Newton–Picard Gauss-Seidel

by

Joseph P. Simonis

A Project Report

Submitted to the Faculty

of

WORCESTER POLYTECHNIC INSTITUTE

in Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

in

Mathematical Sciences

by

May 2004

APPROVED:

Dr. Homer Walker, Project Advisor

1 Introduction

Sandia National Laboratories is interested in numerical methods for solving continuation problems on large scale systems. Some methods of particular interest are the Recursive Projection Method (RPM) of Shroff and Keller [8] and the Newton-Picard methods (NPs) of Lust et al.[5]

Often information about a dynamical system is given in the form a time-stepper. A time-stepper is a map that brings a solution from a time t to a time $t+T$. It may be described by the integration of a system of ODEs or PDEs. It may also be given by a "black-box" time integrator; a compiled code which takes the state of a system at time t as input, and outputs the state of the system at time $t+T$. The RPM and NP algorithms are used in tracking parameter dependent steady states of systems for which only "black-box" time-steppers are available.

Many dynamical systems of interest to Sandia National Laboratories are only available in the form of "black-box" time-steppers. John Shadid at Sandia is particularly interested in studying the macroscopic behavior of systems for which there exist only microscopic physical descriptions, for example using molecular dynamics codes to describe reaction-diffusion systems. Here the molecular dynamics simulation acts as a time–stepper for macroscopic parameters of interest.

In this project I studied a specific NP algorithm, Newton-Picard Gauss-Seidel (NPGS) for tracking periodic solutions of parameter dependent problems. I implemented a version of the algorithm as a subroutine in a code under development at Sandia to gain insights into practical considerations and limitations of the method.

Section two of this report gives a general idea of the NPGS method. For a more thorough presentation, see the original paper by Lust et. al. [5]. Section three presents a continuation problem used to verify the functionality of my code. The fourth section describes the algorithms as coded in the subroutine, including information about constants and specific routines called by the code. The final section presents the numerical solution of the test problem, reproducing the results presented in [5].

2 Derivation of Algorithm

2.1 Problem Formulation and Notation

Newton–Picard Gauss–Seidel was developed to "compute and determine the stability of both stable and unstable periodic orbits"[5] in large-scale dynamical systems within a continuation framework.

Problem: Given

$$\frac{dx}{dt} = f(x, \lambda),\ x \in I\!R^N,\ \lambda \in I\!R, \tag{1}$$

fix λ and find $x(0)$ and T such that

$$\begin{array}{rcl} x(T) - x(0) & = & 0 \\ s(x(0), T, \lambda) & = & 0 \end{array} \tag{2}$$

It is assumed that there is an accurate initial guess of the solution $x(0)$ and the period T, for a given λ. In the second equation, s is a phase condition needed to eliminate the translational invariance of periodic solutions. In the computations discussed below, s is a linear phase condition independent of T:

$$f(x(0)^{(0)}, \lambda)^T (x(0) - x(0)^{(0)}) = 0 \tag{3}$$

where $x(0)^{(0)}$ is the starting value for the Newton–Picard iterations.

Clearly we do not have a direct method of obtaining $x(T)$. An approximate solution is obtained by numerically solving the initial value problem formed by (1) with initial data $x(0)$. Its solution is denoted by $\varphi(x(0), T, \lambda)$. Then (2) is written

$$F(x(0), T) \equiv \left[\begin{array}{c} \varphi(x(0), T, \lambda) - x(0) \\ s(x(0), T, \lambda) \end{array} \right] = 0 \tag{4}$$

and the solution is denoted by $(x(0)^*, T^*, \lambda)$. The functions φ and s are assumed to be twice differentiable with respect to $x(0)$, T, and λ.

The matrix M^* given by

$$M^* := \left. \frac{\partial \varphi}{\partial x} \right|_{(x(0)^*, T^*, \lambda)} \tag{5}$$

is the monodromy matrix. Its eigenvalues (called Floquet multipliers) are denoted by μ_i^*, $i = 1 \ldots N$. The stability of the periodic orbit is determined by the magnitude of the eigenvalues. The orbit is *linearly stable* if $|\mu_i^*| < 1$ for all i, otherwise the orbit is considered *unstable*.

