30

1968 Summer Institute on Symbolic Mathematics by Computer

COMPUTING TIME ANALYSES FOR SOME ARITHMETIC AND ALGEBRAIC ALGORITHMS

by

George E. Collins*

Computer Sciences Technical Report #36

July 1968

*Computer Sciences Department and Computing Center

1. Introduction

In a recent paper, [6], the author presented a new algorithm, the reduced p.r.s. algorithm, for computing the g.c.d. of two multivariate polynomials with integer coefficients. It was asserted there that the computing time for this algorithm, when applied to two univariate polynomials of degree n whose coefficients are d digits long, is approximately proportional to $n^4 d^2$. In Section 3 of the present paper we analyze thoroughly and rigorously the computing time for this algorithm in the univariate case, proving that if the two polynomials are weakly normal and d bounds their norms then the computing time is bounded by a function which is $O(n^4(\ln d)^2)$, the norm of a polynomial being the sum of the absolute values of its coefficients. This is obtained as a corollary of a more general theorem, which bounds the computing time as a function of four variables.

In Section 4 of the present paper we present a new and faster algorithm for computing the g.c.d. of two univariate polynomials with integer coefficients. The new algorithm uses congruence arithmetic (arithmetic performed in finite fields. GF(p), containing a prime number of elements) and is based on the theory of reduced polynomial remainder sequences and subresultants developed in [6]. Several theorems are proved in Section 4 to show that the algorithm always terminates and produces the greatest common divisor.

In Section 5, the computing time of the new algorithm is analyzed. The bound that is obtained for the computing time of the new algorithm is

$O(n^4(\ln d) + n^3(\ln d)^2)$. Moreover, it is shown that the average computing time for the new algorithm is $O(n^3(\ln d)^2)$, a substantial improvement. Furthermore, it is shown that the average computing time for the new algorithm when applied to polynomials with a g.c.d. of degree zero (a case of frequent occurrence in practice) is $O(n^2 + n(\ln d)^2)$. Finally, a method is proposed for extending the algorithm to multivariate polynomials.

In order to analyze the computing times of polynomial algorithms, we must have at our disposal bounds for the computing times for performing operations on large integers. Such bounds, easily obtained for the operations of addition, subtraction, multiplication and division (with or without a remainder), are stated in Section 2. We also obtain bounds in Section 2 for the time to compute the g.c.d. of two integers using the Euclidean algorithm, and show that these bounds also apply to the extended Euclidean algorithm.

In the congruence arithmetic g.c.d. algorithm, as in other congruence arithmetic algorithms (see, for example, [1] and [14]), one must apply the Chinese remainder theorem algorithm. A bound for the computing time of this algorithm is also derived in Section 2.

2. Operations on Large Integers

We assume throughout that integers are represented in radix form, using an arbitrary base $\beta \geq 2$. Computing times for arithmetic algorithms are then naturally expressed as functions of the number of β-digits in certain numbers N which occur in the algorithms, i.e. $[\log_\beta N] + 1$. However, since $\log_\beta N = (\ln N)/(\ln \beta)$, where $\ln$ is the natural logarithm function, and since we will uniformly ignore constant multipliers, these being dependent in any case on the particular computer that is used, on numerious details of the version of the algorithm used and the precise manner in which data are represented in the computer, we shall express computing times in terms of $\ln N$.

The following theorem, on addition and subtraction, illustrates the general form in which our theorems will be stated.

Theorem 2.1. Let $t(a, b)$ be the time required to compute $a + b$ (or $a - b$). Let $T(d) = \max \{t(a, b): |a|, |b| \leq d\}$. Then $T(d) = O(\ln d)$.

The statement that $T(d) = O(\ln d)$ means that there exists a constant C (independent of d) such that $T(d) \leq C \ln d$ for all sufficiently large d. Of course the theorem appears to be still quite ambiguous since it does not specify what algorithm is to be used, nor what computer. The exact statement would

be that if we choose any one of the standard classical algorithms, any one of the familiar data representations which use radix canonical form, and any one of various well-known computers, then such a C will exist for that combination of choices. The theorem can be readily verified for several choices by consulting [5], [8] and [13].

The next theorem states a similarly well-known fact about various classical multiplication algorithms.

Theorem 2.2. Let $t(a, b)$ be the time to compute $a \cdot b$. Let $T(d, e) = \max \{t(a, b): |a| \leq d \ \& \ |b| \leq e\}$. Then $T(d, e) = O((\ln d)(\ln e))$.

Here we have applied the O-notation to a function $T(d, e)$ of more than one variable. The meaning of the statement is that, for some C, $T(d, e) \leq C(\ln d)(\ln e)$ whenever $d \geq d_0$ and $e \geq e_0$.

Theorem 2.2 applies to the classical multiplication algorithms. As a special case we have $T(d, d) = O((\ln d)^2)$, although the stronger form of the theorem containing two variables will be important in many applications where one argument is much smaller than the other. In recent years several multiplication algorithms have been devised which are much faster for very large integers. One such fast algorithm, based on earlier work by A. L. Toom, is given by Cook in [11]. It has the property that $T(d, d) = O((\ln d) \cdot 2^{5\sqrt{\ln(\ln d)}})$. It follows that, for every $\epsilon > 0$, $T(d, d) = O((\ln d)^{1+\epsilon})$. It is easy to construct a simple version of this algorithm for which $T(d, d) = O((\ln d)^{1.5})$. Throughout the present paper, however, it will be assumed that the classical multiplication algorithm is employed, and Theorem 2.2 will be applied.

So far, we have bounded computing times as functions of bounds on the inputs to the algorithms. In order to obtain a tight bound on the time for division, it is necessary to express it in terms of bounds on one input (the divisor) and one output (the quotient), as follows.

Theorem 2.3. Let $t(a, b)$ be the time to compute q and r, given a and b, such that $a = b \cdot q + r$, $0 \le |r| < |b|$, $ar \ge 0$ and $abq \ge 0$. Let $T(d, e) = \max \{t(a, b): |b| \le d \ \& \ |q| < e\}$. Then $T(d, e) = O((\ln d)(\ln e))$.

The truth of Theorem 2.3 follows from the observation that most of the computation required to produce q and r from a and b is essentially the same as that required to produce a from b, q and r.

Next we bound the computing time for the Euclidean algorithm. The bound is derived from the following lemma, which bounds the product of the quotients computed by the algorithm.

Lemma 2.4. