Parameter Identification by Continuation Methods

C. Martin *, J. Miller and K. Pearce

Abstract

A method is discussed for calculating a dirichlet polynomial with positive coefficients. The method employs continuation techniques used for finding the zeros of a mapping in n-dimensional euclidean space $\mathbb{R}^n$.

1 Introduction

Suppose one were given a sample consisting of a number of unknown radioactive materials for which it is required not only to detect what materials are present but also how much of each. One might go about solving this problem in the following way. First, one recalls that each radioactive substance decays at a rate proportional to an exponential term of the form $e^{-\lambda t}$, where λ is a term peculiar to the given substance. So that if, say, c grams of such a substance were present in the given sample, then it is probable that between t and $t + dt$, $ce^{-\lambda t}$ particles would decay. Thus if $c_1, c_2, \cdots, c_n$ grams of n substances are in the sample, with $\lambda_1, \cdots, \lambda_n$ their respective decay factors, then it is probable between t and $t + dt$ that $c_1 e^{-\lambda_1 t} + \cdots + c_n e^{-\lambda_n t}$ particles would decay.

Next, in $2n$ equally spaced intervals $t = \Delta t, 2\Delta t, \cdots 2n\Delta t$ one observes the values $p_1, p_2, \cdots, p_{2n}$ of the expression $c_1 e^{-\lambda_1 t} + \cdots + c_n e^{-\lambda_n t}$. This gives rise to the following $2n$-equations in $2n$-unknowns:

$$\begin{aligned} c_1 e^{-\lambda_1 \Delta t} + \cdots + c_n e^{-\lambda_n \Delta t} &= p_1 \\ c_1 e^{-2n\lambda_1 \Delta t} + \cdots + e^{-2n\lambda_n \Delta t} &= p_{2n} \end{aligned} \tag{1.1}$$

*Supported in part by grants from NASA and NSA.

which is then to be solved.

Otherwise put, one is to solve for the root of the function $F : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ where the coordinate functions of F are

$$\begin{aligned} &f_1(c_1, \cdots, c_n, x_1, \cdots, x_n) = c_1x_1 + c_2x_2 + \cdots + c_nx_n - p_1 \\ &\vdots \\ &f_{2n}(c_1, \cdots, c_n, x_1, \cdots, x_n) = c_1x_1^{2n} + c_2x_2^{2n} + \cdots + c_nx_n^{2n} - p_{2n}, \end{aligned} \tag{1.2}$$

where $x_1 = e^{-\lambda_1 \Delta t}, \cdots, x_n = e^{-\lambda_n \Delta t}$ and the c_i's are all positive. The solution of this system of equations is the solution of the problem.

In practice we would, of course, use some least square approximation technique, for example the algorithm developed in [1], [2], or [4]. However we feel that it is important to point out that given small amounts of data "exact" solutions to this interpolation problem are possible and the continuation methods for this can be used in this class of exponential interpolation problems. We also feel that it is important to understand the deterministic problem in order to understand the stochastic problem.

2 Preliminaries

In this section we give some basic definitions and the method which we will use in finding the zeros of the nonlinear function is explained. We will exploit certain continuation techniques in this paper.

First, all mapping $F : \mathbb{R}^n \to \mathbb{R}^n$ are to be at least continously differentiable, C^1. Thus $JF(x)$, the Jacobian of f at x, is a continuous real valued function. If $y(t) = (y_1(t), \cdots, y_n(t)), 0 \leq t \leq 1$, is a path in $\mathbb{R}^n$, it is said to be lifted by the mapping f to a covering path $x(t) = (x_1(t), \cdots, x_n(t))$ if $y(t) = F(x(t))$ for all $0 \leq t \leq 1$. The following theorem gives conditions which enable us to state when the paths of the form $(1-t)F(a)$ for some a in an open subset W of $\mathbb{R}^n$ may be lifted by F to a covering path which lies in W. It is Theorem 2 of [3].