2.2 Newton-Picard Gauss-Seidel

A common method for computing $x(0)$ and T satisfying (4) is Newton's method. Newton's method generates a sequence $\{(x(0)^{(k)}, T^{(k)})\}$ such that under certain conditions $\{(x(0)^{(k)}, T^{(k)})\} \to (x(0)^{(*)}, T^{(*)})$ where $F(x(0)^{(*)}, T^{(*)}) = 0$. Given a current guess $(x(0)^{(k)}, T^{(k)})$, the $(k+1)$ guess is the root of the local linear model of $F(x(0), T)$ in the neighborhood of $(x(0)^{(k)}, T^{(k)})$.

Algorithm NEWT: Newton's method

GIVEN: $(x(0)^{(0)}, T^{(0)})$

WHILE (NOT CONVERGED)

$(x(0)^{(k+1)}, T^{(k+1)}) = -J^{-1}F((x(0)^{(k)}, T^{(k)})) + (x(0)^{(k)}, T^{(k)})$

where J is the $(N+1)\times(N+1)$ Jacobian matrix of F evaluated at $(x(0)^{(k)}, T^{(k)})$.

When N is large, Newton's method is infeasible because the Jacobian must be calculated using finite differences, with each function evaluation requiring the solution of an initial value problem. The Newton–Picard methods seek to reduce the computational cost of Newton's method on problems with F of the form (4) and for which the following assumption holds:

Assumption 1 *Let $y^* = (x(0)^*, T^*)$ denote an isolated solution to the system (4), and let $\mathcal{B}$ be a small neighborhood of y^*. Let $M(y) = \frac{\partial \varphi}{\partial x}(y)$ for $y \in \mathcal{B}$ and denote its eigenvalues by μ_i, $i = 1, \ldots, N$. Assume that for all $y \in \mathcal{B}$ precisely p eigenvalues lie outside the disk $C_\rho = \{|z| < \rho\}$, $0 < \rho < 1$ and that no eigenvalue has modulus ρ; i.e., for all $y \in \mathcal{B}$, $|\mu_1| \geq |\mu_2| \geq \ldots \geq |\mu_p| > \rho > |\mu_{p+1}|, \ldots, |\mu_N|$.*

Each iteration of Newton's method requires the solution of a linear system of the form:

$$\begin{bmatrix} \frac{\partial \varphi}{\partial x(0)} - I & \frac{\partial \varphi}{\partial T} \\ \frac{\partial s}{\partial x(0)} & \frac{\partial s}{\partial T} \end{bmatrix} \begin{bmatrix} \Delta x(0) \\ \Delta T \end{bmatrix} = - \begin{bmatrix} \varphi(x(0), T, \lambda) - x(0) \\ s(x(0), T, \lambda) \end{bmatrix} \tag{6}$$

Denote by $\mathcal{U}$ the subspace spanned by the eigenvectors associated with the p eigenvalues of M with largest magnitude. Its orthogonal complement is given by $\mathcal{U}^\perp$. Denote by $V_p \in \mathbb{R}^{N\times p}$ and $V_q \in \mathbb{R}^{N\times(N-p)}$ matrices the columns of which are orthonormal bases for the spaces $\mathcal{U}$ and $\mathcal{U}^\perp$ respectively. Now $x(0) \in \mathbb{R}^N$ can be decomposed:

$$x(0) = V_p \tilde{p} + V_q \tilde{q}$$

with $\tilde{p} \in \mathbb{R}^p$ and $\tilde{q} \in \mathbb{R}^{N-p}$. Left multiply the first N equations of (6) with $[V_q, V_p]^T$. The following facts are easily deduced; $V_p^T V_q = 0$, $V_q^T V_p = 0$ and $V_q^T \frac{\partial \varphi}{\partial x(0)} V_p = 0$. Also $V_q^T \frac{\partial \varphi}{\partial T} \to 0$ as the Newton iterates converge because at the solution $\frac{\partial \varphi}{\partial T}$ is an eigenvector of $\frac{\partial \varphi}{\partial x(0)}$ with eigenvalue 1 ([4],pp.29-30) and thus a vector in $\mathcal{U}$. Equation (6) can be rewritten

