LC structures often have impractical element values. All of these problems can usually be avoided by the synthesis approach that is now going to be described.

The basic topology to be used is referred to as "additive amplification" [9,l0,11,12]. This topology involves injecting currents into nodes of an LC ladder filter. The output voltage of this design is the sum of the voltages due to the individual current sources, hence the name "additive amplification."

Consider a voltage-controlled current source driving node r of an all-pole singly-terminated LC filter. The output voltage due to this single-current source is well known and is given by [13]:

$$V_o^{(r)} = \frac{gm^{(r)} z^{(r)} R\, Vi}{Z_{zz} + R} \qquad (4)$$

where $V^{(r)}_o$ is the voltage across R due to current sourcing at node r and gm is transconductance of the current source.

Using equation (4), the transfer function of the filter will be:

$$H(s) = \sum_r \frac{gm^{(r)} z_{12}^{(r)} R}{z_{22} + R}$$

The transconductance of the r'th source can be related to the transconductance of the 1'st source (unterminated end of LC filter) by a multiplicative constant:

$$gm^{(r)} = (c_r) gm^{(1)}$$

Also, the transfer impedances between the nodes and the output can be related. Using these ideas in equation (5) yields:

$$H(s) = \frac{gm^{(1)} z_{12}^{(1)} R \,(1 + (s^2 a_2 + 1) c_2 + \ldots + (s^{2k-2} a_k + \ldots) c_k)}{z_{22} + R} \quad (7)$$

where k is the number of nodes being driven by current sources and a_k are constants that relate the component values of the filter.

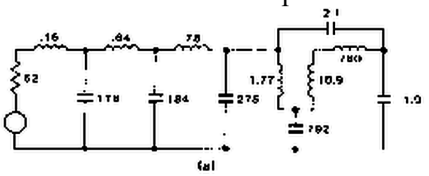
It is clear that the transfer function of this realization is related to the LC ladder transfer function by a multiplicative even polynomial. This results in k-1 unknowns and k-1 linearly independent equations.

Since all-pole LC filters are guaranteed not to contain mutual inductance and the element values are nearly always positive and do not change by more than about a factor of ten, this realization procedure circumvents many of the problems attendant with insertion loss design.

Two filters with the same pole-zero constellation are shown in Fig. 2. The first was designed with standard insertion loss techniques while the second is similar to Fig. 6-11 in [12]. The improvement, as far as practical implementation goes, in the "additive amplifier" approach is self-evident.

The all-pole filter can be realized by using insertion loss theory (the driving point impedance is the ratio of the even and odd parts of the transfer function numerator) or closed form expressions can be derived. The closed form expressions can be derived by expressing the transfer function in terms of its pole locations and in terms of its element values. By equating the coefficients of the denominator of these two transfer functions, a set of linear equations will be formed that will result in closed form expressions for the element values in terms of the pole locations.

(a)



(b)

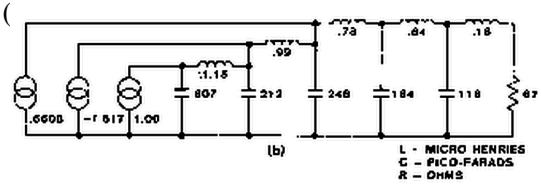


Fig. 2a. Comparison of an insertion loss realization

b. With a multi-input realization

FIGURE OF MERIT

An important consideration for an equalizer design involves the change in the signal to noise ratio (SNR) introduced by the equalizer. Here the SNR is defined as the peak signal to the rms noise voltage. The figure of merit (FM) of the filter is the ratio of the input to the output SNR expressed in dB. The computation was done numerically with the input signal being Lorentz, the output signal being Van der Maas and the noise power spectral density was taken directly from a disc. The FM as a function of the ratio TM/PW50 is shown in Fig. 3 where TM is one-half the distance between the zero slope points on the Van der Maas (VDM) time function.

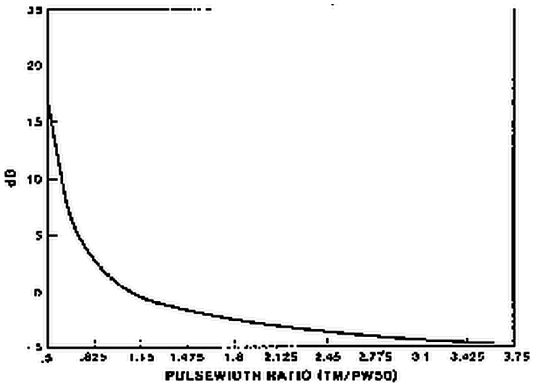


Fig. 3. Figure of merit

FILTER DESIGN EXAMPLE

An equalizer to remove intersymbol interference in the time derivative of the slimmed output is now designed to illustrate the ideas discussed thus far. The input signal

has a PW50 of about 110 ns and the output VDM frequency function has a cutoff frequency of 12.36 MHz. Initially, the pole locations of the filter were adjusted to equalize the group delay to 10 MHz. This resulted in a time function error (deviation from a constant) of 1.5 ns. Then the zeros were adjusted to minimize the magnitude error. A SPICE analysis of the equalizer showing the input and the output are shown in Fig. 4.

