NUMERICAL SOLUTIONS OF A NON-LINEAR CLASSICAL FIELD THEORY OF LEPTONS AND QUARKS

by

Philip Howard Sharman,
B.A.A., Ryerson Polytechnical Institute, 1978,
B.Sc., University of Calgary, 1986.

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the Department
of Physics and Astronomy.

We accept this thesis as conforming
to the required standard.

[redacted]

Dr. F. I. Cooperstock, Department of Physics and Astronomy, University of Victoria.

[redacted]

Dr. C. E. Picciotto, Department of Physics and Astronomy, University of Victoria.

[redacted]

Dr. G. G. Miller, Department of Mathematics, University of Victoria.

[redacted]

Dr. J. Carminati, Department of Applied Mathematics, University of Waterloo.

ISBN 0-315-62350-0

Supervisor: Dr. F. I. Cooperstock.

ABSTRACT

This paper studies a non-linear, classical field theory of elementary particles. This theory is gauge-invariant and singularity-free and proposes that these particles are made up purely of three fundamental fields whose non-linear interactions produce a particle-like region of localized energy. Numerical solutions of the resulting equations are presented which show that the model can be used to describe not only the lepton family, but also the two known quark families.

Examiners:

[redacted]

Dr. F. I. Cooperstock, Department of Physics and Astronomy, University of Victoria.

[redacted]

Dr. G. E. Picciotto, Department of Physics and Astronomy, University of Victoria.

[redacted]

Dr. G. G. Miller, Department of Mathematics, University of Victoria.

[redacted]

Dr. J. Carminati, Department of Applied Mathematics, University of Waterloo.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. F. I. Cooperstock, for his support and encouragement.

I would also like to thank Diane Fox, Paul Lim, Joel Newman, and Bryce Young for much help, and the Natural Sciences and Engineering Research Council for financial support under grant #A5340.

NUMERICAL SOLUTIONS OF A NON-LINEAR CLASSICAL FIELD THEORY OF LEPTONS AND QUARKS

CHAPTER 1: INTRODUCTION

Physics now has well-developed theories of quantum and particle physics; they accurately predict many things about particles' interactions, but do not explain these particles in terms of more fundamental constructs nor how or why they have their observed properties. They take the fundamental particles, including leptons and quarks, to be "given". Nor do they even attempt to make such an explanation; such gaps are not deficiencies of quantum or nuclear physics, they simply reflect the fact that such questions are extra-theoretical for those disciplines. There are some current attempts to describe elementary particles in terms of simpler constituents ("rishons", "preons", etc.)[1] but these sub-units are not themselves explained or described in any more fundamental sense than the particles they supposedly make up.

[1] See Das et.al. [1] for more details.

Their hypothetical existence explains the *patterns*, but not the fundamental *existence* of matter.

The work presented here is the latest in a long series of attempts by physicists to determine the relationship between *matter* and *fields*. This particle/field duality has been a feature of physics for centuries.

> In the history of fundamental physics no other basic concepts have assumed a more important role than that of *fields* and *particles*. In fact it seems to be a strange characteristic of the human mind that it is forced to describe the physical properties of matter either as fields or as particles. The whole history of physics appears as a struggle to either clarify or escape from this *either or* dichotomy.[2]

On the one hand we have *matter* made up of particles, and on the other we have the concept of *fields*. In the course of scientific development there has been frequent controversy about the ontological priority of fields versus matter. Newtonian mechanics was eminently successful at using the particle and action-at-a-distance concepts. But then came James Clerk Maxwell's electromagnetism; he was a pioneer in maintaining that the electromagnetic field concept was a fundamentally important one. In his "A Dynamical Theory of the Electromagnetic Field", he ushers in the era of classical field theory; now the field becomes a real entity where energy can reside.

> In speaking of the Energy of the field ... I wish to be understood literally. All energy is the same as mechanical energy, whether it exists in the form of motion or in that of elasticity, or in any other form. The energy in electromagnetic phenomena is mechanical energy. The only question is, where

---

[2] Sen, [2], page viii.

> does it reside? On the old theories it resides in the electrified bodies, conducting circuits, and magnets, in the form of an unknown quality called potential energy, or the power of producing certain effects at a distance. On our theory it resides in the electromagnetic field, in the space surrounding the electrified and magnetic bodies, as well as in those bodies themselves....[3]

We can distinguish three types of approaches.[4] First, there are the *dualistic* theories, such as classical electromagetism, which assume that the field-sources (i.e, the particles) are ontologically separate from the fields themselves. Secondly there are the non-dualistic theories which attempt to describe physics in terms of either pure fields *or* only particles. And thirdly there are the *unified non-dualistic* theories attempting to unify *all* fields or particles. Figure 1 illustrates some of these kinds of theories.

The work presented here is firmly in the non-dualistic, field-theory camp. It follows the vision of Einstein and Nathan Rosen who wrote:

> A complete field theory knows only fields and not the concepts of particle and motion. for these must not exist independently of the field but are to be treated as part of it. On the basis of the description of a particle without singularity one has the possibility of a logically more satisfactory treatment of the combined problem: The problem of the field and that of motion coincide.[5]

Naturally, such fields are abstract, remote, and (presently) immune from direct observation. So in what sense do they "explain" the existence

---

[3] James Clerk Maxwell, from "A Dynamical Theory of the Electromagnetic Field", quoted in Williams, [3], page 134.

[4] (This classifcation scheme, and much of this section, is due to Sen [2], pp. viii-x.)

[5] Einstein and Rosen, [4], page 76.

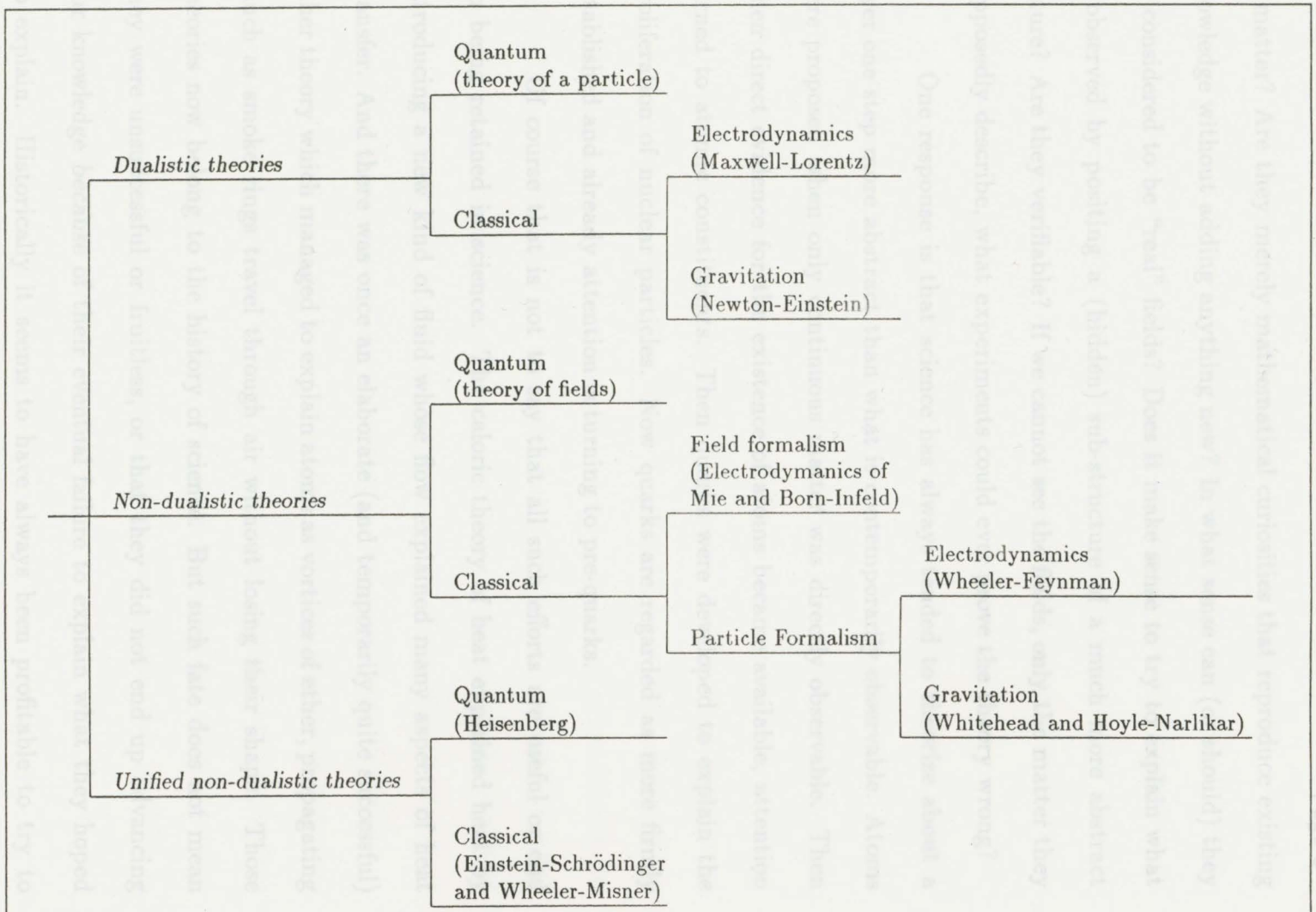


FIGURE 1: *A comparison of the different approaches to the field/particle question (from Sen [2], page x).*

of matter? Are they merely mathematical curiosities that reproduce existing knowledge without adding anything new? In what sense can (or should) they be considered to be "real" fields? Does it make sense to try to explain what is observed by positing a (hidden) sub-structure of a much more abstract nature? Are they verifiable? If we cannot see the fields, only the matter they supposedly describe, what experiments could ever prove the theory wrong?

One response is that science has always tended to theorize about a layer one step more abstract than what is contemporarily observable. Atoms were proposed when only continuous matter was directly observable. Then when direct evidence for the existence of atoms became available, attention turned to atomic constituents. Then quarks were developed to explain the proliferation of nuclear particles. Now quarks are regarded as more firmly established and already attention is turning to pre-quarks.

Of course that is not to say that all such efforts are useful or end up being retained in science. The caloric theory of heat explained heat by introducing a new kind of fluid whose flow explained many aspects of heat transfer. And there was once an elaborate (and temporarily quite successful) ether theory which managed to explain atoms as vortices of ether, propagating much as smoke rings travel through air without losing their shape. Those theories now belong to the history of science. But such fate does not mean they were unsuccessful or fruitless, or that they did not end up advancing our knowledge *because* of their eventual failure to explain what they hoped to explain. Historically it seems to have always been profitable to try to

guess what lies hidden beyond our current powers of observation. Whether the work presented here will ultimately be successful is presently unknown, but it is worth pursuing. And even though presumably the full explanation of matter will have to be quantized, it is hoped that pursuing the strictly *classical* approach as far as possible, as is done here, will lay the correct groundwork for the future.

One attractive feature of the present work is that it needs a small number of assumptions and *ad hoc* hypotheses. Apart from numeric values (for mass, charge, and for various coupling constants whose magnitude is unknown), the main assumption used here is that the behaviour of matter can be described by an appropriate Lagrangian density which satisfies an action-extremizing principle. The correct Lagrangian must still be deduced (or guessed), but once that is done almost everything else flows automatically. In particular, as we shall see, the non-linearities of the fields involved here automatically produce their own field equations. Thus the beautfy of the pure field theory approach is that it gives both the equations of the field development *and* the equations of motion of matter which is composed from the fields.

We must "plug in" from the outside values for the charge and mass of the particles described, but then the equations produce an independent prediction of the size of the particle. And as we have available experimentally-determined upper limits for the size of leptons and quarks, this provides an important independent check of the theory.

* * *

CHAPTER 2: PREVIOUS WORK

Classical field theory starts by positing a Lagrangian density (L) for a given system (where L is a function of the fields and their first partial derivatives), and assumes that the action

$$S = \int L \, dV \, dt$$

satisfies an extremal principle

$$\delta S = 0. \tag{1}$$

If we let B represent the fields, so that $L = L(B, \partial_u B)$ then[6] in the same way that in classical mechanics Hamilton's principle yields Lagrange's equations, (1) yields the "Euler-Lagrange" equations:

$$\left(\frac{\partial L}{\partial B_{,a}}\right)_{;a} - \frac{\partial L}{\partial B} = 0. \tag{2}$$

Also,

$$T_{ij} = \left(\frac{\partial L}{\partial g^{ij}}\right) - \frac{L\, g_{ij}}{2} \tag{3}$$

where T_{ij} is the "energy-momentum" tensor and g_{ij} is the metric tensor. In particular, T_{00} is the energy density of the system.

[6] ∂_u and other conventions used here are defined in Appendix A.

A) STANDARD ELECTROMAGNETISM

It will be useful to look first at how this works in the case of standard, classical (dualistic) electromagnetism to show both similarities and differences with the later, non-dualistic, pure field-theory approach. For example, for the simple case of charged particles in an electromagnetic field we have

$$L_{\mathrm{EM}} = -\frac{F_{ab}F^{ab}}{8\pi} - j_a A^a$$

where

$$F_{kl} \equiv A_{l,k} - A_{k,l}$$

$$\mathbf{j} \equiv \rho(1, \vec{v})$$

$\rho \equiv$ the charge density

$\vec{v} \equiv$ velocity of the charged particles

$$\mathbf{A} \equiv (\varphi, \vec{A})$$

$\varphi \equiv$ the scalar electromagnetic potential

$\vec{A} \equiv$ the vector electromagnetic potential.

So now (2) yields

$$F^{ka}{}_{;a} = -4\pi j^k \tag{4}$$

(which is in fact the tensor formulation of two of Maxwell's equations). Note that here **j**, the current four-vector, must be defined *outside* the field theory. Both it and the concept of a charged mass-point itself are *auxiliary* concepts, thus illustrating the fact the classical electromagnetism is indeed a dualistic theory.

B) MIE, BORN AND INFELD

Classical electromagnetism with its Maxwell's equations has great power, but there are two fundamental difficulties. Firstly, the concept of a point-charge leads to difficulties of inifinite self-energies. Secondly, the field equations break down at the point-charge itself because of the singularity there. So, as we have seen, the equations of motion have to be added separately from the field equations.

G. Mie [5] proposed a non-dualistic field theory where the particle's mass and charge are determined solely by the fields. But there are two criticisms of his work.[7] One is that Mie's Lagrangian does not provide unique solutions for each particle. Another is that his theory is not gauge-invariant; because his Lagrangian and his field equations both depend explicitly on φ, adding a constant external potential to φ changes the solution.

When it was determined that quantum electrodynamics also had difficulties with the notion of a point-electron, Born and Infeld [6] tried to develop a non-linear theory of electrodynamics. However Rosen [7] showed that there are still singularities present at the origin.

C) ROSEN

Rosen [7] concluded that L should depend on the potentials, to avoid the singularities of Born and Infeld, but must somehow involve other variables which will make the Lagrangian gauge-invariant.

---

[7] Sen, [2], page 41.

He investigated a Lagrangian formed by combining an electromagnetic field with a complex, scalar matter-field ψ, and demanded that gauge invariance hold when the electromagnetic vector potential undergoes a local gauge transformation

$$A_u \rightarrow A'_u = A_u + \frac{\partial \lambda}{\partial x^u} \tag{5}$$

(where λ is an arbitrary function of the space-time coordinates). He found he could make L gauge-invariant by imposing the condition that under the gauge transformation (5) we also have

$$\psi \rightarrow \psi' = e^{i\epsilon\lambda(x)}\psi$$

(where ϵ is a real constant of dimensions $1/charge$.[8]) Then $\psi\psi^*$ is gauge-invariant since the phase factors cancel out. (F^{uv} is already gauge-invariant since the second derivatives of λ cancel out.) But if L contains factors like $\partial_u\psi$ they will *not* be gauge-invariant since

$$\partial_u\psi \rightarrow e^{i\epsilon\lambda}(\partial_u + i\epsilon\lambda_{,u})\psi.$$

So, to cancel out these extra $\lambda_{,u}$ terms, ∂_u must be replaced by the so-called "gauge-covariant derivative"[9]

$$D_u \equiv \partial_u - i\epsilon A_u.$$

Then

$$(D_u\psi) \rightarrow \left[\partial_u - i\epsilon(A_u + \lambda_{,u})\right]e^{i\epsilon\lambda}\psi$$

$$= e^{i\epsilon\lambda}(D_u\psi)$$

---

[8] (The dimensions of a number of quantities are summarized for convenience in Appendix B.)

[9] Griffiths, [8], page 349.

and so factors like $(D^a\psi)(D_a\psi)^*$ are gauge-invariant. Rosen combined the simpler possible scalars (omitting more complicated ones like $F_{ab}F^{ab}\psi\psi^*$) to form

$$L_{\text{ROSEN}} = -\frac{F_{ab}F^{ab}}{8\pi} - (D^a\psi)(D_a\psi)^* + \sigma^2\psi\psi^*$$

where σ is a (real) constant. (The signs of the last two terms were chosen as shown as it was found that only this choice of sign structure gave particle-like solutions.) Variation with respect to A_u and ψ^* then gives two field equations

$$F^{ua}{}_{;a} = -4\pi J^u \tag{6}$$

$$\left(\text{where} \quad J^u \equiv \frac{1}{2}i\epsilon\left[\psi^*(D^u\psi) - \psi(D^u\psi)^*\right]\right)$$

$$(D^a\psi)_{;a} - i\epsilon A_a(D^a\psi) + \sigma^2\psi = 0.$$

Notice that now the current four-vector **J** is produced by the fields themselves; the theory is self-contained. It then follows via (3) that the energy-momentum tensor is

$$T_{uv} = \frac{1}{16\pi}(g_{uv}F_{ab}F^{ab} - 4F_{ua}F_v{}^a)$$
$$+ \frac{1}{2}\left[g_{uv}\left\{(D^a\psi)(D_a\psi)^* - \sigma^2\psi\psi^*\right\} - (D_u\psi)(D_v\psi)^* - (D_u\psi)^*(D_v\psi)\right].$$

For the spherically-symmetric case we can use the gauge freedom of (5) to choose a gauge where $\vec{A} = 0$ so we have

$$\mathbf{A} = (\varphi(r), \vec{0}).$$

Rosen now (implicitly) assumes that the **J** of (6) should be identified with the usual four-vector current **j** of standard electromagnetic theory. (This is reasonable since we want Maxwell's equations (4) to hold in both cases.) So

$$J^0 \equiv \rho(r),$$

and, for the *static* case where the charge does not alter with time, $\vec{v} = 0$ so

$$J^\alpha \equiv v^\alpha = 0 \quad (\alpha = 1, 2, 3).$$

Rosen takes ψ to be of the form

$$\psi = \theta(r) e^{-i\epsilon\mu t}.$$

Then the field equations (6) become

$$\nabla^2\theta + \left[\epsilon^2(\varphi + \mu)^2 - \sigma^2\right]\theta = 0$$

$$\nabla^2\varphi + 4\pi\rho = 0$$

$$\left(\text{where} \quad \rho = \epsilon^2(\varphi + \mu)\theta^2\right).$$

For solutions to represent a particle some boundary conditions must be imposed. To ensure that there are no discontinuities at the origin we require that

$$\frac{d\theta}{dr} = \frac{d\varphi}{dr} = 0 \quad \text{at} \quad r = 0.$$

We also require that

$$\theta \to 0 \text{ exponentially} \quad \text{as} \quad r \to \infty$$

so that the particle is concentrated near the origin, and require that

$$\varphi \to \frac{\text{constant}}{r} \quad \text{as} \quad r \to \infty$$

so the electic field behaves like a Coulomb field at large distances from the particle.

It is easier to work with dimensionless variables by defining

$$x \equiv r\,\sigma, \quad z \equiv \frac{\epsilon}{\sigma}(\varphi + \mu), \quad y \equiv \sqrt{4\pi}\,\epsilon\,\frac{\theta}{\sigma}. \tag{7}$$

Then the field equations are

$$y'' + \frac{2}{x}y' - y + yz^2 = 0 \tag{8}$$

$$z'' + \frac{2}{x}z' + y^2 z = 0.$$

The total charge (q) of the particle is

$$q = \int_{\text{all space}} \rho(r)dV \tag{9}$$

$$= 4\pi \int_0^\infty \rho\, r^2 dr$$

$$= 4\pi \int_0^\infty \epsilon^2(\varphi + \mu)\theta^2 r^2 dr$$

$$= \frac{\alpha}{\epsilon}$$

$$(\text{where } \alpha \equiv \int_0^\infty y^2 z x^2 dx).$$

The energy density is

$$T_{00} = \frac{1}{8\pi}\left(\frac{d\varphi}{dr}\right)^2 - \frac{1}{2}\left[\epsilon^2(\varphi + \mu)^2\theta^2 - \left(\frac{d\theta}{dr}\right)^2 - \sigma^2\theta^2\right].$$

By integrating by parts and using the field equations we can find the total energy (E) is

$$E = \int_{\text{all space}} T_{00}\, dV \qquad (10)$$

$$= 4\pi \int_0^\infty T_{00}\, r^2 dr$$

$$= -\frac{\sigma\, \mathrm{w}}{2\,\epsilon^2}$$

where

$$\mathrm{w} \equiv \alpha\beta + \gamma, \quad \beta \equiv \frac{\epsilon\mu}{\sigma}, \quad \gamma \equiv \int_0^\infty y^2 z^2 x^2 dx.$$

We require β be less than one for the following reason. From (8), the field equation for y, we have

$$y'' + \frac{2}{x}y' + [z^2 - 1]y = 0.$$

And we also require that

$$\varphi \to \varphi_{\text{coulombic}} = \frac{q}{r} \qquad \text{as} \quad r \to \infty.$$

So, using (9),

$$z = \frac{\epsilon(\varphi + \mu)}{\sigma} \quad \to \quad \frac{\epsilon}{\sigma}\frac{q}{r} + \frac{\epsilon\mu}{\sigma} \qquad (11)$$

$$= \frac{\alpha}{x} + \beta.$$

As $x \to \infty$, $z \to \beta$ so if we define

$$C \equiv (z^2 - 1)$$

then

$$C \quad \rightarrow (\beta^2 - 1) \quad \text{as} \quad x \rightarrow \infty.$$

So as $x \rightarrow \infty$ we have

$$y'' + \frac{2}{x}y' + C\,y = 0.$$

If $\beta^2 > 1$ then $C > 0$ and

$$y(x) = A\frac{\sin(\sqrt{C}x)}{x} + B\frac{\cos(\sqrt{C}x)}{x} \quad \text{(A, B = constants).} \tag{12}$$

If $\beta^2 < 1$ then $C < 0$ and

$$y(x) = A\,\frac{e^{\sqrt{-C}\,x}}{x} + B\,\frac{e^{-\sqrt{-C}\,x}}{x}. \tag{13}$$

Of these two solutions, (12) has oscillatory behaviour whereas (13) has the desired exponentially-decaying solution, so we need $\beta^2 < 1$.

This model will produce particles, in the sense that there are localized solutions satisfying the boundary conditions, but all the solutions found have w>0, and hence, via (10) the total energy is *negative*. And since mass is related to energy via E=mc^2, to produce positive-mass particles a more complicated Lagrangian must be used.

D) ROSEN AND ROSENSTOCK

Rosen and Rosenstock [9] investigated the case of the purely scalar field, without electromagnetism. They took

$$L_{\mathrm{RR}} = (\partial^a\psi)(\partial_a\psi)^* - \sigma^2\psi\psi^* + \frac{1}{2}g\,\psi^2\psi^{*2}$$

(where g is a positive constant). For the static, spherically-symmetric case they take

$$\psi = \theta(r)\, e^{i\epsilon\mu t}$$

(where $\epsilon\mu$ is a real constant). The field equation is then

$$\frac{d^2\theta}{dr^2} + \frac{2}{r}\frac{d\theta}{dr} - [\sigma^2 - (\epsilon\mu)^2]\theta + g\,\theta^3 = 0 \qquad (14)$$

(where $[\sigma^2 - (\epsilon\mu)^2]$ is assumed to be positive to prevent oscillatory solutions). The energy density is

$$T_{00} = \frac{1}{2}\left[(\epsilon\mu)^2\theta^2 + \sigma^2\theta^2 + \left(\frac{d\theta}{dr}\right)^2 - \frac{1}{2}\,g\,\theta^4\right].$$

Again, integrating by parts yields a total energy

$$E = 2\pi \int_0^\infty \left[2(\epsilon\mu)^2\theta^2 + \frac{1}{2}\,g\,\theta^4\right] r^2 dr$$

which is positive definite, so we have positive-energy particles. Rosen and Rosenstock identify **J**, the current four-vector arising from the field equations, with **j**, the current four-vector of standard electromagnetic theory, the same way as was done before (in Section C, page 11). They then identify a quantity they call "charge" via

$$\text{"charge"} \equiv \int_{\text{all space}} \rho\, dV = \int J^0 dV.$$

If ψ is real then the charge is zero, but if $\psi = \theta e^{i\epsilon\mu t}$ then

$$\text{"charge"} = 2\epsilon^2\mu \int \theta^2 dV.$$

But this scalar "charge" is *not* the usual electromagnetic charge. Whereas before **J** was identified with **j** through the appearance of Maxwell's equations, here there is not the same incentive. So this L is successful at producing *neutral* particles, but not really satisfactory for modelling *charged* particles.

However, it is still useful to look at the solutons to (14) since many of the properties of this field equation are also characteristics of the more complicated field equations which will appear later. Using the same dimensionless variables the Rosenstock equation (14) becomes

$$y'' + \frac{2}{x}y' - y + y^3 = 0. \tag{15}$$

We are interested in the case where we only have one free parameter: $y(0)$. (The other boundary condition is $y'(0)=0$ which is imposed to ensure that $\theta'(r=0)=0$ for smoothness at the origin.) This is an interesting equation because it looks deceptively simple yet its non-linearity produces some interesting behaviour. What are the solutions to this equation?

First, we can note that (15) has three constant solutions:

$$y = 0, \quad y = -1, \quad y = +1.$$

We are interested in solutions where the particle is concentrated in a localized region near the origin and has very little mass far away from the origin. The mass depends on T_{00}, which depends on y, so we need to find solutions where $y \to 0$ as $x \to \infty$.

We can deal with this equation in two ways. Firstly, we can set $y(0)$ to our desired start parameter and then integrate y numerically along the

x-axis to find $y(x)$ for any desired $x>0$ value (subject to the limitations of numerical accuracy to be discussed below). Secondly, we can treat the equation analytically by making some linear approximations for asymptotic solutions for large x.

We can start by looking at solutions where y approaches one of the constant solutions as x grows large.

First, if we look at the behaviour of y where $y \to \pm 1$ as $x \to 0$ we can set

$$y = \pm 1 + u(x) \quad (\text{where } u(x) \text{ is small})$$

then (15) approaches

$$x^2 u'' + 2x\, u' + (2x^2)u = 0$$

which can be solved (via spherical Bessel functions) to show

$$y(x) = A\frac{\sin(\sqrt{2}x)}{x} + B\frac{\cos(\sqrt{2}x)}{x} \quad (\text{A, B = constants}).$$

So $y \to \pm 1$ with a decaying oscillation around the ± 1 value. These obviously are not suitable candidates for particle-solutions, but we will see below that numerically we often run into these oscillatory solutions because of numerical instability.

More interesting for us are the particle-like solutions where $y \to 0$ as $x \to \infty$. In this case the $(y)^3$ term in (15) is negligible for large x and we have

$$y'' + \frac{2}{x}y' - y = 0$$

which has the (exact) solution

$$y = A\frac{e^x}{x} + B\frac{e^{-x}}{x} \quad (\text{A, B } = \text{ constants}). \tag{16}$$

Analytically we can set A=0 and assume the e^{-x}/x term is our desired solution, but numerically this presents some problems as we will see below.

The important fact is that there are these particle-like solutions to (15) where y goes exponentially to 0 as x goes to ∞. But how many of these particle-like solutions are there? And what do they look like?

Finkelstein, LeLevier, and Ruderman [10] indicated that there are quantized "eigensolutions" which have the following characteristics: if $y(0)$ is a very specific value (an eigenvalue) then $y \to 0$, but if $y(0)$ is raised even slightly above or lowered slightly below this value then $y \to \pm 1$ as $x \to \infty$. Furthermore there are an infinite number of such eigensolutions. And each eigensolution can be characterized by the number of "nodes" it has (i.e., how many times it crosses the x-axis). It turns out the nth eigensolution has exactly n nodes.

Table 1 shows the first fifty eigenvalues and Figures 2 and 3 show the $n = 1$ to $n = 6$ eigensolutions.

Numerically this sort of equation presents certain problems which occur both here and in the full model. For the solutions (16) we can analytically set A=0 and thus ensure that $y \to 0$ as $x \to 0$. But when integrating (16) numerically from $y(0)$ we cannot numerically set A=0 and so eventually round-off error and lack of precision accumulate to the point where the e^x/x term starts

n	$y(0)_n$
1	4.33738
2	14.10358
3	29.13121
4	49.36079
5	74.77307
6	105.38181
7	141.16164
8	181.91134
9	226.57026
10	279.37381
11	335.02740
12	393.64247
13	452.41196
14	508.20404
15	559.10233
16	604.71920
17	645.61402
18	683.52714
19	721.00998
20	757.24527
21	791.07109
22	822.21383
23	850.71010
24	876.70201
25	900.37976

n	$y(0)_n$
26	921.95804
27	941.66012
28	959.70671
29	976.30937
30	991.66159
31	1005.92686
32	1019.23873
33	1031.72519
34	1043.54078
35	1054.87351
36	1065.90501
37	1076.70618
38	1087.07776
39	1096.53831
40	1104.70036
41	1111.56826
42	1117.37910
43	1122.39493
44	1126.82993
45	1130.83396
46	1134.49822
47	1137.88114
48	1141.03927
49	1144.03985
50	1146.95335

TABLE 1: *The $y(0)$ values for the first fifty eigensolutions for the Rosenstock equation.*

to show up and the solution "blows up" as x gets large. So numerically we must take care to adjust the step size of integration and carry enough precision internally in the computer programs to suppress the e^x/x term until we have integrated far enough down the x-axis to extract the information we need (i.e.,

Figure 2: The first three Rosenstock eigensolutions

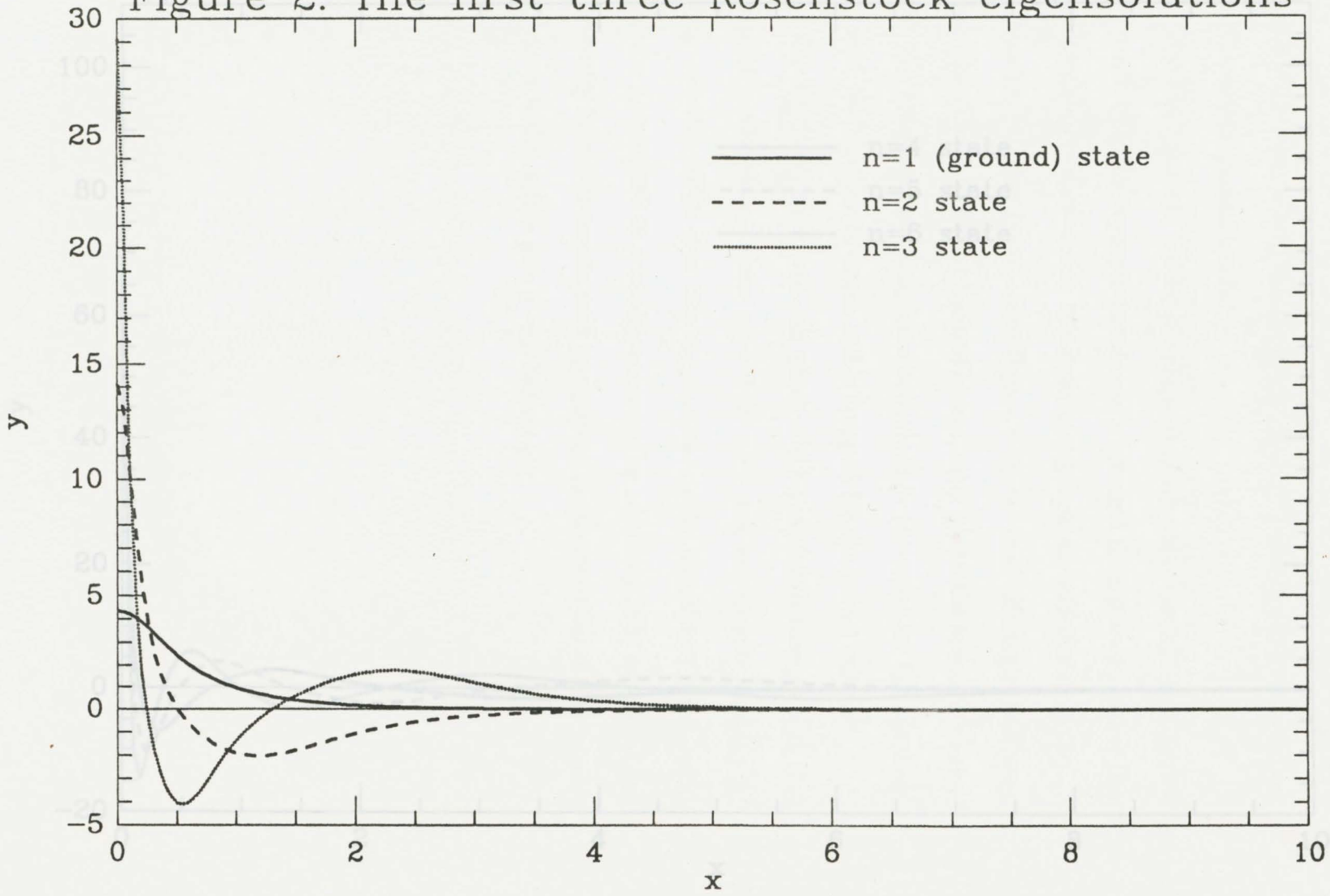
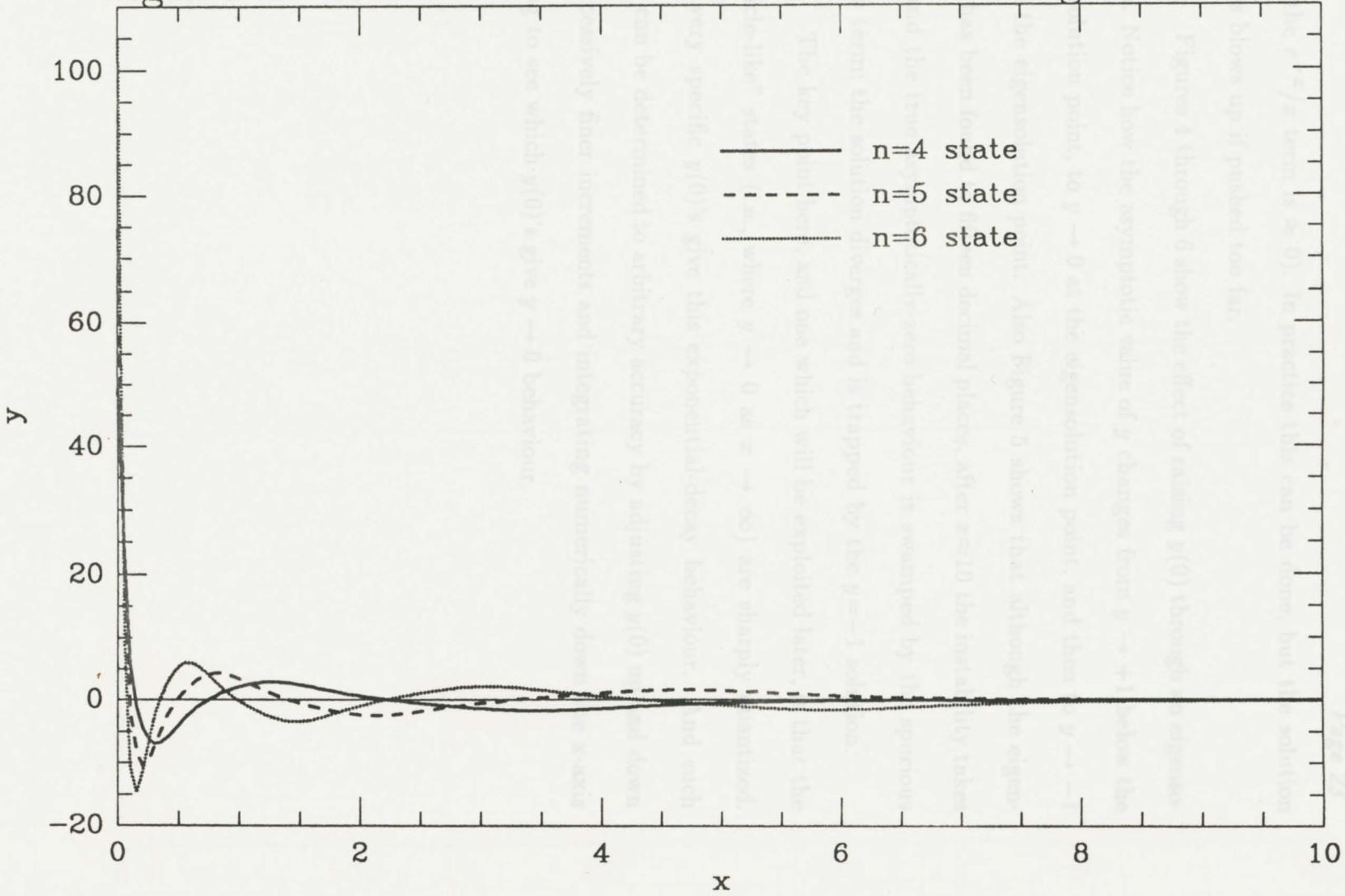


Figure 3: The second three Rosenstock eigensolutions



until the e^{-x}/x term is ≈ 0). In practice this can be done, but the solution always blows up if pushed too far.

Figures 4 through 6 show the effect of raising $y(0)$ through an eigensolution. Notice how the asymptotic value of y changes from $y \to +1$ below the eigensolution point, to $y \to 0$ at the eigensolution point, and then to $y \to -1$ above the eigensolution point. Also Figure 5 shows that although the eigenvalue has been found to fifteen decimal places, after $x \approx 10$ the instability takes over and the true asymptotically-zero behaviour is swamped by the spurious Ae^{x}/x term; the solution diverges and is trapped by the $y=-1$ solution.