Theorem 2.1 [3] *Let C be a bounded convex domain of $\mathbb{R}^n$ and $F : C \to \mathbb{R}^n$ be C^1. If $JF(x) \neq 0$ for x in C, then for each zero x^* of F in C there is an open subset W of C such that for each a in W the path $(1-t)f(a)$ may be lifted by F to a covering path $x(t)$ in W such that $x^* = x(1)$.*

The open subset W may be described as follows: Let ∂C be the boundary of C and U the nonempty component of $\mathbb{R}^n - F(\partial C)$ which contains the origin. Define $St(0)$, the star of 0, as those points y in U for which $(1-t)y$, $0 \leq t \leq 1$, lies in U. W is the connected component of $F^{-1}(St(0))$ which contains the zero x^*.

In practice we do not find a covering path, rather we observe that a covering path $x(t)$ is a trajectory of the differential equation

$$F'(x)\frac{dx}{dt} = -F(a) \tag{2.1}$$

with initial condition $x(0) = a$. We then numerically follow this solution to $t = 1$.

It is of interest to note that at no point does our computation take us into regions where the variables become negative if we start with positive variables. This is important because it is well known that exponential equations of the sort defined by (1.1) become ill-conditioned and intractable when there are mixed weights. (See Wiscombe and Evans [1] or Evans, et. al. [4] for this point.) Our experience is that even simple examples in which solutions are selected with some c_i negative become numerically impossible.

It is clear that in order to use this method, knowledge of the zeros of $JF(x)$ is necessary. To this end we calculate the Jacobian of the system in equation (1.1).

Proposition 2.1 *Let* $F : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ *have as its coordinate functions the system of equations given in (1.2). Then* $JF(c_1, \cdots, c_n, x_1, \cdots, x_n)$ *is* $\pm c_1 c_2 \cdots c_n x_1^2 \cdots x_n^2 (x_2 - x_1)^4 \cdots (x_j - x_i)^4 \cdots (x_n - x_{n-1})^4$, $j > i$.

Proof: The matrix representation for F' is the $2n \times 2n$ matrix

$$F' = \begin{bmatrix} x_1 & \cdots & x_n & c_1 & \cdots & c_n \\ x_1^2 & \cdots & x_n^2 & 2c_1x_1 & \cdots & 2c_nx_n \\ \vdots & & & & & \\ x^{2n} & \cdots & x_n^n & nc_1x_1^{2n-1} & \cdots & nc_nx_n^{2n-1} \end{bmatrix} \tag{2.2}$$

To evaluate the determinant of this matrix we factor out $c_1, c_2, \cdots, c_n$ from columns $n+1$ to $2n$ respectively. Then we note that what remains is

a symmetric function in the variables $x_1, x_2, \cdots, x_n$ which we denote by $S(x_1, \cdots, x_n)$. To evaluate the determinant, with the c's removed, we just need to look at the 1st column, nth, $n+1st$ and $2n$th columns and then make use of the symmetry of the determinant.

First we factor x_1 and x_n from the 1st and the nth columns and then subtract the first column from the $(n+1)$'st and the n'th column from the $2n$th. Factoring out again an x_1 and x_n out of the resulting $(n+1)$'st and $2n$'th columns there results the following expression:

$$S(x_1, \cdots, x_n) = \\ x_1^2 x_n^2 \begin{vmatrix} 1 & \cdots & 1 & 0 & \cdots & 0 \\ x_1 & \cdots & x_n & 1 & \cdots & 1 \\ x_1^2 & \cdots & x_n^2 & 2x_1 & & 2x_n \\ \vdots & & & & & \\ x_1^{2n-1} & \cdots & x_n^{2n-1} & (2n-1)x_1^{2n-2} & \cdots & (2n-1)x_n^{2n-2} \end{vmatrix} \tag{2.3}$$

Next, in the above determinant in which only the 1st, nth, $(n+1)$st and $2n$th columns are presented, subtract the first column from the nth and the $(n+1)$st column from the $2n$th and then factor out $x_n - x_1$ from column n and $2n$ to get:

$$S(x_1, \cdots, x_n) = \\ x_1^2 x_n^2 (x_n - x_1)^2 \begin{vmatrix} \cdots & & 0 \\ \cdots & 1 & 1 \\ \cdots & \dfrac{x_n^2 - x_1^2}{x_n - x_1} & 2x_1 \\ & \vdots & \vdots \\ \cdots & \dfrac{x_n^{2n-1} - x_1^{2n-1}}{x_n - x_1} & (2n-1)x_1^{2n-2} \end{vmatrix} \tag{2.4}$$

in which only the nth and $(n+1)$st column are presented. Observe that if $x_n = x_1$ then these two columns are identical. Hence there is yet another factor of $x_n - x_1$ present.