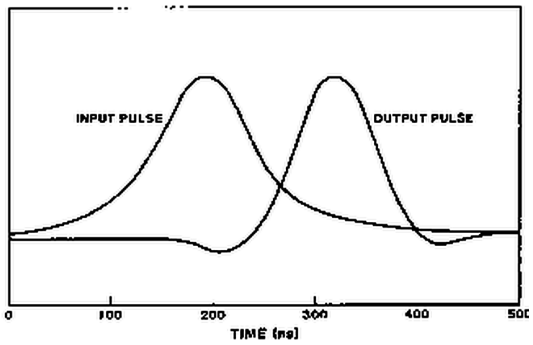


Fig. 4. Spice simulation of example design

CONCLUSIONS

During the process of developing this design approach, it became clear that a more appropriate approach would be to specify the objective function in the time domain. This would completely circumvent the need for having precise information about the group delay, for example. Only a modest change is required to change the procedure described here into a time domain design.

REFERENCES

[1] H.M. Sierra, "Increased Magnetic Recording Read-back Resolution by Means of a Linear Passive Network", *IBM Journal*, Jan. 1963.

[2] D.E. Vakman, *Sophisticated Signals and the Uncertainly Principle in Radar*, Springer-Verlag New York Inc., 1968.

[3] J.C. Mallinson and C.W. Steele, 'Theory of Linear Superposition in Tape Recording', *IEEE Transactions on Magnetics*, Vol. MAG-5, No. 4, Dec. 1969.

[4] B.K. Middleton and P.L. Wisley, 'Pulse Superposition and High Density Recording', *IEEE Transactions On Magnetics,* Vol. MAG-14, No. 5, Sept. 1978.

[5] PROSE, Inc., Palos Verdes Estates, CA 90274.

[6] T. Fujisawa, 'Realizability Theorem for Mid-series or Mid-shunt Low-pass Ladders Without Mutual Induction', *IRE Transaction-Circuit Theory*, Dec. 1955.

[7] A.V. Oppenheim and R.W. Schafer, *Digital Signal Processing*, Prentice Hall, Inc., 1975, p 21.

[8] Korn and Korn, *Malhemafical Handbook for Scientist and Engineers*, McGraw Hill, 1968, pp 134-136.

[9] W.C. Percival, *Thermionic Valve Circuits*, British Patent 460562, July 1935.

[10] E.L. Ginrton, W.A. Hewlett, J.H. Jasberg and J.D. Noe, *Distributed Amplification*, Proc IRE, vol 36, pp 956-969, Aug. 1948.

[11] P.H. Rodgers and L.H. Enloe, *Transistor Distributed Amplifier*, U.S Signal Corps Contract DA-36-039 SC-75021, March 1958.

[12] J.M. Pettit and M.M. McWhorter, *Electronic Amplifier Circuits*, McGraw Hill, 1961, pp 147-163.

[13] G.C. Temes and J.W. LaPatra, *Introduction to Circuit Synthesis and Design*, McGraw Hill, 1977, pp 157-159.

Incomplete Problems: can you help complete one or more?

These problems started some time ago and contact was lost with authors. If you recognize a problem and understand what the author was trying to do and known the goal/objective for a problem, please contact us with additions and changes so we can publish a complete problem with solution. Your name will be added to problem in order to give you the credit for defining the problem.