The key point here, and one which will be exploited later, is that the "particle-like" states (i.e., where $y \to 0$ as $x \to \infty$) are sharply quantized. Only very specific $y(0)$'s give this exponential-decay behaviour. And such $y(0)$'s can be determined to arbitrary accuracy by adjusting $y(0)$ up and down in successively finer increments and integrating numerically down the x-axis looking to see which $y(0)$'s give $y \to 0$ behaviour.

Figure 4: The Rosenstock equation with y(0)=4.0

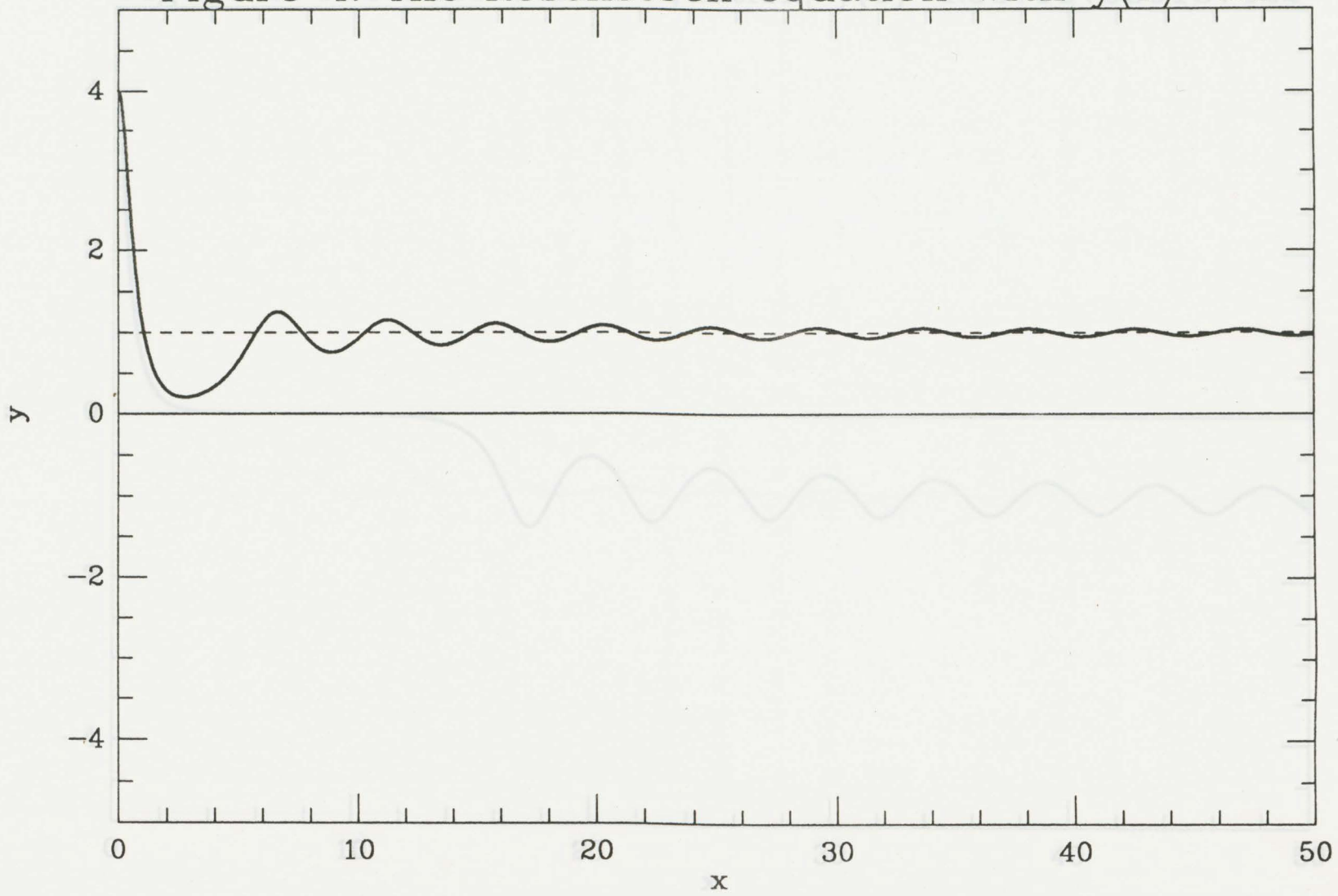


Figure 5: The Rosenstock equation, y(0)=4.337387680187422

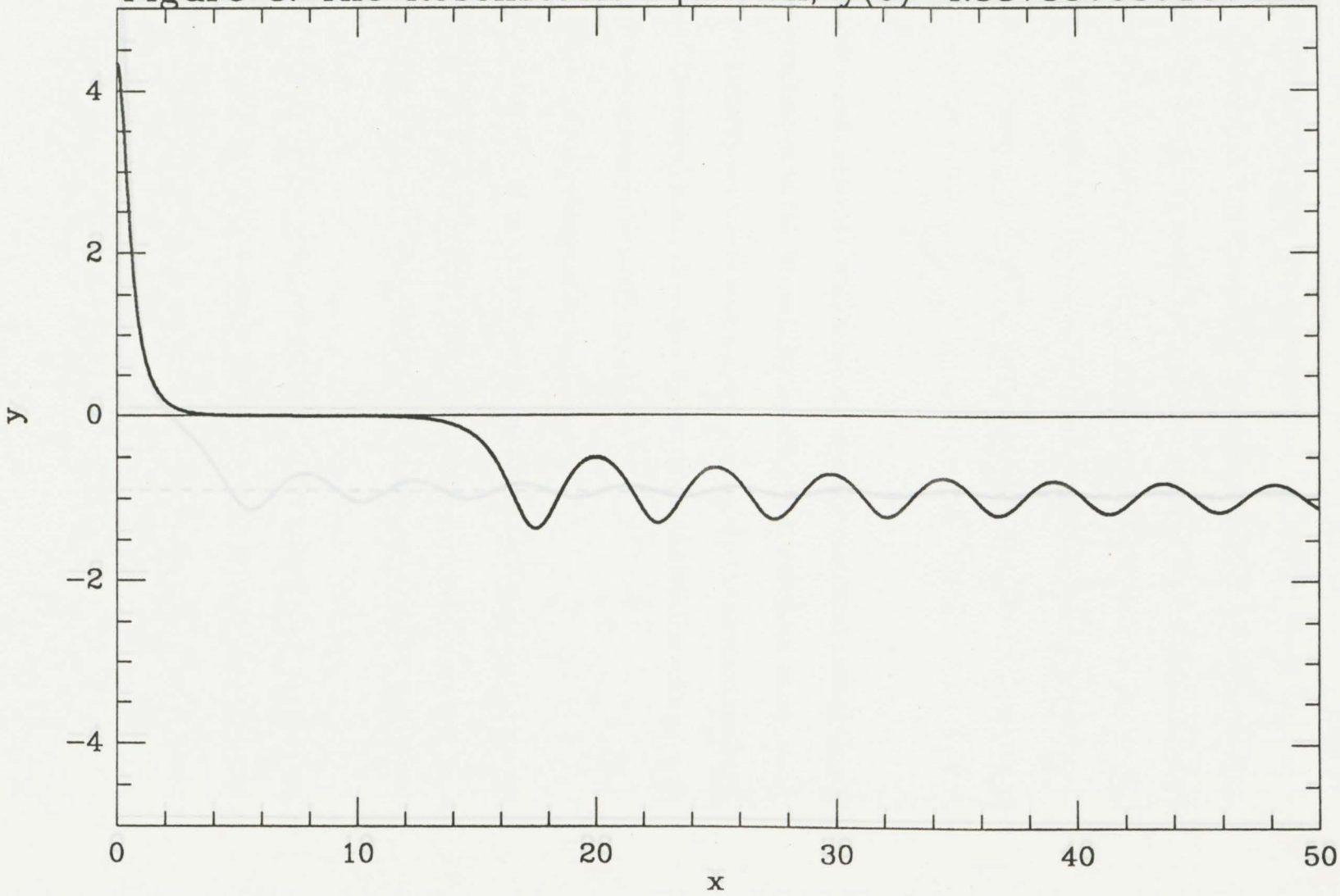
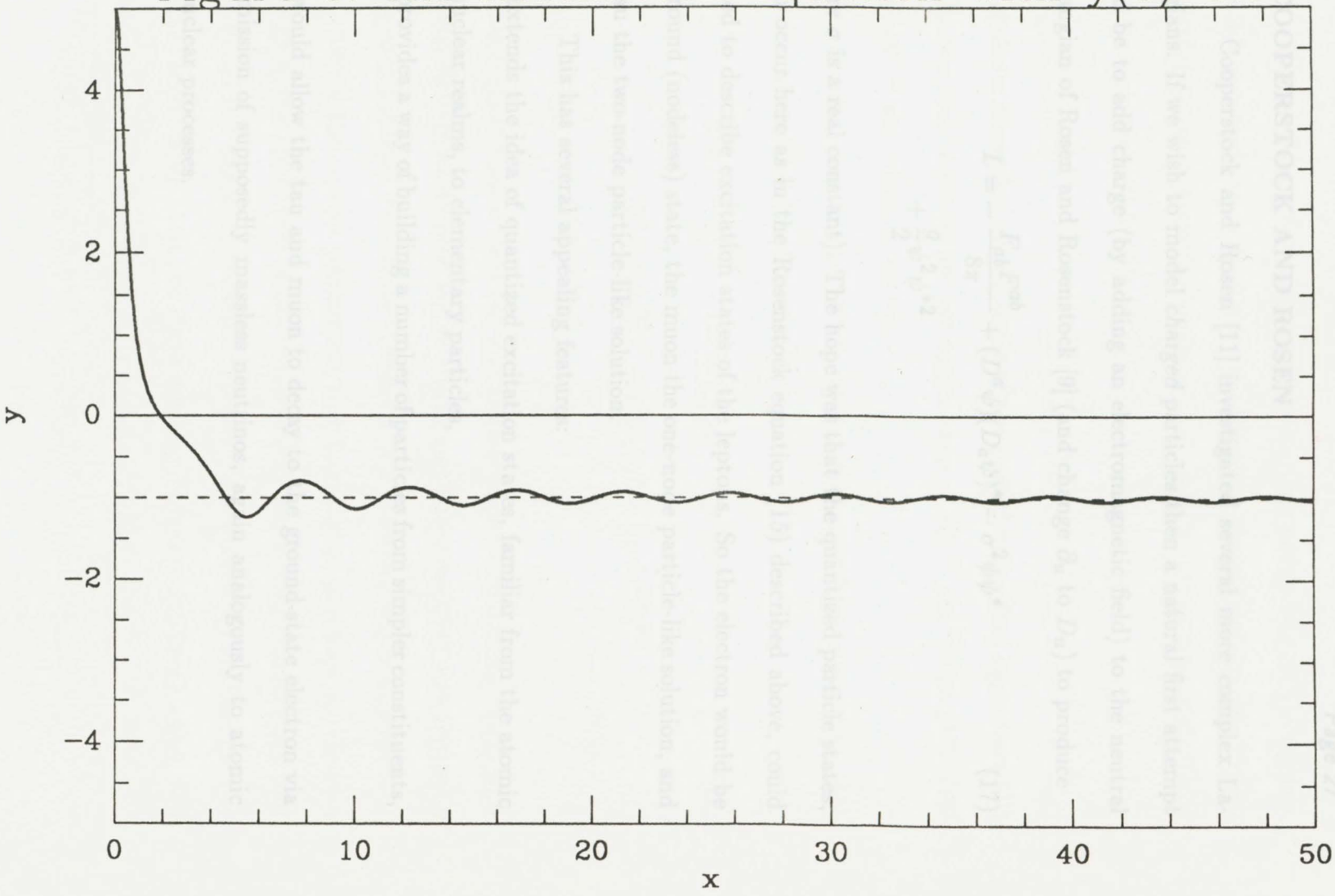


Figure 6: The Rosenstock equation with y(0)=5.0



E) COOPERSTOCK AND ROSEN

Cooperstock and Rosen [11] investigated several more complex Lagrangians. If we wish to model *charged* particles then a natural first attempt would be to add charge (by adding an electromagnetic field) to the neutral Lagrangian of Rosen and Rosenstock [9] (and change ∂_u to D_u) to produce

$$L = -\frac{F_{ab}F^{ab}}{8\pi} + (D^a\psi)(D_a\psi)^* - \sigma^2\psi\psi^* + \frac{g}{2}\psi^2\psi^{*2} \tag{17}$$

(where σ is a real constant). The hope was that the quantized particle states, which occur here as in the Rosenstock equation (15) described above, could be used to describe excitation states of the leptons. So the electron would be the ground (nodeless) state, the muon the one-node particle-like solution, and the tau the two-node particle-like solution.

This has several appealing features:

a) it extends the idea of quantized excitation states, familiar from the atomic and nuclear realms, to elementary particles,

b) it provides a way of building a number of particles from simpler constituents, and

c) it would allow the tau and muon to decay to the ground-state electron via the emission of supposedly massless neutrinos, again analogously to atomic and nuclear processes.

However, there is a problem with this Lagrangian. For all the models, the fields approach zero very quickly. Figure 2 shows that for the Rosen-Rosenstock equation (16), for example, $y \approx 0$ when $x > \sim 5$ so x is on the order of the particle size. And since

$$r = \frac{x}{\sigma}$$

$1/\sigma$ is also a dimension on the order of the particle size. And experimentally it is known that electrons are not bigger than $\sim 10^{-16}$ cm.[10] But (17) gives $1/\sigma$ of 10^{-12} cm, which is too big.

Another Lagrangian which ultimately proved unsatisfactory, but which has some attractive features, is

$$L = -\frac{F_{ab}F^{ab}}{8\pi} - (D^a\psi)(D_a\psi)^* + \sigma^2\psi\psi^* \qquad (18)$$
$$+ (\partial^a\psi_1)(\partial_a\psi_1)^* - \sigma^2\psi_1\psi_1{}^* + \frac{1}{2}g\,\psi_1{}^2\psi_1{}^{*2}$$
$$- f\psi\psi^*\psi_1\psi_1{}^*.$$

This combines the scalar field (ψ) and electromagnetic field of Rosen [7] with the scalar field (now denoted ψ_1) of Rosen and Rosenstock [9]. In addition, a simple coupling term $f\psi\psi^*\psi_1\psi_1{}^*$ (where f is a constant) has been added.

With the correct parameters this Lagrangian produces an electron well within the upper size limit, but when ψ_1 is excited, Cooperstock and Rosen found this Lagrangian cannot represent the muon since the particle size predicted is too large. Also, no single set of parameters was found which will give the correct tau mass when ψ_1 is doubly excited.

---

[10] Quigg, [12], page 3.

However there are two features which will be used later. The negative coupling term $(-f\psi\psi^*\psi_1\psi_1{}^*)$ adds a negative term to the energy integral of the form

$$-\int_0^\infty y^2(y_1)^2x^2dx.$$

This binding energy term grows larger in abolute value as y_1 is excited, thereby holding the particle constituents together, even when highly-excited. (It would be a problem for this theory if the fields were free to separate since one of the free fields would be the negative-energy ψ scalar field.) Also, via (9), the charge is proportional to

$$\alpha = \int_0^\infty y^2 z\, x^2 dx$$

which depends mainly on the charged scalar ψ (i.e., y) and only slightly, via the coupling, on the neutral scalar ψ_1 (i.e. y_1). So it is numerically easy to adjust the parameters to keep the charge conserved for all three states in a family.

Cooperstock and Rosen achieved their goal of modelling the complete lepton family by adding another neutral scalar field (ψ_2) to the two previous fields: the charged field (ψ) and the first neutral scalar field (ψ_1). So

$$\begin{aligned} L_{\mathrm{CR}} = &-\frac{F_{ab}F^{ab}}{8\pi} - (D^a\psi)(D_a\psi)^* + \sigma^2\psi\psi^* \\ &+ (\partial^a\psi_1)(\partial_a\psi_1)^* - \sigma^2\psi_1\psi_1^* + \frac{1}{2}g_1\,\psi_1{}^2\psi_1{}^{*2} \\ &+ (\partial^a\psi_2)(\partial_a\psi_2)^* - \sigma^2\psi_2\psi_2{}^* + \frac{1}{2}g_2\,\psi_2{}^2\psi_2{}^{*2} \\ &- f_1\psi\psi^*\psi_1\psi_1{}^* - f_2\psi\psi^*\psi_2\psi_2{}^* - f_3\psi_1\psi_1{}^*\psi_2\psi_2{}^*. \end{aligned}$$

This Lagrangian includes couplings between all three fields. (The last $\psi_1\psi_2$ coupling term was not used in [11] as it proved unecessary for that work.) It turns out this Lagrangian gives particles of the correct masses and size to model the lepton family (the electron, muon, and tau).

The field equations obtained by varying ψ^*, ${\psi_1}^*$, ${\psi_2}^*$, and A^u are

$$(D^a\psi)_{;a} - i\epsilon A_a(D^a\psi) + \sigma^2\psi - f_1(\psi_1{\psi_1}^*)\psi - f_2(\psi_2{\psi_2}^*)\psi = 0$$

$$(\partial^a\psi_1)_{;a} + \sigma^2\psi_1 - g_1({\psi_1})^2{\psi_1}^* + f_1(\psi\psi^*)\psi_1 + f_3(\psi_2{\psi_2}^*)\psi_1 = 0$$

$$(\partial^a\psi_2)_{;a} + \sigma^2\psi_2 - g_2({\psi_2})^2{\psi_2}^* + f_2(\psi\psi^*)\psi_2 + f_3(\psi_1{\psi_1}^*)\psi_2 = 0$$

$$F^{ua}{}_{;a} = -4\pi J^u$$

where

$$J^u \equiv \frac{1}{2}i\epsilon[\psi^*(D^u\psi) - \psi(D^u\psi)^*].$$

(Note, the subscripts 1 and 2 on ψ_1 and ψ_2 are not tensor indices; they serve only to distinguish the two neutral fields.)

Again looking at the case of static, spherical symmetry we have

$$A^0 = \varphi(r), \quad A^\alpha = 0 \quad (\alpha = 1,2,3)$$

$$\psi = \theta(r)e^{-i\epsilon\mu t} \quad \text{(charged field)}$$

$$\psi_1 = \theta_1(r) \quad \text{(neutral field)}$$

$$\psi_2 = \theta_2(r) \quad \text{(neutral field)}$$

possibilities. Using $-$ to represent the unexcited state, $\uparrow$ to represent the first excited state, and $\Uparrow$ the second excited state, one formulation is to only excite the ψ_2 (i.e., the y_2) field:

	y	y_1	y_2
TAU:	$-$	$-$	$\Uparrow$
MUON:	$-$	$-$	$\uparrow$
ELECTRON:	$-$	$-$	$-$

But as with (18) this does not successfully model all three particles.

But by exciting alternatively ψ_1 and ψ_2 (i.e., y_1 and y_2) in this scheme

	y	y_1	y_2
TAU:	$-$	$-$	$\uparrow$
MUON:	$-$	$\uparrow$	$-$
ELECTRON:	$-$	$-$	$-$

the correct masses are found for all three particles. But this scheme seems to suggest that the tau would naturally decay into an electron, whereas experimentally it is found to decay into both muons and electrons.

So Cooperstock and Rosen preferred the scheme

	y	y_1	y_2
TAU:	$-$	$\uparrow$	$\uparrow$
MUON:	$-$	$\uparrow$	$-$
ELECTRON:	$-$	$-$	$-$

where a tau can decay into a muon by de-exciting y_2, or decay into an electron by de-exciting both y_1 and y_2. Just de-exciting y_1 would presumably lead to a short-lived intermediate state. (However this is a purely *static* model and such questions of transitions may or may not be meaningful with the current model.)

So now

$$mass_{\text{low state}} = \frac{\sigma}{2\epsilon^2}\, \mathrm{w}_{\text{low state}} \tag{22}$$

$$mass_{\text{middle state}} = \frac{\sigma}{2\epsilon^2}\, \mathrm{w}_{\text{middle state}}$$

$$mass_{\text{high state}} = \frac{\sigma}{2\epsilon^2}\, \mathrm{w}_{\text{high state}}$$

where (for the EMT group as we have here) the "low state" is the electron state, the "middle state" is the muon state, and the "high state" is the tau state. So $\mathrm{w}_{\text{low state}}$ contains integrals with the unexcited (ground-state) y_1 and y_2 fields, and higher-state w's contain the excited fields. For this Lagrangian

$$\begin{aligned} \mathrm{w} = &\left(\frac{2\pi\epsilon^2}{g_1}\right)\int_0^\infty (y_1)^4 x^2 dx + \left(\frac{2\pi\epsilon^2}{g_2}\right)\int_0^\infty (y_2)^4 x^2 dx \\ &- \left(\frac{f_1}{g_1}\right)\int_0^\infty y^2 (y_1)^2 x^2 dx - \left(\frac{f_2}{g_2}\right)\int_0^\infty y^2 (y_2)^2 x^2 dx \\ &- \left(\frac{4\pi\epsilon^2 f_3}{g_1 g_2}\right)\int_0^\infty (y_1)^2 (y_2)^2 x^2 dx - (\alpha\beta + \gamma) \end{aligned} \tag{23}$$

where

$$\alpha \equiv \int_0^\infty y^2 z x^2 dx$$

$$\beta \equiv \frac{\epsilon\mu}{\sigma}$$

$$\gamma \equiv \int_0^\infty y^2 z^2 x^2 dx$$

as before.

It will also be useful to introduce some abbreviations for the factors appearing in the equation for w:

$$\text{FG1} \equiv -f_1/g_1 \tag{24}$$

$$\text{FG2} \equiv -f_2/g_2$$

$$\text{FE1} \equiv -f_1/4\pi\epsilon^2$$

$$\text{FE2} \equiv -f_2/4\pi\epsilon^2$$

$$\text{FG3} \equiv \frac{+f_1 f_3}{g_1 g_2}$$

$$\text{I}_1 \equiv \int_0^\infty {y_1}^4 x^2 dx/2$$

$$\text{I}_2 \equiv \int_0^\infty {y_2}^4 x^2 dx/2$$

$$\text{I}_3 \equiv \int_0^\infty y^2 {y_1}^2 x^2 dx$$

$$\text{I}_4 \equiv \int_0^\infty y^2 {y_2}^2 x^2 dx$$

$$\text{I}_5 \equiv \int_0^\infty {y_1}^2 {y_2}^2 x^2 dx$$

so that

$$w = \left(\frac{FG1}{FE1}\right) I_1 + \left(\frac{FG2}{FE2}\right) I_2 + (FG1)\ I_3 \qquad (25)$$
$$+ (FG2)\ I_4 + \left(\frac{FG3}{FE1}\right) I_5 - (\alpha\beta + \gamma).$$

So if we choose our input parameters FG1, FG2, FG3, FE1, and FE2, and then find $y(x)$, $z(x)$, $y_1(x)$, and $y_2(x)$ satisifying their respective boundary conditions and giving particle-like solutions, how do we work backwards to obtain the other parameters?

1) $\alpha = \int_0^\infty y^2 z x^2 dx$ and $\gamma = \int_0^\infty y^2 z^2 x^2 dx$ so α and γ can both be integrated easily. (At least they can be approximated since we can only integrate a finite distance down the x-axis before the previously-mentioned instability of the e^x/x term causes the integrals to diverge. But if care is taken, the exponential decay of y and x ensures that the integrals can be approximated to enough accuracy with little trouble.)

2) We also know via (11) that

$$z \to \frac{\alpha}{x} + \beta \quad \text{as } x \to \infty$$

so once we determine α we can also determine β.

3) $w_{\text{low state}}$, $w_{\text{middle state}}$, and $w_{\text{high state}}$ are calculated via (25) (given input values for FG1 etc.).

4) From (22) and (21) we have

$$\sigma = \frac{\text{mass}_{\text{low state}}}{w_{\text{low state}}} 2\epsilon^2$$

$$= \frac{\text{mass}_{\text{low state}}}{w_{\text{low state}}} \frac{2\alpha^2}{q^2}$$

so σ can be evaluated once $w_{\text{low state}}$ and α have been calculated (and using the *known* charges and masses).

5) Then ϵ is obtained from

$$\epsilon = \frac{\alpha}{q}.$$

(We assume that $\epsilon_{\text{low state}} = \epsilon_{\text{middle state}} = \epsilon_{\text{high state}}$, and that q (the charge) is the same for all three states, so we have to adjust the parameters to keep $\alpha_{\text{low state}} = \alpha_{\text{middle state}} = \alpha_{\text{high state}}$.)

The next step is to adjust the initial values (i.e., $y(0)_{\text{low state}}$, $y(0)_{\text{middle state}}$, and $y(0)_{\text{high state}}$) to try to make $\alpha_{\text{low state}} = \alpha_{\text{middle state}} = \alpha_{\text{high state}}$ (so that charge is conserved), and to adjust the F's (i.e., FG1, FG2, FG3, FE1, and FE2) to try to make

$$\frac{w_{\text{middle state}}}{w_{\text{low state}}} = \frac{\text{mass}_{\text{middle state}}}{\text{mass}_{\text{low state}}} \qquad (26)$$

$$\frac{w_{\text{high state}}}{w_{\text{low state}}} = \frac{\text{mass}_{\text{high state}}}{\text{mass}_{\text{low state}}}.$$

Of course these new F's and $y(0)$'s will alter the solution so the whole process has to be repeated until, after many trials, everything converges. The aim is to find a set of parameters (the F's and the initial-values) so that: 1) charge is conserved, and 2) the correct mass-ratios (26) are obtained. Then $1/\sigma$ provides a check of the theory when compared to the particle size.

Table 2 shows the results as found by Cooperstock and Rosen.

	ELECTRON	MUON	TAU
$y(0)$	1.6300	1.6300	1.6243
$z(0)$	2.2112	2.2112	2.2025
$y_1(0)$	4.3383	14.1038	14.1049
$y_2(0)$	4.3431	4.4341	14.1049
w	4.286×10^{-3}	0.8869	14.974
α	1.9051	1.9051	1.9051
β	0.00264	0.00264	0.00489
γ	2.8168	2.8168	2.8077

$$
\begin{aligned}
\text{FG1} &= -2.770872 \times 10^{-05} \\
\text{FG2} &= -2.652788 \times 10^{-03} \\
\text{FE1} &= -5.0 \times 10^{-04} \\
\text{FE2} &= -3.0 \times 10^{-03} \\
\sigma &= 6.013 \times 10^{15}\ \text{cm}^{-1} \\
1/\sigma &= 1.663 \times 10^{-16}\ \text{cm} \\
\epsilon &= -1.379 \times 10^{34}\ \text{cm}^{-1}
\end{aligned}
$$

$$
\begin{aligned}
w_{\text{muon}}/w_{\text{electron}} &= 206.9 \quad \text{error} = -0.06\% \\
w_{\text{tau}}/w_{\text{electron}} &= 3494 \quad \text{error} = -0.06\%
\end{aligned}
$$

TABLE 2: *The solution for the EMT group as found by Cooperstock and Rosen.*

(Here ϵ, which is related to the charge, has been expressed in dimensions of length via the use of "geometric units." See Appendix C.)

In geometric units

$$e = 6.77 \times 10^{-56}\text{cm}$$

$$\text{mass of an electron} = 1.38 \times 10^{-34}\text{cm}$$

so e/m $\sim 10^{21}$. However, Cooperstock and Rosen found that when gravity was studied in this model, every charged particle where gravity played a significant

role led to an e/m ratio ~ 1, with a radius of $<\sim 10^{-33}$cm. So they conclude that gravity does *not* play a significant role in lepton structure, and that the electron radius has a lower bound well above 10^{-33} cm.

* * *

CHAPTER 3: NEW WORK: FITTING LEPTONS AND QUARKS

Now suppose we wish to extend the model to include quarks (which, like the leptons, are also spin 1/2 particles). They can be grouped into families paralleling the EMT (electron/muon/tau) group:

	EMT group	UCT group	DSB group
Third generation:	TAU	TOP	BOTTOM
Second generation:	MUON	CHARM	STRANGE
First generation:	ELECTRON	UP	DOWN
Charge (in units of e):	-1	$+2/3$	$-1/3$

At this point we do not know what parameters can be held constant and which should be allowed to vary. Another problem is that the masses of quarks are model-dependent and not clearly defined. Various estimates of quark masses are given in Table 3.

	Griffiths				
	Bare mass	Effective mass in mesons	Effective mass in baryons	Veltman	Physics Letters
TOP:	>23,000	>23,000	>23,000	22,500	50,000
CHARM:	1,100	1,500	1,500	1,500	1,350
UP:	4.2	310	363	310	5.6
CHARM/UP:	261.9	4.8	4.1	4.8	241.0
TOP/UP:	5,476.2	74.2	63.4	72.6	8,928.6
BOTTOM:	4,200	4,700	4,700	≃5,000	5,000
STRANGE:	150	483	538	505	199
DOWN:	7.5	310	363	310	9.9
STRANGE/DOWN:	20.0	1.6	1.5	1.6	20.1
BOTTOM/DOWN:	560.0	15.2	12.9	16.1	505.0

TABLE 3: *Different values of estimates for the masses of quarks and the resulting mass ratios (in MeV/c^2) from Griffiths [8], Veltman [13], and* Physics Letters *[14].*

The masses of the leptons are[11]

$$\text{electron mass} = 0.51099906 \text{ MeV/c}^2$$

$$\text{muon mass} = 105.65839 \text{ MeV/c}^2$$

$$\text{tau mass} = 1784.1 \text{ MeV/c}^2$$

So to fit the quarks we need to find the correct set of coupling constants (F's) and initial conditions for each particle so that: 1) we get the correct α's, and hence the correct charges, and 2) we get the correct w ratios, and hence the correct mass ratios. Then, as an independent check of the model, we will have a prediction of the particle sizes (through $1/\sigma$).

[11] *Physics Letters*, [14], page 14.

The first attempt made was to use Cooperstock and Rosen's values for the EMT group as in Table 2, and to try to find different F's to fit the UCT (up/charm/top) group. This proved difficult to do as it forced the search into areas where $\beta \simeq 1$, which is numerically unstable, so another approach was tried.

In looking at previously unpublished data [15] of solutions to the pure Rosen equations for negative-mass particles

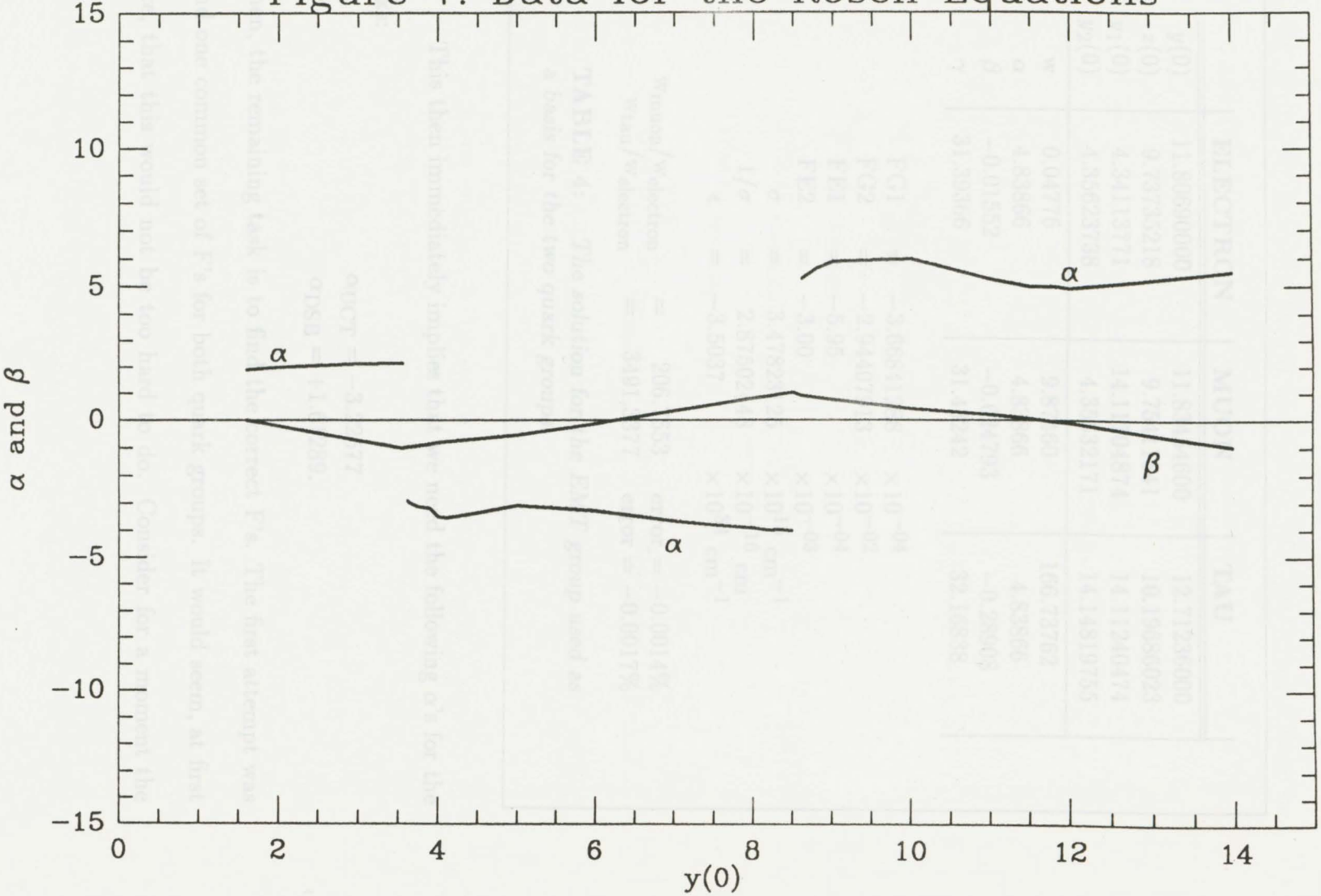
$$y'' + \frac{2}{x}y' - y + yz^2 = 0$$

$$z'' + \frac{2}{x}z' + y^2 z = 0$$

it can be seen that there naturally occur three different "regimes" (see Figure 7). In particular notice how α naturally changes sign in the middle regime. This sign change, because of α's connection with the charge via (9), easily accomodates the alternating charges of the three groups. (If α did *not* change sign it could still be forced to do so by changing the sign of z, but the approach used here seems more elegant.)

We assume that solutions to the full equations are not too far removed from solutions to the simpler Rosen equations, so the first step was to fix an EMT solution in the high regime and use that as a basis for the other two quark groups. Once α_{EMT} is fixed, then, because of the known charge ratios, α_{UCT} and α_{DSB} are thereby fixed also. This was done; the solution is shown in Table 4. In particular we have $\alpha_{\mathrm{EMT}} = +4.83866$.

Figure 7: Data for the Rosen Equations



	ELECTRON	MUON	TAU
$y(0)$	11.80690000	11.83454600	12.71236000
$z(0)$	9.73735218	9.75411841	10.19686023
$y_1(0)$	4.34113771	14.11104874	14.11240474
$y_2(0)$	4.35623738	4.35632171	14.14819755
w	0.04776	9.87460	166.73762
α	4.83866	4.83866	4.83866
β	−0.01552	−0.024793	−0.28908
γ	31.39366	31.43242	32.16838

$$
\begin{aligned}
\text{FG1} &= -3.66841288 \times 10^{-04} \\
\text{FG2} &= -2.94407913 \times 10^{-02} \\
\text{FE1} &= -5.95 \times 10^{-04} \\
\text{FE2} &= -3.00 \times 10^{-03} \\
\sigma &= 3.47823125 \times 10^{15}\ \text{cm}^{-1} \\
1/\sigma &= 2.87502448 \times 10^{-16}\ \text{cm} \\
\epsilon &= -3.5037 \times 10^{34}\ \text{cm}^{-1}
\end{aligned}
$$

$$
\begin{aligned}
w_{\text{muon}}/w_{\text{electron}} &= 206.7653 \quad \text{error} = -0.0014\% \\
w_{\text{tau}}/w_{\text{electron}} &= 3491.3377 \quad \text{error} = -0.0017\%
\end{aligned}
$$

TABLE 4: *The solution for the EMT group used as a basis for the two quark groups.*

This then immediately implies that we need the following α's for the quarks:

$$
\begin{aligned}
\alpha_{\text{UCT}} &= -3.22577 \\
\alpha_{\text{DSB}} &= +1.61289.
\end{aligned}
$$

So then, the remaining task is to find the correct F's. The first attempt was to find one common set of F's for both quark groups. It would seem, at first glance, that this would not be too hard to do. Consider for a moment the

integrals in (23) to be fixed. Then if we consider FG1, FG2, and FE2 to be variables and FE1 and FG3 to be fixed, we can define

$$a \equiv \mathrm{FG1}$$

$$b \equiv \mathrm{FG2/FE2}$$

$$c \equiv \mathrm{FG2}.$$

Then we define

$$\mathrm{A} \equiv \frac{\mathrm{I}_{1,\ \text{low state}}}{\mathrm{FE1}}$$

$$\mathrm{B} \equiv \mathrm{I}_{2,\ \text{low state}}$$

$$\mathrm{C} \equiv \mathrm{I}_{3,\ \text{low state}}$$

$$\mathrm{D} \equiv \mathrm{I}_{4,\ \text{low state}}$$

$$\mathrm{E} \equiv \frac{\mathrm{FG3}}{\mathrm{FE1}} \cdot \mathrm{I}_{5,\ \text{low state}} - (\alpha\beta + \gamma)_{\text{low state}}.$$

Then F, G, H, J, K are defined similarly for the middle state, and M, N, P, Q, and R are defined similarly for the high state so that

$$\mathrm{w_{up}} = a \cdot \mathrm{A_{UCT}} + b \cdot \mathrm{B_{UCT}} + a \cdot \mathrm{C_{UCT}} + c \cdot \mathrm{D_{UCT}} + \mathrm{E_{UCT}}$$

$$\mathrm{w_{charm}} = a \cdot \mathrm{F_{UCT}} + b \cdot \mathrm{G_{UCT}} + a \cdot \mathrm{H_{UCT}} + c \cdot \mathrm{J_{UCT}} + \mathrm{K_{UCT}}$$

$$\mathrm{w_{down}} = a \cdot \mathrm{A_{DSB}} + b \cdot \mathrm{B_{DSB}} + a \cdot \mathrm{C_{DSB}} + c \cdot \mathrm{D_{DSB}} + \mathrm{E_{DSB}}$$

$$\mathrm{w_{strange}} = a \cdot \mathrm{F_{DSB}} + b \cdot \mathrm{G_{DSB}} + a \cdot \mathrm{H_{DSB}} + c \cdot \mathrm{J_{DSB}} + \mathrm{K_{DSB}}$$

$$\mathrm{w_{bottom}} = a \cdot \mathrm{M_{DSB}} + b \cdot \mathrm{N_{DSB}} + a \cdot \mathrm{P_{DSB}} + c \cdot \mathrm{Q_{DSB}} + \mathrm{S_{DSB}}.$$

If we define

$$\mathrm{R1} \equiv \frac{\mathrm{mass}_{\text{middle state}}}{\mathrm{mass}_{\text{low state}}} = \frac{\mathrm{w}_{\text{middle state}}}{\mathrm{w}_{\text{low state}}}$$

$$R2 \equiv \frac{\text{mass}_{\text{high state}}}{\text{mass}_{\text{low state}}} = \frac{w_{\text{high state}}}{w_{\text{low state}}}$$

then we have

$$R1 \cdot w_{\text{low state}} = w_{\text{middle state}}$$

so that

$$R1 \cdot \left[a(A + C) + bB + cD + E\right] \quad = \quad a(F + H) + bG + cJ + K$$

for each family. Therefore

$$a\left[R1(A + C) - (F + H)\right] + b\left[R1 \cdot B - G\right] + c\left[R1 \cdot D - J\right] \quad = \quad \left[K - R1 \cdot E\right].$$

Similarly

$$a\left[R2(A + C) - (M + P)\right] + b\left[R2 \cdot B - N\right] + c\left[R2 \cdot D - Q\right] \quad = \quad \left[S - R2 \cdot E\right].$$

If we consider the mass of the top quark unknown then we are led to the matrix equation

$$\begin{bmatrix} R1_{UCT} \cdot (A_{UCT} + C_{UCT}) - (F_{UCT} + H_{UCT}) & R1_{UCT} \cdot B_{UCT} - G_{UCT} & R1_{UCT} \cdot D_{UCT} - J_{UCT} \\ R1_{DSB} \cdot (A_{DSB} + C_{DSB}) - (F_{DSB} + H_{DSB}) & R1_{DSB} \cdot B_{DSB} - G_{DSB} & R1_{DSB} \cdot D_{DSB} - J_{DSB} \\ R2_{DSB} \cdot (A_{DSB} + C_{DSB}) - (M_{DSB} + P_{DSB}) & R2_{DSB} \cdot B_{DSB} - N_{DSB} & R2_{DSB} \cdot D_{DSB} - Q_{DSB} \end{bmatrix}$$

$$\times \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} K_{UCT} - R1_{UCT} \cdot E_{UCT} \\ K_{DSB} - R1_{DSB} \cdot E_{DSB} \\ S_{DSB} - R2_{DSB} \cdot E_{DSB} \end{bmatrix}.$$

This then gives us a set of three equations in three unknowns (a, b, and c), which can be solved for "perfect" F's that, with the given integrals, will give the desired mass ratios: bottom/down, strange/down, and charm/up.