$$\begin{bmatrix} V_q^T \frac{\partial \varphi}{\partial x} V_q - I_q & 0 & 0 \\ V_p^T \frac{\partial \varphi}{\partial x} V_q & V_p^T \frac{\partial \varphi}{\partial x} V_p - I_p & V_p^T \frac{\partial \varphi}{\partial T} \\ \frac{\partial s}{\partial x} V_q & \frac{\partial s}{\partial x} V_p & \frac{\partial s}{\partial T} \end{bmatrix} \begin{bmatrix} \Delta \tilde{q} \\ \Delta \tilde{p} \\ \Delta T \end{bmatrix} = - \begin{bmatrix} V_q^T(\varphi - x) \\ V_p^T(\varphi - x) \\ s \end{bmatrix} \tag{7}$$

where the explicit function dependencies have been dropped. The first set of $N - p$ equations is solved using Picard iterations:

$$\begin{cases} \Delta \tilde{q}^{[0]} = 0, \\ \Delta \tilde{q}^{[i]} = V_q^T \frac{\partial \varphi}{\partial x} V_q \Delta \tilde{q}^{[i-1]} + V_q^T(\varphi - x), \ i = 1, \ldots, l, \\ \Delta \tilde{q} = \Delta \tilde{q}^{[l]} \end{cases} \tag{8}$$

Substitute $\Delta\tilde{q}$ into (7); a linear system in $\Delta\tilde{p}$ and ΔT remains. This new system is $p+1$ equations in $p+1$ unknowns.

$$\begin{bmatrix} V_p^T \frac{\partial \varphi}{\partial x} V_p - I_p & V_p^T \frac{\partial \varphi}{\partial T} \\ \frac{\partial s}{\partial x} V_p & \frac{\partial s}{\partial T} \end{bmatrix} \begin{bmatrix} \Delta\tilde{p} \\ \Delta T \end{bmatrix} = - \begin{bmatrix} V_p^T(\varphi - x) - V_p^T \frac{\partial \varphi}{\partial x} V_q \Delta\tilde{q} \\ s - \frac{\partial s}{\partial x} V_q \Delta\tilde{q} \end{bmatrix} \tag{9}$$

If we assume $p \ll N$ then V_q has dimension close to $N \times N$, making the computation of V_q expensive. To avoid explicitly computing and storing V_q we rewrite (8) and (9) in terms of $\Delta q = V_q \Delta\tilde{q}$ and $Q = V_q V_q^T = I_N - V_p V_p^T$:

$$\begin{cases} \Delta q^{[0]} = 0, \\ \Delta q^{[i]} = Q \frac{\partial \varphi}{\partial x} \Delta q^{[i-1]} + Q(\varphi - x), \ i = 1, \ldots, l, \\ \Delta q = \Delta q^{[l]} \end{cases} \tag{10}$$

$$\begin{bmatrix} V_p^T \frac{\partial \varphi}{\partial x} V_p - I_p & V_p^T \frac{\partial \varphi}{\partial T} \\ \frac{\partial s}{\partial x} V_p & \frac{\partial s}{\partial T} \end{bmatrix} \begin{bmatrix} \Delta\tilde{p} \\ \Delta T \end{bmatrix} = - \begin{bmatrix} V_p^T(\varphi - x) - V_p^T \frac{\partial \varphi}{\partial x} \Delta q \\ s - \frac{\partial s}{\partial x} \Delta q \end{bmatrix} \tag{11}$$

This formulation does not require the matrix V_q. The term $\frac{\partial \varphi}{\partial x} V_p$ is formed with only p matrix–vector products using the approximation $\frac{\partial \varphi}{\partial x} v \approx \frac{1}{\epsilon}[\varphi(x(0) + \epsilon v, T) - \varphi(x(0), T)]$. Similarly each iteration of (10) requires a single matrix–vector product. Therefore, approximately $p+l$ integrations are needed to obtain Δq and solve (11). Then $(\Delta x(0), \Delta T) = (\Delta q + V_p \Delta\tilde{p}, \Delta T)$ solves the original system (6). With $p \ll N$ this is a significant reduction in computational cost over Newton's method using a Jacobian formed by finite differences.

Solving first for Δq and using it to obtain $\Delta\tilde{p}$ and ΔT is similar to a single sweep of a Gauss–Seidel method, hence the name *Newton–Picard Gauss–Seidel* method.