This means we may write $S(x_1, \cdots, x_n)$ as $S(x_1, \cdots, x_n) = x_1^2 x_n^2 (x_n - x_1)^3 Q(x_1, \cdots, x_n)$, where $Q(x_1, \cdots, x_n)$ is a polynomial. In the same fashion, by working with the columns headed by x_1 and x_2, say, we can write

$S(x_1, \cdots, x_n)$ as $S(x_1, \cdots, x_n) = x_1^2 x_2^2 (x_2 - x_1)^3 Q'(x_1, \cdots, x_n)$, where again $Q'(x_1, \cdots, x_n)$ is a polynomial. But this clearly shows by divisibility properties and induction that $S(x_1, \cdots, x_n)$ may be written as

$$S(x_1, \cdots, x_n) = x_1^2 x_2^2 \cdots x_n^2 (x_2 - x_1)^3 \cdots (x_j - x_i)^3 \cdots (x_n - x_{n-1})^3 A(x_1, \cdots, x_n),$$

where $j > i$ and $A(x_1, \cdots, x_n)$ is an alternating polynomial in its arguments. This last point is clear since $S(x_1, \cdots, x_n)$ is symmetric. But $A(x_1, \cdots, x_n)$ an alternating form implies there are yet again factors of the form $x_j - x_i$, for all $i, j, j > i$. Finally by simply noting the degree of the arguments it follows that $S(x_1, \cdots, x_n) = \pm x_1^2 \cdots x_n^2 (x_2 - x_1)^4 \cdots (x_n - x_{n-1})^4$ and the proposition follows.

3 Computing Experience

In this section we look at the computational experience we have had in formulating a method for solving polynomial systems of nonlinear equations of the type (1.2). There is a variety of general methods in the literature for finding the roots of systems of polynomial equations [11], [12], [10]. Our overall experience has been that we have not been able to apply such methods to (1.2) because of the deficiency of (1.2) as a system of polynomial equations.

For a polynomial system of equations the deficiency of the system is defined as the difference between the degree of the system and the number of roots of the system. In general, a system of n polynomial equations

$$\begin{aligned} f_1(y_1, \cdots, y_n) &= 0 \\ f_2(y_1, \cdots, y_n) &= 0 \\ &\cdots \\ f_n(y_1, \cdots, y_n) &= 0 \end{aligned}$$

will have $d_1 d_2 \cdots d_n$ roots, where d_k is the degree of f_k, $1 \leq k \leq n$. The degree of (1.2) is thus $(2n+1)!$, where n is state dimension of the problem, i.e., the number of independent factors x_j. The construction and symmetry of (1.2), however, show that all of the roots are permutations, in their components, of a fundamental root whose x_j components are ordered, i.e., $x_1 < x_2 < \cdots x_n$. (The terms x_j in (1.2) are uniquely associated and

determined by the factors x_j in the system.) Thus, there are $n!$ roots for (1.2).

Li, Sauer and Yorke in [5] and Morgan and Sommese in [5,6] describe general (homotopy) methods for finding all of the roots for any polynomial system of equations. While those homotopy methods can be applied to (1.2), the deficiency of (1.2) makes it inadvisable, because a large percentage of the paths generated by those homotopies diverge (in C^{2n}) to infinity.

We considered several different approaches of varying complexity and sophistication for finding the roots of (1.2):

(i) applying both direct and modified (quasi trust region) forms of Newton's method;

(ii) using a package nonlinear solver (NS01A [MJDP], written by M. J. D. Powell) which is a hybrid crossing of the method of steepest descent with Newton's method;

(iii) using an IMSL package nonlinear solver (DNEQNF [IMSL], which was based on a MINPACK subroutine HYBRD1 [MP]);

(iv) using a package differential equation solver (ODE, described in [S]) to solve the differential equation (2.1) for the homotopy trajectory described in Section 2;

(v) implementing a predictor-corrector path-following procedure to trace the homotopy covering path described in Section 2.