Of course in actuality the integrals are *not* fixed, but instead depend on the functions y, z, y_1, and y_2, which in turn depend on the F's, so these calculated "perfect" F's then have to be used to obtain new solutions for y, z, y_1, and y_2, and hence obtain new integrals. The hope was that this process would iteratively converge to an answer with the desired accuracy (and simultaneously provide an independent estimate of the mass of the top quark). However the problem arises that the calculated "perfect" F's are very large, on the order of 1, and with such strong coupling between the equations, solutions are almost impossible to obtain. The other possible excitation schemes were tried as well, but they also require similar stong couplings. (Unfortunately it seems impossible to *prove* that such a set of F's does not exist.)

Various other schemes were tried too. Using the two mass ratios ($w_{\text{middle state}}/w_{\text{low state}}$, and $w_{\text{high state}}/w_{\text{low state}}$) for the EMT, UCT, and DSB groups yields six equations in five unknowns (the five F's), so the overdetermined system can be solved if one equation is discarded. Or, putting FG3=0 and fitting only the UCT and DSB groups gives four equations in four unknowns. Another approach is to specify (arbitrarily) w_{up} and w_{down} and then use the known mass ratios to calculate w_{charm}, w_{strange}, and w_{bottom}, and hence generate five equations in five unknowns. All these various matrix methods, and others, proved fruitless.

A slightly different approach was also tried. A parameter-fitting program[12] was used to try to adjust each parameter in turn to try to converge

[12] Listings and full details of the programs used are in Appendices D and E.

to the desired solution. This has two advantages: it does not require an equal number of equations and unknowns and so can deal effectively with both under-determined and over-determined systems. Also, it does not seek an *exact* solution only a solution which minimizes the error. Thus one could hope that it might be able to iteratively converge to a solution where the matrix methods fail. But both it and another parameter-fitting program that was written failed to find a common set of F's that would fit even just the two quark families.

Such a task does not seem, at first glance, to be so hard; why does it fail? One reason is that when fitting just *one* group the F's can be adjusted so that $w_{\text{low state}}$ is small. Then minor variations in $w_{\text{middle state}}$ and $w_{\text{high state}}$ produce large variations in the ratios $w_{\text{middle state}}/w_{\text{low state}}$ and $w_{\text{high state}}/w_{\text{low state}}$. So if $w_{\text{low state}}$ is small, only minor variations in F's can fit the two mass ratios for that group very closely. However the situation is completely different as soon as it is attempted to fit two groups with the same F's. Now a typical situation is that the F's that make $w_{\text{low state}}$ small for one group also make $w_{\text{low state}} \approx 10$ for the other group. And so much larger changes in $w_{\text{middle state}}$ and $w_{\text{high state}}$ are needed to fit the two ratios for that group. It is difficulties like these that stop the iterative procedure from working.

But can we achieve something less restrictive? After all, we do not know whether there is *any* necessary relation between the F's of one group and another. So suppose we attempt to find solutions to all three groups where

two of the F's (take FE1 and FE2 for convenience) are the same for all three groups, and FG1 and FG2 vary from group to group. (FG3 is put equal to zero since it is not needed now to achieve the desired mass ratios.) Such solutions *can* be found. They are obviously not unique since they depend on: 1) the decision of *which* two F's to hold constant, 2) the values for those constant F's, and 3) the value of $y(0)$, and hence α, used for the basis reference group (in this case the EMT group). One such solution is given in Table 5.

This solution gives the small mass ratios of the quarks' effective masses in baryons (as in Table 3). However, with the freedom of two F's to vary for each group, we can adjust the mass ratios to fit either small or large ratios. So another solution is given in Table 6 which fits the large mass ratios of the quarks' bare masses (also as in Table 3).

The graphs of the basic EMT solution is shown in Figures 8 through 10, and the small-ratio UCT and DSB solutions are shown in Figures 11 through 16. (The large-ratio UCT and DSB solutions look very similar.)

Both solutions yield a particle size (through $1/\sigma$) on the order of 10^{-17} or 10^{-18} cm.

So this work shows clearly that this classical field-theory approach is useful for both leptons *and* quarks.

* * *

	UP	CHARM	TOP
$y(0)$	5.95643900	5.95954400	6.24493500
$z(0)$	5.65725624	5.65907746	5.82410012
$y_1(0)$	4.33858952	14.10955750	14.10987056
$y_2(0)$	4.34342337	4.34342533	14.13530840
w	1.23183	5.090370	78.07905
α	−3.22577	−3.22577	−3.22577
β	−0.15296	−0.15213	−0.07695
γ	12.76646	12.76822	12.91677

$$
\begin{aligned}
\text{FG1} &= -1.44096790 \times 10^{-04} \\
\text{FG2} &= -1.37288390 \times 10^{-02} \\
\text{FE1} &= -5.95 \times 10^{-04} \\
\text{FE2} &= -3.00 \times 10^{-03} \\
\sigma &= 9.57933143 \times 10^{16}\ \text{cm}^{-1} \\
1/\sigma &= 1.04391418 \times 10^{-17}\ \text{cm} \\
\epsilon &= -3.5037 \times 10^{34}\ \text{cm}^{-1}
\end{aligned}
$$

$$
\begin{aligned}
w_{charm}/w_{up} &= 4.1324 \quad \text{error} = 0.0031\% \\
w_{top}/w_{up} &= 63.3845 \quad \text{error} = 0.0373\%
\end{aligned}
$$

	DOWN	STRANGE	BOTTOM
$y(0)$	0.91072000	0.91079000	0.91860000
$z(0)$	1.64385770	1.64386984	1.64485315
$y_1(0)$	4.33796458	14.10395660	14.10395619
$y_2(0)$	4.34029683	4.34029705	14.10545745
w	2.13685	3.16700	27.66712
α	1.61289	1.61289	1.61289
β	0.41928	0.41924	0.41474
γ	2.00596	2.00596	2.005663

$$
\begin{aligned}
\text{FG1} &= -3.84780260 \times 10^{-05} \\
\text{FG2} &= -4.61288970 \times 10^{-03} \\
\text{FE1} &= -5.95 \times 10^{-04} \\
\text{FE2} &= -3.00 \times 10^{-03} \\
\sigma &= 5.52225788 \times 10^{16}\ \text{cm}^{-1} \\
1/\sigma &= 1.81085355 \times 10^{-17}\ \text{cm} \\
\epsilon &= -3.5037 \times 10^{34}\ \text{cm}^{-1}
\end{aligned}
$$

$$
\begin{aligned}
w_{strange}/w_{down} &= 1.4821 \quad \text{error} = 0.0001\% \\
w_{bottom}/w_{down} &= 12.9476 \quad \text{error} = 0.0001\%
\end{aligned}
$$

TABLE 5: *Solutions for the UCT and DSB groups which achieve the small mass ratios of the quarks' effective masses in baryons.*

	UP	CHARM	TOP
$y(0)$	5.94912000	5.95182000	6.21557000
$z(0)$	5.65519001	5.65677482	5.80965891
$y_1(0)$	4.33858466	14.10954358	14.10983931
$y_2(0)$	4.34339885	4.34340053	14.13515024
w	0.01286	3.36865	70.44521
α	−3.22577	−3.22577	−3.22577
β	−0.15585	−0.15513	−0.08555
γ	12.76655	12.76808	12.90706

$$
\begin{aligned}
\text{FG1} &= -1.25321400 \times 10^{-04} \\
\text{FG2} &= -1.26169700 \times 10^{-02} \\
\text{FE1} &= -5.95 \times 10^{-04} \\
\text{FE2} &= -3.00 \times 10^{-03} \\
\sigma &= 1.06104654 \times 10^{17}\ \text{cm}^{-1} \\
1/\sigma &= 9.42465733 \times 10^{-18}\ \text{cm} \\
\epsilon &= -3.5037 \times 10^{34}\ \text{cm}^{-1}
\end{aligned}
$$

$$
\begin{aligned}
w_{\text{charm}}/w_{\text{up}} &= 261.7944 \quad \text{error} = -0.0421\% \\
w_{\text{top}}/w_{\text{up}} &= 5474.6512 \quad \text{error} = -0.0281\%
\end{aligned}
$$

	DOWN	STRANGE	BOTTOM
$y(0)$	0.90281000	0.90284900	0.90736000
$z(0)$	1.63971508	1.63972539	1.64037538
$y_1(0)$	4.33795939	14.10395815	14.10395790
$y_2(0)$	4.34027067	4.34027080	14.10546611
w	0.02568	0.51355	14.38032
α	1.61289	1.61289	1.61289
β	0.42185	0.42183	0.41925
γ	2.00349	2.00350	2.00340

$$
\begin{aligned}
\text{FG1} &= -1.82228970 \times 10^{-05} \\
\text{FG2} &= -2.61084380 \times 10^{-03} \\
\text{FE1} &= -5.95 \times 10^{-04} \\
\text{FE2} &= -3.00 \times 10^{-03} \\
\sigma &= 9.49548009 \times 10^{16}\ \text{cm}^{-1} \\
1/\sigma &= 1.05313264 \times 10^{-17}\ \text{cm} \\
\epsilon &= -3.5037 \times 10^{34}\ \text{cm}^{-1}
\end{aligned}
$$

$$
\begin{aligned}
w_{\text{strange}}/w_{\text{down}} &= 20.0012 \quad \text{error} = 0.0058\% \\
w_{\text{bottom}}/w_{\text{down}} &= 560.0705 \quad \text{error} = 0.0126\%
\end{aligned}
$$

TABLE 6: *Solutions for the UCT and DSB groups which achieve the large mass ratios of the quarks' bare masses.*