In summary the Newton-Picard methods combine the efficiency of the Picard algorithm with the robustness of Newton's Method. The idea, put forth by Shroff and Keller[8] and Jarausch and Mackens[1, 2, 3] and summarized by Lust et. al.[5], is "to use Newton's Method on a small-dimensional subspace, where the Picard iteration is slowly convergent or unstable, and to use the Picard iteration on the complement of that subspace."

3 Brusselator Problem

3.1 Problem

The Brusselator problem [6] involves a coupled nonlinear system of equations derived from a hypothetical set of chemical reactions:

$$\frac{\partial X}{\partial t} = \frac{D_x}{L^2} \frac{\partial^2 X}{\partial z^2} + X^2 Y - (B+1)X + A \tag{12}$$

$$\frac{\partial Y}{\partial t} = \frac{D_y}{L^2}\frac{\partial^2 Y}{\partial z^2} - X^2 Y + BX \tag{13}$$

X and Y are concentrations of different chemical species; the parameters A and B are fixed at values of 2 and 5.45 respectively, $D_x = .008$ and $D_y = .004$. The characteristic length L is the bifurcation parameter of the problem. The Dirichlet boundary conditions are given by

$$X(t, z = 0) = X(t, z = 1) = A \tag{14}$$

$$Y(t, z = 0) = Y(t, z = 1) = B/A \tag{15}$$

A steady-state solution of this problem is given by $X = A$ and $Y = B/A$. From this we may form a trivial steady-state branch of the corresponding continuation problem. Hopf bifurcations occur at the parameter values of $L_k = 0.5130k,\ k = 1, 2, \ldots$. A Hopf bifurcation is characterized by "the appearance, from equilibrium state, of small–amplitude periodic oscillations." [4] Branches of periodic solutions extend from these bifurcation points. In the Lust et. al paper [5] the NPGS algorithm is used to compute solutions along the first three of these branches. As a test of the code written for this project, these curves are reproduced.

3.2 Integrator

The NPGS algorithm requires a function which acts as a time-stepper. (Recall that a time-stepper is a map that brings a solution from a time t to a time $t + T$). In the context of the Brusselator problem this is a function which takes the concentrations at $t = 0$, $X(0)$ and $Y(0)$, and returns $X(T)$ and $Y(T)$. The integration of the system is performed with the operator splitting method discussed by D. Ropp and J. Shadid in [7].

Given an initial guess $u(0) = [X(0), Y(0)]^T$ and some time t, find $u(t) = [X(t), Y(t)]^T$. Form two operators $F_R(u)$ and $F_D(u)$ for the reaction and diffusion terms respectively.

$$F_R(u) = \begin{bmatrix} A - (B+1)X + X^2 Y \\ BX - X^2 Y \end{bmatrix} \tag{16}$$

$$F_D(u) = \begin{bmatrix} \frac{D_x}{L^2}\frac{\partial^2 X}{\partial^2 z} \\ \frac{D_y}{L^2}\frac{\partial^2 Y}{\partial^2 z} \end{bmatrix} \tag{17}$$

The system is written

$$\frac{du}{dt} = F_D(u) + F_R(u),\ z \in [0, 1],\ t > 0 \tag{18}$$

"A single step of a first-order splitting method advancing the solution from t^n to $t^{n+1} = t^n + \Delta t$ amounts to an application of time discretizations applied to the system

$$\begin{aligned} \tfrac{du^*}{dt} &= F_R(u^*) \quad on\ (t^n, t^{n+1}), \qquad u^*(t^n) = u^n, \\ \tfrac{du^{**}}{dt} &= F_D(u^{**}) \quad on\ (t^n, t^{n+1}), \quad u^{**}(t^n) = u^*(t^{n+1}), \end{aligned} \tag{19}$$

with $u^{n+1} = u^{**}(t^{n+1})$."[7] The reaction part has no spatial dependence; its integration is performed by the classical 4^{th}–order Runge-Kutta method. The diffusion part does have a spatial dependence; its integration is performed by the backward Euler method.