In each case we contrasted the efficiency of the method against standard sets of 1000 randomly generated problems at a test state dimension of $n = 3$ and $n = 4$. Also, using the homotopy methods we examined a broader range of problems where the state dimension was not restricted as above. For each of the problems we tested, we used a standardized entry point $\mathbf{A}$ with components $c_j = 1$ and $x_j = j/(n+1)$, $1 \leq j \leq n$.

There is a secondary problem which needs to be addressed before we look at a summary of our computational experience. That problem is that in general the state dimension, n, of the system (1.2) is not known. If, indeed, the system (1.2) arises from sampling data as suggested by the

prototype in Section 1, the state dimension of the problem is not *apriori* discernable.

We employed the following strategy for determining the state dimension of the problem:

1. We start with a postulate for the initial state dimension, say n_i, and then solve (1.2) for that initial state dimension using a standardized entry point. The system (1.2) involves $2n_i$ observations of actual state values.

2. Upon finding the solution of (1.2), we then compute predicted values and, also, measure actual values for the next n_i observations. If the L_2 norm of the difference between the predicted values and the actual values is sufficiently small, we presume that we have identified the state dimension of the problem (as n_i) and, also, the solution of the problem as well.

3. Otherwise, we increment n_i and cycle back to solving (1.2) with an augmented state dimension. However, for this second pass (and all subsequent passes) we have *apriori* information for constructing the entry point - namely, information in the form of the solution, which was just generated, for the lower-dimensional problem.

We note that there is an intrinsic problem with the assumption in (2). Suppose that the actual state dimension for a given problem is N and that two (or more) of the actual solution components, say x_j and x_{j+1} are numerically close to each other. The above procedure may call for termination at a test state dimension of $N-1$ (or lower), because the computed values for the function $\mathbf{F}$ in (1.2) at the test state dimension $N-1$ may numerically agree with the actual values up to the specified error tolerance, i.e., we may have a test component x_j which approximates both x_j and x_{j+1} in such a way that

$$c_j(x_j)^k \sim c_j x_j^k + c_j + 1^x j + 1^k, \ 1 \leq k \leq 3(N-1).$$

For all practical purposes the solution at the test state dimension of $N-1$ is the solution for the measured data. However, theoretically, we have not found the exact solution.

In the discussion which follows we will use $\mathbf{C}$ to denote the (n-dimensional) vector of constants c_j, $\mathbf{X}$ the (n-dimensional) vector of factors x_j and $\mathbf{Y}$ the ($2n$-dimensional) vector whose first n components are the components of $\mathbf{C}$ and whose second n components are the components of $\mathbf{X}$. We will use $\mathbf{SC}$, $\mathbf{SX}$ and $\mathbf{SY}$ to denote the analogs of $\mathbf{C}$, $\mathbf{X}$ and $\mathbf{Y}$, respectively, which represent the exact solution for (1.2). Also, we will let $\mathbf{G}$ denote the complement in $\mathbf{R}^{2n}$ of the singular set on which the Jacobian vanishes.

We note that the initial state of the vector $\mathbf{C}$ appeared to have little impact on whether any of the methdods converged or failed.

(I) Newton's Method

Newton's method can be applied (easily) to the problem of solving (1.2), because the Jacobian matrix for the function $\mathbf{F}$ in (1.2) is directly computable. (See (2.2).) Rather than using the inverse of the Jacobian matrix to compute successive iterates as

$$\mathbf{Y}_+ = \mathbf{Y} - (\mathbf{JF}(\mathbf{Y}))^{-1} F(\mathbf{Y})$$

we alternatively solved the system

$$\mathbf{JF}(\mathbf{Y})\ \mathbf{V} = F(\mathbf{Y})$$

for $\mathbf{V}$ and then calculated

$$\mathbf{Y}_+ = \mathbf{Y} - \mathbf{V}.$$

Newton's method was by far the weakest method. Unless the nature of the test case was such that the initial vector $\mathbf{X}$ was very close to $\mathbf{SX}$, then Newton's method usually diverged or the program crashed. The termination generally occurred, because in the simplistic form of the above implementation, we made no modification in the construction of successive iterates for sensitivity to the singularness of the Jacobian matrix.