Figure 8: The Basic Electron Solution

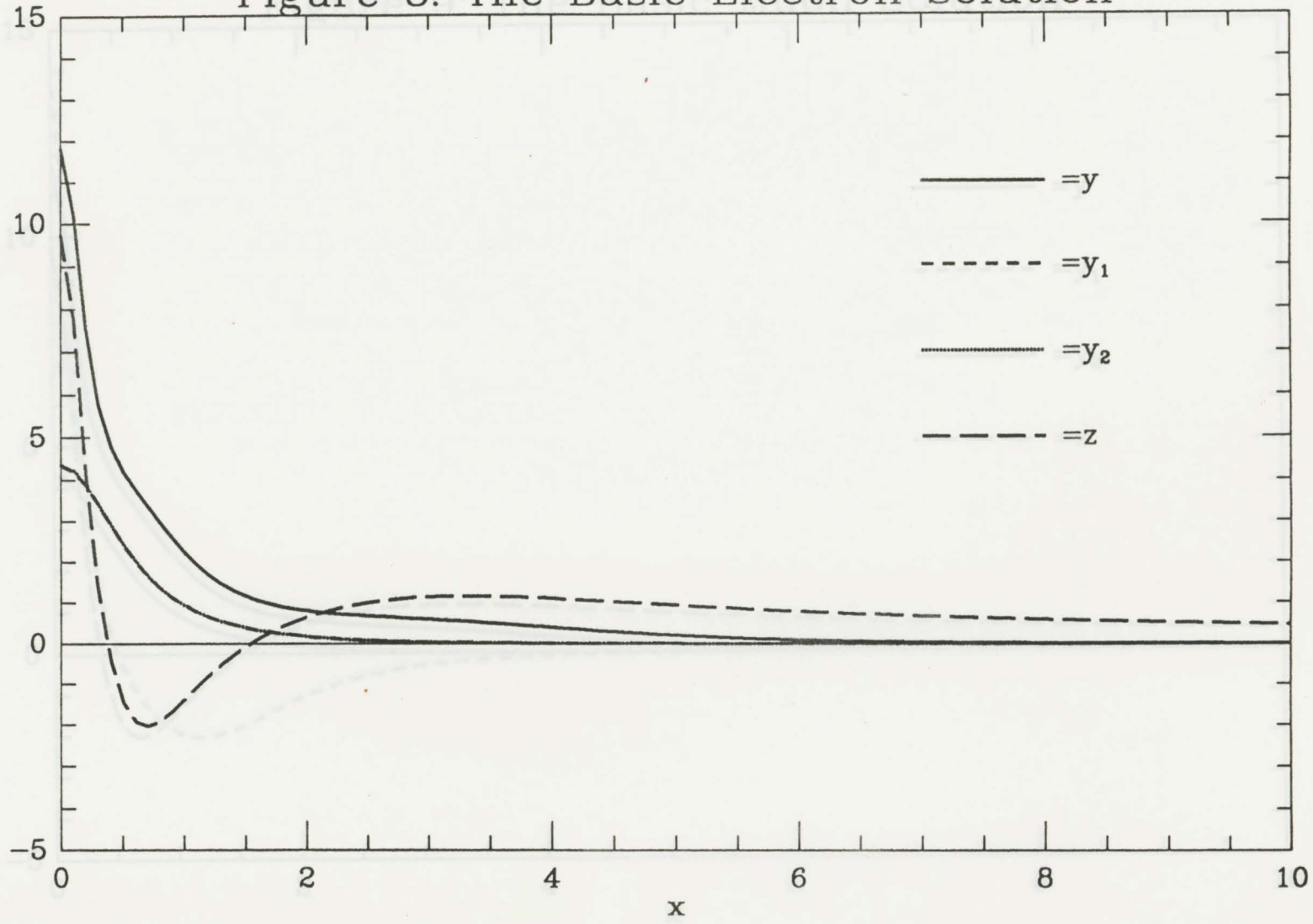


Figure 9: The Basic Muon Solution

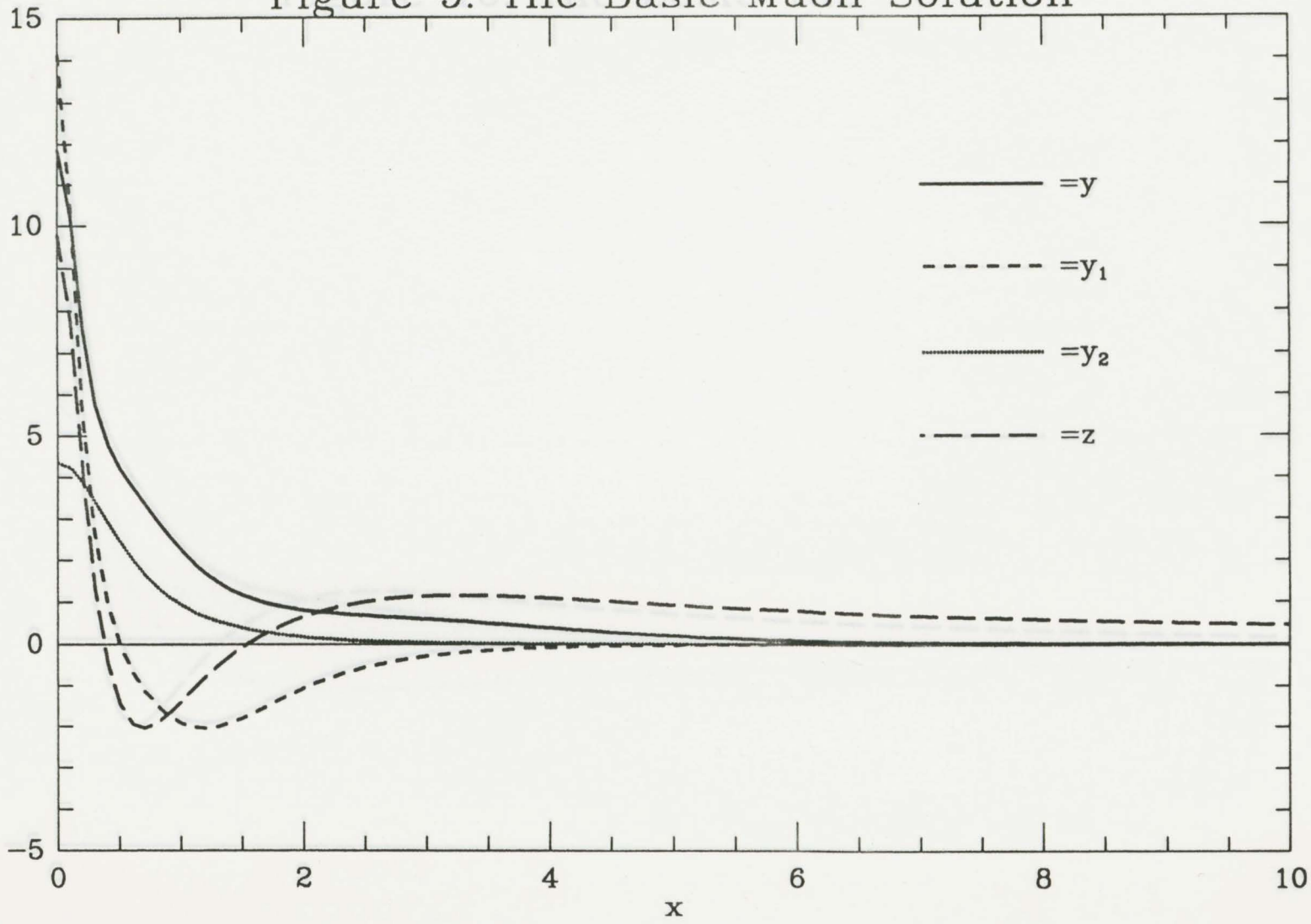


Figure 10: The Basic Tau Solution

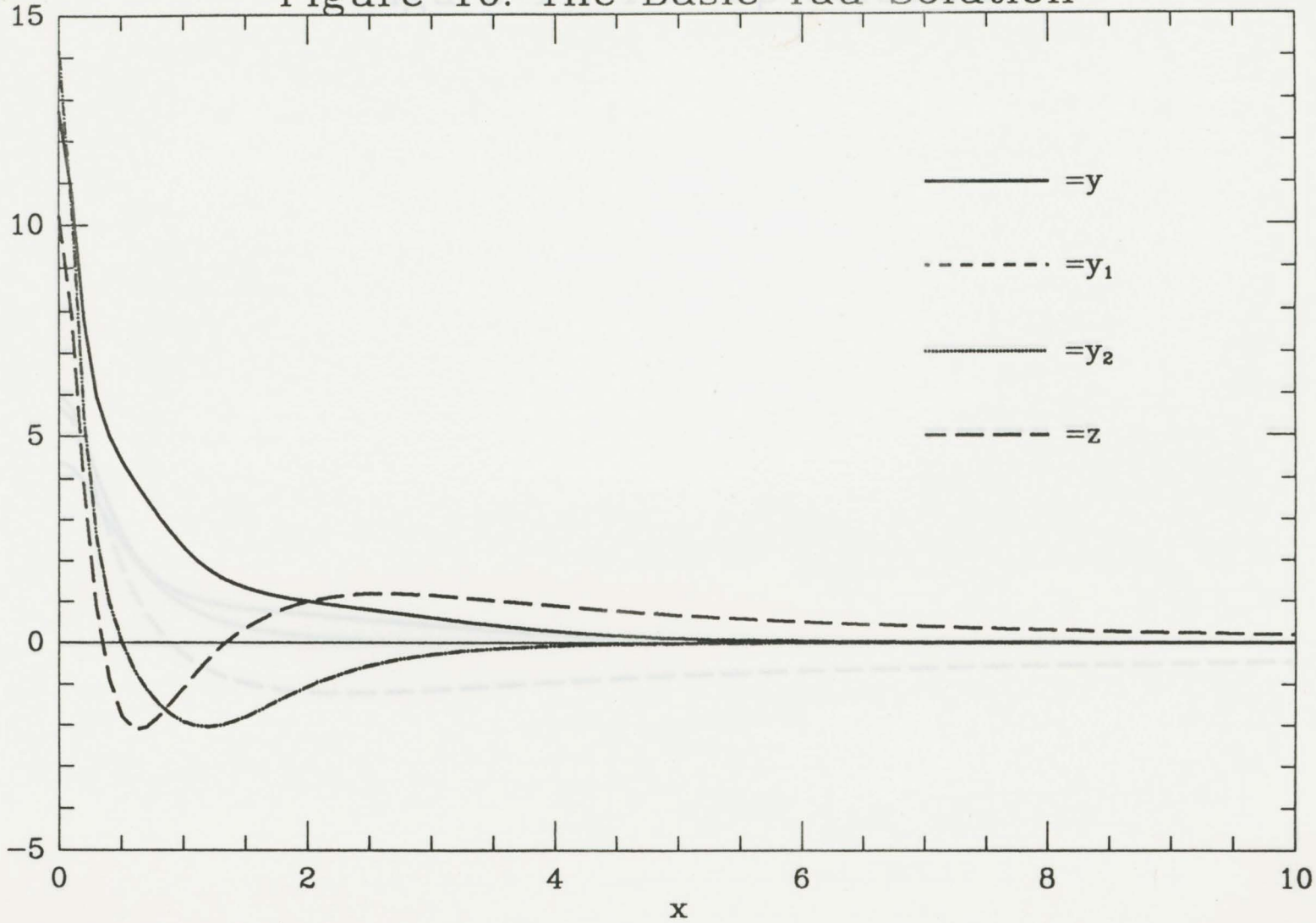


Figure 11: The Up Solution

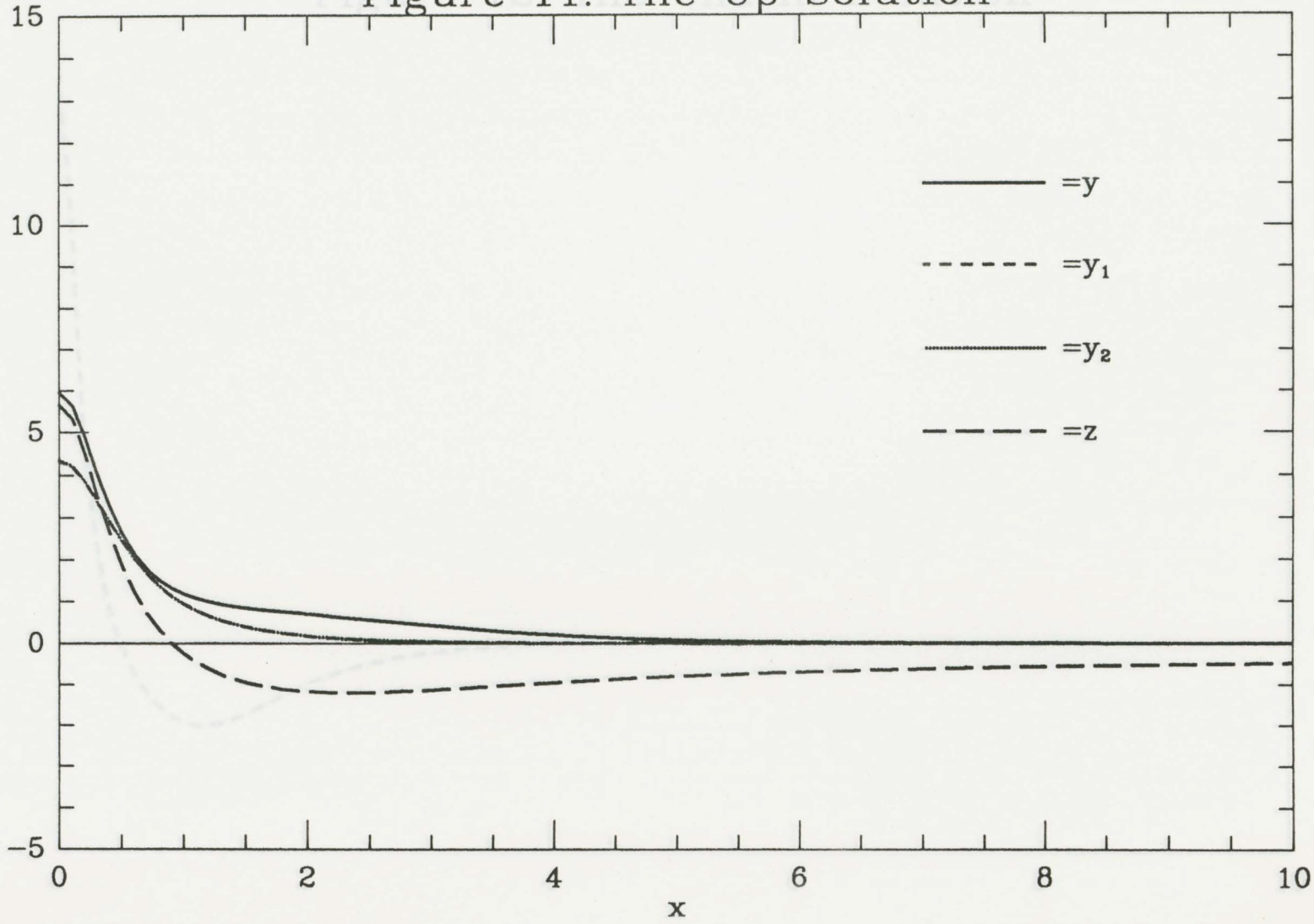


Figure 12: The Charm Solution

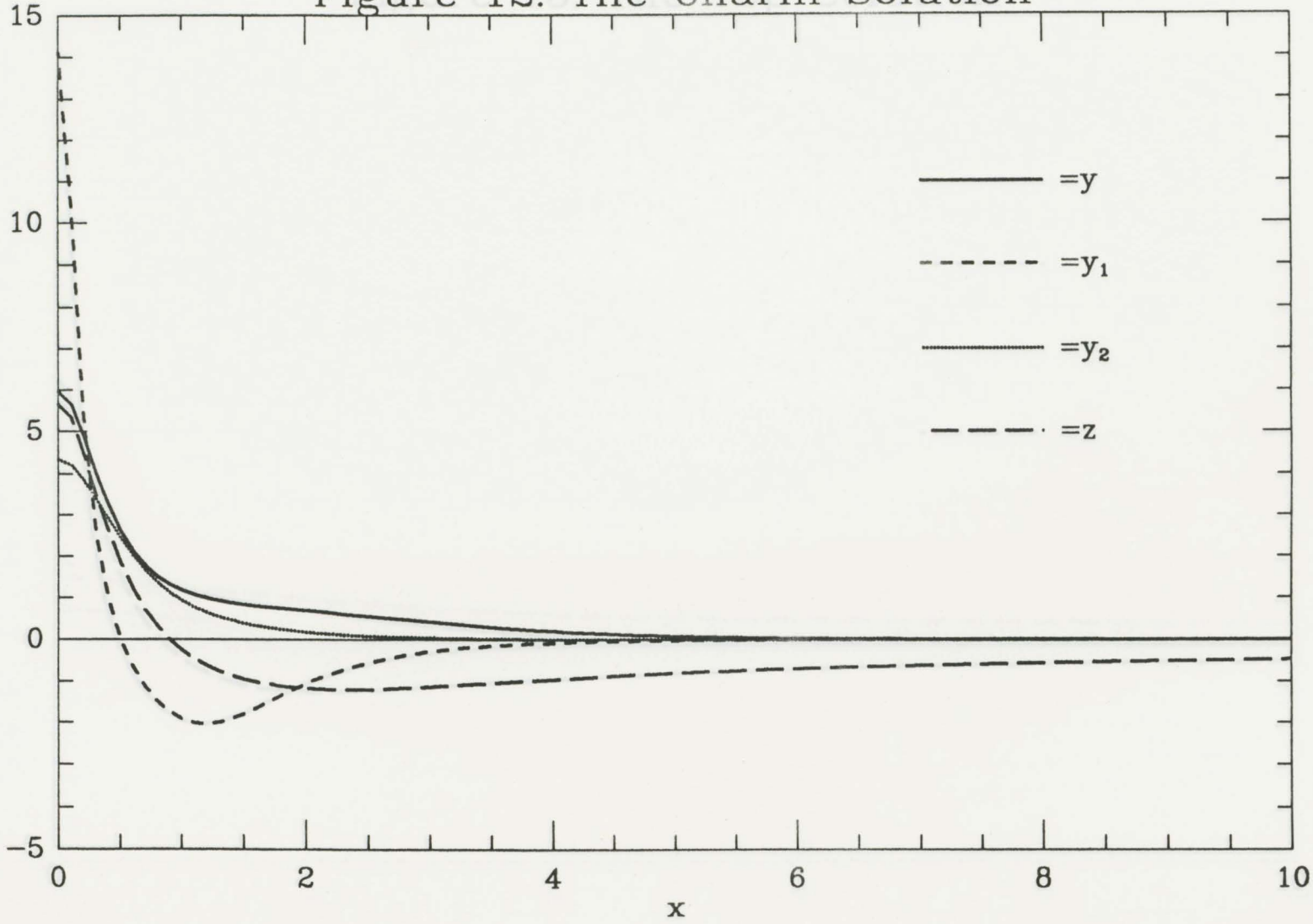


Figure 13: The Top Solution

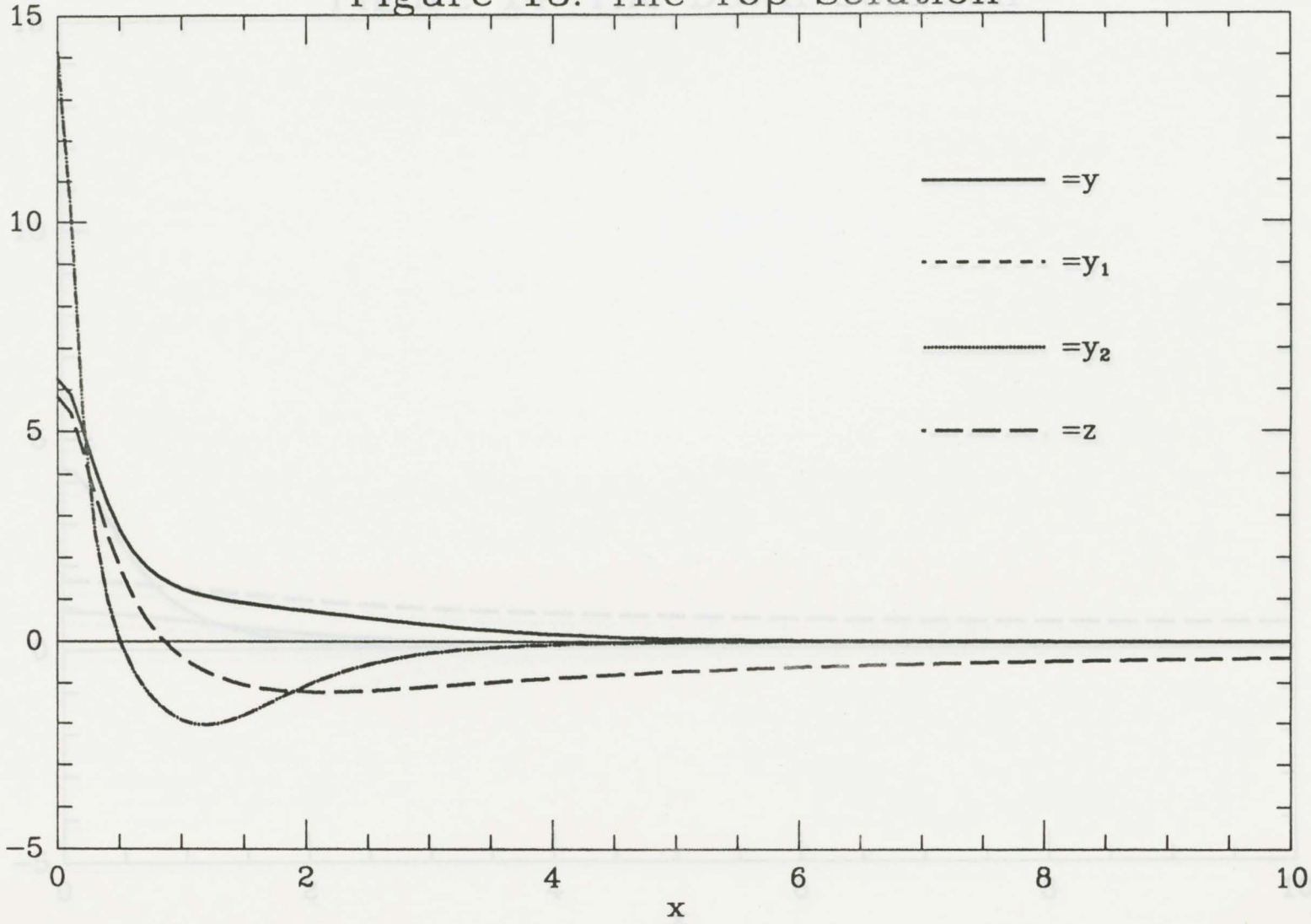


Figure 14: The Down Solution

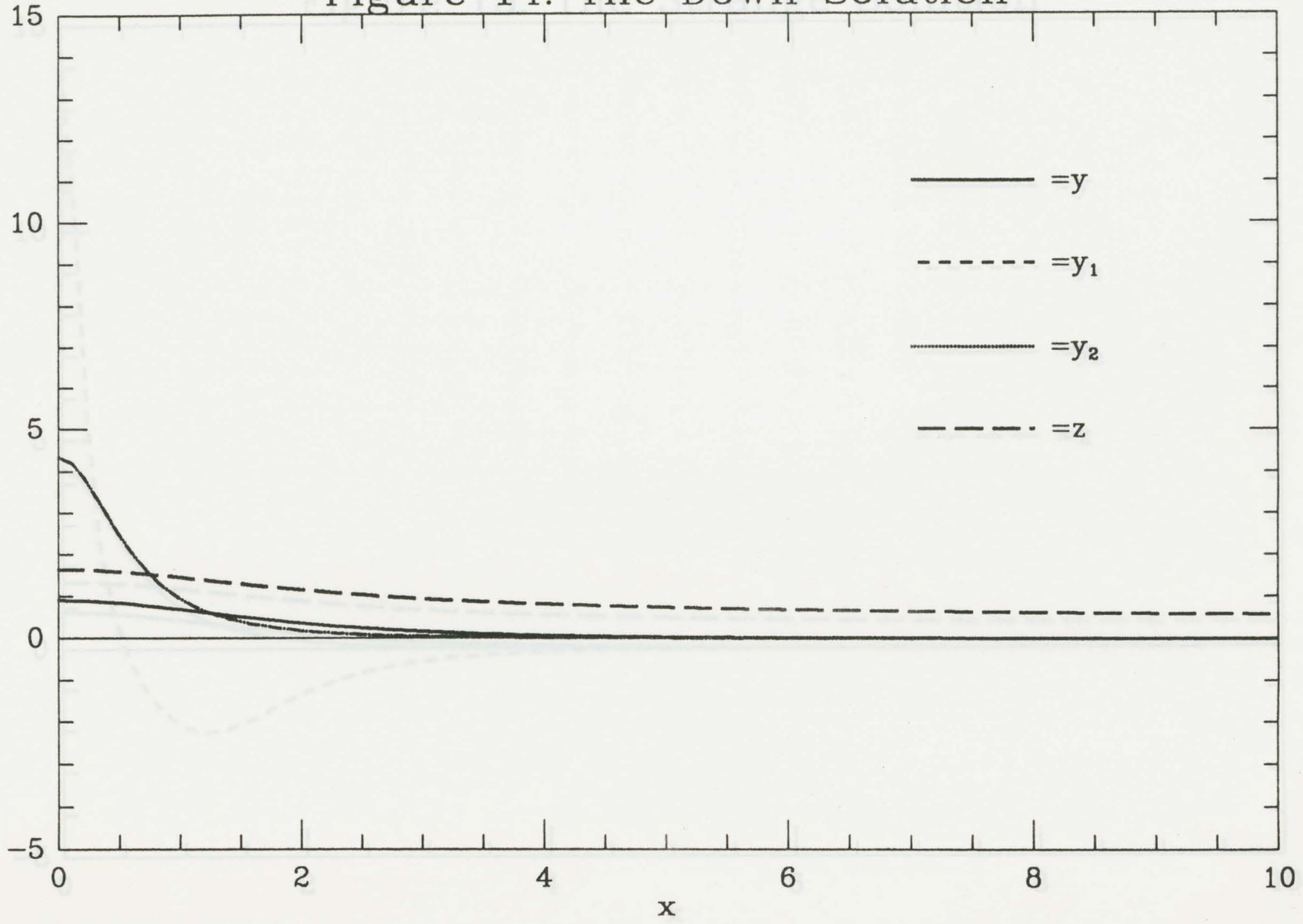


Figure 15: The Strange Solution

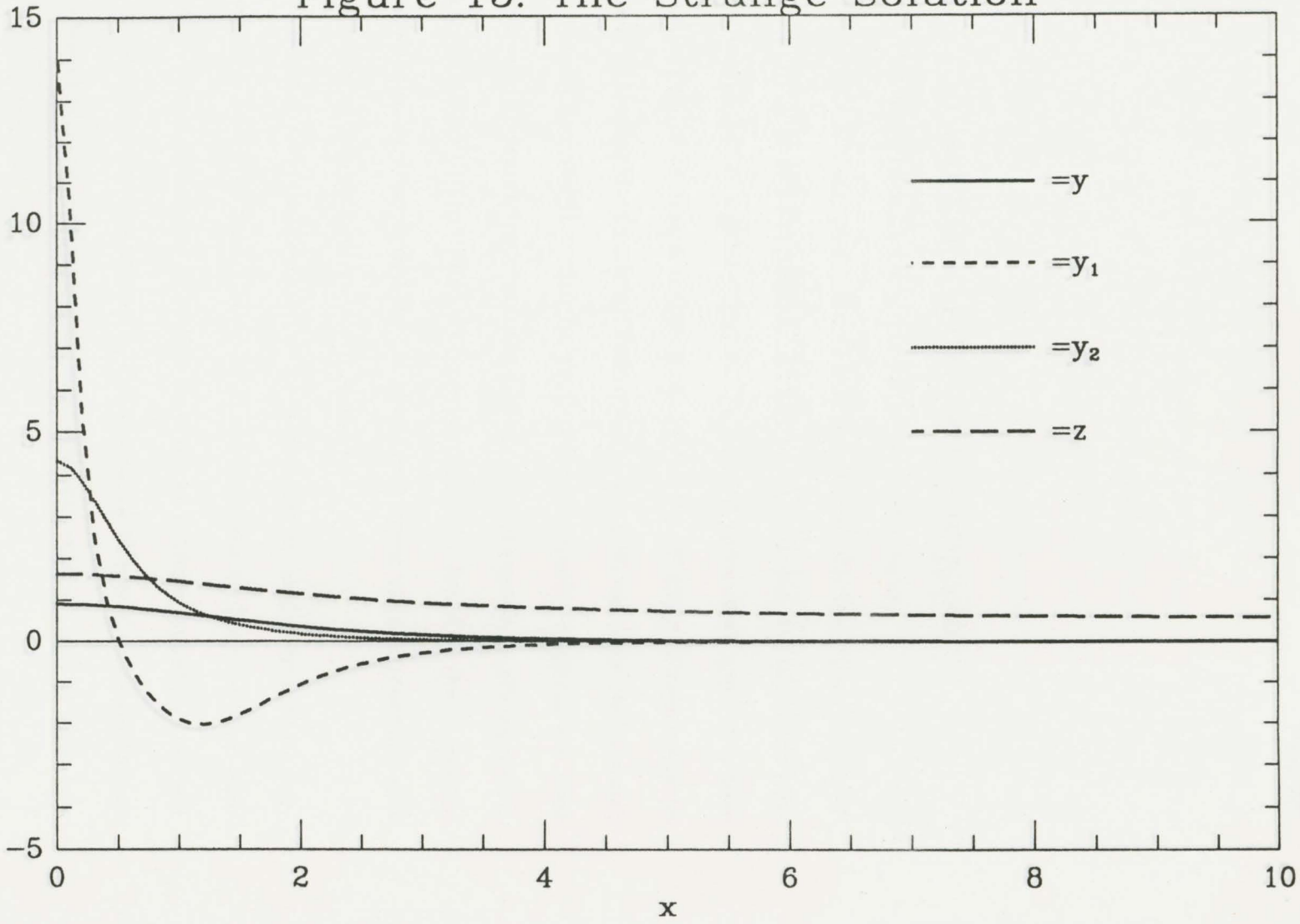
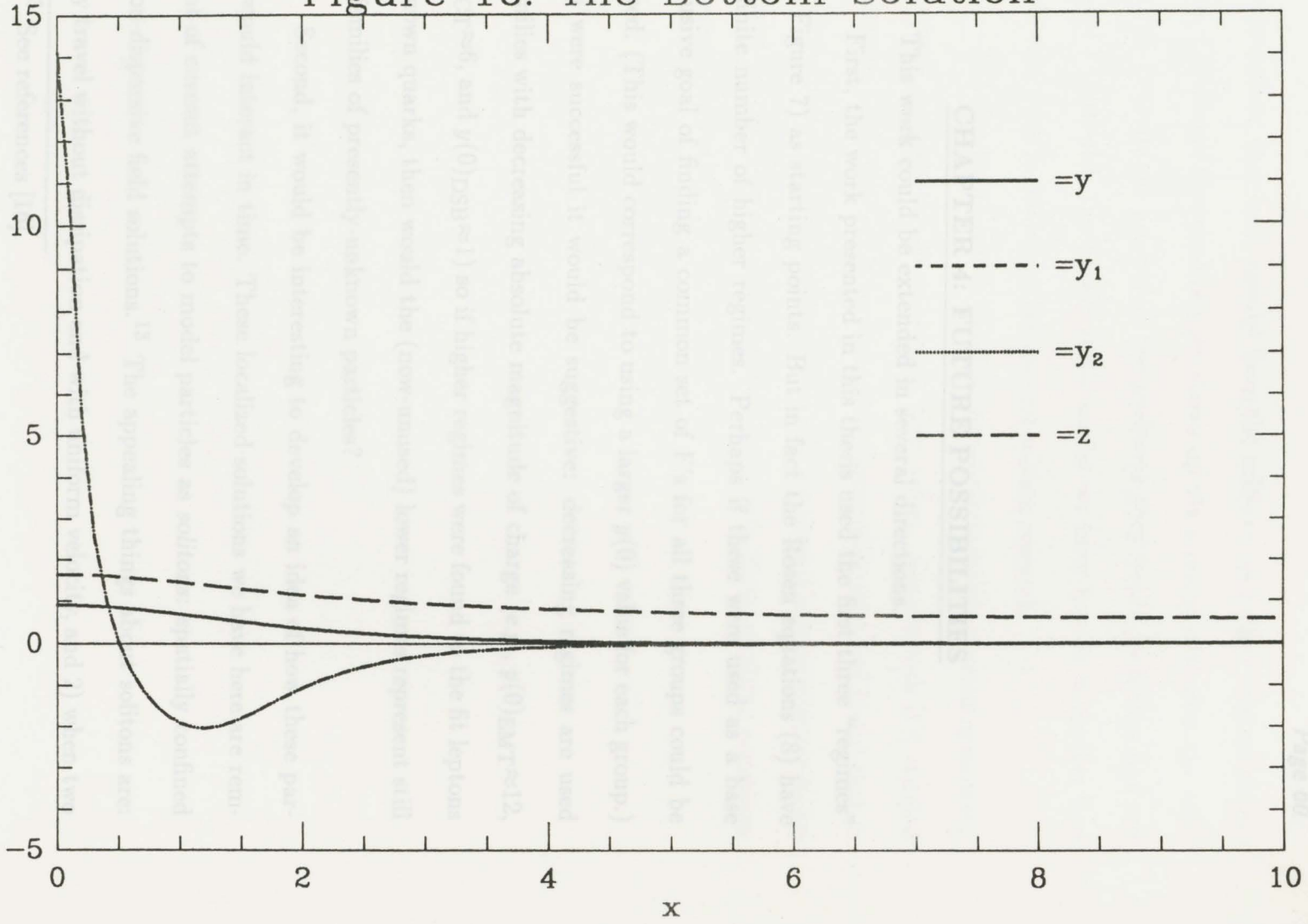


Figure 16: The Bottom Solution



CHAPTER 4: FUTURE POSSIBILITIES

This work could be extended in several directions.

First, the work presented in this thesis used the first three "regimes" (as in Figure 7) as starting points. But in fact the Rosen equations (8) have an infinite number of higher regimes. Perhaps if these were used as a base the elusive goal of finding a common set of F's for all three groups could be achieved. (This would correspond to using a larger $y(0)$ value for each group.) If this were successful it would be suggestive: decreasing regimes are used for families with decreasing absolute magnitude of charge (e.g., $y(0)_{\mathrm{EMT}} \approx 12$, $y(0)_{\mathrm{UCT}} \approx 6$, and $y(0)_{\mathrm{DSB}} \approx 1$) so if higher regimes were found to the fit leptons and known quarks, then would the (now-unused) lower regimes represent still more families of presently-unknown particles?

Second, it would be interesting to develop an idea of how these particles would interact in time. These localized solutions we have here are reminiscent of current attempts to model particles as *solitons*: spatially confined and non-dispersive field solutions.[13] The appealing things about solitons are: 1) they travel without dissipation and with uniform velocity, and 2) when two

[13] See references [16].

solitons collide, they each emerge from the collision with their original shapes intact. (Although the non-linearity shows up via a phase-shift after the collision; the two solitons are not in the positions they would be if they had not collided.) But with the purely *static* model we have here it is unclear how closely this work is related to the current soliton research.

The third approach is to relax the requirement of spherical symmetry. Some steps towards this have already been taken. Cooperstock [17] studied the simpler Lagrangian (17) for the axially-symmetric case and found that it led to a quantized magnetic moment and spin for the particle, in addition to its mass and charge. The complete numerical solutions of the resulting equations are presently unkown.

* * *

BIBLIOGRAPHY

[1] S. N. Das, "Rishon Constituents of Higher-Generation Leptons and Global Conservation Law," *International Journal of Theoretical Physics*, Volume 25, No. 3, 1986, pp. 247-253.

Michael A. Shupe, "A Composite Model of Leptons and Quarks," *Physics Letters*, Volume 86B, No. 1, 10 September 1979, pp. 87-92.

Haim Harari, "A Schematic Model of Quarks and Leptons," *Physics Letters*, Volume 86B, No. 1, 10 September 1979, pp. 83-86.

Haim Harari, "The Structure of Leptons and Quarks," *Scientific American*, Volume 248, Number 4, April 1983, pp. 56-68.

[2] D. K. Sen, *Fields and/or Particles*, (Ryerson Press, Toronto: 1968).

[3] L. Pearce Williams, *The Origins of Field Theory*, (Random House, New York: 1966).

[4] A. Einstein and N. Rosen, "The Particle Problem in the General Theory of Relativity," *Physical Review*, Volume 48, July 1, 1935.

[5] G. Mie, *Annalen der Physik*, Volume 37, 1912, p. 511.

G. Mie, *Annalen der Physik*, Volume 39, 1912, p. 1.

G. Mie, *Annalen der Physik*, Volume 40, 1913, p. 40.

[6] M. Born and L. Infeld, "Foundations of the New Field Theory," *Proceedings of the Royal Society*, Volume A144, 1934, pp. 425-451.

M. Born and L. Infeld, "On the Quantization of the New Field Equations I," *Proceedings of the Royal Society*, Volume A147, 1934, pp. 522-546.

M. Born and L. Infeld, "On the Quantization of the New Field Equations II," *Proceedings of the Royal Society*, Volume A150, 1935, pp. 141-166.

[7] N. Rosen, "A Field Theory of Elementary Particles," *Physical Review*, Volume 55, January 1, 1939, pp. 94-101.

[8] David Griffiths, *Introduction To Elementary Particles*, (Harper and Row, New York: 1987).

[9] N. Rosen and H. B. Rosenstock, "The Force Between Particles in a Nonlinear Field Theory," *Physical Review*, Volume 85, January 15, 1952, pp. 257-259.

[10] R. Finkelstein, R. LeLevier, and M. Ruderman, "Nonlinear Spinor Fields," *Physical Review*, Volume 83, No. 2, July 15, 1951, pp. 326-332.

[11] F. I. Cooperstock and N. Rosen, "A Nonlinear Gauge-Invariant Field Theory of Leptons," *International Journal of Theoretical Physics*, Volume 28, Number 4, 1989.

[12] Chris Quigg, *Gauge Theories of the Strong, Weak, and Electromagnetic Interactions*, (Benjamin/Cummings, Massachusetts: 1983).

[13] Martinus J. G. Veltman, "The Higgs Boson," *Scientific American*, Volume 252, Number 11, November 1986, pp. 76-105.

[14] "Summary Tables of Particle Properties," *Physics Letters B*, Volume 204, 14 April 1988.

[15] F. I. Cooperstock, unpublished data.

[16] T. D. Lee, *Particle Physics and Introduction to Field Theory*, (Harwood; Chur, Switzerland: 1981), pp. 117-160.

R. Rajaraman, *Solitons and Instantons*, (North-Holland, Amsterdam: 1982).

P. G. Drazin and R. S. Johnson, *Solitons: An Introduction*, (Cambridge University Press, Cambridge: 1988).

[17] F. I. Cooperstock, "Structure of Elementary Particles in a Non-Linear Field Theory," to appear in *Foundations of Physics Letters*.

[18] *CERN Program Library* manual, 1987.03.01.

[19] F. R. Ruckdeschel, *BASIC Scientific Subroutines, Volume II*, (McGraw-Hill, New York: 1981).

* * *

APPENDIX A: CONVENTIONS USED

The conventions used here are common ones:

1) Roman subscripts are taken to run 0, 1, 2, 3. Greek subscripts are taken to run 1, 2, 3. Also, the summation convention is used where repeated indices are assumed to have an implied summation.

2) $f_{,k}$ and $\partial_k f$ both denote

$$\frac{\partial f}{\partial x^k}$$

and $f_{;k}$ denotes the covariant derivative of f.

3) Gaussian electromagnetic units are used.

4) Factors of c have generally been omitted.

5) $\mathbf{T} = (x, \vec{y})$ indicates that the components of the four-vector $\mathbf{T}$ are:

$$T^0 = x, \quad T^1 = y^1$$
$$T^2 = y^2$$
$$T^3 = y^3.$$

6) A metric with a $[+\ -\ -\ -]$ signature is used.

* * *

APPENDIX B: THE DIMENSIONS OF VARIOUS SYMBOLS

Quantity:	Dimensions:
r, $\left(\frac{1}{\sigma}\right)$	length
L	energy/volume
ψ, θ, φ, μ, $\left(\frac{\sigma}{\epsilon}\right)$	$\sqrt{\text{energy/length}}$ = charge/distance
ϵ	$\frac{1}{\sqrt{\text{energy} \cdot \text{length}}} = \frac{1}{\text{charge}}$
f_1, f_2, f_3, g_1, g_2	$\frac{1}{\text{energy} \cdot \text{length}}$
x, y, z, y_1, y_2 FG1, FG2, FG3, FE1, FE2 w, α, β, γ	(dimensionless)

* * *

APPENDIX C: GEOMETRIC UNITS

It is convenient to use so-called "geometric units" where G (the gravitational constant) and c (the speed of light) are used to convert various quantities from conventional dimensions to dimensions of length. So:

---

1)

$$\text{mass}_{\text{geometric}} = \left(\frac{G}{c^2}\right) \cdot \text{mass}_{\text{conventional}}$$

therefore

$$7.428 \times 10^{-29}\text{cm} = 1\,\text{g}.$$

---

2)

$$\text{charge}_{\text{geometric}} = \left(\sqrt{\frac{G}{c^4}}\right) \cdot \text{charge}_{\text{conventional}}$$

(in Gaussian units). Therefore

$$1.381 \times 10^{-34}\text{cm} = e \quad \text{(the proton charge)}$$

---

3)

$$\text{energy}_{\text{geometric}} = \left(\frac{G}{c^2}\right) \cdot \text{energy}_{\text{conventional}}$$

therefore

$$1.324 \times 10{-}61\text{cm} = 1\,\text{eV}.$$

---

* * *

APPENDIX D: COMPUTER PROGRAM NOTES

This appendix details some of the computer programs and procedures used for this work. All the programs are written in FORTRAN and were run on a VAX minicomputer. The only non-standard FORTRAN features used are: 1) the `/extend_source` parameter on the VAX FORTRAN compiler is used to make the compiler read input from colums 7 to 132 (instead of stopping at column 72 as in some FORTRAN versions), and 2) the inline-comment feature of the VAX FORTRAN compiler is occasionally used; the rest of the line following an exclamation mark is ignored by the compiler.

I found it convenient to make the programs read, write, and transfer data-files between themselves, thereby reducing the amount of information that must be typed in by hand. Details of the data-file structure are given below, and a diagram of the data-flow is given in Figure 17. Listings of the programs are in Appendix E.

I found it useful to keep a semi-permanent set of input data files (with different files for each family) as well as a set of temporary, generic data files that are the direct input for any particular program. For example, I have three data files `FCONEMT.DAT`, `FCONUCT.DAT`, and `FCONDSB.DAT` which contain the current F's for each group. The program GETBCD needs to read this information and needs the correct file depending on which group it is currently doing. Rather than have to tell GETBCD which file to read, I have GETBCD simply read the generic file `FCONSTANTS.DAT`. It is then a simple matter to copy the desired data file (e.g., `FCONEMT.DAT`) to `FCONSTANTS.DAT` before running

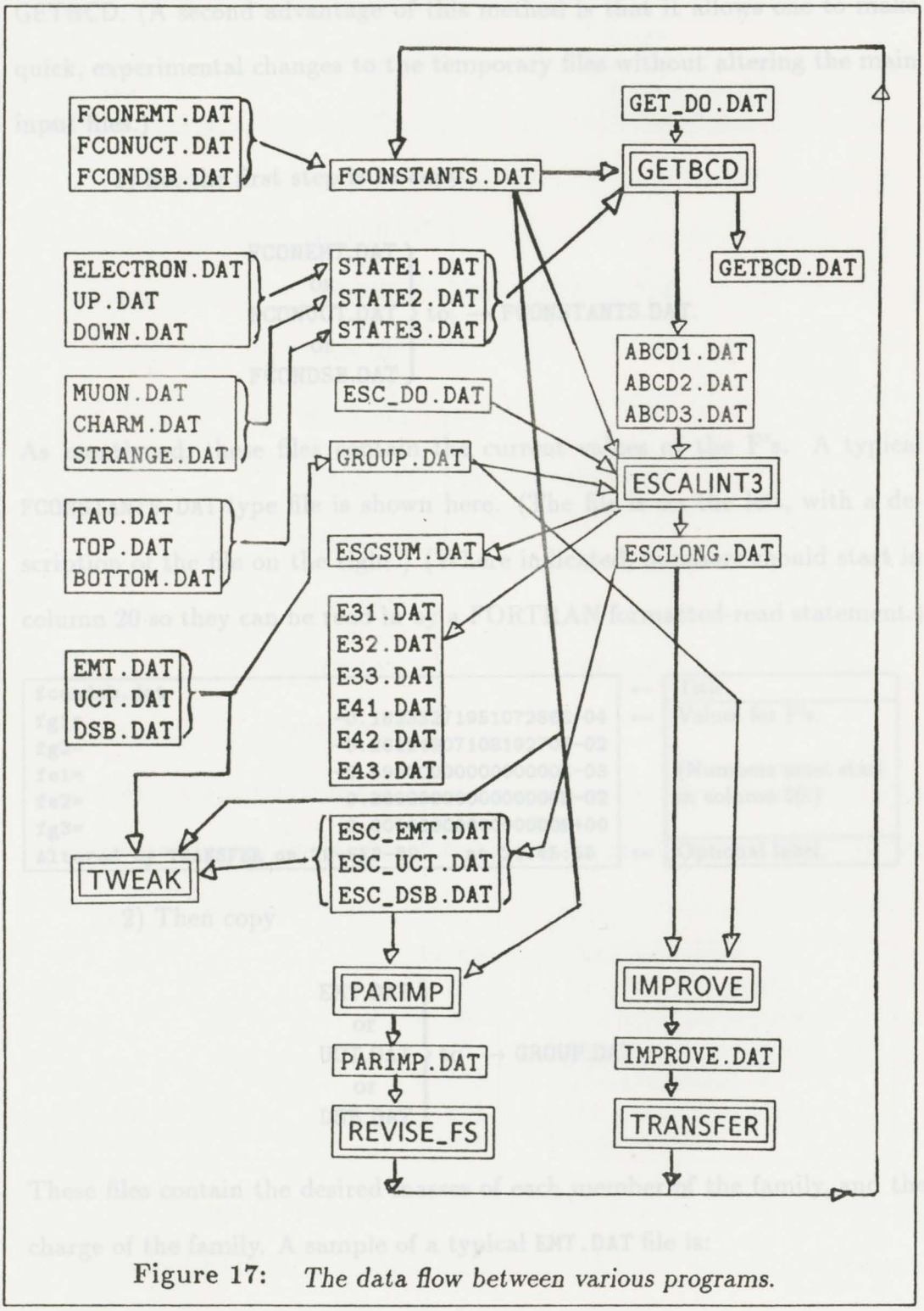


Figure 17: *The data flow between various programs.*

GETBCD. (A second advantage of this method is that it allows one to make quick, experimental changes to the temporary files without altering the main input files.)

1) So, the first step is to copy

FCONEMT.DAT
or
FCONUCT.DAT } to → FCONSTANTS.DAT.
or
FCONDSB.DAT

As mentioned, these files contain the current values of the F's. A typical FCONSTANTS.DAT-type file is shown here. (The file is on the left, with a description of the file on the right.) (Where indicated, numbers should start in column 20 so they can be read in by a FORTRAN formatted-read statement.)

fcondsb.dat	←	Title.
fg1= -0.1823327195107286E-04	←	Values for F's.
fg2= -0.2612490710819270E-02		
fe1= -0.5950000000000000E-03		(Numbers must start
fe2= -0.3000000000000000E-02		in column 20.)
fg3= 0.0000000000000000E+00		
Altered by TRANSFER on 29-SEP-89 at 21:45:55	←	Optional label.

2) Then copy

EMT.DAT
or
UCT.DAT } to → GROUP.DAT.
or
DSB.DAT

These files contain the desired masses of each member of the family, and the charge of the family. A sample of a typical EMT.DAT file is:

E/M/T data from Physics Letters Vol 204.	←	Title.
mass electron = 0.51099906	←	Masses
mass muon = 105.65839		(in MeV/c^2).
mass tau = 1784.1		(Numbers start in col. 20.)
charge = -1.0	←	Charge (in e).

3) Copy

ELECTRON.DAT
or
UP.DAT
or
DOWN.DAT
} to → STATE1.DAT

MUON.DAT
or
CHARM.DAT
or
STRANGE.DAT
} to → STATE2.DAT

TAU.DAT
or
TOP.DAT
or
BOTTOM.DAT
} to → STATE3.DAT.

These files contain the starting guesses for the initial conditions for each of the three members of the family. A sample of a typical ELECTRON.DAT file is:

EMT	←	Title.
A= 11.80	←	Starting values
B= 9.73		for $y(0)$, $z(0)$,$y_1(0)$,
C= 4.34		and $y_2(0)$ respectively.
D= 4.35		(Numbers start in col. 20.)
DELTAB= 0.001	←	The placement of
DELTAC= 0.001		the 1 indicates which
DELTAD= 0.001		digit to test.

4) Change (if desired) GET_DO.DAT. This is a file which contains instructions for GETBCD. Through it one can run GETBCD normally, or just run

it for one member of the group, and/or have GETBCD prompt for any one of the desired initial values (over-riding what is in the STATEi.DAT file). LINE_ONE controls the state being done. If LINE_ONE=0, then it runs normally (and does all three states). If LINE_ONE=1 it does only the low state, if LINE_ONE=2 it does only the middle state, and if LINE_ONE=3 it does only the high state. LINE_TWO controls the prompting. If LINE_TWO=0 it runs normally (and takes the input directly from the STATEi.DAT files). If LINE_TWO=1 it prompts the user for ASTART, if LINE_TWO=2 it prompts for BSTART, if LINE_TWO=3 it prompts for CSTART, and if LINE_TWO=4 it prompts for DSTART. A sample of a typical GET_DO.DAT file follows. (All values can start in column 1.) (The other variables, such as PREC, are defined below.)

0	←	LINE_ONE
0	←	LINE_TWO
1.0d-8	←	PREC
50	←	MAXX
400	←	1/step size

5) Run GETBCD. (This takes about 45 minutes usually, so it is best done as a batch-job.) GETBCD finds values for $y(0)$, $z(0)$, $y_1(0)$, and $y_2(0)$ that give particle-like solutions using the current F's.

It takes input from: GROUP.DAT, FCONSTANTS.DAT, and GET_DO.DAT.

As it runs it shows a rough graph on the screen of the equation's behaviour ($z/y_1/y_2$ are horizontal, x is vertical). This is useful for spotting instability problems and for generally seeing what is going on.

It works by testing the current guess of the initial value and seeing if that value causes the graph to cross the x-axis with too many nodes. If that value is acceptable it then increments the last digit and tries again. In this way it finds the next digit of each start value in turn and keeps going until the desired precision is reached. The desired precision is indicated through the variable PREC. Set PREC $= 10^{-n}$ where n is the number of desired digits.

GETBCD integrates down the x-axis until

a) $x >$MAXX (and the program assumes that that initial value is okay), or

b) the graph crosses with too many nodes (in which case the program knows that initial value is too high), or

c) the program can tell, via the deriviatives of the functions, that the graph is not going to cross again (in which case is knows that initial value is okay).

There are two flags set inside the program: DO_FORWARD and DO_BACKUPS. If DO_FORWARD is set to TRUE, GETBCD will keep incrementing the last digit until it finds it has gone too far. This is in contrast to the normal behaviour where it will abort if it has to increment the last digit more than ten times. If DO_BACKUPS is set to TRUE, GETBCD will decrement the start-value if it fails on the first iteration. (Normally it aborts if the first iteration fails.) *Both these flags should be set to* FALSE *for normal operation.* They are intended only as a last resort when strong coupling makes normal operation impossible. (If either flag is enabled there is an iteration-counter built in that will stop the program running after too many iterations, so infinite-loops will

be avoided. But strong couplings can still send the program off searching for non-existent solutions. Normally changing the last digit in one value will not alter the current last digits in the other values, but with strong couplings this is no longer true and each change can mean re-determining all the other values. The problem arises when *those* changes change the value that caused the trouble in the first place. In such instances there *are* no stable solutions and GETBCD will finally abort through the loop-counter.)

GETBCD uses the subroutine DRKSTP from the CERN Computer Centre Program Library.[14] This is a double-precision Runge-Kutta integration routine for solving the system of n ($n \geq 1$) simultaneous first-order differential equations of the form:

$$\frac{dy_i}{dx} = f_i(x, y_1, \ldots, y_n) \quad (i = 1, 2, \ldots, n).$$

It is called as

```
CALL DRKSTP(N,H,X,Y,FCN,W)
```

where the arguments are as follows:

N (Integer.) On entry, N must equal n, the number of differential equations. (It is left unchanged.)

H (Real, double-precision.) On entry, H must equal the desired step length (i.e., Δx). (It is left unchanged.)

X (Real, double-precision.) On entry, X must equal the initial value of x. On exit it equals $x + \Delta x$.

---

[14] *CERN Program Libary* manual, [18], pp. 6.106-6.107.

Y (Real, double-precision, one-dimensional array.) On entry, Y(i) must contain $y_i(x)$ $(i = 1, \ldots, n)$. On exit, Y(i) contains the approximate values of $y_i(x + \Delta x)$.

W (Real, double-precision array.) W should contain $\geq$ 3N elements. It is used as a scratch work-array.

FCN This is the name of a user-written, double-precision subroutine (declared EXTERNAL in the calling program) of the form:

```
FCN(X, Y, F)
```

The subroutine must set

$$f(i) = \frac{dy_i}{dx}(X, Y(1), \ldots, Y(N)).$$

So in our case the four second-order differential equations of the field equations (20) are broken into eight first-order differential equations:

$$u_1 \equiv y'$$

$$u_2 \equiv y''$$

$$u_3 \equiv z'$$

$$u_4 \equiv z''$$

$$u_5 \equiv y_1'$$

$$u_6 \equiv y_1''$$

$$u_7 \equiv y_2'$$

$$u_8 \equiv y_2''$$

$$u_1 = y'$$

$$u_2 = -\left(\frac{2}{x}y' + z^2y - y - (\mathrm{FG1})(\mathrm{y_1})^2\mathrm{y} - (\mathrm{FG2})(\mathrm{y_2})^2\mathrm{y}\right)$$

$$u_3 = z'$$

$$u_4 = -\left(\frac{2}{x}z' + y^2z\right)$$

$$u_5 = y_1'$$

$$u_6 = -\left(\frac{2}{x}y_1 - y_1{}' + (y_1)^3 + (\mathrm{FE1})\mathrm{y}^2\mathrm{y_1} + \left(\frac{\mathrm{FG3}}{\mathrm{FG1}}\right)(\mathrm{y_2})^2\mathrm{y_1}\right)$$

$$u_7 = y_2{}'$$

$$u_8 = -\left(\frac{2}{x}y_2 - y_2{}' + (y_2)^3 + (\mathrm{FE2})\mathrm{y}^2\mathrm{y_2} + \left(\frac{\mathrm{FG3\,FE2}}{\mathrm{FG2\,FE1}}\right)(\mathrm{y_1})^2\mathrm{y_2}\right)$$

A good step size for most operations is 1/400 or 1/800. A good MAXX value is 50. (Although 20 speeds things up and is good for rough work.)

GETBCD produces GETBCD.DAT which is a short summary of its attempts to find solutions, and ABCDi.DAT. ($i = 1, 2, 3$). The latter files record the A, B, C, and D values (i.e., $y(0)$, $z(0)$, $y_1(0)$, and $y_2(0)$) that GETBCD found – if it was sucessful. These files are used for input to other programs.

6) Run ESCALINT3. This program calculates all the various integrals and α, ω, σ, etc.

It uses ABCDi.DAT, FCONSTANTS.DAT, and GROUP.DAT as input.

It also takes instructions from ESC_DO.DAT. This controls the regime being studied and the step size, and contains a flag called RECORD.

A sample of a typical ESC_DO.DAT file is:

.false.	←	RECORD
0	←	regime
400	←	1/step size

ESCALINT3 automatically finds the point where α starts diverging and stops integrating there. But it cannot start checking for divergence until α is the correct side of the x-axis, so it uses information about the regime being studied. It turns out that if the lowest regime is labelled 'regime 0' then the nth regime has n nodes in α. (This can be seen from the behaviour of the Rosen pair of equations (8).)

RECORD is a flag which will optionally produce an additional two output files E3i.DAT and E4i.DAT. These contain values useful for plotting.

It produces ESCSUM.DAT and ESCLONG.DAT. The former is a short summary of the most-useful quantities. The latter is a more comprehensive datafile designed to be used as input for other programs.

7) Copy

$$\text{ESCLONG.DAT} \rightarrow \text{to} \begin{cases} \text{ESC\_EMT.DAT} \\ \text{or} \\ \text{ESC\_UCT.DAT} \\ \text{or} \\ \text{ESC\_DSB.DAT.} \end{cases}$$

8) Now at this point we should have a set of initial values that give particle-like solutions for the current group with the current set of F's. The next thing to do is to a) change the F's to give the desired mass-ratios, and/or b) change the $y(0)$ for each member of the family to bring the α's into line. (In practice (a) changes things more than (b) so do it seems best to first get

the F's reasonably close, then worry about the α's. In the last stages changes can be made to both at the same time.)

8a) To improve the mass-ratios either run IMPROVE, PARIMP, or TWEAK.

i) Run IMPROVE:

IMPROVE uses `FCONSTANTS.DAT`, `GROUP.DAT`, and `ESCLONG.DAT` as input.

It produces `IMPROVE.DAT` as output.

This program finds a "perfect" FG1 and FG2 pair that will give the correct mass-ratios. (At least they would be correct if nothing else changed; in practice these new F's will change everything else, hopefully slightly.)

ii) Run TRANSFER. This takes the data from `IMPROVE.DAT` and inserts the "perfect" F's into `FCONSTANTS.DAT`.

iii) Copy

$$\texttt{FCONSTANTS.DAT} \rightarrow \text{to} \begin{cases} \texttt{FCONEMT.DAT} \\ \text{or} \\ \texttt{FCONUCT.DAT} \\ \text{or} \\ \texttt{FCONUCT.DAT.} \end{cases}$$

or

i) Run PARIMP or TWEAK. They are both parameter-fitting programs that attempt to find common F's to fit two or more groups.

PARIMP is based on PARAFIT[15] and uses `ESC_EMT.DAT`, `ESC_UCT.DAT`, and `ESC_DSB.DAT` for input. It produces `PARIMP.DAT` which records the F's which give the best (although not necessarily perfect) mass-ratios.

PARAFIT should be called with the following variables set by the calling program:

L: the number of coefficients to be fitted.

A(i): ($i = 1, \ldots, L$) This array contains the initial guesses for the coefficients. Theses guesses should be over-estimated in magnitude and of the correct sign.

N: the number of data pairs.

X(i), Y(i): ($i = 1, \ldots, N$) These pairs hold the data itself.

E: a convergence criterion. The iteration is stopped when the variance is less than this fraction. (A good starting value is E=0.1.)

E1: a convergence factor. ($0 < E1 \leq 1.$)

E1=0 will stall calculation.

E1=1 will speed up the search, but may cause instability.

PARAFIT returns:

A(i): ($i = 1, \ldots, L$) These are the calculated coefficients that fit the data best.

D: the standard deviation of the fit.

---

[15] PARAFIT is from *BASIC Scientific Subroutines, Vol. II*, [19], pp. 75-82.

M: the number of iterations performed.

PARAFIT also needs access to a subroutine to evaluate the function

$$y = f(x, A(1), \ldots, A(L)).$$

TWEAK uses ESC_EMT.DAT, ESC_UCT.DAT, and ESC_DSB.DAT for input. It produces only output to the screen. TWEAK is an itelligent parameter-tweaker. It assumes there are some variables, $x(1{:}n_x)$, and some functions of these variables, $y(1{:}n_y)$ and some desired values, $d(1{:}n_y)$, of these functions. (No assumption is made as to whether the number of x's is equal to the number of functions.) It tries to tweak the variables to give the desired values, but instead of just doing each variable in turn (as PARAFIT does for example) it also assumes there is some other criterion for judging what tweakings are better than others. (This might be "don't move the variables far from where they are now", or it might be "keep the variables as small as possible", or "keep the variables as close to certain values as possible.") So the basic strategy is this: for each variable, and each desired output, work out what change would be required of each variable for each output. Then see what change has the least penalty value, do that change, then the next lowest penalty value, and so on. In this way, it is hoped, the variables will converge to some sort of stable solution (and hopefully also one with low undesirability) although it is not guaranteed that this will hapen.

ii) If PARIMP was used, use REVISE_FS to insert the newly-found F's into FCONSTANTS.DAT.

iii) Copy

$$\texttt{FCONSTANTS.DAT} \rightarrow \text{to} \begin{cases} \texttt{FCONEMT.DAT} \\ \text{or} \\ \texttt{FCONUCT.DAT} \\ \text{or} \\ \texttt{FCONUCT.DAT.} \end{cases}$$

8b) To improve the α's:

i) Run PAR_ALPHA.

This is a parameter-fitting program set up to fit a parabola through two or more (α,A) data pairs. (If you only have one data point use (0,0) for the other data point to start with.) It works reasonably well at honing in on the correct α in a dozen or fewer iterations usually.

Note: the desired α is coded into the program; change it to what is desired and re-compile PAR_ALPHA.

ii) Change the necessary file (e.g., ELECTRON.DAT) to include the new $y(0)$ value suggested by PAR_ALPHA.

9) Reiterate the process all over again. This may take some time; over 2,500 iterations were used for this work. The basic procedure is simple, but it can take months to achieve results.