Algorithm RK: 4^{th}–order Runge-Kutta Method

GIVEN A TIME STEP h AND INITIAL VALUE u_n
$K_1 = F_R(u_n)$
$K_2 = F_R(u_n + \frac{1}{2}hK_1)$
$K_3 = F_R(u_n + \frac{1}{2}hK_2)$
$K_4 = F_R(u_n + hK_3)$
$u_{n+1} = u_n + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4)$

Algorithm BE: backward Euler Method

GIVEN A TIME STEP h AND INITIAL VALUE u_n
$u_{n+1} = u_n + hF_D(u_{n+1})$

Algorithm BI: Brusselator Integrator

GIVEN A TIME INTERVAL $[0, T]$ AND INITIAL VALUE u_0.
LET $h = T/1000$
FOR $i = 0, i < 1000, i++$
    $u^*_{i+1} = RK(h, u_i)$
    $u_{i+1} = BE(h, u^*_{i+1})$
END LOOP
$u_T = u_{1000}$

The backward Euler method applied to the diffusion term of the Brusselator problem requires the solution of a sparse linear system. A tri-diagonal matrix solver is used. The Runge-Kutta method requires four function evaluations.

4 Code

This project involved implementing the Newton–Picard Gauss–Siedel method as a subroutine for Andrew Salinger's CAPO code. The CAPO code is envisioned to be a set of numerical routines for performing standard continuation and bifurcation computations using only "black box" time integrators to obtain necessary information of function behavior.

The NPGS(2) algorithm is Algorithm 3.2 in [5]. I present it here including details specific to my implementation. Following the algorithm are the additional subroutines called by NPGS(2).

Algorithm NPGS(2): Newton–Picard Gauss-Siedel

GIVEN: $x(0)^{(0)}$, $T^{(0)}$, STARTING BASIS V_e,
    $M(x(0), T)v$ ROUTINE, $\varphi(x(0), T)$ ROUTINE.

COMPUTE: $f_{init} = f(x(0)^{(0)}, T, \lambda) = (\varphi(x + \epsilon, T) - \varphi(x, T))/\epsilon$
$v = \varphi(x(0)^{(0)}, T)$
$r = v - x(0)^{(0)}$
WHILE (NOT CONVERGED)
CREATE W_e,S_e,R_e
CALL SUBSPACEITERATIONS(W_e,S_e,R_e,p)
IF $(p > 0)$
BUILD V_p
CALCULATE dq
CALCULATE $d\overline{p}$, dT
$x(0) += dq + V_p d\overline{p}$
$T += dT$
ELSE
$dq = M(x(0), T)r + r$
$lhs = \varphi(x(0), \epsilon, \lambda) - x(0)/\epsilon$
$rhs = f_{init} \cdot (x(0) - x(0)^{(0)} + dq)$
$dt = -rhs/lhs$
$x(0) += dq$
$T += dT$
ADJUST BASIS SIZE V_e
ORTHONORMALIZE V_e
$v = \varphi(x(0), T)$
$r = v - x(0)$
END WHILE

Algorithm SI: Subspace Iterations

GIVEN: W_e, S_e, R_e, p.
FOR $(1 : subspaceIters)$
$W_e = M(x(0), T)V_e$
$S_e = V_e^T W_e$
PERFORM ORDERED SCHUR DECOMPOSITION ON $S_e \rightarrow Y_e R_e Y_e^T$
IF $(i < subspaceIters)$
$V_e = W_e Y_e$
ORTHONORMALIZE V_e
END IF
END FOR

An approximation of the initial subspace is calculated by Algorithm InitSub below. The general idea is to form vectors of random numbers and perform subspace iterations until a "good" approximation of the dominant eigenspace is found. Through trial and error it was found performing 15 iterations tends to converge the vectors to the desired eigenvectors. A future implementation should use a more sophisticated method for stopping the iterations. For the tests an initial subspace consisting of four vectors is used. The random vectors are produced in an ad hoc manner; the only restriction used is there are no zero

entries in the vector.