We considered a modification to Newton's method, a quasi trust region approach. The errors ensued in the above direct implementaion because the unmodified Newton correction was either abnormally large or because the constructed iterate fell on the singular set for the Jacobian matrix. We modified the implementation to restrict the size of the Newton correction

so that each successive iterate remained in the same component of **G**. However, that restriction was too stringent as Newton's method then made no observable movement towards the system solution.

(II) NS01A

The nonlinear solver NS01A performed only moderately successfully against our set of test problems. NS01A solved 79% of the test problems with state dimension $n = 3$, but only 55% of the problems with state dimension $n = 4$. NS01A requires an input parameter, MAXFUN, to be set, which determines the number of iterations NS01A will carry out before returning control to the calling program. The majority of unsuccessful returns for NS01A on our set of test problems were coded with an exit error that the limit MAXFUN on iteration steps for NS01A had been reached. Increasing MAXFUN would have increased the impact of NS01A on our set of test problems, but at an expense of increased CPU time.

There were test problems for which NS01A exited unsuccessfully with other error codes (failure to detect decrease in the residuals and encountering fixed points of F). While these particular test problems could have been restarted at alternate entry points, we did not pursue that approach.

(III) DNEQNF

DNEQNF performed less than adequately against the problems we used for comparative evaluation. DNEQNF solved only 56% of the test problems with state dimension $n = 3$ and 34% of the problems with state dimension $n = 4$. The exit errors for DNEQNF were usually that the initial entry point was ill-chosen and that further progress could be initiated only after supplying a more suitable entry point.

The last two methods which we considered constructed covering paths for the homotopy

$$F(\mathbf{Y}(t) = (1-t)F(\mathbf{A}),\ 0 \leq t \leq 1, \tag{3.1}$$

with **A** being the standardized entry point we used for reference with all of the test problems.

(IV) ODE

As noted in Section 2, rather than treat (3.1) directly we can transform (3.1) to

$$F(\mathbf{Y}(t))d\mathbf{Y}(t)/dt = -F(\mathbf{A}),\ 0 \leq t \leq 1, \tag{3.2}$$

and solve (3.2) for the trajectory which passes through $\mathbf{A}$. We note that since the x_j components of $\mathbf{A}$ are ordered and since the solution trajectory to (3.2) must lie in a component of $\mathbf{G}$, the x_j components of $\mathbf{SF}$ must also be ordered, i.e., the description of the Jacobian in Proposition 2.1 implies that the x_j components of $\mathbf{Y}(t)$ cannot change their relative ordering for $0 \leq t \leq 1$.

Solving (3.2) using the differential equation solver ODE was substantially more expensive (in terms of accessed CPU time) than the above methods. The time required to follow the homotopy trajectory usually accumulated because of a 'hooking' effect which was forced on the trajectory because the components x_j of $\mathbf{Y}(t)$ cannot change their relative ordering for $0 \leq t \leq 1$. On the otherhand, using ODE we were able to solve each of the problems in our test set with state dimension $n = 3$ and 99% of the problems with state dimension $n = 4$.

When we considered more general problems where the state dimension of the problem was also to be determined, the effects of the 'hooking' on the CPU time required by ODE became more pronounced. While ODE was generally successful for problems with state dimensions $n \leq 5$, the program generally stagnated at higher state dimensions, $6 \leq n \leq 8$.

(V) Predictor-Corrector

An alternative viewpoint for using the homotopy (3.1) is to implement a more direct path following procedure to move from $\mathbf{A}$ to $\mathbf{SY}$. We implemented the following predictor-corrector method:

(a) At each iteration of the procedure we construct a polynomial predictor to move form the current location $(\mathbf{Y}(t), t)$ on the homotopy path to a new point $(\hat{\mathbf{Y}}(t_+), t_+)$.