* * *

APPENDIX E: COMPUTER PROGRAM LISTINGS

1) GETBCD

```
c ------------------------------------------------------------------------------
c ------------------------------------------------------------------------------
c       GETBCD
c       ======
c       Determines B,C, and D.  (Initial values of z,y1,y2.)
c       Loops three times to do the ground, middle, and highest excited states.
c       Reads:  STATEi.DAT, FCONSTANTS.DAT, GET_DO.DAT
c       Writes: ABCDi.DAT       (i=1 for ground ... i=3 for highest),
c               GETBCD.DAT.
c
c       If DO_BACKUP is enabled, it will back up the attempted solution.
c       If DO_FORWARD is enabled, it will go forward more than 10 iterations.
c       (Use only for strong coupling problems.)
c
c       It reads instructions from GET_DO.DAT:
c               1) Line 1: 0=process normally, i=do only state i. (i=1,2,3)
c               2) Line 2: 0=process normally, i=prompt for A,B,C, or D.
c                          (i=1 -> A, ..., i=4 -> D)
c               3) Line 3: prec (Controls # of digits. EG. prec=1.0d-4
c                          means do 4 digits.) (real*8)
c               4) Line 4: maxx (Controls how far down the x-axis to go.)
c                          (integer)
c
c       (All variables are declared, none are implicitly typed.)
c
c       Written:        - Oct. 1988 by Philip Sharman,
c                       (based on BVAXB3 by F.I. Cooperstock).
c       Last revised: 1988/Nov/30       -scale added for doplot
c                     Dec/5             -minor changes to scale stuff
c                   1989/Jan/17         -STOP changed to EXIT
c                     Jan/18            -prints out tiles of input STATEi files
c                     Jan/26            -show f's scaled by 10^4
c                     Feb/16            -Changed things because of extra
c                                       coupling term....
c                                       -Changed COMMON, FCN; FG3 added.
c                                       -Also added RETURN to DOPLOT.
c                     Mar/11            -Added some quit tests.
c                                       -reads GET_DO.DAT to read special
c                                       instructions.
c                                       -writing to GETBCD.DAT moved to DOIT.
c                     Mar/25            -SAVE added for instructions.
c                     May/19            -'Nnodes is still okay at' message added.
c                                       -special handling put into FCN to prevent
c                                       division by zero in FG3 term.
c                     May/28            -MAXX lowered from 30 to 20.
c                     May/29            -made it back up if it fails on first
c                                       iteration.
c                     Jun/1             -HMAX deleted.
```

```
c                       Jun/2       -DO_BACKUP and N_BACKUPS added.
c                       Jun/4       -SCHEME added.
c                       Jun/10      -ability added to go more than 10 forward.
c                                   (via NORMAL)
c                       Jun/14      -NORMAL taken out, DO_FORWARD put in.
c                                   -formatting cleaned up
c                       Jun/21      -MAXX and PREC added to GET_DO.DAT.
c                       Aug/29      -if CSTART and DSTART are zero,
c                                   then the program skips C and/or D loops.
c                                   (Useful for doing just uncoupled Rosen
c                                   equations ... remember to also set
c                                   all the F's to zero.)
c                                   -modified to also show the quantities
c                                   (like FG1/FE1 that appear in the field
c                                   equations.
c                       Aug/31      -step size stuff cleaned up.  (JK
c                                   renamed INVERSE_STEP_SIZE and moved
c                                   all to one place.)
c                       Sep/1       -the quit tests changed to only abort
c                                   if slopes are > 0.5 (instead of just >0
c                                   as before).
c                       Sep/3       -reads inverse_step_size from GET_DO.DAT.
c                       Sep/19      -bug fixed in initialization of
c                                   backup and forwarding parameters at
c                                   top of KSTEPB loop.
c                                   And missing line fixed.
c ------------------------------------------------------------------------------
c ------------------------------------------------------------------------------

c     MAIN LOOP
      real*8        a, b, c, d, zero, one, three,
     &                 fg1, fe1, fg2, fe2, fg3, ten, prec
      integer       scale, instructions_line1, instructions_line2, maxx,
     &                 inverse_step_size
      character*100 junk
      character*12  buffer, buffer2

      common    a, b, c, d, fg1, fe1, fg2, fe2, fg3
      common    /const/ zero, one, three
      common    /plots/ scale
      common    /special/ instructions_line1, instructions_line2

      zero = 0.d0
      one = 1.d0
      three = 3.d0
      ten = 10.d0

c     READ FG1 ETC
      print*, '(Reading FCONSTANTS.DAT)'
      open(unit=3, file='fconstants.dat', status='old')
      read(unit=3, fmt=91), junk
      read(unit=3, fmt=93), fg1
```

```
      read(unit=3, fmt=93), fg2
      read(unit=3, fmt=93), fe1
      read(unit=3, fmt=93), fe2
      read(unit=3, fmt=93), fg3
      close(unit=3)
c     TAKE SPECIAL INSTRUCTIONS (IF ANY) AND PREC & MAXX FROM 'GET_DO.DAT'
      print*, '(Reading GET_DO.DAT)'
      open(unit=3, file='get_do.dat', status='old')
      read(unit=3, fmt=90), instructions_line1
      read(unit=3, fmt=90), instructions_line2
      read(unit=3, fmt=*), prec
      read(unit=3, fmt=90), maxx
      read(unit=3, fmt=90), inverse_step_size
      close(unit=3)
90    format (i)

c     GROUND STATE
      call doit(1, prec, maxx, inverse_step_size)

c     MIDDLE STATE
      call doit(2, prec, maxx, inverse_step_size)

c     HIGHEST STATE
      call doit(3, prec, maxx, inverse_step_size)

c     ALL DONE
      print*, 'Program is complete.'

c ----------------formats---------------------
91    format (a)
83    format (' ', a, t20, e30.16)
93    format (t20, e30.16)
c ----------------------------------------------
      end


c ---------------------------------------------------------------
c     DOIT - does the particular case
      subroutine doit(case, prec, maxx, inverse_step_size)
      real*8  x, h, y(8), w(99),
     &             a,b,c,d,
     &      astart, bstart, cstart, dstart, deltab, deltac,
     &      deltad, prec, fg1, fg2, fe1, fe2, fg3,
     &             zero, one, three, ten
      logical ok,okay,hunt,tryx,lastpos(3),lastneg(3), backed_up,
     &             do_backup, do_forward, did_more_than_ten
      integer nnodes(3), nstate(3), maxx, case,
     &             kstepb, istepc, jstepd, inverse_step_size, ki,
     &             instructions_line1, instructions_line2,
     &             scale, n_backups, scheme, max_forward
```

```
      character*100   junk, title
      character*12    buffer,buffer2
      common    /special/ instructions_line1, instructions_line2

c     TRYX determines whether to keep searching down the x-axis.
c     OKAY determines whether the axis has been crossed too many times.
c     NNODES(i) records the number of graph crossings for y(1),
c     y(5), and y(7) [y,y1,& y2].
c     NSTATE(i) records the maximum allowable number of graph crossings.

      common    a,b,c,d,fg1,fe1,fg2,fe2, fg3
      common    /const/ zero,one,three
      common    /plots/ scale

      external  fcn

      ten=10.0d0

c     (DO_BACKUP LETS IT BACK UP IF IT FAILS ON FIRST ITERATION)
      do_backup=   .false.
      if (do_backup) print*, 'Ability to do backups is enabled.'
c     (DO FORWARD LETS IT GO FORWARD MORE THAN 10 ITERATIONS)
      do_forward=  .false.
      if (do_forward) print*, 'Ability to go more than 10 forward is enabled.'
      if (do_forward) then
             max_forward= 100
      else
             max_forward= 10
      endif

      print*, 'case=', case
      if (case.ne.1 .and. case.ne.2 .and. case.ne.3) call exit(0)

c     TAKE SPECIAL INSTRUCTIONS (IF ANY) FROM 'GET_DO.DAT'
      if (instructions_line1.eq.1 .and. case.ne.1) return
      if (instructions_line1.eq.2 .and. case.ne.2) return
      if (instructions_line1.eq.3 .and. case.ne.3) return
c     USE ZERO TO MEAN PROCESS NORMALLY
      if (    instructions_line1.ne.1 .and.
     &            instructions_line1.ne.2 .and.
     &            instructions_line1.ne.3) instructions_line1=0
      if (    instructions_line2.ne.1 .and.
     &            instructions_line2.ne.2 .and.
     &            instructions_line2.ne.3 .and.
     &            instructions_line2.ne.4) instructions_line2=0


c     READ STARTING VALUES
      print*, '(Reading STATEi.DAT)'
      if (case.eq.1)
     &          open(unit=3, file='state1.dat', status='old')
      if (case.eq.2)
```

```
     &              open(unit=3, file='state2.dat', status='old')
        if (case.eq.3)
     &              open(unit=3, file='state3.dat', status='old')
        read(unit=3, fmt=91), title
        print*, 'Title: ', title
        read(unit=3, fmt=93), astart
        read(unit=3, fmt=93), bstart
        read(unit=3, fmt=93), cstart
        read(unit=3, fmt=93), dstart
        read(unit=3, fmt=93), deltab
        read(unit=3, fmt=93), deltac
        read(unit=3, fmt=93), deltad
        close(unit=3)

c       SPECIAL INSTRUCTIONS MAYBE
        if (instructions_line1.ne.0) then
                if (instructions_line2.eq.1) then
                        print*,'astart?'
                        read*,astart
                        print*,'astart=',astart
                elseif (instructions_line2.eq.2) then
                        print*,'bstart?'
                        read*,bstart
                        print*,'bstart=',bstart
                elseif (instructions_line2.eq.3) then
                        print*,'cstart?'
                        read*,cstart
                        print*,'cstart=',cstart
                elseif (instructions_line2.eq.4) then
                        print*,'dstart?'
                        read*,dstart
                        print*,'dstart=',dstart
                endif
        endif

c     PARAMETERS
ccc     maxx = 20
        print*, 'maxx= ', maxx
cccc    prec = 1.d-8
        print*, 'prec= ', prec
        scheme= 2
c       (SCHEME 0= 000,001, 002)
c       (SCHEME 1= 000,010, 001  ... ROSEN'S)
c       (SCHEME 2= 000,010, 011  ... NORMAL, FRED'S)
        print*, 'scheme= ', scheme
        if (case.eq.1) then
                nstate(1)=0
                nstate(2)=0
                nstate(3)=0
        else if (case.eq.2) then
                if (scheme.eq.0) then
                        nstate(1)=0
```

```
                nstate(2)=0
                nstate(3)=1
                print*, 'DOING #0 EXCITATION SCHEME.'
        else if (scheme.eq.1) then
                nstate(1)=0
                nstate(2)=1
                nstate(3)=0
                print*, 'DOING #1 (ROSEN''S) EXCITATION SCHEME.'
        else
                nstate(1)=0
                nstate(2)=1
                nstate(3)=0
                print*, 'DOING NORMAL EXCITATION SCHEME.'
        endif
    else if (case.eq.3) then
        if (scheme.eq.0) then
                nstate(1)=0
                nstate(2)=0
                nstate(3)=2
                print*, 'DOING #0 EXCITATION SCHEME.'
        else if (scheme.eq.1) then
                nstate(1)=0
                nstate(2)=0
                nstate(3)=1
                print*, 'DOING #1 (ROSEN''S) EXCITATION SCHEME.'
        else
                nstate(1)=0
                nstate(2)=1
                nstate(3)=1
                print*, 'DOING NORMAL EXCITATION SCHEME.'
        endif
    endif
c   (NSTATE(I) IS THE NUMBER OF NODES ALLOWED. 0 FOR GROUND STATE,
c   1 FOR NEXT, ETC.  I=1 FOR Y, I=2 FOR Y1, I=3 FOR Y2)


    call date(buffer)
    call time(buffer2)
    print*,'GETBCD running on ', buffer, ' at ', buffer2

    a = astart
    b = bstart
    c = cstart
    d = dstart

    if (        (instructions_line1.eq.1).and.(case.eq.1)
   &            .or. (instructions_line1.eq.2).and.(case.eq.2)
   &            .or. (instructions_line1.eq.3).and.(case.eq.3)
   &            .or. (instructions_line1.eq.0).and.(case.eq.1))
   &           open(unit=4,file='getbcd.dat',status='new')
    print*, '(Writing GETBCD.DAT)'
```

```
      print*,'fg1=',fg1, ' = ', (fg1*1.0d4), ' x10^-4'
      print*,'fg2=',fg2, ' = ', (fg2*1.0d4), ' x10^-4'
      print*,'fe1=',fe1, ' = ', (fe1*1.0d4), ' x10^-4'
      print*,'fe2=',fe2, ' = ', (fe2*1.0d4), ' x10^-4'
      print*,'fg3=',fg3, ' = ', (fg3*1.0d4), ' x10^-4'
      if (fe1.ne.zero .and. fe2.ne.zero .and. fg1.ne.zero .and. fg2.ne.zero) then
            print*, ' Field equation quantities:'
            print*,'fg1/fe1 = ', fg1/fe1
            print*,'fg2/fe2 = ', fg2/fe2
            print*,'fe1 = ', fe1
            print*,'fe2 = ', fe2
            print*,'fg3/fg1 = ', fg3/fg1
            print*,'(fg3*fe2)/(fg2*fe1) = ', (fg3*fe2)/(fg2*fe1)
            print*,' '
      endif
      print*,'astart=',astart
      print*,'bstart=',bstart
      print*,'cstart=',cstart
      print*,'dstart=',dstart

      if (case.eq.1)
     &      write(unit=4,fmt=*) 'Results of GETBCD run on ',
     &           buffer, ' at ', buffer2

      write(unit=4,fmt=*) ' '
      write(unit=4,fmt=*) 'case=', case
      write(unit=4,fmt=*) 'fg1=', fg1, ' = ', (fg1*1.0d4), ' x10^-4'
      write(unit=4,fmt=*) 'fg2=', fg2, ' = ', (fg2*1.0d4), ' x10^-4'
      write(unit=4,fmt=*) 'fe1=', fe1, ' = ', (fe1*1.0d4), ' x10^-4'
      write(unit=4,fmt=*) 'fe2=', fe2, ' = ', (fe2*1.0d4), ' x10^-4'
      write(unit=4,fmt=*) 'fg3=', fg3, ' = ', (fg3*1.0d4), ' x10^-4'
      write(unit=4,fmt=*) 'astart=', astart
      write(unit=4,fmt=*) 'bstart=', bstart
      write(unit=4,fmt=*) 'cstart=', cstart
      write(unit=4,fmt=*) 'dstart=', dstart



c     MAIN PART
c     =========
      hunt = .true.
      print*, 'Looking for ',nstate(1),nstate(2),nstate(3),
     &        ' type solution.'
      write(unit=4,fmt=*) 'Looking for ',nstate(1),nstate(2),
     &        nstate(3),' type solution.'

      h = one/inverse_step_size
      print*, 'Step size = 1/', inverse_step_size, ' = ', h

c     SKIP AHEAD IF OBEYING SPECIAL INSTRUCTIONS
      if (instructions_line2.eq.4) goto 774        !(GOTO D LOOP)
      if (instructions_line2.eq.3) goto 773        !(GOTO C LOOP)
```

```
c       KEEP LOOKING FOR B, C, & D UNTIL THE DESIRED PRECISION ("PREC") IS REACHED
        do 154 while(hunt)      !(THIS HUGE OUTER LOOP NOT INDENTED)

c -------------------------KSTEPB LOOP-----------------------------------
        backed_up= .false.
        did_more_than_ten= .false.
        n_backups= 0

270     continue        !(THIS IS A TARGET LINE FOR A LATER GOTO)
        if (do_backup .or. do_forward) then
                print*, 'After line 270 (top of KSTEPB loop):'
                print*, ' backed_up = ', backed_up
                print*, ' did_more_than_ten = ', did_more_than_ten
                print*, ' n_backups = ', n_backups
        endif

        scale= int(abs(a)+1)
        okay=.true.
        kstepb=0
        do 152 while(okay .and. kstepb.le.max_forward)
                lastpos(1)=.true.
                lastpos(2)=.true.
                lastpos(3)=.true.
                lastneg(1)=.false.
                lastneg(2)=.false.
                lastneg(3)=.false.
c               (LASTPOS & LASTNEG RECORD WHAT SIDE OF THE AXIS THE GRAPH WAS LAST ON)
                nnodes(1)=0
                nnodes(2)=0
                nnodes(3)=0
c               (NNODES RECORDS NUMBER OF NODES FOUND)
                print*, 'Now inside the KSTEPB loop.  KSTEPB=', kstepb
                b = bstart + deltab*kstepb
                print 044, a
                print 144, b, deltab
                print 244, c, deltac
                print 344, d, deltad
                x = zero
                y(1) = a
                y(2) = zero
                y(3) = b
                y(4) = zero
                y(5) = c
                y(6) = zero
                y(7) = d
                y(8) = zero
                ki=1
                tryx = .true.
                do 151 while(okay .and. tryx .and. x.le.maxx)
                        call drkstp(8, h, x, y, fcn, w)
                        if (mod(ki, 2*inverse_step_size).eq.1) then
```

```
                call drkstp(8, h, x, y, fcn, w)
                if (mod(ki, 2*inverse_step_size).eq.1) then
                        call doplot(y(1))
                endif
c               HAVE WE CROSED THE AXIS?
                if ((lastpos(1) .and. y(1).lt.0) .or.
     &                  (lastpos(2) .and. y(5).lt.0) .or.
     &                  (lastpos(3) .and. y(7).lt.0)) then
c                       YES WE'VE CROSSED IT FROM + TO -
                        call doplot(y(1))
                        if (lastpos(1) .and. y(1).lt.0) then
                                print*,'y(1) gone pos to neg. x,y(1)=',x,y(1)
                                nnodes(1)=nnodes(1)+1
                                lastpos(1)=.false.
                                lastneg(1)=.true.
                        endif
                        if (lastpos(2) .and. y(5).lt.0) then
                                print*,'y(5) gone pos to neg. x,y(5)=',x,y(5)
                                nnodes(2)=nnodes(2)+1
                                lastpos(2)=.false.
                                lastneg(2)=.true.
                        endif
                        if (lastpos(3) .and. y(7).lt.0) then
                                print*,'y(7) gone pos to neg. x,y(7)=',x,y(7)
                                nnodes(3)=nnodes(3)+1
                                lastpos(3)=.false.
                                lastneg(3)=.true.
                        endif
                        if (nnodes(1).gt.nstate(1)) then
                                print*, 'TOO BIG: nnodes(1),nstate(1)=',
     &                                  nnodes(1),nstate(1)
                                okay=.false.
                        endif
                        if (nnodes(2).gt.nstate(2)) then
                                print*, 'TOO BIG: nnodes(2),nstate(2)=',
     &                                  nnodes(2),nstate(2)
                                okay=.false.
                        endif
                        if (nnodes(3).gt.nstate(3)) then
                                print*, 'TOO BIG: nnodes(3),nstate(3)=',
     &                                  nnodes(3),nstate(3)
                                okay=.false.
                        endif
                else if ((lastneg(1) .and. y(1).gt.0) .or.
     &                  (lastneg(2) .and. y(5).gt.0) .or.
     &                  (lastneg(3) .and. y(7).gt.0)) then
c                       YES WE'VE CROSSED IT FROM - TO +
                        call doplot(y(1))
                        if (lastneg(1) .and. y(1).gt.0) then
                                print*,'y(1) gone neg to pos. x,y(1)=',x,y(1)
                                nnodes(1)=nnodes(1)+1
                                lastpos(1)=.true.
```

```
                        if (lastneg(2) .and. y(5).gt.0) then
                                print*,'y(5) gone neg to pos. x,y(5)=',x,y(5)
                                nnodes(2)=nnodes(2)+1
                                lastpos(2)=.true.
                                lastneg(2)=.false.
                        endif
                        if (lastneg(3) .and. y(7).gt.0) then
                                print*,'y(7) gone neg to pos. x,y(7)=',x,y(7)
                                nnodes(3)=nnodes(3)+1
                                lastpos(3)=.true.
                                lastneg(3)=.false.
                        endif
                        if (nnodes(1).gt.nstate(1)) then
                                print*, 'TOO BIG: nnodes(1),nstate(1)=',
     &                               nnodes(1),nstate(1)
                                okay=.false.
                        endif
                        if (nnodes(2).gt.nstate(2)) then
                                print*, 'TOO BIG: nnodes(2),nstate(2)=',
     &                               nnodes(2),nstate(2)
                                okay=.false.
                        endif
                        if (nnodes(3).gt.nstate(3)) then
                                print*, 'TOO BIG: nnodes(3),nstate(3)=',
     &                               nnodes(3),nstate(3)
                                okay=.false.
                         endif
                    endif
c                   QUIT TEST -- TEST Y(1)=Y (EVEN THOUGH WE ARE STEPPING Z?)
                    if (    nstate(1).eq.0 .and. x.gt.3.0 .and.
     &                  y(2).gt.0.5) then
                         call doplot(y(1))
                         print *, 'Quitting: slope of y(1)=y is >0.5'
                         print 1000, '(x= ', x, ')'
                         tryx=.false.
1000                     format (a, f7.2)
                    endif
                    if (    nstate(1).eq.1 .and.
     &                  y(1).lt.(-1*a) ) then
                         call doplot(y(1))
                         print*, 'Quitting: y(1)=y is less than -A.'
                         print 1000, '(x= ', x, ')'
                         tryx=.false.
                    endif
                    if (    nstate(1).eq.2 .and.
     &                        y(1).gt.a) then
                         call doplot(y(1))
                         print*, 'Quitting: y(1)=y is greater than A.'
                         print 1000, '(x= ', x, ')'
                         tryx=.false.
                    endif
```

```
                ki=ki+1
151             continue
                if (okay) then
                        kstepb=kstepb+1
                        if (tryx) print*, 'Number of nodes is still okay at x= ', x
                endif
152     continue
c ------------------------

c       CHECK WHAT WE HAVE
        if (kstepb.eq.0) then
c               IT BOMBED OUT ON THE FIRST TRY - ABORT OR BACKUP
                if (do_backup) then
                        print*, 'WARNING: Graph failed on first iteration.'
c                       BACKUP
                        backed_up= .true.
                        n_backups= n_backups + 1
                        if (n_backups.gt.100 .or. b.lt.zero) then
                                print*, 'FAILURE: backups aborted.'
                                call exit(0)
                        endif
c                       LOWER THE LAST DIGIT OF BSTART
                        bstart= bstart- deltab
                        b= bstart
                        print*, 'Re-doing B loop with BSTART= ', bstart
                        print*, ' backed_up = ', backed_up
                        print*, ' did_more_than_ten = ', did_more_than_ten
                        print*, ' n_backups = ', n_backups
                        print*, ' '
                        goto 270                !(GO TO KSTEPB LOOP AGAIN)
                else
c                       ABORT
                        print*, 'FAILURE: Graph failed on first iteration. (B loop)'
                        print 044, a
                        print 144, b, deltab
                        print 244, c, deltac
                        print 344, d, deltad
                        call exit(0)
                endif
        else if (kstepb.ge.max_forward+1) then
c               WE FAILED TO FIND A GOOD VALUE - ABORT
                print*, 'FAILURE: Graph did not cross axis after ',
     &                  max_forward, ' iterations.  (B loop)'
                print 044, a
                print 144, b, deltab
                print 244, c, deltac
                print 344, d, deltad
                call exit(0)
        else
c               WE ARE ON THE RIGHT TRACK.  NARROW DOWN THE PRECISION.
                if (kstepb.ge.11) then
                        did_more_than_ten= .true.
```

```
                print*, '(WARNING: Graph took more than 10 iterations to cross.)'
            endif
            print*, 'Look for next digit of B.'
            bstart=bstart + (kstepb-1)*deltab
            deltab = deltab / ten
            b = bstart
            kstepb=0
        endif

c ------------------------ISTEPC LOOP-----------------------------------
773     continue        !(TARGET FOR A POSSIBLE GOTO)
        if (do_backup .or. do_forward) then
            print*, 'After line 773 (top of ISTEPC loop):'
            print*, ' backed_up = ', backed_up
            print*, ' did_more_than_ten = ', did_more_than_ten
            print*, ' n_backups = ', n_backups
        endif

        scale= int(abs(c)+1)
        print*,' '
        okay = .true.
        istepc=0
        if (cstart.eq.zero) then
            print*, '(Skipping the C loop as CSTART=0.)'
            goto 774        ! (GOTO JSTEP D LOOP)
        endif
        do 252 while(okay .and. istepc.le.max_forward)
            lastpos(2)=.true.
            lastneg(2)=.false.
            nnodes(2)=0
            print*, 'Now inside the ISTEPC loop.  ISTEPC=', istepc
            c = cstart + deltac*istepc
            print 044, a
            print 144, b, deltab
            print 244, c, deltac
            print 344, d, deltad
            x = zero
            y(1) = a
            y(2) = zero
            y(3) = b
            y(4) = zero
            y(5) = c
            y(6) = zero
            y(7) = d
            y(8) = zero
            ki=1
            tryx = .true.
            do 251 while(okay .and. tryx .and. x.le.maxx)
                call drkstp(8, h, x, y, fcn, w)
                if (mod(ki, 2*inverse_step_size).eq.1) then
                    call doplot(y(5))
                endif
```

```
c                       HAVE WE CROSED THE AXIS?
                        if (lastpos(2) .and. y(5).lt.0) then
c                               YES WE'VE CROSSED IT FROM + TO -
                                print*,'gone pos to neg. x,y(5)=',x,y(5)
                                call doplot(y(5))
                                nnodes(2)=nnodes(2)+1
                                lastpos(2)=.false.
                                lastneg(2)=.true.
                                if (nnodes(2).gt.nstate(2)) then
                                     print*, 'TOO BIG: nnodes(2),nstate(2)=',
     &                                    nnodes(2),nstate(2)
                                     okay=.false.
                                endif
                        else if (lastneg(2) .and. y(5).gt.0) then
c                               YES WE'VE CROSSED IT FROM - TO +
                                print*,'gone neg to pos. x,y(5)=',x,y(5)
                                call doplot(y(5))
                                nnodes(2)=nnodes(2)+1
                                lastpos(2)=.true.
                                lastneg(2)=.false.
                                if (nnodes(2).gt.nstate(2)) then
                                     print*, 'TOO BIG: nnodes(2),nstate(2)=',
     &                                    nnodes(2),nstate(2)
                                     okay=.false.
                                endif
                        endif
c                       QUIT TEST -- TEST Y(5)=y1
                        if (    nstate(2).eq.0 .and. x.gt.3.0 .and.
     &                      y(6).gt.0.5) then
                                call doplot(y(5))
                                print*, 'Quitting: slope of y(5)=y1 is >0.5'
                                print 1000, '(x= ', x, ')'
                                tryx=.false.
                        endif
                        if (    nstate(2).eq.1 .and.
     &                      y(5).lt.(-1*c) ) then
                                call doplot(y(5))
                                print*, 'Quitting: y(5)=y1 is less than -C.'
                                print 1000, '(x= ', x, ')'
                                tryx=.false.
                        endif
                        if (    nstate(2).eq.2 .and.
     &                      y(5).gt.c) then
                                call doplot(y(5))
                                print*, 'Quitting: y(5)=y1 is greater than C.'
                                print 1000, '(x= ', x, ')'
                                tryx=.false.
                        endif

                        ki=ki+1
251             continue
                if (okay) then
```

```
                istepc=istepc+1
                if (tryx) print*, 'Number of nodes is still okay at x= ', x
            endif
252     continue
c ---------------------

c       CHECK WHAT WE HAVE
        if (istepc.eq.0) then
c           IT BOMBED OUT ON THE FIRST TRY - ABORT OR BACKUP
            if (do_backup) then
                print*, 'WARNING: Graph failed on first iteration.'
c               BACKUP
                backed_up= .true.
                n_backups= n_backups + 1
                if (n_backups.gt.100 .or. c.lt.zero) then
                    print*, 'FAILURE: backups aborted'
                    call exit(0)
                endif
c               LOWER THE LAST DIGIT OF CSTART
                cstart= cstart- deltac
                c= cstart
                print*, 'Re-doing C loop with CSTART= ', cstart
                print*, ' backed_up = ', backed_up
                print*, ' did_more_than_ten = ', did_more_than_ten
                print*, ' n_backups = ', n_backups
                print*, ' '
                goto 773        !(GOTO C LOOP)
            else
c               ABORT
                print*, 'FAILURE: Graph failed on first iteration. (C loop)'
                print 044, a
                print 144, b, deltab
                print 244, c, deltac
                print 344, d, deltad
                call exit(0)
            endif
        else if (istepc.ge.max_forward+1) then
c           WE FAILED TO FIND A GOOD VALUE - ABORT
            print*, 'FAILURE: Graph did not cross axis after ',
     &          max_forward, ' iterations.  (C loop)'
            print 044, a
            print 144, b, deltab
            print 244, c, deltac
            print 344, d, deltad
            call exit(0)
        else
c           WE ARE ON THE RIGHT TRACK.  NARROW DOWN THE PRECISION.
            if (istepc.ge.11) then
                did_more_than_ten= .true.
                print*, '(WARNING: graph took more than 10 iterations to cross.)'
            endif
            print*, 'Look for next digit of C.'
```

```
                cstart=cstart + (istepc-1)*deltac
                deltac = deltac / ten
                c = cstart
                istepc=0
        endif

c ------------------------JSTEPD LOOP-----------------------------------
774     continue        !(TARGET FOR A POSSIBLE GOTO)
        if (do_backup .or. do_forward) then
                print*, 'After line 774 (top of JSTEPD loop):'
                print*, ' backed_up = ', backed_up
                print*, ' did_more_than_ten = ', did_more_than_ten
                print*, ' n_backups = ', n_backups
        endif

        scale= int(abs(d)+1)
        print*,' '
        okay = .true.
        jstepd=0
        if (dstart.eq.zero) then
                print*, '(Skipping the D loop as DSTART=0.)'
                goto 850        ! (GOTO END OF JSTEPD LOOP)
        endif
        do 352 while(okay .and. jstepd.le.max_forward)
                lastpos(3)=.true.
                lastneg(3)=.false.
                nnodes(3)=0
                print*, 'Now inside the JSTEPD loop.  JSTEPD=', jstepd
                d = dstart + deltad * jstepd
                print 044, a
                print 144, b, deltab
                print 244, c, deltac
                print 344, d, deltad
                x = zero
                y(1) = a
                y(2) = zero
                y(3) = b
                y(4) = zero
                y(5) = c
                y(6) = zero
                y(7) = d
                y(8) = zero

                ki=1
                tryx = .true.
                do 351 while(okay .and. tryx .and. x.le.maxx)
                        call drkstp(8, h, x, y, fcn, w)
                        if (mod(ki, 2*inverse_step_size).eq.1) then
                                call doplot(y(7))
                        endif
c                       HAVE WE CROSED THE AXIS?
                        if (lastpos(3) .and. y(7).lt.0) then
```

```
c                       YES WE'VE CROSSED IT FROM + TO -
                        print*,'gone pos to neg. x,y(7)=',x,y(7)
                        call doplot(y(7))
                        nnodes(3)=nnodes(3)+1
                        lastpos(3)=.false.
                        lastneg(3)=.true.
                        if (nnodes(3).gt.nstate(3)) then
                              print*, 'TOO BIG: nnodes(3),nstate(3)=',
     &                              nnodes(3),nstate(3)
                              okay=.false.
                        endif
                     else if (lastneg(3) .and. y(7).gt.0) then
c                       YES WE'VE CROSSED IT FROM - TO +
                        print*,'gone neg to pos. x,y(7)=',x,y(7)
                        call doplot(y(7))
                        nnodes(3)=nnodes(3)+1
                        lastpos(3)=.true.
                        lastneg(3)=.false.
                        if (nnodes(3).gt.nstate(3)) then
                              print*, 'TOO BIG: nnodes(3),nstate(3)=',
     &                              nnodes(3),nstate(3)
                              okay=.false.
                        endif
                     endif
c                    QUIT TEST -- TEST Y(7)=y2
                     if (    nstate(3).eq.0 .and. x.gt.3.0 .and.
     &                    y(8).gt.0.5) then
                        call doplot(y(7))
                        print*, 'Quitting: slope of y(7)=y2 is >0.5'
                        print 1000, '(x= ', x, ')'
                        tryx=.false.
                     endif
                     if (    nstate(3).eq.1 .and.
     &                   y(7).lt.(-1*d) ) then
                        call doplot(y(7))
                        print*, 'Quitting: y(7)=y2 is less than -D.'
                        print 1000, '(x= ', x, ')'
                        tryx=.false.
                     endif
                     if (    nstate(3).eq.2 .and.
     &                    y(7).gt.d) then
                        call doplot(y(7))
                        print*, 'Quitting: y(7)=y2 is greater than D.'
                        print 1000, '(x= ', x, ')'
                        tryx=.false.
                     endif
                     ki=ki+1
351               continue
                  if (okay) then
                     jstepd=jstepd+1
                     if (tryx) print*, 'Number of nodes is still okay at x= ', x
```

```
                endif
352     continue
c -----------------

c       CHECK WHAT WE HAVE
        if (jstepd.eq.0) then
c               IT BOMBED OUT ON THE FIRST TRY - ABORT OR BACKUP
                if (do_backup) then
                        print*, 'WARNING: Graph failed on first iteration.'
c                       BACKUP
                        backed_up= .true.
                        n_backups= n_backups + 1
                        if (n_backups.gt.100 .or. d.lt.zero) then
                                print*, 'FAILURE: backups aborted.'
                                call exit(0)
                        endif
c                       LOWER THE LAST DIGIT OF DSTART
                        dstart= dstart- deltad
                        d= dstart
                        print*, 'Re-doing D loop with DSTART= ', dstart
                        print*, ' backed_up = ', backed_up
                        print*, ' did_more_than_ten = ', did_more_than_ten
                        print*, ' n_backups = ', n_backups
                        print*, ' '
                        goto 774        !(GOTO D LOOP)
                else
c                       ABORT
                        print*, 'FAILURE: Graph failed on first iteration. (D loop)'
                        print 044, a
                        print 144, b, deltab
                        print 244, c, deltac
                        print 344, d, deltad
                        call exit(0)
                endif
        else if (jstepd.eq.max_forward+1) then
c               WE FAILED TO FIND A GOOD VALUE - ABORT
                print*, 'FAILURE: Graph did not cross axis after',
     &                  max_forward, ' iterations.  (D loop)'
                print 044, a
                print 144, b, deltab
                print 244, c, deltac
                print 344, d, deltad
                call exit(0)
        endif

        if (jstepd.ge.11) then
                print*, '(WARNING: graph took more than 10 iterations to cross.)'
                did_more_than_ten= .true.
        endif

c       DO WE HAVE TO LOOP AGAIN, OR ARE WE FINALLY DONE?
        if (backed_up .or. did_more_than_ten) then
```

```
c               LOOP AGAIN
                if (backed_up) print*, 'WARNING: We backed up somewhere.'
                if (did_more_than_ten) print*, 'WARNING: We did more than 10 itera-
tions somewhere.'
                print*, 'Increasing DELTAB and DELTAC and going back to B loop again.'
                deltab= deltab*10
                deltac= deltac*10
                print*, 'Now re-setting flags and counter.'
                backed_up = .false.
                did_more_than_ten = .false.
                n_backups = 0
                print*, ' backed_up = ', backed_up
                print*, ' did_more_than_ten = ', did_more_than_ten
                print*, ' n_backups = ', n_backups
                print*, ' '
                goto 270                    !(GO TO STEPB LOOP AGAIN)
        else
c               WE ARE ON THE RIGHT TRACK.  NARROW DOWN THE PRECISION.
                print*, '(We are okay.  No backups or >10 iterations.)'
                print*, 'Look for next digit of d.'
                dstart=dstart + (jstepd-1)*deltad
                deltad = deltad / ten
                d = dstart
                jstepd=0
        endif

c -------------------------------------------------------------------------
c -------------------------------------------------------------------------
850     continue        ! (TARGET FOR AN EARLIER GOTO)
c       HAVE WE ACHEIVED THE DESIRED PRECISION?  IF NOT, LOOP BACK AND
c       GET ANOTHER DIGIT FOR B, C, AND D.
        if (deltab.ge.prec) then
                print*, 'Looping again to increase precision.'
                print 111, bstart, deltab
                print 211, cstart, deltac
                print 311, dstart, deltad
                write (unit=4,fmt=*) 'Looping again to increase precision.'
                write (unit=4,fmt=111) bstart, deltab
                write (unit=4,fmt=211) cstart, deltac
                write (unit=4,fmt=311) dstart, deltad
        else
                print*, 'Desired precision (', prec,
     &                  ') has been reached.'
                print*, 'a,b,c,d='
                print*, a
                print*, b
                print*, c
                print*, d
                write(unit=4,fmt=*) 'Desired precision (', prec,
     &                  ') has been reached.'
                write(unit=4,fmt=*) 'a,b,c,d='
                write(unit=4,fmt=*) a
```

```
              write(unit=4,fmt=*) b
              write(unit=4,fmt=*) c
              write(unit=4,fmt=*) d
              hunt=.false.
       endif

154    continue      !(THIS HUGE OUTER LOOP NOT INDENTED)
       print*, '(Writing ABCDi.DAT)'
       if (case.eq.1) then
              open(unit=3, file='abcd1.dat', status='new')
       else if(case.eq.2) then
              open(unit=3, file='abcd2.dat', status='new')
       else if(case.eq.3) then
              open(unit=3, file='abcd3.dat', status='new')
       endif
       call date(buffer)
       call time(buffer2)
       write(unit=3, fmt=*), ' Results found by GETBCD run on ',
     &       buffer, ' at ', buffer2
       write(unit=3, fmt=83), 'a=', a
       write(unit=3, fmt=83), 'b=', b
       write(unit=3, fmt=83), 'c=', c
       write(unit=3, fmt=83), 'd=', d
       endfile(unit=3)
       close(unit=3)

c ---------------------------------------------------------------------
111    format (' new precision: bstart=',f18.15,' deltab=',f18.15)
211    format (' new precision: cstart=',f18.15,' deltac=',f18.15)
311    format (' new precision: dstart=',f18.15,' deltad=',f18.15)
044    format (' A=', F18.15)
144    format (' B=', F18.15, '   DeltaB=', F18.15)
244    format (' C=', F18.15, '   DeltaC=', F18.15)
344    format (' D=', F18.15, '   DeltaD=', F18.15)
91     format (a)
93     format (t20, e30.16)
83     format (' ', a, t20, e30.16)
c ---------------------------------------------------------------------
       print*,'Doit ends.'
       write(unit=4,fmt=*) 'Doit ends.'
       if (          (instructions_line1.eq.1).and.(case.eq.1)
     &              .or. (instructions_line1.eq.2).and.(case.eq.2)
     &              .or. (instructions_line1.eq.3).and.(case.eq.3)
     &              .or. (instructions_line1.eq.0).and.(case.ge.3))
     &             then
                    endfile(unit=4)
                    close(unit=4)
       endif

       return
```

```
      end

c ======================================================================
c     FCN
c     ===
      subroutine fcn(x,y,yprime)
c     (Evaluates derivatives.)
c     (Taken from ESCALINT3, 1989/Feb/16.  RETURN added at end.)
c     (FG3 term not added if FG3=0.  This allows the denominator to
c     be zero for when the uncoupled case is wanted.)

      real*8  a,b,c,d, fg1,fg2,fe1,fe2, fg3, zero,one,three
      real*8  p,q,u,v, pb,qb,pbb,qbb
      real*8  x,y(8),yprime(8)

      common a,b,c,d, fg1,fe1,fg2,fe2, fg3
      common /const/ zero,one,three

      p = y(1)
      q = y(2)
      u = y(3)
      v = y(4)
      pb = y(5)
      qb = y(6)
      pbb = y(7)
      qbb = y(8)
      if (x.gt.zero) then
            yprime(1) = q
            yprime(2) = (one-u**2)*p - 2*q/x +(pb**2)*p*fg1 + (pbb**2)*p*fg2
            yprime(3) = v
            yprime(4) = -u*p**2 - 2*v/x
            yprime(5) = qb
            yprime(6) = -2*qb/x + (one - pb*pb)*pb - p*p*pb*fe1
            if (fg3.ne.zero) then
                  yprime(6)= yprime(6) - (fg3/fg1)*pbb*pbb*pb
            endif
            yprime(7) = qbb
            yprime(8) = -2*qbb/x + (one -pbb*pbb)*pbb - p*p*pbb*fe2
                  if (fg3.ne.zero) then
                        yprime(8) = yprime(8) - (fg3*fe2/(fg2*fe1))*pb*pb*pbb
                  endif
      else
            yprime(1) = zero
            yprime(2) = (one-b*b+c*c*fg1+d*d*fg2)*a/three
            yprime(3) = zero
            yprime(4) = -b*a**2/three
            yprime(5) = zero
            yprime(6) = (one - a*a*fe1 -c*c )*c/three
            if (fg3.ne.zero) then
                  yprime(6)= yprime(6) - (fg3/fg1)*d*d *c/three
            endif
```

```
              yprime(7) = zero
              yprime(8) = (one - a*a*fe2 -d*d )*d/three
              if (fg3.ne.zero) then
                      yprime(8) = yprime(8)  - (fg3*fe2/(fg2*fe1))*c*c *d/three
              endif
       endif

       return
       end
c ------------------------------------------------------------------------------
c ------------------------------------------------------------------------------
c      DOPLOT
c      (DOES QUICK AND DIRTY SIDEWAYS PLOT ON SCREEN)
       subroutine doplot(y)
       integer         swidth,posn, scale
       real            m,b,ymin,ymax,y
       character*80    bar

       common          /plots/ scale

       parameter (swidth=80)
c      (swidth=screen width)

       if (scale.lt.1) scale=1
c      print*,'y, scale=', y,scale
       ymax= scale
       ymin=-ymax
       bar =        '--------------------------------------:'
     &          // ':++++.++++.++++.++++.++++.++++.++++.++++'

c      (Determine equation of line relating y-value to position to plot in)
       m=(1-swidth)/(ymin-ymax)
       b=1-m*ymin
       posn=int(m*y + b)
       if (posn.ge.1 .and. posn.le.swidth) then
              print*, bar(1:posn)
       else if (posn.lt.1) then
              print*, '<<<(off scale)'
       else
              print*, '(off scale)>>>'
       endif

       return
       end
c ------------------------------------------------------------------------------
c ------------------------------------------------------------------------------
```