For **latest work** on *Math Problem-Solving*, click here

Index

2nd order non-linear ode, 47
3rd order non-linear ode, 50
algebraic equations, 9
algebraic models
  custom thermistor design, 93
  damped sinusoidal signal, 23, 44
  magnetic recording intro, 9
  optimum matched filter, 34
  pharmaco-kinetics, 26
  plane crash reconstruction, 103
  robot arm movement, 100
  sinusoidal signal, 21
  tfh design, 12, 15, 17, 19
automatic differentiation, 6
bode plot, 35
boundary value problems, 6, 46, 65, 83, 95
burgers' equations, 86
bvp. *See* boundary value problems
bvp models
  3rd order non-linear ode, 50
  burgers equation, 91
  drug development, 95
  telegrapher's equations, 89
company objectives, 124, 125
compiler, calculus-level, 6, 12, 47, 50, 66, 120, 124
  FortranCalculus, 3
  prose, 3, 72
continuously differentiable, 6
converting
  boundary value problem, 91
  initial value problem, 82
curve fitting, 9
  damped sine series, 23, 44
  lorentzian series, 12, 14, 15
  mod. lorentzian series, 17, 19
  sine series, 6, 21
  thin-film-head, 14
data file, 128, 129
download source code, 12
errors in model, 7
exercises, 33, 91
find statement, 6, 21, 34, 36, 46, 65, 83, 92, 93, 106, 122
FortranCalculus language, 3
fourier transform, 44
frequency parameters, 6, 22
  spectral estimation, 128
future outlook, 123
heat transfer over slab, 97
implicit equations, 106
  drug development, 95
implicit models
  2nd order implicit differential equation, 110
  algebraic, 107
incomplete problems
  body plasma chemistry, 71
inequalities, 29
initial value problems, 6, 46, 65, 69, 83
initial values, 7
integration statements, 46, 62
inverse problems, 26, 42, 92, 93, 94, 100, 103
ip. *See* inverse problems
ip models
  custom thermistor design, 93, 94
  drug development, 95
  heat transfer over slab, 97
  optimum matched filter, 42
  pharmaco-kinetics, 27
  plane crash reconstruction, 103
  robot arm movement, 100
ivp. *See* initial value problems
ivp models
  2nd order non-linear ode, 47
  non-linear equation of motion, 62
  system of pdes, 69
jacobian matrix, 29
laplace domain, 34, 35
laplace transforms, 34, 42
lorentz equations, 66
lorentz function, 10, 12, 50
lorentzian series, 6, 9, 10, 12
  modified, 10, 17
magnetic recording intro, 9
mainlobe & sidelobe plots, 35
manage by objectives, 3, 124, 125
matched filter, 34, 42, 132
math model, 7
math models
  algebraic, 9
  laplace transforms, 34
  ode, 46
  pde, 83
  systems of ode/pde, 65
method of lines, 6, 97
monte carlo simulation, 120
nasa project, 3
nesting, 6, 36, 53, 117
non-linear equation of motion, 62
objective function, 46
objective levels, 125
Objective-Driven Design, 31
ode. *See* ordinary differential equations
ode models
  2nd order non-linear ode, 47
  3rd order non-linear ode, 50
  bang-bang control, 53
  non-linear equation of motion, 62
  voice coil motor, 53
oil refinery production, 115, 124
one's vision
  mr. algebra, 8, 123
  mr. arithmetic, 8, 123

mr. calculus, 8, 123
operator overloading, 3
ordinary differential equations, 46
over-determined system, 47
parameter estimation, 5, 9, 22, 126
2nd order implicit differential equation, 110
2nd order non-linear ode, 47
3rd order non-linear ode, 50
bang-bang control, 53
burgers' equations, 86
bvp models, 6
custom thermistor design, 93
damped sine series, 23, 44
drug development, 95
future outlook, 123
heat transfer over slab, 97
inverse problems, 92
laplace models, 34
oil refinery production, 115
optimum matched filter, 34
pde models, 83
pharmaco-kinetics, 26
plane crash reconstruction, 103
robot arm movement, 100
sine series, 21
system of odes, 66
telegrapher's equations, 89
tfh design, 11, 12, 15, 17, 19
trouble shooting, 121
parameters
lacking, 7
partial differential equations, 6, 83
pde. *See* partial differential equations
pde models
burgers' equations, 86
heat transfer over slab, 97
oil refinery production, 115
stock market to biology, 84
telegrapher's equations, 89
peak shift, 41
pendulum problem, 62
pharmaco-kinetics, 26
plane crash reconstruction, 103
poles & zeroes, 34
production monitoring, 126
prose language, 3
requirements for model, 6
robot arm movement, 100
slack variable, 29
solar cell model, 76
spc. *See* statistical process control
spectral estimation, 6
statistical calculations, 120
statistical process control, 5, 15, 19, 126
stiff equations, 121
stock market to biology, 84
system of differential equations, 65
system of equations, 107
system of odes
lorentz equations, 66
system of pdes
convection reaction equations, 69
telegrapher's equations, 89
tfh. *See* thin-film-head
tfh model
lorentzian series, 10, 12, 15
mod. lorentzian series, 10, 17, 19
thermistor design, 93
thin-film-head
math model, 9, 10
results, 20
time domain, 42
tolerant designs/analysis, 120
total derivative, 120
transfer function, 34
poles & zeros, 34, 44
trouble shooting, 122
under-determined system, 47, 122
variance calculations, 120