(b) Using a modified Newton's method we then correct back from $(\hat{\mathbf{Y}}(t_+), t_+)$ to a point $(\mathbf{Y}(t_+), t_+)$ on the homotopy path.

(c) We adjust the step size for the next iteration to reflect the relative error of the previous prediction (primarily in terms of the number of Newton steps which were required for the correction step).

While there are several choices for predictors in step (a), we chose to use a form of (two-point) Hermite extrapolation, with which we constructed a third order polynomial predictor.

We found that if we did not modify Newton's method and restrict the size of the Newton correction, we usually jumped from the homotopy path we were following to a path in an alternative component of $\mathbf{G}$. We employed a modified trust region approach, which modified the size of the Newton correction only when the direction and size of the Newton correction would result (without correction) in jumping paths.

The predictor-corrector scheme described above was very robust and very fast. While it did not solve every problem in our test set, the exceptions generally occurred because of coalesing of x_j components of the solution. The predictor-corrector scheme solved 93% of the test problems with state dimension $n = 3$ and 83% of the test problems with state dimension $n = 4$. In every case it was substantially faster than the ODE program.

When we considered more general problems where the state dimension was also unknown, the predictor-corrector scheme produced very reliable results, for state dimensions $n \leq 6$. In comparison with the ODE program, the predictor-corrector scheme was significantly cheaper in terms of use of CPU time.

For problems with higher state dimensions, $7 \leq n \leq 8$, the program suffered difficulties which suggested that a better predictor, such as a higher order Hermite extrapolator, might be required.

References

[1] W. J. Wiscombe and J. W. Evans, Exponential-sum fitting of radioactive transmission functions, *J. Comp. Physics*, 24: 416-444 (1977).

[2] A. Ruhe, Fitting empirical data by positive sums of exponentials, *SIAM J. Sci. Stat. Comp.*, Vol. 1, No. 1, December 1980: 481-498.

[3] J. Miller, Finding roots of equations in given neighborhoods, *Appl. Math. Comp.*, Vol. 23, No. 3, Sept. 1987, 185-192.

[4] J. W. Evans, W. B. Gragg, R. J. Le Vegue, On least squares exponential sum approximation with positive coefficients, *Math. Comp.*, Vol. 34, No. 149, Jan. 1980, 203-211.

[5] T. Y. Li, T. Sauer, and J. A. Yorke, Numerically determining solutions of systems of polynomial equations, *Bull. AMS*, 18 (1988), 173-177.

[6] A. Morgan and A. Sommese, A homotopy for solving general homotopy systems that respects m-homogeneous structures, *Appl. Math. Comp.*, 24: 101-113 (1987).

[7] _____, Computing all solutions to polynomial systems using homotopy continuation, *Appl. Math. Comp.*, 24: 115-138 (1987).

[8] L. Shampine and M. Gordon, Computer Solution of Ordinary Differential Equations: The Initial Value Problem, Freeman, San Francisco, 1975.

[9] M. J. D. Powell, Technical Report AERE-R-5947, Harwell, England, 1968.

[10] L. T. Watson, T. C. Billaps and A. P. Morgan, "Algorithms G52:HOMPACK: A suite of codes for globally convergent homotopy algorithms," *ACM Transactions on Math. Software*, **13**, No. 3, (1987) 281-310.

[11] S. N. Chow, J. Mallet-Paret and J. A. Yorke, "A homotopy method for locating all zeros of a system of polynomials," *Functional Differential Equations and Approximation of Fixed Points*, Lecture Notes in Math. 730, Springer, New York, 1979, 228-237.

[12] C. B. Garcia and W. I. Zangwill, "Finding all solutions to polynomial systems and other systems of equations," *Math. Programming* **16** (1979), 159-176.

[13] J. Moré, B. Garbow and K. Hillstrom, "User guide for MINPACK-1," Argonne National Labs Report ANL-80-74, Argonne, Illinois 1980.

[14] IMSL Math/Library, User's Manual **2** (1987), 776-779.