2) ESCALINT3

```
c ------------------------------------------------------------------------
c ------------------------------------------------------------------------
c       ESCALINT3
c       =========
c
c       Computes psi, w, beta, sigma, etc.
c       Writes output to: ESCSUM.DAT, ESCLONG.DAT.
c       Reads: ABCDi.DAT, GROUP.DAT, FCONSTANTS.DAT, ESC_DO.DAT.
c       From ESC_DO.DAT it reads: record, regime, inverse_step_size.
c
c       If "record" is set to true, it writes out a number of
c       variables to "E3i.DAT" and "E4i.DAT" for later plotting.
c
c       Stops integrating when alpha starts to diverge (or when
c       x .ge. 100 [via ijlast] ).
c       (It starts looking for divergence at x+2 after the point
c       when alpha has crossed the axis "regime" number of times.)
c
c       (All variables declared; none implicit)
c
c       Unit            File
c       ----            ----
c       3               FCONSTANTS.DAT (read)
c       "               ABCDi.DAT (read)
c       4               GROUP.DAT (read)
c       "               ESC_DO.DAT (read)
c       6               ESCSUM.DAT (write)
c       7               ESCLONG.DAT (write)
c       8               E3i.DAT (write)
c       9               E4i.DAT (write)
c
c       Written: 1988/Nov       (by Philip Sharman)
c                               (modifed from ESCALINT by F.I. Cooperstock)
c       Revised: 1988/Nov/27    (added ESCSUM.DAT)
c                        28     (fixed bug about -b in /CONST/)
c                               (uses abs(alphaf) to test for alpha divergence)
c                               (print out psi instead of w properly in SUM.dat)
c                               (added w1,w2,w3 in /OMEGAS/)
c                        29     (alter order of fg's printout)
c                1988/Dec/4     (add ESCLONG.DAT writing)
c                               (reversed order of printing pua,pta etc)
c                               (added units to sigma, epsilon)
c                        10     (show more digits in escsum.dat)
c                        11     (added E3.DAT stuff) (deleted ESCALINT3.DAT)
c                        13     (took out the idea of using +b or -b; use a
c                                negative bstart as input instead)
c                1989/Jan/15    (E5.DAT added)
c                        18     (ESCSUM.DAT altered for better printout)
c                        26     (scale f's by 10^4 in ESCSUM.DAT)
c                     Feb/15    (Added the extra coupling term....
```

```
c                       Altered FCN to include F3 term.
c                       Altered COMMON to include FG3.
c                       Altered FG3 I/O.
c                       Changed omega, pextra, etc.)
c                 Mar/1 (Added warnings about when things diverge, etc.)
c                 May/19 Changed FCN to only add FG3 term if it is non
c                       zero.  Other places changed for that too.
c                   /21 (Added PUA etc to ESCSUM.DAT)
c                 Jun/4 (Do not compute psif if fe1 or fe2 are 0)
c                       (WARN10 and WARN11 added to show when y' and
c                       z' go positive)
c                       (NOTE: this warning not always very useful
c                       for z.  Could be more sophisticated?)
c                 Jun/12 (More digits shown in ESCSUM.DAT)
c                 Jun/13 (Percentage error shown in ESCSUM.DAT)
c                 Jun/14 (Formatting cleaned up)
c                 Jun/18 (IMPLICIT typing removed)
c                 Aug/4 (E3i.DAT file should have x=0 values in now
c                       for x,y,z,y1, and y2.)
c                 Aug/29 (Changed y**2*z*x**2 to z-alpha/x (for Beta)
c                       in E3i.DAT)
c                 Aug/31 (IJLAST was set to stop at 20; changed it
c                       to go to 100 if necessary)
c                 Sep/3 (ESC_DO.DAT added.)
c ------------------------------------------------------------------------
c ------------------------------------------------------------------------

      character*80   junk
      real*8         a, b, c, d, fg1, fe1, fg2, fe2, fg3,
     &                    zero, one, two, three,
     &                    w1, w2, w3
      common         a, b, c, d, fg1, fe1, fg2, fe2, fg3
      common /const/  zero, one, two, three
      common /omegas/ w1, w2, w3

      zero = 0.d0
      one = 1.d0
      two = 2.d0
      three = 3.d0

c     READ FG1 ETC
      print*, '(Reading FCONSTANTS.DAT)'
      open(unit=3, file='fconstants.dat', status='old')
      read(unit=3, fmt=80), junk
      read(unit=3, fmt=93), fg1
      read(unit=3, fmt=93), fg2
      read(unit=3, fmt=93), fe1
      read(unit=3, fmt=93), fe2
      read(unit=3, fmt=93), fg3
      close(unit=3)

c     MAIN LOOP:
```

```
c       DO GROUND CASE
        print*, '(Reading ABCD1.DAT)'
        open(unit=3, file='abcd1.dat', status='old')
        read(unit=3, fmt=80), junk
        read(unit=3, fmt=93), a
        read(unit=3, fmt=93), b
        read(unit=3, fmt=93), c
        read(unit=3, fmt=93), d
        close(unit=3)
        call doit(1)

c       DO MIDDLE CASE
        print*, '(Reading ABCD2.DAT)'
        open(unit=3, file='abcd2.dat', status='old')
        read(unit=3, fmt=80), junk
        read(unit=3, fmt=93), a
        read(unit=3, fmt=93), b
        read(unit=3, fmt=93), c
        read(unit=3, fmt=93), d
        close(unit=3)
        call doit(2)

c       DO HIGHEST EXCITED CASE
        print*, '(Reading ABCD3.DAT)'
        open(unit=3, file='abcd3.dat', status='old')
        read(unit=3, fmt=80), junk
        read(unit=3, fmt=93), a
        read(unit=3, fmt=93), b
        read(unit=3, fmt=93), c
        read(unit=3, fmt=93), d
        close(unit=3)
        call doit(3)

c       ALL DONE
        print*, 'Program ends.'

c       -------formats---------
80      format  (a)
93      format  (t20, e30.16)
c       ------------------------
        end
c ---------------------------------------------------------------------------
c ---------------------------------------------------------------------------
c       DOIT
        subroutine doit(case)

        real*8          charge, x, h, delta, alpha, beta, gamma, psi,
     &                  xold, p, u, pb, pbb, deltax,
     &                  deltaf, oldalphaf, alphaf, gammaf,
     &                  oldpsif, oldpextraf, aa, olddeltaf, oldgammaf,
     &                  omega, charge2, sigma, epsilon, pi, f1, f2, g1, g2, f3,
     &                  pua, pub, pta, ptb, pextra, abg,
```

```
&                puaf, pubf, ptaf, ptbf, pextraf, psif,
&                pua1, pub1, pta1, ptb1, pextra1, abg1,
&                pua2, pub2, pta2, ptb2, pextra2, abg2,
&                pua3, pub3, pta3, ptb3, pextra3, abg3,
&                pe1, pe2,
&                y(8), wk(232), masslow, massmed, masshigh, mass2,
&                a, b, c, d, fg1, fe1, fg2, fe2, fg3,
&                zero, one, two, three,
&                a1, b1, c1, d1, w1, alpha1, beta1,
&                a2, b2, c2, d2, w2, alpha2, beta2,
&                a3, b3, c3, d3, w3, alpha3, beta3,
&                check_alpha
      integer    ij, ijlast, case, regime, inverse_step_size, nodes
      logical    okay, record, lastpos
      logical    warn1, warn2, warn3, warn4, warn5, warn6, warn7, warn8, warn9,
&                warn10, warn11
      character*12   buffer, buffer2, recfile, rec2
      character*100  junk
      common     a, b, c, d, fg1, fe1, fg2, fe2, fg3
      common /const/  zero, one, two, three
      common /case1/  a1, b1, c1, d1, w1, alpha1, beta1
      common /case2/  a2, b2, c2, d2, w2, alpha2, beta2
      common /case3/  a3, b3, c3, d3, w3, alpha3, beta3
      save /case1/
      save /case2/
      save /case3/
      external fcn

      warn1=.true.
      warn2=.true.
      warn3=.true.
      warn4=.true.
      warn5=.true.
      warn6=.true.
      warn7=.true.
      warn8=.true.
      warn9=.true.
      warn10=.true.
      warn11=.true.

      print*, ' '
      print*, 'case=', case
      if (case.ne.1 .and. case.ne.2 .and. case.ne.3) call exit(0)
      print *, 'A=', a
      print *, 'B=', b
      print *, 'C=', c
      print *, 'D=', d
      print*, 'FG1=', fg1
      print*, 'FG2=', fg2
      print*, 'FE1=', fe1
      print*, 'FE2=', fe2
      print*, 'FG3=', fg3
```

```
      if (case.eq.1) then
            call date(buffer)
            call time(buffer2)
            print*, 'ESCALINT3 run on ', buffer, ' at ', buffer2
      endif
      if (case.eq.1) then
C     (READ MASS OF GROUND PARTICLE, AND CHARGE OF GROUP)
            print*, '(Reading GROUP.DAT)'
            open(unit=4, file='group.dat', status='old')
            read(unit=4, fmt=80), junk
            read(unit=4, fmt=95), masslow
            read(unit=4, fmt=95), massmed
            read(unit=4, fmt=95), masshigh
            read(unit=4, fmt=95), charge
            close(unit=4)
c     (READ ESC_DO.DAT)
            print*, '(Reading ESC_DO.DAT)'
            open(unit=4, file='esc_do.dat', status='old')
            read(unit=4, fmt=*) record
            read(unit=4, fmt=*), regime
            read(unit=4, fmt=*), inverse_step_size
            close(unit=4)
            print*, 'Regime = ', regime
      endif

      if (record) then
            print*, '(Record is .true.)'
            if (case.eq.1) recfile='e31.dat'
            if (case.eq.2) recfile='e32.dat'
            if (case.eq.3) recfile='e33.dat'
            if (case.eq.1) rec2='e41.dat'
            if (case.eq.2) rec2='e42.dat'
            if (case.eq.3) rec2='e43.dat'
            open(unit=8, file=recfile, status='new')
            print*, 'Opening file: ', recfile
            open(unit=9, file=rec2, status='new')
            print*, 'Opening file: ', rec2
      endif

c     CALCULATE
      x = zero
      y(1) = a
      y(2) = zero
      y(3) = b
      y(4) = zero
      y(5) = c
      y(6) = zero
      y(7) = d
      y(8) = zero
      h = one/inverse_step_size
      print*,'step size =1/', inverse_step_size, ' = ', h
      delta  = zero
```

```
      alpha  = zero
      gamma  = zero
      psi = zero
      pta = zero
      ptb = zero
      pua = zero
      pub = zero
      pextra = zero
      ijlast = 20*inverse_step_size
      check_alpha = 999.9
c     (OKAY STOPS US WANDERING INTO THE REGION WHERE THE FUNCTION BLOWS UP)
      okay=.true.
c     (LASTPOS RECORDS WHETHER ALPHA IS + OR -)
      if (y(3).ge.zero) then
            lastpos = .true.
      else
            lastpos = .false.
      endif
c     (NODES RECORDS HOW MANY TIMES ALPHA HAS CROSSED THE AXIS)
      nodes = 0

      print*,'Integrating...'
      ij= 1
c -------------------------------INTEGRATION LOOP-----------------------
      do 30 while (okay .and. ij.le.ijlast)
            xold = x
            call drkstp(8, h, x, y, fcn, wk)

c           PERFORM THE INTEGRATION OF ALPHA, DELTA, GAMMA, PSI,PTA,PTB,ETC
            p = y(1)
            u = y(3)
            pb = y(5)
            pbb = y(7)
            deltax=x-xold
            if ((abs(deltax-h)).gt.1.0d-15) print*,'deltax,h=',deltax,h
            deltaf = p**2*u*x
            delta  = delta  + deltaf*deltax
            oldalphaf=alphaf
            alphaf = deltaf*x
            alpha  = alpha  + alphaf*deltax

c           HAS ALPHA CROSSED THE AXIS?
            if (lastpos .and. alpha.lt.zero) then
                  lastpos = .false.
                  nodes = nodes + 1
                  print 292, 'Alpha has crossed from + to -.  Number of nodes =',
     &                  nodes, ', x = ', x
292               format (' ', a, i2, a, f5.2)
                  if (nodes.eq.regime) check_alpha = x + 2.0
            endif
            if (.not.lastpos .and. alpha.gt.zero) then
                  lastpos = .true.
```

```
                nodes = nodes + 1
                print 292, 'Alpha has crossed from - to +.  Number of nodes =',
     &                nodes, ', x = ', x
                if (nodes.eq.regime) check_alpha = x + 2.0
              endif
              if (nodes.gt.regime) then
                print*, 'Something is wrong.  Nodes=', nodes, 'regime=', regime
                stop
              endif
c             (STOP INTEGRATING WHEN ALPHA STARTS DIVERGING)
              if (x.ge.check_alpha .and. abs(alphaf).gt.abs(oldalphaf) ) then
                print*, 'Stopping.  Alpha starts diverging at x=', x
                print*, '(check_alpha = ', check_alpha, ')'
                print*, 'Here y,y1,y2,z= ', y(1), y(5), y(7), y(3)
                okay=.false.
              endif

              gammaf = alphaf*u
              gamma  = gamma  + gammaf*deltax
              if (fe1.ne.zero .and. fe2.ne.zero) then
                psif = fg1*x*x*pb*pb*(p*p+ pb*pb/(two*fe1)) +
     &                 fg2*x*x*pbb*pbb*(p*p+pbb*pbb/(two*fe2))
              else
c               (PSI NOT COMPUTED TO AVOID DIVISION BY ZERO ERROR)
                psif= zero
              endif
              psi = psi + psif*deltax
              ptaf = x*x*p*p*pb*pb
              pta = pta+ ptaf*deltax
              ptbf = x*x*p*p*pbb*pbb
              ptb = ptb + ptbf*deltax
              puaf = x*x*pb*pb*pb*pb/two
              pua = pua + puaf*deltax
              pubf = x*x*pbb*pbb*pbb*pbb/two
              pub = pub + pubf*deltax
              pextraf = pb*pb* pbb*pbb* x*x
              pextra= pextra + pextraf*deltax
              olddeltaf=deltaf
              oldgammaf=gammaf
              oldpsif=psif
              oldpextraf=pextraf
c             WARNINGS
              if (x.ge.check_alpha .and. abs(alphaf).gt.abs(oldalphaf) .and. warn1) then
                warn1=.false.
                print*,'WARNING: alpha starts to diverge at x=',x
              endif
c             (DON'T EVER USE DELTA, SO DON'T REALLY CARE ABOUT IT.)
c             if (x.gt.5  .and. abs(deltaf).gt.abs(olddeltaaf) .and. warn2) then
c               warn2=.false.
c               print*,'WARNING: delta starts to diverge at x=',x
```

```
c               endif
                if (x.gt.5 .and. abs(gammaf).gt.abs(oldgammaf) .and. warn3) then
                        warn3=.false.
                        print*,'WARNING: gamma starts to diverge at x=',x
                endif
                if (x.gt.5  .and. abs(psif).gt.abs(oldpsif) .and. warn4) then
                        warn4=.false.
                        print*,'WARNING: psi starts to diverge at x=',x
                endif
                if (x.gt.5  .and. abs(pextraf).gt.abs(oldpextraf) .and. warn9) then
                        warn9=.false.
                        print*,'WARNING: psi starts to diverge at x=',x
                endif
c               CHECK DERIVATIVES OF Y AND Z
                if (x.ge.check_alpha) then
                        if (lastpos .and. y(2).gt.zero .and. warn10) then
                                print*,'y''>0 at x=',x
                                warn10= .false.
                        endif
                        if (.not.lastpos .and. y(2).lt.zero .and. warn10) then
                                print*,'y''<0 at x=',x
                                warn10= .false.
                        endif
                        if (lastpos .and. y(4).gt.zero .and. warn11) then
                                print*,'z''>0 at x=',x
                                warn11= .false.
                        endif
                        if (.not.lastpos .and. y(4).lt.zero .and. warn11) then
                                print*,'z''<0 at x=',x
                                warn11= .false.
                        endif
                endif

c               (SELECTIVE PRINTOUT CAN GO HERE IF DESIRED)

c               WRITE TO "E3i.DAT" IF WANTED.
c               (DO x=0 STUFF)
                if (record .and. ij.le.1) then
                        write(unit=8, fmt=654), zero, a, b, c, d,
     &                       zero, zero, zero, b
                endif
c               (WRITE OUT IN INCREMENTS OF 0.1 ALONG THE X AXIS)
                if (record .and. int(mod(ij, int(inverse_step_size*0.1))).eq.1 ) then
                        write(unit=8, fmt=654), x,y(1),y(3),y(5),y(7),
     &                       alpha,gamma,psi, y(3)- alpha/x
654                     format (f8.3, 1x, f8.3, 1x, f8.3, 1x, f8.3, 1x,
     &                       f8.3, 1x, f8.3, 1x, f8.3, 1x, f12.3, 1x, f12.3)
                endif

                ij=ij+1
30      continue
c -----------(END OF INTEGRATION LOOP)---------------------------------------
```

```
c       CHECK: DID Z CROSS ENOUGH TIMES?
        if (nodes.lt.regime) then
                print*, 'ERROR! Nodes = ', nodes, ' Regime = ', regime
                stop
        endif

c       CLOSE E3i.DAT FILE IF NECESSARY
        if (record) then
                endfile(unit=8)
                close(unit=8)
        endif

c       PRINTOUT
        print*,'Stopped at X=',x
        print*,'ALPHA=',alpha
        print*,'GAMMA=',gamma
        print*,'PSI=',psi

c       COMPUTE BETA
        beta=y(3)-(alpha/x)
        print*,'BETA=', beta
        print*,'z,x=',y(3),x

        print*,'alpha,beta+gamma=', (alpha*beta+gamma)

c       CHECK ALPHA
        aa=(y(3) - beta)*x
c       print*, 'ALPHA should equal:', aa, ' Difference=', (alpha-aa)
        if (abs(aa-alpha) .gt. 1.0d-12) then
                print*, 'ERROR!!!! aa does not equal alpha!!!!!'
                call exit(0)
        endif

c       COMPUTE OMEGA
        omega=psi - (alpha*beta + gamma)
        if (fg3.ne.zero) then
                omega = omega + (fg3/fe1)*pextra
        endif
        print*,'OMEGA(w)=', omega
        if (case.eq.1) w1=omega
        if (case.eq.2) w2=omega
        if (case.eq.3) w3=omega

        if (case.eq.1) then
c               COMPUTE SIGMA ETC
                print*, 'mass of lowest state=', masslow, ' MeV'
                print*, 'charge of group=', charge, ' (in units of e)'
c               (convert mass from MeV to cm, scaled by 10^60)
c               (1ev=1.324x10^-61cm [MTW], 1Mev=10^6ev)
                mass2 = masslow * (1.324d-1) * 1d6
c               (convert charge to cm, scaled by 10^30)
```

```
c               (e=1.381x10^-34cm [MTW])
                charge2 = charge * (1.381d-4)
                print*, 'mass of ground state=   ',mass2, ' x10^-60 cm'
                print*, 'charge of ground state=',charge2,' x10^-30 cm'

c               (equation 4.19, page 22)
c               (the scaling factors cancel out)
                sigma=(2.0d0 * alpha**2/omega)*(mass2/charge2**2)
                print*,'SIGMA=', sigma, ' cm^-1'
                print*,'1/SIGMA=', (1/sigma), ' cm'

c               COMPUTE EPSILON
c               (Scaled)
                epsilon=alpha/charge2
                print*,'EPSILON=', epsilon, 'x10^30 cm^-1'

c               COMPUTE F1 ETC
                pi=3.14159265358979323846 d0
c               (SKIP THIS SECTION IF THERE IS NO COUPLING)
                if (fg1.ne.zero .and. fg2.ne.zero .and. fg3.ne.zero
     &              .and. fe1.ne.zero .and. fe2.ne.zero) then
                        f1 = (-fe1)*(4.0d0*pi*epsilon**2)
                        f2 = (-fe2)*(4.0d0*pi*epsilon**2)
                        g1 = f1/(-fg1)
                        g2 = f2/(-fg2)
                        f3 = -g2*(fg3/fg1)
                        print99,'F1=', f1, 'x10^60'
                        print99,'F2=', f2, 'x10^60'
                        print99,'G1=', g1, 'x10^60'
                        print99,'G2=', g2, 'x10^60'
                        print99,'F3=', f3, 'x10^60'
                        if ( abs(-f1/g1 - (fg1) ).gt.1.0d-15) call exit(0)
                        if ( abs(-f2/g2 - (fg2) ).gt.1.0d-15) call exit(0)
                        if ( abs(-f3/g2 - (fg3/fg1) ).gt.1.0d-15) call exit(0)
                        if ( abs(-f3/g1 - (fg3*fe2/(fg2*fe1) ) ).gt.1.0d-15) call exit(0)
                endif
        endif

c       PRINT INTEGRALS TO SCREEN
        print*,'pua=',pua
        print*,'pub=',pub
        print*,'pta=',pta
        print*,'ptb=',ptb
        print*,'pextra=',pextra

c       SAVE THE VARIABLES
        if (case.eq.1) then
                a1=a
                b1=b
                c1=c
                d1=d
                w1=omega
```

```
              alpha1=alpha
              beta1=beta
              pua1=pua
              pub1=pub
              pta1=pta
              ptb1=ptb
              pextra1=pextra
              abg1=alpha*beta + gamma
      endif

      if (case.eq.2) then
              a2=a
              b2=b
              c2=c
              d2=d
              w2=omega
              alpha2=alpha
              beta2=beta
              pua2=pua
              pub2=pub
              pta2=pta
              ptb2=ptb
              pextra2=pextra
              abg2=alpha*beta + gamma
      endif

      if (case.eq.3) then
              a3=a
              b3=b
              c3=c
              d3=d
              w3=omega
              alpha3=alpha
              beta3=beta
              pua3=pua
              pub3=pub
              pta3=pta
              ptb3=ptb
              pextra3=pextra
              abg3=alpha*beta + gamma
      endif


c     WRITE TO LONG FILE
      if (case.eq.1) then
              print*, '(Writing to ESCLONG.DAT)'
              open(unit=7, file='esclong.dat', status='new')
              write(unit=7, fmt=*),
     &              'Long output of ESCALINT3 run on ', buffer,
     &       ' at ', buffer2
              write(unit=7,fmt=92),'fg1=', fg1
```

```
            write(unit=7,fmt=92),'fg2=', fg2
            write(unit=7,fmt=92),'fe1=', fe1
            write(unit=7,fmt=92),'fe2=', fe2
            write(unit=7,fmt=92),'fg3=', fg3
         endif
c        (NEXT FIVE LINES ARE WRITTEN FOR ALL 3 LEVELS)
         write(unit=7,fmt=92),'pua=', pua
         write(unit=7,fmt=92),'pub=', pub
         write(unit=7,fmt=92),'pta=', pta
         write(unit=7,fmt=92),'ptb=', ptb
         write(unit=7,fmt=92),'pextra=', pextra
         write(unit=7,fmt=92),'alpha*beta+gamma=',
     &          (alpha*beta + gamma)
         write(unit=7, fmt=92),'psi=', psi
         write(unit=7,fmt=92),'w=',  omega
         if (case.ge.3) then
                 endfile(unit=7)
                 close(unit=7)
         endif


         if (case.eq.3) then
                 print*, '(Writing to ESCSUM.DAT)'
                 open(unit=6,file='escsum.dat',status='new')
                 write(unit=6,fmt=*),'Summary of ESCALINT3 ',
     &                buffer, '  ', buffer2
                 write(unit=6, fmt=200), 'A=', a1, a2, a3
                 write(unit=6, fmt=200), 'B=', b1, b2, b3
                 write(unit=6, fmt=200), 'C=', c1, c2, c3
                 write(unit=6, fmt=200), 'D=', d1, d2, d3
                 write(unit=6, fmt=*),
     &                '-------------------------------------------------------------
-------',
                 write(unit=6, fmt=207), 'w=', w1, w2, w3
                 write(unit=6, fmt=208), 'Alpha=', alpha1, alpha2, alpha3
                 write(unit=6, fmt=207), 'Beta=', beta1, beta2, beta3
                 write(unit=6, fmt=201), (fg1*1.0d4), (fe1*1.0d4)
                 write(unit=6, fmt=202), (fg2*1.0d4), (fe2*1.0d4),
     &                                   (fg3*1.0d4)
                 write(unit=6, fmt=203), (w2/w1), (w3/w1)
                 pe1= 100* ( ((w2/w1)-(massmed/masslow)) /(massmed/masslow) )
                 pe2= 100* ( ((w3/w1)-(masshigh/masslow))/(masshigh/masslow) )
                 write(unit=6, fmt=209), pe1, pe2
                 write(unit=6, fmt=*), ' '
                 write(unit=6, fmt=*), 'pua, pub, pta, ptb, pextra, abg...'
                 write(unit=6, fmt=901), '1', pua1, pub1, pta1,
     &                          ptb1, pextra1, abg1
                 write(unit=6, fmt=901), '2', pua2, pub2, pta2,
     &                          ptb2, pextra2, abg2
                 write(unit=6, fmt=901), '3', pua3, pub3, pta3,
     &                          ptb3, pextra3, abg3
                 endfile(unit=6)
                 close(unit=6)
```

```
      endif
901   format (' ', a1, ': ', 6(f7.3, 1x) )
200   format (' ', a6,  ' : ',  3(f17.10, ' : ') )
201   format (' ', 'FG1=', f12.3,  3x,  'FE1=', f12.3 , ' [x10^-4 for all F''s]' )
202   format (' ', 'FG2=', f12.3,  3x,  'FE2=', f12.3 ,
     &          3x, 'FG3=', f12.3)
203   format (' ', 'w2/w1=', f12.4, 3x,    'w3/w1=', f12.4)
207   format (' ', a6,  ' : ',  3(f10.3, 7x, ' : ') )
208   format (' ', a6,  ' : ',  3(f12.5, 5x, ' : ') )
209   format (' ', 'error='  f12.4, '

c     WRITE TO E4i.DAT
      if (record) then
            do 345, x=0.2, 10, 0.2
                  write(unit=9,fmt=*), x, (alpha/x + beta), beta
345         continue
c           CLOSE E4i.DAT FILE IF NECESSARY
            endfile(unit=9)
            close(unit=9)
      endif


c     -------------FORMATS---------------------------------
101   format (' ', 'A=', f9.5, ' B=', f9.5,
     &        ' C=', f9.5, ' D=', f9.5)
109   format (' ', 'w=', f12.3, ' alpha=', f12.4,
     &    ' psi=', e12.4)
102   format (' ', 'psi=',
     &      '(', f10.2,'*',f5.2, ')',
     &      ' + ',
     &      '(', f10.2,'*',f5.2, ')',
     &    ' + ',
     &      '(', f10.2,'*',f5.2, ')',
     &      ' + ',
     &      '(', f10.2,'*',f5.2, ')' )
103   format(' ', ' =',
     &     f10.2, '+', f10.2, '+', f10.2, '+', f10.2, '.')
104   format(' ', 'w=  (', f10.2, ')-(', f10.2, ')' )
105   format(' ', ' =    ', f10.2, '.' )


90    format (a, f12.4)
99    format (' ', a, e15.6, a)
92    format (' ', a, t20, e30.16)
80    format (a)
95    format (t20, e30.16)
c -------------------------------------

      return
      end
c -------------------------------------------------------------------------
c -------------------------------------------------------------------------
```

```
c       FCN     (EVALUATES DERIVATIVES)
        subroutine fcn(x, y, yprime)
        real*8          y(8), yprime(8), p, q, u, v, pb, pbb, qb, qbb,
     &                      a, b, c, d, fg1, fe1, fg2, fe2, fg3,
     &                      zero, one, two, three, x
        common          a, b, c, d, fg1, fe1, fg2, fe2, fg3
        common /const/  zero, one, two, three

        p = y(1)
        q = y(2)
        u = y(3)
        v = y(4)
        pb = y(5)
        qb = y(6)
        pbb = y(7)
        qbb = y(8)
        if (x.gt.zero) then
                yprime(1) = q
                yprime(2) = (one-u**2)*p - 2*q/x +(pb**2)*p*fg1 + (pbb**2)*p*fg2
                yprime(3) = v
                yprime(4) = -u*p**2 - 2*v/x
                yprime(5) = qb
                yprime(6) = -2*qb/x + (one - pb*pb)*pb - p*p*pb*fe1
                if (fg3.ne.zero) then
                        yprime(6)= yprime(6) - (fg3/fg1)*pbb*pbb*pb
                endif
                yprime(7) = qbb
                yprime(8) = -2*qbb/x + (one -pbb*pbb)*pbb - p*p*pbb*fe2
                if (fg3.ne.zero) then
                        yprime(8) = yprime(8) - (fg3*fe2/(fg2*fe1))*pb*pb*pbb
                endif
        else
                yprime(1) = zero
                yprime(2) = (one-b*b+c*c*fg1+d*d*fg2)*a/three
                yprime(3) = zero
                yprime(4) = -b*a**2/three
                yprime(5) = zero
                yprime(6) = (one - a*a*fe1 -c*c )*c/three
                if (fg3.ne.zero) then
                        yprime(6)= yprime(6) - (fg3/fg1)*d*d *c/three
                endif
                yprime(7) = zero
                yprime(8) = (one - a*a*fe2 -d*d )*d/three
                if (fg3.ne.zero) then
                        yprime(8) = yprime(8)  - (fg3*fe2/(fg2*fe1))*c*c *d/three
                endif
        endif

        return
        end
c -------------------------------------------------------------------------
c -------------------------------------------------------------------------
```

3) IMPROVE

```
c ------------------------------------------------------------------------
c ------------------------------------------------------------------------
c       IMPROVE
c       =======
c
c       Calculates better FG1, FG2 given PUA, etc.
c       Takes input from ESCLONG.DAT, GROUP.DAT.
c
c       Writes better FG's to file IMPROVE.DAT.
c
c       Written:
c       1988/Nov        -by Philip Sharman.
c       Revised:
c       1988/Dec/9      - bug fixed (w1 not constant)
c            Mar/25     - FG3 and PEXTRA terms added.
c ------------------------------------------------------------------------
c ------------------------------------------------------------------------

        implicit real*8 (a-z)
        character*79    title
        character*12    buffer, buffer2

        call date(buffer)
        call time(buffer2)
        print*,'IMPROVE run on ', buffer, ' at ', buffer2

c       READ DATA
        print*, 'Reading ESCLONG.DAT:'
        open(unit=3, file='esclong.dat', status='old')
        read(unit=3, fmt=90), title
        print*, title
        read(unit=3,fmt=93), fg1
        read(unit=3,fmt=93), fg2
        read(unit=3,fmt=93), fe1
        read(unit=3,fmt=93), fe2
        read(unit=3,fmt=93), fg3

        read(unit=3,fmt=93), pua1
        read(unit=3,fmt=93), pub1
        read(unit=3,fmt=93), pta1
        read(unit=3,fmt=93), ptb1
        read(unit=3,fmt=93), pextra1
        read(unit=3,fmt=93), abg1
        read(unit=3,fmt=93), junk
        read(unit=3,fmt=93), w1

        read(unit=3,fmt=93), pua2
        read(unit=3,fmt=93), pub2
        read(unit=3,fmt=93), pta2
        read(unit=3,fmt=93), ptb2
```

```
      read(unit=3,fmt=93), pextra2
      read(unit=3,fmt=93), abg2
      read(unit=3,fmt=93), junk
      read(unit=3,fmt=93), w2

      read(unit=3,fmt=93), pua3
      read(unit=3,fmt=93), pub3
      read(unit=3,fmt=93), pta3
      read(unit=3,fmt=93), ptb3
      read(unit=3,fmt=93), pextra3
      read(unit=3,fmt=93), abg3
      read(unit=3,fmt=93), junk
      read(unit=3,fmt=93), w3

      close(unit=3)

      print*, '      pua     pub     pta     ptb  pextra     abg'
      print 901, '1', pua1, pub1, pta1,
     &                    ptb1, pextra1, abg1
      print 901, '2', pua2, pub2, pta2,
     &                    ptb2, pextra2, abg2
      print 901, '3', pua3, pub3, pta3,
     &                    ptb3, pextra3, abg3

      print*
      print*, 'old fg1=', fg1
      print*, 'old fg2=', fg2
      print*, 'fe1=', fe1
      print*, 'fe2=', fe2
      print*, 'fg3=', fg3
901   format (' ', a1, ': ', 6(f7.3, 1x) )

      print*, '(Reading GOUP.DAT)'
      open(unit=3, file='group.dat', status='old')
      read(unit=3, fmt=90), title
      read(unit=3, fmt=93), mass1
      read(unit=3, fmt=93), mass2
      read(unit=3, fmt=93), mass3
      close(unit=3)

c -------------------------------
c     (NEW VARIABLE NAMES)
      b1=(mass2/mass1)
      b2=(mass3/mass1)
      print*, 'desired wm/we=', b1
      print*, 'desired wt/we=', b2

      a=    pua1
      b=    pub1
      c=    pta1
      d=    ptb1
      e=    abg1 - (fg3/fe1)*pextra1
```

```
      f=      pua2
      g=      pub2
      h=      pta2
      j=      ptb2
      k=      abg2 - (fg3/fe1)*pextra2

      l=      pua3
      m=      pub3
      n=      pta3
      p=      ptb3
      q=      abg3 - (fg3/fe1)*pextra3


ccc   print 901, '1', a,b,c,d,e, w1
ccc   print 901, '2', f,g,h,j,k, w2
ccc   print 901, '3', l,m,n,p,q, w3

      c1= (b1*a - f)/fe1 + b1*c - h
      c2= (b1*b - g)/fe2 + b1*d - j
      c3= b1*e - k

      c4= (b2*a - l)/fe1 + b2*c - n
      c5= (b2*b - m)/fe2 + b2*d - p
      c6= b2*e - q

c -------------------------------------------------------
c     CALCULATE
c     Solve: xC1 + yC2 = C3
c            xC4 + yC5 = C6

      x= (c3*c5 - c6*c2)/(c1*c5 - c4*c2)
      y= (c3*c4 - c6*c1)/(c2*c4 - c5*c1)
      print*, 'Perfect fg1=', x
      print*, 'Perfect fg2=', y

c     WRITE TO OUTPUT FILE
      print*, '(Writing to IMPROVE.DAT)'
      open(unit=3,file='improve.dat',status='new')
      write(unit=3,fmt=*), 'IMPROVE run on ',
     &                buffer, ' at ', buffer2
      write(unit=3,fmt=80), 'Perfect fg1=', x
      write(unit=3,fmt=80), 'Perfect fg2=', y
80    format (' ', a, t20, e30.16)

c     CHECK
c     (lhs should equal rhs in all these checks)
      criteria= 1.0 d-10
c     (error criteria)

      lhs= x*c1 + y*c2
      rhs= c3
```

```
      if (abs(lhs-rhs).gt.criteria) then
              print*,'**** WARNING! *** Check #1 fails'
              print*,'Difference = ', (lhs-rhs)
              write(unit=3,fmt=*),'**** WARNING! *** Check #1 fails'
              print*, 'lhs rhs=', lhs, rhs
              write(unit=3,fmt=*),'Difference = ', (lhs-rhs)
      endif

      lhs= x*c4 + y*c5
      rhs= c6
      if (abs(lhs-rhs).gt.criteria) then
              print*,'**** WARNING! *** Check #2 fails'
              print*,'Difference = ', (lhs-rhs)
              print*, 'lhs rhs=', lhs, rhs
              write(unit=3,fmt=*),'**** WARNING! *** Check #2 fails'
              write(unit=3,fmt=*),'Difference = ', (lhs-rhs)
      endif

c     (use the old fg's)
      lhs=w1
      rhs=(fg1*pua1)/fe1 + (fg2*pub1)/fe2
     &              + fg1*pta1 + fg2*ptb1 - (abg1 - (fg3/fe1)*pextra1)
      if (abs(lhs-rhs).gt.criteria) then
              print*,'**** WARNING! *** Check #3 fails'
              print*,'Difference = ', (lhs-rhs)
              print*, 'lhs rhs=', lhs, rhs
              write(unit=3,fmt=*),'**** WARNING! *** Check #3 fails'
              write(unit=3,fmt=*),'Difference = ', (lhs-rhs)
      endif


c     (use the new fg's)
      fg1=x
      fg2=y
c     (figure out what w1 is now with the new fg's)
      w1new=(fg1*pua1)/fe1 + (fg2*pub1)/fe2
     &              + fg1*pta1 + fg2*ptb1 - (abg1 - (fg3/fe1)*pextra1)


      lhs=(b1*w1new)
      rhs=(fg1*pua2)/fe1 + (fg2*pub2)/fe2
     &              + fg1*pta2 + fg2*ptb2 - (abg2 - (fg3/fe1)*pextra2)
      if (abs(lhs-rhs).gt.criteria) then
              print*,'**** WARNING! *** Check #4 fails'
              print*,'Difference = ', (lhs-rhs)
              print*, 'lhs rhs=', lhs, rhs
              write(unit=3,fmt=*),'**** WARNING! *** Check #4 fails'
              write(unit=3,fmt=*),'Difference = ', (lhs-rhs)
      endif

      lhs=(b2*w1new)
      rhs=(fg1*pua3)/fe1 + (fg2*pub3)/fe2
```

```
     &          + fg1*pta3 + fg2*ptb3 - (abg3 - (fg3/fe1)*pextra3)
           if (abs(lhs-rhs).gt.criteria) then
                  print*,'**** WARNING! *** Check #5 fails'
                  print*,'Difference = ', (lhs-rhs)
                  print*, 'lhs rhs=', lhs, rhs
                  write(unit=3,fmt=*),'**** WARNING! *** Check #5 fails'
                  write(unit=3,fmt=*),'Difference = ', (lhs-rhs)
           endif

           print*,'Checks are done.'

           write(unit=3,fmt=*), 'Checks are done.'
           endfile(unit=3)
           close(unit=3)


c          ---------------------------------------------
90         format (a)
93         format ( t20, e30.16)
           print*,'Program ends.'
           end
```