Algorithm InitSub: Initialize Subspace

FOR $(1:4)$
  $T_{perturbed} = (rand()\%10) + 20T$
  $r = \varphi(x(0)^{(0)}, T_{perturbed})$
  $r+ = .01$ (AN N VECTOR OF .01'S)
  ADD r AS A COLUMN OF V_e.
END FOR
ORTHONORMALIZE V_e
FOR $(1:15)$
  PERFORM SUBSPACE ITERATIONS
END FOR
ORTHONORMALIZE V_e

In NPGS(2) the matrix V_p is formed by multiplying the first p columns of V_e by the $p \times p$ matrix consisting of the upper left $p \times p$ block of S_e. The Ordered Schur Decomposition routine is the LAPACK routine dgees. The details for calculating dq, $d\overline{p}$, and dT can be found in [5]. NPGS(2) as written here assumes a linear phase condition of the form

$$f(x(0)^{(0)}, \lambda)^T \cdot (x(0) - x(0)^{(0)}) = 0 \tag{20}$$

5 Results

This implementation of the Newton–Picard Gauss–Seidel algorithm successfully reproduces the results presented in Figure 4.1 of [5] for the Brusselator problem. The following three figures are the branches of periodic solutions extending from the Hopf bifurction points along the trivial steady–state solution $X = A/B$. The horizontal axis represents the reactor length, L, while the vertical axis represents the period of the solution, T.

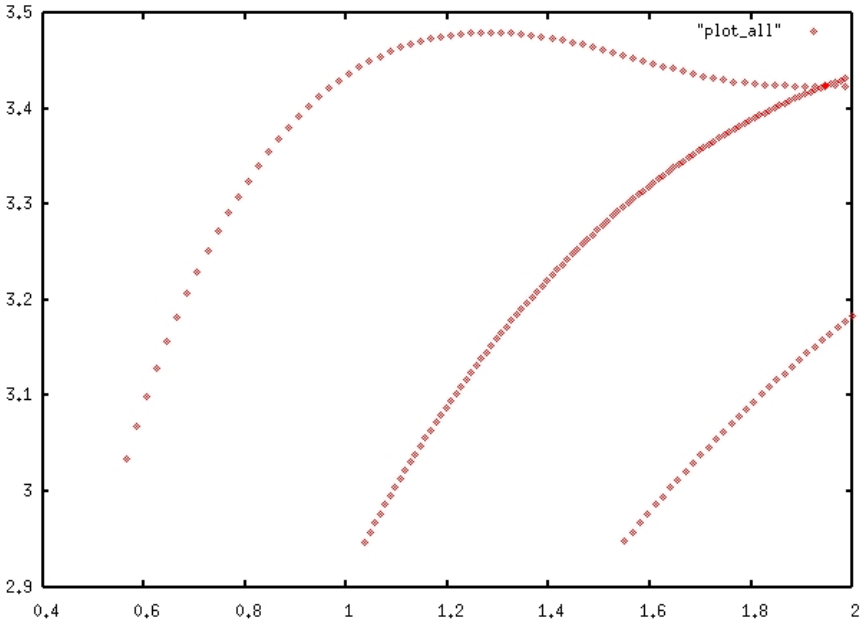

Figure 1: Solution Branches

References

[1] H. Jarausch and W. Mackens. Computing bifurcation diagrams for large nonlinear variational problems. In P. Deuflhard and B. Enguist, editors, *Large Scale Scientific Computing*, Progress in Scientific Computing 7. Birkhäuser-Verlag, Basel, 1987.

[2] H. Jarausch and W. Mackens. Numerical treatment of bifurcation branches by adaptive condensation. In T. Küpper, H. Mittelman, and H. Weber, editors, *Numerical Methods for Bifurcation Problems*. Birkhäuser-Verlag, Basel, 1987.

[3] H. Jarausch and W. Mackens. Solving large nonlinear systems of equations by an adaptive condensation process. *Numer. Math.*, 50:633–653, 1987.

[4] Yuri A. Kuznetsov. *Elements of Applied Bifurcation Theory*. Applied Mathematical Sciences. Springer, second edition, 1995.

[5] K. Lust, D. Roose, A. Spence, and A. Champneys. An adaptive Newton-Picard algorithm with subspace iteration for computing periodic solutions. *SIAM J. Sci. Comput.*, 19:1188–1209, 1998.

[6] I. Prigogine and R. Lefever. Symmetry breaking instabilities in dissipative systems. *J. Chem. Phys.*, 48(4):1695–1700, 1968.

[7] D. L. Ropp and J. N. Shadid. Stability of operator splitting methods for systems with indefinite operators: reaction-diffusion systems. *Journal of Computational Physics*, 2004. To Appear.

[8] G. M. Shroff and H. B. Keller. Stablization of unstable procedures: The recursive projection method. *SIAM J. Numer. Anal.*, 30:1099–1120, 1993.