4) TRANSFER

```
c-------------------------------------------------------------------------------
c-------------------------------------------------------------------------------
c       TRANSFER
c       ========
c
c       Takes "perfect" FG's from IMPROVE.DAT and inserts them
c       into FCONSTANTS.DAT.
c
c       Written:
c       1988/Dec/10     -by Philip Sharman.
c       Revised:
c       1989/Jan/19     -keep original title; add comment at bottom
c                        of output file.
c            Jun/12     -FG3 term added.
c-------------------------------------------------------------------------------
c-------------------------------------------------------------------------------
        implicit        real*8 (a-z)
        character*40    title
        character*100   character_junk
        character*12    buffer, buffer2

        call date(buffer)
        call time(buffer2)

c       READ THE OLD VALUES
        open(unit=3, file='fconstants.dat', status='old')
        read(unit=3, fmt=90), title
        read(unit=3, fmt=91), junk
        read(unit=3, fmt=91), junk
        read(unit=3, fmt=91), fe1
        read(unit=3, fmt=91), fe2
        read(unit=3, fmt=91), fg3
        close(unit=3)
90      format (a)
91      format (t20, e30.16)

c       READ THE NEW VALUES
        open(unit=3, file='improve.dat', status='old')
        read(unit=3, fmt=90), character_junk
        read(unit=3, fmt=91), fg1
        read(unit=3, fmt=91), fg2
        close(unit=3)

c       WRITE THE FILE BACK OUT
        open(unit=3, file='fconstants.dat', status='new')
        write(unit=3, fmt=*), title
        write(unit=3, fmt=81), 'fg1=', fg1
        write(unit=3, fmt=81), 'fg2=', fg2
        write(unit=3, fmt=81), 'fe1=', fe1
        write(unit=3, fmt=81), 'fe2=', fe2
```

```
      write(unit=3, fmt=81), 'fg3=', fg3
      write(unit=3, fmt=*), 'Altered by TRANSFER on ',
     &              buffer, ' at ', buffer2
      endfile(unit=3)
      close(unit=3)
81    format (' ', a, t20, e30.16)

      print*, '"Perfect" F''s have been copied from IMPROVE.DAT to FCONSTANTS.DAT.'
      end
```

5) PARIMP

```
c -----------------------------------------------------------------
c -----------------------------------------------------------------
c
c       PARIMP.FOR
c       ==========
c       This is "Parafit" modified to improve F's.
c       Does only the 2 quark groups.
c       FG3 is included.
c
c       (See Basic Scientific Subroutines, Vol 2, page 80)
c
c       Reads:  ESC_EMT.DAT, ESC_UCT.DAT, ESC_DSB.DAT,
c               FCONSTANTS.DAT.
c       Writes: PARIMP.DAT.
c
c       Written:
c       1989/Jan/16     (by Philip Sharman)
c
c       Revised:
c       1989/Jan/16     (changed from BASIC to FORTRAN)
c                 17    (added input/output)
c                 19    (keep original title; add comment to bottom)
c                 26    (character_junk fixed to absorb unwanted titles)
c                 28    (REAL changed to REAL*8 or DIMENSION)
c                 28    (printout scaled.  And aborts if #iterations too big)
c              Feb/18   (FG3 term added, and DSB added.)
c                 21    (Top quark equation deactivated.)
c                 21    (LT changed to LE when making w's positive)
c                 23    (Read mass ratios from files.)
c              Mar/8    (Tried not making negative w's positive)
c                 14    (Changed to make sure initial guess are over-estimates)
c                 24    (Bug changed: had wrong charm/up desired mass ratio)
c                 28    (Tried making the initial guesses all -1)
c                 28    (Tried making the initial guesses all -200d-4)
c              May/22   (EQ changed to LE in RESIDUALS)
c                       (check for proper xx added in FUNC)
c -----------------------------------------------------------------
c -----------------------------------------------------------------
      implicit real*8(a-z)
      common L, n, x, y, e, e1, d, m, ee1
      common /params/ a
      common /fs/ fg1_initial_guess, fg2_initial_guess,
     &            fe1_initial_guess, fe2_initial_guess,
     &            fg3_initial_guess
      common /title/  fs_title
      integer n, L, I, m
      dimension       x(6), y(6), a(5), e1(6)
      character*80    fs_title
      character*12    buffer, buffer2
```

```
      call date(buffer)
      call time(buffer2)
      print*,'PARIMP run on ', buffer, ' at ', buffer2

c     !---DRIVER SECTION---
      n=6
c     !(number of equations)
      L=5
c     !(number of parameters)

c     READ IN pua's ETC
      call readstuff

c     ! (DATA)
      x(1)=1
      x(2)=2
      x(3)=3
      x(4)=4
      x(5)=5
      x(6)=6
c     READ MASS DATA
      open(unit=3, file='emt.dat', status='old')
      read(unit=3, fmt=90), title
      read(unit=3, fmt=93), mass_1_emt
      read(unit=3, fmt=93), mass_2_emt
      read(unit=3, fmt=93), mass_3_emt
      close(unit=3)
      open(unit=3, file='uct.dat', status='old')
      read(unit=3, fmt=90), title
      read(unit=3, fmt=93), mass_1_uct
      read(unit=3, fmt=93), mass_2_uct
      read(unit=3, fmt=93), mass_3_uct
      close(unit=3)
      open(unit=3, file='dsb.dat', status='old')
      read(unit=3, fmt=90), title
      read(unit=3, fmt=93), mass_1_dsb
      read(unit=3, fmt=93), mass_2_dsb
      read(unit=3, fmt=93), mass_3_dsb
      close(unit=3)
90    format (a)
93    format ( t20, e30.16)

      print*, 'uct masses:', mass_1_uct, mass_2_uct, mass_3_uct
      print*, 'dsb masses:', mass_1_dsb, mass_2_dsb, mass_3_dsb

cccc  y(1)=mass_2_emt/mass_1_emt
cccc  y(2)=mass_3_emt/mass_1_emt
      y(1)=1.0
      y(2)=2.0
      y(3)=mass_2_uct/mass_1_uct
      y(4)=mass_3_uct/mass_1_uct
```

```
      y(5)=mass_2_dsb/mass_1_dsb
      y(6)=mass_3_dsb/mass_1_dsb

      print*, 'Input data:'
      print*, '#       x              y'
      do 111, I=1, n
            print*, I, x(I), y(I)
111   continue

      e= 1e-6
      ee1= 0.9
c     SHOW THE OLD VALUES
      a(1)= fg1_initial_guess
      a(2)= fg2_initial_guess
      a(3)= fe1_initial_guess
      a(4)= fe2_initial_guess
      a(5)= fg3_initial_guess
      print*, 'The old values are...'
      print*, 'FG1= ', a(1), ' = ', a(1)*1.0d4, ' x10^-4'
      print*, 'FG2= ', a(2), ' = ', a(2)*1.0d4, ' x10^-4'
      print*, 'FE1= ', a(3), ' = ', a(3)*1.0d4, ' x10^-4'
      print*, 'FE2= ', a(4), ' = ', a(4)*1.0d4, ' x10^-4'
      print*, 'FG3= ', a(5), ' = ', a(5)*1.0d4, ' x10^-4'
      print*, 'With the old values...'
      print*, 'x                    Calculated',
     &        '                Desired mass ratio'
      do 224, I=1, n
            xx=x(I)
            call func(xx,yy)
            print*, x(I), yy, y(I)
224   continue
      print*, ' '

c     STARTING GUESSES
c     a(1)=   10.0 * fg1_initial_guess
c     a(2)=   10.0 * fg2_initial_guess
c     a(3)=   10.0 * fe1_initial_guess
c     a(4)=   10.0 * fe2_initial_guess
c     a(5)=   10.0 * fg3_initial_guess

      do 135, i=1, 5
            a(i)= -200.0 d-4
135   continue

c     (CHANGE ANY INITIAL GUESS OF ZERO TO SOMETHING ELSE SO PARAFIT WILL WORK)
      do 3, i=1, 5
            if (a(i).eq.0.0d0) a(i)= -1.0d-4
3     continue
      print*, 'The initial guesses are...'
      print*, (a(i), i=1,5)

      call parafit
```

```
      print*, 'Solving for only the 2 quark groups....'

      print*, ' '
      print*, 'The coefficients found by PARAFIT are:'
      print*, 'FG1= ', a(1), ' = ', a(1)*1.0d4, ' x10^-4'
      print*, 'FG2= ', a(2), ' = ', a(2)*1.0d4, ' x10^-4'
      print*, 'FE1= ', a(3), ' = ', a(3)*1.0d4, ' x10^-4'
      print*, 'FE2= ', a(4), ' = ', a(4)*1.0d4, ' x10^-4'
      print*, 'FG3= ', a(5), ' = ', a(5)*1.0d4, ' x10^-4'
      print*, ' '
      print*, 'The standard deviation of the fit is:', d
      print*, 'The number of iterations was: ', m
      print*, ' '
      print*, 'With these parameters...'
      print*, 'x                          Calculated',
     &            '          Desired mass ratio'
      do 124, I=1, n
              xx=x(I)
              call func(xx,yy)
              print*, x(I), yy, y(I)
124   continue

c     WRITE TO OUTPUT FILE
      open(unit=3,file='parimp.dat',status='new')
      write(unit=3, fmt=*), fs_title
      write(unit=3, fmt=80), 'fg1=', a(1)
      write(unit=3, fmt=80), 'fg2=', a(2)
      write(unit=3, fmt=80), 'fe1=', a(3)
      write(unit=3, fmt=80), 'fe2=', a(4)
      write(unit=3, fmt=80), 'fg3=', a(5)
      write(unit=3,fmt=*), 'Fs found by PARIMP on ',
     &            buffer, ' at ', buffer2
80    format (' ', a, t20, e30.16)
      endfile(unit=3)
      close(unit=3)

      end
c -------------------------------------------------------------
c -------------------------------------------------------------
c     !---THE PARAFIT SUBROUTINE---
      subroutine parafit
      implicit real*8(a-z)
      common L, n, x, y, e, e1, d, m, ee1
      common /params/ a
      integer n, L, I, mod_iterations, m
      dimension       x(6), y(6), a(5), e1(6)

      do 333, I=1, L
              e1(I)=ee1
333   continue
```

```
      m=0

      L1=1000000

      ! SWEEP
4257  do 129, I=1, L
            a0=a(I)
            a(I)=a0
c           (Why is that line there?)
c           ! MIDDLE
4261        call residuals(L2)
            m0=L2
            a(I)=a0*(1-e1(I))
            call residuals(L2)
            m1=L2

            if (m1.gt.m0) e1(I)= -e1(I)/2
            if (m1.lt.m0) e1(I)=1.2*e1(I)
            if (m1.gt.m0) a(I)=a0
            if (m1.gt.m0) goto 4261
129   continue

      m=m+1
      mod_iterations= mod(m, 500)
      if (mod_iterations.eq.0) print*,'# of iterations= ', m
      if (m.gt.10000) call exit(0)

      if (L2.eq.0) return
      if (abs((L1-L2)/L2).lt.e) return

      L1=L2
      goto 4257
c     !(goto Sweep)
      end

c -------------------------------------------------------------
c     ! RESIDUALS
      subroutine residuals(L2)
c     (Returns L2, and modifies d [in common block].)
      implicit real*8(a-z)
      common L, n, x, y, e, e1, d, m, ee1
      common /params/ a
      integer n, L, j, m
      dimension       x(6), y(6), a(5), e1(6)

      L2=0
      do 128,j=1, n
            xx=x(j)
            call func(xx, yy)
            L2=L2+(y(j)-yy)*(y(j)-yy)
128   continue
```

```
        if (n.gt.L) d=sqrt(L2/(n-L))
        if (n.le.L) d=sqrt(L2)
c       !(fudged. Not in original pgm.)
        return
        end

c -------------------------------------------------------------
c       READ IN THE PUA'S ETC
        subroutine readstuff
        implicit real*8(a-z)
        common /con_1_emt/ pua_1_emt, pub_1_emt,
     &                     pta_1_emt, ptb_1_emt, abg_1_emt, pextra_1_emt
        common /con_2_emt/ pua_2_emt, pub_2_emt,
     &                     pta_2_emt, ptb_2_emt, abg_2_emt, pextra_2_emt
        common /con_3_emt/ pua_3_emt, pub_3_emt,
     &                     pta_3_emt, ptb_3_emt, abg_3_emt, pextra_3_emt

        common /con_1_uct/ pua_1_uct, pub_1_uct,
     &                     pta_1_uct, ptb_1_uct, abg_1_uct, pextra_1_uct
        common /con_2_uct/ pua_2_uct, pub_2_uct,
     &                     pta_2_uct, ptb_2_uct, abg_2_uct, pextra_2_uct
        common /con_3_uct/ pua_3_uct, pub_3_uct,
     &                     pta_3_uct, ptb_3_uct, abg_3_uct, pextra_3_uct

        common /con_1_dsb/ pua_1_dsb, pub_1_dsb,
     &                     pta_1_dsb, ptb_1_dsb, abg_1_dsb, pextra_1_dsb
        common /con_2_dsb/ pua_2_dsb, pub_2_dsb,
     &                     pta_2_dsb, ptb_2_dsb, abg_2_dsb, pextra_2_dsb
        common /con_3_dsb/ pua_3_dsb, pub_3_dsb,
     &                     pta_3_dsb, ptb_3_dsb, abg_3_dsb, pextra_3_dsb

        common /fs/ fg1_initial_guess, fg2_initial_guess,
     &              fe1_initial_guess, fe2_initial_guess,
     &              fg3_initial_guess

        common /title/ fs_title
        character*80   fs_title
        character*100  character_junk

        save    /con_1_emt/, /con_2_emt/, /con_3_emt/
        save    /con_1_uct/, /con_2_uct/, /con_3_uct/
        save    /con_1_dsb/, /con_2_dsb/, /con_3_dsb/
        save    /title/

c       READ F'S
        open(unit=3, file='fconstants.dat', status='old')
        read(unit=3, fmt=90), fs_title
        read(unit=3,fmt=93), fg1_initial_guess
        read(unit=3,fmt=93), fg2_initial_guess
        read(unit=3,fmt=93), fe1_initial_guess
        read(unit=3,fmt=93), fe2_initial_guess
        read(unit=3,fmt=93), fg3_initial_guess
```

```
      close(unit=3)

c     READ EMT DATA
      open(unit=3, file='esc_emt.dat', access='sequential',
     &     form='formatted', status='old')
      read(unit=3, fmt=90), character_junk
      read(unit=3,fmt=93), junk
      read(unit=3,fmt=93), junk
      read(unit=3,fmt=93), junk
      read(unit=3,fmt=93), junk
      read(unit=3,fmt=93), junk

      read(unit=3,fmt=93), pua_1_emt
      read(unit=3,fmt=93), pub_1_emt
      read(unit=3,fmt=93), pta_1_emt
      read(unit=3,fmt=93), ptb_1_emt
      read(unit=3,fmt=93), pextra_1_emt
      read(unit=3,fmt=93), abg_1_emt
      read(unit=3,fmt=93), junk
      read(unit=3,fmt=93), junk

      read(unit=3,fmt=93), pua_2_emt
      read(unit=3,fmt=93), pub_2_emt
      read(unit=3,fmt=93), pta_2_emt
      read(unit=3,fmt=93), ptb_2_emt
      read(unit=3,fmt=93), pextra_2_emt
      read(unit=3,fmt=93), abg_2_emt
      read(unit=3,fmt=93), junk
      read(unit=3,fmt=93), junk

      read(unit=3,fmt=93), pua_3_emt
      read(unit=3,fmt=93), pub_3_emt
      read(unit=3,fmt=93), pta_3_emt
      read(unit=3,fmt=93), ptb_3_emt
      read(unit=3,fmt=93), pextra_3_emt
      read(unit=3,fmt=93), abg_3_emt
      read(unit=3,fmt=93), junk
      read(unit=3,fmt=93), junk

      close(unit=3)

      print*, 'EMT GROUP:'
      print*, '   pua, pub, pta, ptb, pextra, abg    ...'
      print 901, '1', pua_1_emt, pub_1_emt, pta_1_emt,
     &                ptb_1_emt, pextra_1_emt, abg_1_emt
      print 901, '2', pua_2_emt, pub_2_emt, pta_2_emt,
     &                ptb_2_emt, pextra_2_emt, abg_2_emt
      print 901, '3', pua_3_emt, pub_3_emt, pta_3_emt,
     &                ptb_3_emt, pextra_3_emt, abg_3_emt

901   format (' ', a1, ': ',
```

```
     &              6(f7.3, 1x) )

c       READ UCT DATA
        open(unit=3, file='esc_uct.dat', access='sequential',
     &               form='formatted', status='old')
        read(unit=3, fmt=90), character_junk
        read(unit=3,fmt=93), junk
        read(unit=3,fmt=93), junk
        read(unit=3,fmt=93), junk
        read(unit=3,fmt=93), junk
        read(unit=3,fmt=93), junk

        read(unit=3,fmt=93), pua_1_uct
        read(unit=3,fmt=93), pub_1_uct
        read(unit=3,fmt=93), pta_1_uct
        read(unit=3,fmt=93), ptb_1_uct
        read(unit=3,fmt=93), pextra_1_uct
        read(unit=3,fmt=93), abg_1_uct
        read(unit=3,fmt=93), junk
        read(unit=3,fmt=93), junk

        read(unit=3,fmt=93), pua_2_uct
        read(unit=3,fmt=93), pub_2_uct
        read(unit=3,fmt=93), pta_2_uct
        read(unit=3,fmt=93), ptb_2_uct
        read(unit=3,fmt=93), pextra_2_uct
        read(unit=3,fmt=93), abg_2_uct
        read(unit=3,fmt=93), junk
        read(unit=3,fmt=93), junk

        read(unit=3,fmt=93), pua_3_uct
        read(unit=3,fmt=93), pub_3_uct
        read(unit=3,fmt=93), pta_3_uct
        read(unit=3,fmt=93), ptb_3_uct
        read(unit=3,fmt=93), pextra_3_uct
        read(unit=3,fmt=93), abg_3_uct
        read(unit=3,fmt=93), junk
        read(unit=3,fmt=93), junk

        close(unit=3)

        print*, 'UCT GROUP:'
        print*, '   pua, pub, pta, ptb, pextra, abg   ...'
        print 901, '1', pua_1_uct, pub_1_uct, pta_1_uct,
     &                  ptb_1_uct, pextra_1_uct, abg_1_uct
        print 901, '2', pua_2_uct, pub_2_uct, pta_2_uct,
     &                  ptb_2_uct, pextra_2_uct, abg_2_uct
        print 901, '3', pua_3_uct, pub_3_uct, pta_3_uct,
     &                  ptb_3_uct, pextra_3_uct, abg_3_uct

c       READ DSB DATA
        open(unit=3, file='esc_dsb.dat', access='sequential',
```

```
     &               form='formatted', status='old')
          read(unit=3, fmt=90), character_junk
          read(unit=3,fmt=93), junk
          read(unit=3,fmt=93), junk
          read(unit=3,fmt=93), junk
          read(unit=3,fmt=93), junk
          read(unit=3,fmt=93), junk

          read(unit=3,fmt=93), pua_1_dsb
          read(unit=3,fmt=93), pub_1_dsb
          read(unit=3,fmt=93), pta_1_dsb
          read(unit=3,fmt=93), ptb_1_dsb
          read(unit=3,fmt=93), pextra_1_dsb
          read(unit=3,fmt=93), abg_1_dsb
          read(unit=3,fmt=93), junk
          read(unit=3,fmt=93), junk

          read(unit=3,fmt=93), pua_2_dsb
          read(unit=3,fmt=93), pub_2_dsb
          read(unit=3,fmt=93), pta_2_dsb
          read(unit=3,fmt=93), ptb_2_dsb
          read(unit=3,fmt=93), pextra_2_dsb
          read(unit=3,fmt=93), abg_2_dsb
          read(unit=3,fmt=93), junk
          read(unit=3,fmt=93), junk

          read(unit=3,fmt=93), pua_3_dsb
          read(unit=3,fmt=93), pub_3_dsb
          read(unit=3,fmt=93), pta_3_dsb
          read(unit=3,fmt=93), ptb_3_dsb
          read(unit=3,fmt=93), pextra_3_dsb
          read(unit=3,fmt=93), abg_3_dsb
          read(unit=3,fmt=93), junk
          read(unit=3,fmt=93), junk

          close(unit=3)

          print*, 'DSB GROUP:'
          print*, '   pua, pub, pta, ptb, pextra, abg    ...'
          print 901, '1', pua_1_dsb, pub_1_dsb, pta_1_dsb,
     &                    ptb_1_dsb, pextra_1_dsb, abg_1_dsb
          print 901, '2', pua_2_dsb, pub_2_dsb, pta_2_dsb,
     &                    ptb_2_dsb, pextra_2_dsb, abg_2_dsb
          print 901, '3', pua_3_dsb, pub_3_dsb, pta_3_dsb,
     &                    ptb_3_dsb, pextra_3_dsb, abg_3_dsb

c         ----------------------------
90        format (a)
93        format ( t20, e30.16)
c         ----------------------------

          return
```

```
      end
c -------------------------------------------------------------
c -------------------------------------------------------------
c     ! ---THE FUNCTION IN QUESTION---
      subroutine func(xx,yy)
      implicit real*8(a-z)
      common /params/ a

      common /con_1_emt/ pua_1_emt, pub_1_emt,
     &              pta_1_emt, ptb_1_emt, abg_1_emt, pextra_1_emt
      common /con_2_emt/ pua_2_emt, pub_2_emt,
     &              pta_2_emt, ptb_2_emt, abg_2_emt, pextra_2_emt
      common /con_3_emt/ pua_3_emt, pub_3_emt,
     &              pta_3_emt, ptb_3_emt, abg_3_emt, pextra_3_emt

      common /con_1_uct/ pua_1_uct, pub_1_uct,
     &              pta_1_uct, ptb_1_uct, abg_1_uct, pextra_1_uct
      common /con_2_uct/ pua_2_uct, pub_2_uct,
     &              pta_2_uct, ptb_2_uct, abg_2_uct, pextra_2_uct
      common /con_3_uct/ pua_3_uct, pub_3_uct,
     &              pta_3_uct, ptb_3_uct, abg_3_uct, pextra_3_uct

      common /con_1_dsb/ pua_1_dsb, pub_1_dsb,
     &              pta_1_dsb, ptb_1_dsb, abg_1_dsb, pextra_1_dsb
      common /con_2_dsb/ pua_2_dsb, pub_2_dsb,
     &              pta_2_dsb, ptb_2_dsb, abg_2_dsb, pextra_2_dsb
      common /con_3_dsb/ pua_3_dsb, pub_3_dsb,
     &              pta_3_dsb, ptb_3_dsb, abg_3_dsb, pextra_3_dsb


      real*8          a(5), xx, yy

      save    /con_1_emt/, /con_2_emt/, /con_3_emt/
      save    /con_1_uct/, /con_2_uct/, /con_3_uct/
      save    /con_1_dsb/, /con_2_dsb/, /con_3_dsb/

      fg1=a(1)
      fg2=a(2)
      fe1=a(3)
      fe2=a(4)
      fg3=a(5)


c     CALCULATE
      w1_emt=       (fg1/fe1)*pua_1_emt + (fg2/fe2)*pub_1_emt
     &              + fg1*pta_1_emt + fg2*ptb_1_emt - abg_1_emt
     &              + (fg3/fe1)*pextra_1_emt
      w2_emt=       (fg1/fe1)*pua_2_emt + (fg2/fe2)*pub_2_emt
     &              + fg1*pta_2_emt + fg2*ptb_2_emt - abg_2_emt
     &              + (fg3/fe1)*pextra_2_emt
      w3_emt=       (fg1/fe1)*pua_3_emt + (fg2/fe2)*pub_3_emt
     &              + fg1*pta_3_emt + fg2*ptb_3_emt - abg_3_emt
```

```
     &          + (fg3/fe1)*pextra_3_emt

          w1_uct=    (fg1/fe1)*pua_1_uct + (fg2/fe2)*pub_1_uct
     &               + fg1*pta_1_uct + fg2*ptb_1_uct - abg_1_uct
     &               + (fg3/fe1)*pextra_1_uct
          w2_uct=    (fg1/fe1)*pua_2_uct + (fg2/fe2)*pub_2_uct
     &               + fg1*pta_2_uct + fg2*ptb_2_uct - abg_2_uct
     &               + (fg3/fe1)*pextra_2_uct
          w3_uct=    (fg1/fe1)*pua_3_uct + (fg2/fe2)*pub_3_uct
     &               + fg1*pta_3_uct + fg2*ptb_3_uct - abg_3_uct
     &               + (fg3/fe1)*pextra_3_uct

          w1_dsb=    (fg1/fe1)*pua_1_dsb + (fg2/fe2)*pub_1_dsb
     &               + fg1*pta_1_dsb + fg2*ptb_1_dsb - abg_1_dsb
     &               + (fg3/fe1)*pextra_1_dsb
          w2_dsb=    (fg1/fe1)*pua_2_dsb + (fg2/fe2)*pub_2_dsb
     &               + fg1*pta_2_dsb + fg2*ptb_2_dsb - abg_2_dsb
     &               + (fg3/fe1)*pextra_2_dsb
          w3_dsb=    (fg1/fe1)*pua_3_dsb + (fg2/fe2)*pub_3_dsb
     &               + fg1*pta_3_dsb + fg2*ptb_3_dsb - abg_3_dsb
     &               + (fg3/fe1)*pextra_3_dsb


c         FUDGE ... IF W'S ARE ZERO OR NEGATIVE, MAKE THEM VERY SMALL BUT POSITIVE
ccc       if (w1_emt.le.0) w1_emt=0.0001*exp(w1_emt)
ccc       if (w2_emt.le.0) w2_emt=0.0001*exp(w2_emt)
ccc       if (w3_emt.le.0) w3_emt=0.0001*exp(w3_emt)
ccc       if (w1_uct.le.0) w1_uct=0.0001*exp(w1_uct)
ccc       if (w2_uct.le.0) w2_uct=0.0001*exp(w2_uct)
ccc       if (w3_uct.le.0) w3_uct=0.0001*exp(w3_uct)
ccc       if (w1_dsb.le.0) w1_dsb=0.0001*exp(w1_dsb)
ccc       if (w2_dsb.le.0) w2_dsb=0.0001*exp(w2_dsb)
ccc       if (w3_dsb.le.0) w3_dsb=0.0001*exp(w3_dsb)
c         TRY A DIFFERENT METHOD
          if (w1_emt.le.0) then
                  w1_emt=0.00001
                  w2_emt=9.0d4 *exp(abs(w1_emt))
                  w3_emt=9.0d4 *exp(abs(w1_emt))
          endif
          if (w1_uct.le.0) then
                  w1_uct=0.00001
                  w2_uct=9.0d4 *exp(abs(w1_uct))
                  w3_uct=9.0d4 *exp(abs(w1_uct))
          endif
          if (w1_dsb.le.0) then
                  w1_dsb=0.00001
                  w2_dsb=9.0d4 *exp(abs(w1_dsb))
                  w3_dsb=9.0d4 *exp(abs(w1_dsb))
          endif


c         RETURN VALUES
```

```
cccc    if (xx.eq.1) yy=(w2_emt/w1_emt)
cccc    if (xx.eq.2) yy=(w3_emt/w1_emt)
        if (xx.eq.1) yy=1.0
        if (xx.eq.2) yy=2.0
        if (xx.eq.3) yy=(w2_uct/w1_uct)
        if (xx.eq.4) yy=(w3_uct/w1_uct)
        if (xx.eq.5) yy=(w2_dsb/w1_dsb)
        if (xx.eq.6) yy=(w3_dsb/w1_dsb)
        if (xx.ne.1 .and. xx.ne.2 .and. xx.ne.3 .and. xx.ne.4
     &              .and. xx.ne.5 .and. xx.ne.6) call exit(0)

        return
c -------------------------------------------------------------
c -------------------------------------------------------------
        end
```

6) REVISE_FS

```
c ------------------------------------------------------------------------
c ------------------------------------------------------------------------
c       REVISE_FS
c       =========
c
c       Takes data from PARIMP.DAT
c       and writes a revised FCONSTANTS.DAT
c
c       Written:
c       1989/Jan/19     -by Philip Sharman.
c       Revised:
c       1989/Feb/20     -FG3 term added.
c ------------------------------------------------------------------------
c ------------------------------------------------------------------------
        implicit real*8 (a-z)
        character*80    title, comment
        character*12    buffer, buffer2

        call date(buffer)
        call time(buffer2)
        print*,'REVISE_FS run on ', buffer, ' at ', buffer2

c       READ DATA
        open(unit=3, file='parimp.dat', status='old')
        read(unit=3, fmt=90), title
        read(unit=3,fmt=93), fg1
        read(unit=3,fmt=93), fg2
        read(unit=3,fmt=93), fe1
        read(unit=3,fmt=93), fe2
        read(unit=3,fmt=93), fg3
        read(unit=3,fmt=90), comment
        close(unit=3)

c       WRITE DATA
        open(unit=3, file='fconstants.dat', status='new')
        write(unit=3, fmt=*), title
        write(unit=3, fmt=92), 'FG1=', fg1
        write(unit=3, fmt=92), 'FG2=', fg2
        write(unit=3, fmt=92), 'FE1=', fe1
        write(unit=3, fmt=92), 'FE2=', fe2
        write(unit=3, fmt=92), 'FG3=', fg3
        write(unit=3, fmt=*),  comment
        endfile(unit=3)
        close(unit=3)

        print*, 'PARIMP.DAT has been transferred to FCONSTANTS.DAT.'

c       --------------------
90      format (a)
92      format (' ', a, t20, e30.16)
```

```
93      format (t20, e30.16)
c       --------------------
        end
```

7) PAR_ALPHA

```
c -------------------------------------------------------------------
c -------------------------------------------------------------------
c       PAR_ALPHA
c       =========
c       This is "Parafit" modified to try to fit a parabola
c       to alpha versus A data.
c       y=A, x=alpha, y=a*x^2 + b*x +c
c
c       (See Basic Scientific Subroutines, Vol 2, page 80)
c
c       (Alter X_DESIRED to be the desired alpha.)
c
c       Written:
c       1989/May/30     -by Philip Sharman.
c                       -Modified from PARIMP.FOR.
c                       (Variable names etc cleaned up a lot.)
c       Revised:
c       1989/Jun/15     -Modified to handle variable number of data pairs.
c            Jun/16     -quick plot added.
c            Jun/17     -min/max added.
c                       -INT() bug fixed.
c            Jun/18     -PLOT subroutine added.
c            Jun/19     -initial guesses changed to put the parabola
c                       through the first two data points.
c -------------------------------------------------------------------
c -------------------------------------------------------------------

c       GLOBAL VARIABLES
        real*8          x(99), y(99), a(99), standard_deviation
        integer         n_parameters, n_data_pairs, n_iterations

c       LOCAL VARIABLES
        integer         i
        real*8          xx, yy
        character*100   array(99)
        integer         ncols, nrows, row, col
        real*8          b_col, b_row, m_col, m_row, xmin, xmax, ymin, ymax,
     &                       x_desired, y_desired

c       GREETINGS
        print*,'Welcome to Par_Alpha'
        print*,'===================='

c       DRIVER SECTION
        print*, 'How many data pairs are there?'
        read*, n_data_pairs
        print*, 'n_data_pairs= ', n_data_pairs
        do i= 1, n_data_pairs
```

```
            print 50, 'A(', i, ')?'
            read*, y(i)
            print*, y(i)
            print 50, 'alpha(', i, ')?'
            read*, x(i)
            print*, x(i)
        end do
50      format (' ', a, i2, a)

        n_parameters= 3

c       DATA
c       (STATE 2)
c       n_data_pairs= 3
c       x(1)= -3.27457
c       x(2)= -3.20235
c       x(3)= -3.21508
c       y(1)= 6.07
c       y(2)= 5.89
c       y(3)= 5.9
c       (STATE 3)
c       n_data_pairs= 2
c       x(1)= 4.83885
c       x(2)= 4.83511
c       y(1)= 12.71303
c       y(2)= 12.70

        print*, 'Input data:'
        print*, '  #                x                    y'
        do i= 1, n_data_pairs
            print*, i, x(i), y(i)
        end do
        print*, ' '

c       STARTING GUESSES (THESE PUT THE PARABOLA THROUGH THE FIRST TWO
C       DATA POINTS)
        a(1)=   (y(1)-y(2))/(x(1)**2 - x(2)**2)
        a(2)=   0.001           !(PARAFIT WON'T ACCEPT 0)
        a(3)=   (y(1)*x(2)**2 - y(2)*x(1)**2)/(x(2)**2 - x(1)**2)

        print*, 'The initial guesses are...'
        print*, (a(i), i= 1,3)
        print*, 'With these parameters...'
        print*, 'x                       Calculated',
     &  '       Desired '
        do i= 1, n_data_pairs
            xx= x(i)
            call func(xx, yy, a)
            print*, x(i), yy, y(i)
        end do

        call parafit(x, y, a, n_data_pairs, n_parameters,
```

```
     &      standard_deviation, n_iterations)

            print*, ' '
            print*, 'The coefficients found by PARAFIT are:'
            print*, 'a= ', a(1)
            print*, 'b= ', a(2)
            print*, 'c= ', a(3)
            print*, 'y= a*x^2 + b*x + c'
            print*, ' '
            print*, 'The standard deviation of the fit is:', standard_deviation
            print*, 'The number of iterations was: ', n_iterations
            print*, ' '
            print*, 'With these parameters...'
            print*, 'x                          Calculated',
     &                    '              Desired '
            do i= 1, n_data_pairs
                    xx= x(i)
                    call func(xx, yy, a)
                    print*, x(i), yy, y(i)
            end do

            print*, 'The vertex is:'
            print*, 'x= ', -a(2)/(2*a(1))
            print*, 'y= ', (4*a(1)*a(3) - a(2)*a(2))/(4*a(1))

c           WHAT WOULD GIVE DESIRED XX?
            x_desired= +1.612885
ccc         x_desired= -3.22577055
            print*, 'To get alpha= ', x_desired
            call func(x_desired, y_desired, a)
            print*, '...choose A= ', y_desired

c           DO A QUICK PLOT
c           INITIALIZE
            xmax= x_desired
            xmin= x_desired
            ymax= y_desired
            ymin= y_desired
            do i= 1, n_data_pairs-1
                    xmax= max(x(i), x(i+1), xmax )
                    ymax= max(y(i), y(i+1), ymax )
                    xmin= min(x(i), x(i+1), xmin )
                    ymin= min(y(i), y(i+1), ymin )
            end do
            print*, 'X goes from ', xmin, ' to ', xmax
            print*, 'Y goes from ', ymin, ' to ', ymax
            ncols=  78
            nrows=  19
            m_col=       (ncols-1)/(xmax-xmin)
            b_col=       1 - m_col*xmin
            m_row=       (nrows-1)/(ymax-ymin)
            b_row=       1 - m_row*ymin
```

```
      do row=1, nrows
            array(row)= '.....'
     &           //           '.....'
      end do
c     PLOT PARABOLA
      do col=1, ncols
            xx=     (col - b_col)/m_col
            call func(xx, yy, a)
            row=    int(m_row*yy + b_row)
            if (col.ge.1 .and. col.le.ncols .and. row.ge.1 .and. row.le.nrows) then
                  call plot('+', row, col, array)
            endif
      end do

c     PLOT X_DESIRED LINE
      do row= 1, nrows
            col=    int(m_col*x_desired + b_col)
            if (col.ge.1 .and. col.le.ncols .and. row.ge.1 .and. row.le.nrows) then
                  call plot('|', row, col, array)
            endif
      end do

c     PLOT O AT X,Y_DESIRED
      col=    int(m_col*x_desired + b_col)
      row=    int(m_row*y_desired + b_row)
      call plot('O', row, col, array)

c     PLOT DATA POINTS
      do i=1, n_data_pairs
            col=    int(m_col*x(i) + b_col)
            row=    int(m_row*y(i) + b_row)
            if (col.ge.1 .and. col.le.ncols .and. row.ge.1 .and. row.le.nrows) then
                  call plot(char(i+48), row, col, array)
            else
                  print*, 'Data point out of range: ', x(i), y(i)
                  print*, 'row= ', row, ' col= ', col
            endif
      end do

c     SHOW IT
      do row= nrows, 1, -1
            print*, array(row)(1:ncols)
      end do

      end
c -------------------------------------------------------------
c -------------------------------------------------------------
c     THE PARAFIT SUBROUTINE
      subroutine parafit(x, y, a, n_data_pairs, n_parameters,
     &      standard_deviation, n_iterations)
```

```
c       INPUT VARIABLES
        real*8          x(*), y(*)
        integer         n_data_pairs, n_parameters

c       IN/OUT VARIABLES
        real*8          a(*)

c       OUTPUT VARIABLES
        real*8          standard_deviation
        integer         n_iterations

c       LOCAL VARIABLES
        integer         i, mod_iterations
        real*8          e1(99), max_variance, convergence_factor, a0, m0, m1,
     &                       l1, xx, yy, residual

c       SOME OF RUCKDESCHEL'S VARIABLE NAMES HAVE BEEN CHANGED:
C       E       -> MAX_VARIANCE
C       E1      -> CONVERGENCE_FACTOR
C       D       -> STANDARD DEVIATION
C       M       -> N_ITERATIONS
C       N       -> N_DATA_PAIRS
C       L       -> N_PARAMETERS
C       L2      -> RESIDUAL

c       MAX VARIANCE DETERMINES WHEN TO STOP
        max_variance= 1.0d-6
c       CONVERGENCE_FACTOR SHOULD BE BETWEEN 0 AND 1.  (1 IS
C       FASTEST BUT IT MAY NOT BE STABLE)
        convergence_factor= 0.1

        do i= 1, n_parameters
           e1(i)= convergence_factor
        end do

        n_iterations= 0
        l1= 1000000
c       SWEEP
4257    do i= 1, n_parameters
           a0= a(i)
           a(i)= a0
c          (WHY IS THAT LINE THERE?)
c          MIDDLE
4261       call residuals(x, y, a, n_data_pairs, n_parameters,
     &                 residual, standard_deviation)
           m0= residual
           a(i)= a0*(1-e1(i))
           call residuals(x, y, a, n_data_pairs, n_parameters,
     &                 residual, standard_deviation)
           m1= residual

           if (m1.gt.m0) e1(i)= -e1(i)/2
```

```
                if (m1.lt.m0) e1(i)= 1.2*e1(i)
                if (m1.gt.m0) a(i)= a0
                if (m1.gt.m0) goto 4261          !(GOTO MIDDLE)
        end do

        n_iterations= n_iterations + 1
        mod_iterations= mod(n_iterations, 500)
        if (mod_iterations.eq.0) print*,'# of iterations= ', n_iterations
        if (n_iterations.gt.90000) call exit(0)

        if (residual.eq.0) return
        if (abs((l1-residual)/residual).lt.max_variance) return

        l1= residual
        goto 4257                    !(GOTO SWEEP)

        end
c -------------------------------------------------------------
c -------------------------------------------------------------
c       RESIDUALS
        subroutine residuals(x, y, a, n_data_pairs, n_parameters,
     &      residual, standard_deviation)

c       INPUT VARIABLES
        integer         n_data_pairs, n_parameters
        real*8          x(*), y(*), a(*)

c       OUTPUT VARIABLES
        real*8          residual, standard_deviation

c       LOCAL VARIABLES
        integer         j
        real*8          xx, yy


c       COMPUTE
        residual= 0
        do j= 1, n_data_pairs
                xx= x(j)
                call func(xx, yy, a)
                residual= residual+(y(j)-yy)*(y(j)-yy)
        end do

        if (n_data_pairs.gt.n_parameters) standard_deviation=
     &              sqrt(residual/(n_data_pairs - n_parameters))
        if (n_data_pairs.le.n_parameters) standard_deviation= sqrt(residual)
c       (ABOVE LINE NOT IN ORIGINAL PGM.)

        return
        end

c -------------------------------------------------------------
```

```
c ------------------------------------------------------------
c       THE FUNCTION IN QUESTION
        subroutine func(xx, yy, a)

c       INPUT VARIABLES
        real*8          xx, a(*)
c       OUTPUT VARIABLE
        real*8          yy

        yy= a(1)*xx*xx + a(2)*xx + a(3)

        return
        end
c ------------------------------------------------------------
c ------------------------------------------------------------
c       PLOT
        subroutine plot(symbol, row, col, array)
        character*1     symbol
        character*100   array(*)
        integer         row, col, nrows, ncols

        if (col.eq.1) then
              array(row)= symbol // array(row)(col+1: )
        else
              array(row)= array(row)( :col-1) // symbol // array(row)(col+1: )
        endif

        return
        end
c ------------------------------------------------------------
c ------------------------------------------------------------
```

8) TWEAK

```
c ------------------------------------------------------------
c ------------------------------------------------------------
c       TWEAK
c       =====
c
c       TWEAK is an itelligent parameter-tweaker.  It assumes there
c       are some variables ,x(1:n_x), and some functions of these variables,
c       y(1:n_y)
c       and some desired values, desired(1:n_y), of these functions.
c       (No assumption is made as to whether the number of x's is
c       equal to the number of functions.)
c       It tries to tweak the variables to give the desired values, but
c       instead of just doing each variable in turn
c       (as PARAFIT does for example) it also assumes there is some other
c       criterion for judging what tweakings are better than others.
c       (This might be 'don't move the variables far from where they are
c       now', or it might be 'keep the variables as small as possible', or
c       'keep the variables as close to certain values as possible.')
c       So the basic strategy is this: for each variable, and each desired
c       output, work out what change would be required of each variable for
c       each output.  Then see what change has the least penalty value, do
c       that change, then the next lowest penalty value, and so on.
c       In this way, it is hoped, the variables will converge to some sort
c       of stable solution (and hopefully also one with low undesirability)
c       although it is not guaranteed that this will hapen.
c
c       Presently this is set up to try fit the two quark groups.
c
c       Reads:          ESC_UCT.DAT, ESC_DSB.DAT, FCONSTANTS.DAT,
c                       UCT.DAT, DSB.DAT.
c       Writes:         (nothing)
c
c       Written:        1989/June/7     -by Philip Sharman.
c       Revised:        (none)
c-------------------------------------------------------------------------------
c-------------------------------------------------------------------------------

c       GLOBAL VARIABLES
        common /global/ debug, n_x, n_y, x, y, desired
        logical         debug
        integer         n_x, n_y
        real*8          x(5), y(4), desired(4)

c       LOCAL VARIABLES
        integer         n_loops, k, maxloops, col, best_trial
        logical         keep_searching
        real*8          sum_error, criterion, minimum_error
        character*9     buffer1, buffer2
        parameter       (maxloops=100)
        save            best_trial, minimum_error
```

```
c       GLOBAL PARAMETERS
        debug=          .true.
        n_x=        5
        n_y=        4

c       WELCOME TO TWEAK
        call date(buffer1)
        call time(buffer2)
        print*, '     ************************************'
        print*, '     TWEAK running on ', buffer1, ' at ', buffer2
        print*, '     ************************************'

c       READ F'S AND SET Y() ARRAY
        call read_stuff
c       SHOW STUFF
        print*, 'The values with the old F''s are...'
        call evaluate_y(0)   !(SHOW W'S)
        call find_actual_ys
        print*, 'y(1)=', y(1)
        print*, 'y(2)=', y(2)
        print*, 'y(3)=', y(3)
        print*, 'y(4)=', y(4)
        call show_fs

c       NOW SEARCH
        keep_searching= .true.
        n_loops= 1
        minimum_error= 9.9d30
c       MAIN LOOP
        do while (keep_searching .and. n_loops.le.maxloops)
                print*, ' '
                call get_matrix
                call show_table
                call evaluate_y(0)
                call change_lowest_penalty
c               ARE WE DONE?
                sum_error= 0.0d0
                do col=1, n_y
                        sum_error= sum_error + abs(desired(col)-y(col))
                end do
                print*, 'Sum of errors is now= ', sum_error
                if (sum_error .lt. minimum_error) then
                        minimum_error= sum_error
                        best_trial= n_loops
                endif
                if (best_trial.gt.0) print*, '(Current best= ',
     &                  minimum_error, 'at trial', best_trial, ')'
                print '(a, i3)', ' Trial: ', n_loops
                criterion= 1.0
                if (sum_error.lt.criterion) then
                        print*, 'Errors in y''s are acceptable. '
```

```
                print*, 'We are successful.'
                keep_searching= .false.
            endif
            n_loops= n_loops + 1
        end do

666     print*, 'Program is done.'
        end
c-------------------------------------------------------------------------------
c-------------------------------------------------------------------------------
        subroutine get_matrix
c       WORK OUT MATRIX
c       MATRIX(r,c) SHOWS HOW X(R) VARIES IF Y(C) IS MADE PERFECT
c       MATRIX(r,0) HOLDS ORIGINAL UNCHANGED X'S

c       GLOBAL VARIABLES
        common /global/ debug, n_x, n_y, x, y, desired
        logical         debug
        integer         n_x, n_y
        real*8          x(5), y(4), desired(4)

c       OTHER COMMON VARIABLES
        common /mat/    matrix
        real*8          matrix(5,0:4)

c       LOCAL VARIABLES
        integer         row, col
        real*8          x_old

        do row= 1, n_x
            x_old= x(row)
            matrix(row, 0)= x_old
            do col= 1, n_y
c               if (debug) print*,'Finding matrix ', row, col
                call solve_x(row, col)
                matrix(row, col)= x(row)
c               (NOW RESTORE ORIGINAL VALUE)
                x(row)= x_old
            end do
        end do


        return
        end
c-------------------------------------------------------------------------------
c-------------------------------------------------------------------------------
        subroutine change_lowest_penalty
c       FINDS ENTRY IN MATRIX WITH LOWEST PENALTY-FUNCTION
c       CHANGES THAT X AND Y

c       GLOBAL VARIABLES
        common /global/ debug, n_x, n_y, x, y, desired
```

```
      logical         debug
      integer         n_x, n_y
      real*8          x(5), y(4), desired(4)

c     OTHER COMMON VARIABLES
      common /mat/    matrix
      real*8          matrix(5,0:4)

c     LOCAL VARIABLES
      integer         row, col, min_row, min_col, max_col, col1, col2
      real*8          minimum, pen, zero, x_new, x_old, penalty(5,4),
     &                     maximum, size, b, d, e, f
      parameter       (zero= 0.0d0)
      save            col1, col2

c     BASE PENALTY ON ABSOLUTE MAGNITUDE OF THE VARIABLES.
      do row= 1, n_x
            do col= 1, n_y
                  x_new= matrix(row, col)
                  size= abs(x_new)
                  b= 1000.0
                  d= 1.0
c     (WOULD REALLY LIKE D ABOUT 1.0d-4)
                  e= 100.0
                  f= 10.0
c     ***ABOVE VARIABLES NOT USED AT PRESENT***
                  penalty(row,col)= size
            end do
      end do

c     PRINT PENALTY MATRIX
ccc   print*, 'Penalty matrix:'
ccc   print*, '                                    (1)          ',
ccc   &       '      (2)          (3)          (4) '
ccc   do row= 1, n_x
ccc         print 150, row, (penalty(row, col), col=1, n_y)
ccc   end do
150   format (' ', 'x(', i1, ')', t21, 4(':',e14.3), ':')


c     CHANGE THE COLUMN WHICH HAS THE least non-zero ERROR, BUT
c     AVOID LAST TWO COLUMNS DONE (COL1 AND COL2)
c     (INITIALIZE IF THIS IS THE FIRST TIME THROUGH)
      if (col1.lt.1 .and. col1.gt.n_y) col1= 0
      if (col2.lt.1 .and. col2.gt.n_y) col2= 0
205   print*, 'Will not do columns: ', col1, col2
      minimum= +9.9d30
      do col= 1, n_y
            pen= abs(desired(col) - y(col))
            if (pen.ne.zero .and.
```

```
     &                pen.lt.minimum .and. col.ne.col1 .and. col.ne.col2) then
                         minimum= pen
                         min_col= col
                 endif
         end do
         col= min_col
         print*, 'The closest column is', col, ' so change that.'


c        ALTERNATIVES:
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c        CHANGE THE COLUMN WHICH HAS THE WORST ERROR, BUT
c        AVOID LAST TWO COLUMNS DONE (COL1 AND COL2)
c        (INITIALIZE IF THIS IS THE FIRST TIME THROUGH)
ccc      if (col1.lt.1 .and. col1.gt.n_y) col1= 0
ccc      if (col2.lt.1 .and. col2.gt.n_y) col2= 0
ccc205   print*, 'Will not do columns: ', col1, col2
ccc      maximum= -9.9d30
ccc      do col= 1, n_y
ccc              pen= abs(desired(col) - y(col))
ccc      if (pen.gt.maximum .and. col.ne.col1 .and. col.ne.col2) then
ccc                      maximum= pen
ccc                      max_col= col
ccc              endif
ccc      end do
ccc      col= max_col
ccc      print*, 'The worst column is', col, ' so change that.'
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c        (INITIALIZE COL_LAST_TIME IF THIS IS THE FIRST TIME THROUGH)
ccc      if (col_last_time.lt.1 .and. col_last_time.gt.n_y) col_last_time= 0
c        STEP THROUGH POSSIBILITIES
c        DO EACH COLUMN IN TURN
ccc      col= col_last_time + 1
ccc      if (col.gt.n_y) col= 1
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

c        FIND LOWEST PENALTY ROW IN COLUMN COL
         minimum= 9.9e30
         do row= 1, n_x
                 pen= penalty(row,col)
                 if (pen.lt.minimum) then
                         minimum= pen
                         min_row= row
                 endif
         end do
c        IF THE MINIMUM PENALTY IS REALLY BAD THEN GO BACK AND DO ANY COLUMN
         if (minimum .gt. 1.0) then
                 if (col1.ne.0 .and. col2.ne.0) then
```

```
                print*, 'Minimum= ', minimum, ' so go back and try again.'
                col2= 0
                goto 205        !(BACK UP)
            else if (col1.ne.0 .and. col2.eq.0) then
c               WE MAY HAVE ALREADY BACKED UP ONCE
                print*, 'Minimum= ', minimum, ' so go back and try again.'
                col1= 0         !(NOW BOTH COL1 AND COL2 = 0)
                goto 205        !(BACK UP)
            elseif (col1.eq.0 .and. col2.eq.0 .and. minimum.gt.1.0) then
c               WE ARE IN TROUBLE NOW
                print*, 'We have no choice.'
            else
                print*, '????', col1, col2
                call exit(0)
            endif
        endif

        print*, 'Minimum penalty= ', minimum
        print '(a, i1, a, i1, a)',
     &          ' Changing x(', min_row, ') to improve y(', col, ')'
        call solve_x(min_row, col)
        col2= col1
        col1= col

        return
        end
c-------------------------------------------------------------------------------
c-------------------------------------------------------------------------------
        subroutine show_table
c       DISPLAY STUFF (AND SET Y() TO ACTUAL VALUES AT END)

c       GLOBAL VARIABLES
        common /global/ debug, n_x, n_y, x, y, desired
        logical         debug
        integer         n_x, n_y
        real*8          x(5), y(4), desired(4)

c       OTHER COMMON VARIABLES
        common /mat/    matrix
        real*8          matrix(5,0:4)

c       LOCAL VARIABLES
        integer         row, col

c       PRINT MATRIX
        print*, ' '
        print*, '          no change          (1)       ',
     &          '    (2)            (3)            (4) '
        do row= 1, n_x
                print 103, row, (matrix(row, col), col=0, n_y)
        end do
        print 105, 'Desired y''s->', (desired(col), col= 1, n_y)
```

```
c       NOW SET Y() TO ACTUAL VALUES
        call find_actual_ys
        print 105, 'Actual y''s-> ', (y(col), col= 1, n_y)
        print 106, 'Difference-> ', ((desired(col)-y(col)), col= 1, n_y)
103     format (' ', 'x(', i1, ')', 1(':', f14.3), 4(':',e14.3), ':')
105     format (' ', a13, t21, 4(':', e14.3), ':')
106     format (' ', a13, t21, 4(':', f14.3), ':')

        call show_fs

        return
        end
c-------------------------------------------------------------------------------
c-------------------------------------------------------------------------------
        subroutine read_stuff
c       READ IN THE PUA'S ETC.  ALSO WORK OUT DESIRED() ARRAY.
c       AND SET INITIAL x() ARRAY.

c       GLOBAL VARIABLES
        common /global/ debug, n_x, n_y, x, y, desired
        logical         debug
        integer         n_x, n_y
        real*8          x(5), y(4), desired(4)

c       COMMON VARIABLES
        common /con_1_uct/ pua_1_uct, pub_1_uct,
     &                  pta_1_uct, ptb_1_uct, abg_1_uct, pextra_1_uct
        common /con_2_uct/ pua_2_uct, pub_2_uct,
     &                  pta_2_uct, ptb_2_uct, abg_2_uct, pextra_2_uct
        common /con_3_uct/ pua_3_uct, pub_3_uct,
     &                  pta_3_uct, ptb_3_uct, abg_3_uct, pextra_3_uct
        common /con_1_dsb/ pua_1_dsb, pub_1_dsb,
     &                  pta_1_dsb, ptb_1_dsb, abg_1_dsb, pextra_1_dsb
        common /con_2_dsb/ pua_2_dsb, pub_2_dsb,
     &                  pta_2_dsb, ptb_2_dsb, abg_2_dsb, pextra_2_dsb
        common /con_3_dsb/ pua_3_dsb, pub_3_dsb,
     &                  pta_3_dsb, ptb_3_dsb, abg_3_dsb, pextra_3_dsb
        save    /con_1_uct/, /con_2_uct/, /con_3_uct/
        save    /con_1_dsb/, /con_2_dsb/, /con_3_dsb/
        real*8  pua_1_uct, pub_1_uct,
     &                  pta_1_uct, ptb_1_uct, abg_1_uct, pextra_1_uct,
     &                  pua_2_uct, pub_2_uct,
     &                  pta_2_uct, ptb_2_uct, abg_2_uct, pextra_2_uct,
     &                  pua_3_uct, pub_3_uct,
     &                  pta_3_uct, ptb_3_uct, abg_3_uct, pextra_3_uct,
     &                  pua_1_dsb, pub_1_dsb,
     &                  pta_1_dsb, ptb_1_dsb, abg_1_dsb, pextra_1_dsb,
     &                  pua_2_dsb, pub_2_dsb,
     &                  pta_2_dsb, ptb_2_dsb, abg_2_dsb, pextra_2_dsb,
     &                  pua_3_dsb, pub_3_dsb,
     &                  pta_3_dsb, ptb_3_dsb, abg_3_dsb, pextra_3_dsb
```

```
c       LOCAL VARIABLES
        character*80    fs_title
        character*100   character_junk, title
        real*8          junk, mass_1_uct, mass_2_uct, mass_3_uct,
     &                        mass_1_dsb, mass_2_dsb, mass_3_dsb,
     &                        fg1, fg2, fe1, fe2, fg3
        integer         i

c       READ F'S
        print*, '(Reading FCONSTANTS.DAT)'
        open(unit=3, file='fconstants.dat', status='old')
        read(unit=3, fmt=90), fs_title
        read(unit=3,fmt=93), fg1
        read(unit=3,fmt=93), fg2
        read(unit=3,fmt=93), fe1
        read(unit=3,fmt=93), fe2
        read(unit=3,fmt=93), fg3
        close(unit=3)

        print*, 'fg1=', fg1
        print*, 'fg2=', fg2
        print*, 'fe1=', fe1
        print*, 'fe2=', fe2
        print*, 'fg3=', fg3

901     format (' ', a1, ': ',
     &          6(f7.3, 1x) )

c       READ UCT DATA
        print*, ' '
        open(unit=3, file='esc_uct.dat', status='old')
        read(unit=3, fmt=90), title
        print*, 'Reading ESC_UCT.DAT: ', title
        read(unit=3,fmt=93), junk
        read(unit=3,fmt=93), junk
        read(unit=3,fmt=93), junk
        read(unit=3,fmt=93), junk
        read(unit=3,fmt=93), junk

        read(unit=3,fmt=93), pua_1_uct
        read(unit=3,fmt=93), pub_1_uct
        read(unit=3,fmt=93), pta_1_uct
        read(unit=3,fmt=93), ptb_1_uct
        read(unit=3,fmt=93), pextra_1_uct
        read(unit=3,fmt=93), abg_1_uct
        read(unit=3,fmt=93), junk
        read(unit=3,fmt=93), junk

        read(unit=3,fmt=93), pua_2_uct
        read(unit=3,fmt=93), pub_2_uct
```

```
      read(unit=3,fmt=93), pta_2_uct
      read(unit=3,fmt=93), ptb_2_uct
      read(unit=3,fmt=93), pextra_2_uct
      read(unit=3,fmt=93), abg_2_uct
      read(unit=3,fmt=93), junk
      read(unit=3,fmt=93), junk

      read(unit=3,fmt=93), pua_3_uct
      read(unit=3,fmt=93), pub_3_uct
      read(unit=3,fmt=93), pta_3_uct
      read(unit=3,fmt=93), ptb_3_uct
      read(unit=3,fmt=93), pextra_3_uct
      read(unit=3,fmt=93), abg_3_uct
      read(unit=3,fmt=93), junk
      read(unit=3,fmt=93), junk

      close(unit=3)

      print*, 'UCT GROUP:'
      print*, '   pua, pub, pta, ptb, pextra, abg    ...'
      print 901, '1', pua_1_uct, pub_1_uct, pta_1_uct,
     &                ptb_1_uct, pextra_1_uct, abg_1_uct
      print 901, '2', pua_2_uct, pub_2_uct, pta_2_uct,
     &                ptb_2_uct, pextra_2_uct, abg_2_uct
      print 901, '3', pua_3_uct, pub_3_uct, pta_3_uct,
     &                ptb_3_uct, pextra_3_uct, abg_3_uct

c     READ DSB DATA
      print*, ' '
      open(unit=3, file='esc_dsb.dat', status='old')
      read(unit=3, fmt=90), title
      print*, 'Reading ESC_DSB.DAT: ', title
      read(unit=3,fmt=93), junk
      read(unit=3,fmt=93), junk
      read(unit=3,fmt=93), junk
      read(unit=3,fmt=93), junk
      read(unit=3,fmt=93), junk

      read(unit=3,fmt=93), pua_1_dsb
      read(unit=3,fmt=93), pub_1_dsb
      read(unit=3,fmt=93), pta_1_dsb
      read(unit=3,fmt=93), ptb_1_dsb
      read(unit=3,fmt=93), pextra_1_dsb
      read(unit=3,fmt=93), abg_1_dsb
      read(unit=3,fmt=93), junk
      read(unit=3,fmt=93), junk

      read(unit=3,fmt=93), pua_2_dsb
      read(unit=3,fmt=93), pub_2_dsb
      read(unit=3,fmt=93), pta_2_dsb
      read(unit=3,fmt=93), ptb_2_dsb
      read(unit=3,fmt=93), pextra_2_dsb
```

```
      read(unit=3,fmt=93), abg_2_dsb
      read(unit=3,fmt=93), junk
      read(unit=3,fmt=93), junk

      read(unit=3,fmt=93), pua_3_dsb
      read(unit=3,fmt=93), pub_3_dsb
      read(unit=3,fmt=93), pta_3_dsb
      read(unit=3,fmt=93), ptb_3_dsb
      read(unit=3,fmt=93), pextra_3_dsb
      read(unit=3,fmt=93), abg_3_dsb
      read(unit=3,fmt=93), junk
      read(unit=3,fmt=93), junk

      close(unit=3)

      print*, 'DSB GROUP:'
      print*, '   pua, pub, pta, ptb, pextra, abg    ...'
      print 901, '1', pua_1_dsb, pub_1_dsb, pta_1_dsb,
     &                    ptb_1_dsb, pextra_1_dsb, abg_1_dsb
      print 901, '2', pua_2_dsb, pub_2_dsb, pta_2_dsb,
     &                    ptb_2_dsb, pextra_2_dsb, abg_2_dsb
      print 901, '3', pua_3_dsb, pub_3_dsb, pta_3_dsb,
     &                    ptb_3_dsb, pextra_3_dsb, abg_3_dsb

c     ------------------------------
90    format (a)
93    format ( t20, e30.16)
c     ------------------------------

c     SET X() ARRAY
      x(1)= fg1
      x(2)= fg2
      x(3)= fe1
      x(4)= fe2
      x(5)= fg3

c     READ MASS DATA
      print*, '(Reading UCT.DAT)'
      open(unit=3, file='uct.dat', status='old')
           read(unit=3, fmt=90), title
           read(unit=3, fmt=93), mass_1_uct
           read(unit=3, fmt=93), mass_2_uct
           read(unit=3, fmt=93), mass_3_uct
      close(unit=3)
      print*, '(Reading DSB.DAT)'
      open(unit=3, file='dsb.dat', status='old')
           read(unit=3, fmt=90), title
           read(unit=3, fmt=93), mass_1_dsb
           read(unit=3, fmt=93), mass_2_dsb
           read(unit=3, fmt=93), mass_3_dsb
      close(unit=3)
```

```
c       WORK OUT DESIRED VALUES
        desired(1)=     0.0d0
        desired(2)=     mass_2_uct/mass_1_uct
        desired(3)=     mass_2_dsb/mass_1_dsb
        desired(4)=     mass_3_dsb/mass_1_dsb

        print*, 'Desired values:'
        print*, 'desired(1)=', desired(1), '(IS UP POSITIVE?)'
        print*, 'desired(2)=', desired(2), '(CHARM/UP)'
        print*, 'desired(3)=', desired(3), '(STRANGE/DOWN)'
        print*, 'desired(3)=', desired(3), '(BOTTOM/DOWN)'

        print*, 'Variables:'
        do i=1, n_x
                print '(a, i1, a, e25.15)',  ' x(', i, ')', x(i)
        end do


        return
        end
c-------------------------------------------------------------------------------
c-------------------------------------------------------------------------------
        subroutine solve_x(m, n)
c       ADJUSTS X(m) TO GIVE DESIRED Y(n)
c       (ALTERS Y(n) BUT RESTORES IT TO ORIGINAL CONDITION AT THE END)
c       IF IT FAILS IT SETS X(m)=1D6

c       GLOBAL VARIABLES
        common /global/ debug, n_x, n_y, x, y, desired
        logical         debug
        integer         n_x, n_y
        real*8          x(5), y(4), desired(4)

c       INPUT VARIABLES
        integer         m, n

c       LOCAL VARIABLES
        integer         n_moves, maxmoves
        parameter       (maxmoves=100)
        real*8          x_original, delta_x, y_original, error,
     &                  error_last_time, two, criterion
        parameter       (two=2.0d0, criterion=1.0d-4)
        logical         keep_moving


c       SET UP
        n_moves= 1
        keep_moving= .true.
        x_original= x(m)
        y_original= y(n)
        delta_x= x_original
        if (delta_x.eq.0.0d0) delta_x= 1.0d-4
```

```
c       START BY MOVING
        x(m)= x_original + delta_x
        call evaluate_y(n)
        error_last_time= desired(n) - y(n)


        do while(keep_moving .and. n_moves.le.maxmoves)
c               NOW MOVE AGAIN IN SAME DIRECTION
                x(m)= x(m) + delta_x
                call evaluate_y(n)
                error= desired(n) - y(n)
c               DID IT GET BETTER OR WORSE?
                if (abs(error).gt.abs(error_last_time)) then
c                       WORSE
c                       HALVE DELTA AND CHANGE DIRECTION
                        delta_x= -(delta_x/two)
c                       if (debug) then
c                               print*, 'worse'
c                               print*, 'x(m)= ', x(m), ' y(n)= ', y(n),
c     &                                 'error=', error
c                               print*, 'delta_x=', delta_x
c                       endif
                else
c                       BETTER
c                       KEEP GOING
                        delta_x= 1.2 * delta_x
c                       if (debug) then
c                               print*, 'better'
c                               print*, 'x(m)=', x(m), 'y(n)=', y(n),
c     &                                 'error=', error
c                               print*, 'delta_x=', delta_x
c                       endif
                endif

                error_last_time= error
                n_moves= n_moves + 1
c               TEST FOR ACCEPTABLE CONVERGENCE
                if (abs(error).lt.criterion) keep_moving= .false.
c               ABORT IF WE ARE WANDERING OFF INTO INFINITY
                if (abs(x(m)).gt.1.0d6) then
                        keep_moving= .false.
c                       print*, 'SOLVE_X not converging. x(m)= ', x(m)
c                       print*, 'Number of iterations= ', n_moves
                        y(n)= y_original
                        return
                endif
c               ABORT IF WE HAVE GONE ON TOO LONG
                if (n_moves.gt.maxmoves) then
                        keep_moving= .false.
c                       print*, 'SOLVE_X failed.'
c                       print*, 'Number of iterations= ', n_moves
```

```
                    x(m)= 1.0 d6
                    y(n)= y_original
c                   (WE REALLY JUST WANT TO PUT A LARGE VALUE IN THE
c                   PENALTY MATRIX ... DO IT DIRECTLY?)
c                   print*, '(Setting x(m)=1.0d6)'
                    return
                endif
        end do

c       FINISH UP - WE DID OKAY
c       if (debug) print*, 'SOLVE_X is done. X(', m, ') is now: ', x(m)
c       if (debug) print*, 'Number of iterations= ', n_moves

c       (PUT ORIGINAL Y VALUE BACK INTO PLACE)
        y(n)= y_original

        return
        end
c-------------------------------------------------------------------------------
c-------------------------------------------------------------------------------
        subroutine evaluate_y(n)
c       FINDS Y(n) (AND CHANGES IT)
c       THIS IS WHERE THE ACTUAL EQUATIONS GO
c       (IF n=0 IT JUST SHOWS THE w's)

c       GLOBAL VARIABLES
        common /global/ debug, n_x, n_y, x, y, desired
        logical         debug
        integer         n_x, n_y
        real*8          x(5), y(4), desired(4)

c       OTHER COMMON VARIABLES (OBTAINED BY READ_STUFF)
        common /con_1_uct/ pua_1_uct, pub_1_uct,
     &                  pta_1_uct, ptb_1_uct, abg_1_uct, pextra_1_uct
        common /con_2_uct/ pua_2_uct, pub_2_uct,
     &                  pta_2_uct, ptb_2_uct, abg_2_uct, pextra_2_uct
        common /con_3_uct/ pua_3_uct, pub_3_uct,
     &                  pta_3_uct, ptb_3_uct, abg_3_uct, pextra_3_uct
        common /con_1_dsb/ pua_1_dsb, pub_1_dsb,
     &                  pta_1_dsb, ptb_1_dsb, abg_1_dsb, pextra_1_dsb
        common /con_2_dsb/ pua_2_dsb, pub_2_dsb,
     &                  pta_2_dsb, ptb_2_dsb, abg_2_dsb, pextra_2_dsb
        common /con_3_dsb/ pua_3_dsb, pub_3_dsb,
     &                  pta_3_dsb, ptb_3_dsb, abg_3_dsb, pextra_3_dsb
        save    /con_1_uct/, /con_2_uct/, /con_3_uct/
        save    /con_1_dsb/, /con_2_dsb/, /con_3_dsb/
        real*8  pua_1_uct, pub_1_uct,
     &                  pta_1_uct, ptb_1_uct, abg_1_uct, pextra_1_uct,
     &                  pua_2_uct, pub_2_uct,
     &                  pta_2_uct, ptb_2_uct, abg_2_uct, pextra_2_uct,
     &                  pua_3_uct, pub_3_uct,
     &                  pta_3_uct, ptb_3_uct, abg_3_uct, pextra_3_uct,
```

```
     &     pua_1_dsb, pub_1_dsb,
     &     pta_1_dsb, ptb_1_dsb, abg_1_dsb, pextra_1_dsb,
     &     pua_2_dsb, pub_2_dsb,
     &     pta_2_dsb, ptb_2_dsb, abg_2_dsb, pextra_2_dsb,
     &     pua_3_dsb, pub_3_dsb,
     &     pta_3_dsb, ptb_3_dsb, abg_3_dsb, pextra_3_dsb

c    INPUT VARIABLES
     integer   n

c    LOCAL VARIABLES
     real*8          zero, w1_uct, w2_uct, w3_uct,
     &                w1_dsb, w2_dsb, w3_dsb,
     &                fg1, fg2, fe1, fe2, fg3
     parameter       (zero=0.0d0)

c    THE NITTY GRITTY
     fg1= x(1)
     fg2= x(2)
     fe1= x(3)
     fe2= x(4)
     fg3= x(5)
     if (fe1.ne.zero .and. fe2.ne.zero) then
          w1_uct=      (fg1/fe1)*pua_1_uct + (fg2/fe2)*pub_1_uct
     &          + fg1*pta_1_uct + fg2*ptb_1_uct - abg_1_uct
     &          + (fg3/fe1)*pextra_1_uct
          w2_uct=      (fg1/fe1)*pua_2_uct + (fg2/fe2)*pub_2_uct
     &          + fg1*pta_2_uct + fg2*ptb_2_uct - abg_2_uct
     &          + (fg3/fe1)*pextra_2_uct
          w3_uct=      (fg1/fe1)*pua_3_uct + (fg2/fe2)*pub_3_uct
     &          + fg1*pta_3_uct + fg2*ptb_3_uct - abg_3_uct
     &          + (fg3/fe1)*pextra_3_uct

          w1_dsb=      (fg1/fe1)*pua_1_dsb + (fg2/fe2)*pub_1_dsb
     &          + fg1*pta_1_dsb + fg2*ptb_1_dsb - abg_1_dsb
     &          + (fg3/fe1)*pextra_1_dsb
          w2_dsb=      (fg1/fe1)*pua_2_dsb + (fg2/fe2)*pub_2_dsb
     &          + fg1*pta_2_dsb + fg2*ptb_2_dsb - abg_2_dsb
     &          + (fg3/fe1)*pextra_2_dsb
          w3_dsb=      (fg1/fe1)*pua_3_dsb + (fg2/fe2)*pub_3_dsb
     &          + fg1*pta_3_dsb + fg2*ptb_3_dsb - abg_3_dsb
     &          + (fg3/fe1)*pextra_3_dsb
     else
          print*, 'PROBLEM in EVALUATE.  (Division by zero) FE1= ',
     &         fe1, ' FE2= ', fe2
c         THEN CARRY ON?  STOP? RETURN? FORCE THEM NON-ZERO?
          call exit(0)
     endif


c    JUST SHOW W'S IF N=0
```

```
      if (n.eq.0)then
            print*, 'w1_uct= ', w1_uct
            print*, 'w2_uct= ', w2_uct
            print*, 'w3_uct= ', w3_uct
            print*, 'w1_dsb= ', w1_dsb
            print*, 'w2_dsb= ', w2_dsb
            print*, 'w3_dsb= ', w3_dsb
c     OTHERWISE UPDATE Y(n)
      else if (n.eq.1) then
c           IS UP MASS POSITIVE?
            if (w1_uct.lt.zero) then
                  y(1)= abs(w1_uct)
            else if (w1_uct.eq.zero) then
                  y(1)= 9.9d6
            else
                  y(1)= zero
            endif
      else if (n.eq.2) then
            y(2)= w2_uct/w1_uct        !(CHARM/UP)
      else if (n.eq.3) then
            y(3)= w2_dsb/w1_dsb        !(STRANGE/DOWN)
      else if (n.eq.4) then
            y(4)= w3_dsb/w1_dsb        !(BOTTOM/DOWN)
      else
            print*, 'ERROR: n= ', n, 'n_x= ', n_x
            call exit(0)
      endif

      return
      end
c-------------------------------------------------------------------------------
c-------------------------------------------------------------------------------
      subroutine find_actual_ys
c     UPDATE ALL THE Y() ARRAY

c     GLOBAL VARIABLES
      common /global/ debug, n_x, n_y, x, y, desired
      logical         debug
      integer         n_x, n_y
      real*8          x(5), y(4), desired(4)

c     LOCAL VARIABLE
      integer col

c     FIND THEM
      do col= 1, n_y
            call evaluate_y(col)
      end do

      return
      end
```

```
c-------------------------------------------------------------------------------
c-------------------------------------------------------------------------------
      subroutine show_fs
c     SHOW THE FG'S AND FE'S ETC.

c     GLOBAL VARIABLES
      common /global/ debug, n_x, n_y, x, y, desired
      logical         debug
      integer         n_x, n_y
      real*8          x(5), y(4), desired(4)

c     LOCAL VARIABLES
      real*8          fg1, fg2, fe1, fe2, fg3, eq(6)
      integer         i

c     WORK THEM OUT
      fg1=    x(1)
      fg2=    x(2)
      fe1=    x(3)
      fe2=    x(4)
      fg3=    x(5)

c     ALSO WORK OUT THE QUANTITIES APPEARING IN THE FIELD EQUATIONS
      eq(1)=  fg1
      eq(2)=  fg2
      eq(3)=  fe1
      eq(4)=  fg3/fg1
      eq(5)=  fe2
      eq(6)=  fg3*fe2/(fg2*fe1)

c     PRINT THEM
      print*, '       fg1          fg2          fe1          fe2          fg3'
      print 755, fg1, fg2, fe1, fe2, fg3
      print 855, fg1, fg2, fe1, fe2, fg3

ccc   print*,
ccc   &    '       fg1          fg2',
ccc   &    '       fe1    (fg3/fg1)        fe2 (fg3*fe2)/(fg2*fe1)'
ccc   print 760, (eq(i), i=1, 6)
ccc   print 860, (eq(i), i=1, 6)

755   format  (' ', 5(f12.7) )
855   format  (' ', 5(e12.4) )
760   format  (' ', 6(f12.7) )
860   format  (' ', 6(e12.4) )

      return
      end
c-------------------------------------------------------------------------------
c-------------------------------------------------------------------------------
```

* * *

VITA

Surname: Sharman. Given names: Philip Howard.
Place of birth: London, England.
Date of birth: 1956/October/18.

Educational institutions attended:

Ryerson Polytechnical Institute, Toronto,	1975 - 1978,
University of Calgary, Calgary,	1982 - 1986,
University of Victoria, Victoria,	1987 - 1989.

Degrees awarded:

Bachelor of Applied Arts, Ryerson Polytechnical Institute, 1978,
Bachelor of Science (Honours), University of Calgary, 1986.

Honors:

All-Canada Award (Ryerson),
Andrew Kirkor Memorial Bursary (Calgary),
NSERC Undergraduate Student Research Award (Calgary).

Publications:

"A Non-Linear Gauge-Invariant Field Theory Of Elementary Particles", P. Sharman and F. I. Cooperstock, in press.

Cosmic Ray Intensity Records, D. Venkatesan, T. Mathews, H. Grauman, and P. Sharman, (University of Calgary: 1989).

"The Trifair Cipher: The Trifair extends the Playfair into a ternary cipher system," *The Cryptogram*, March-April 1983.

"Trifair Variations and Extensions," *The Cryptogram*, May-June 1983.

PARTIAL COPYRIGHT LICENSE

I hereby grant the right to lend my thesis entitled "Numerical Solutions of a Non-Linear Classical Field Theory of Leptons and Quarks" to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the library of any other university, or similar institution, on its behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Author: [redacted]

Philip Howard Sharman.

Date: 1990/Feb./6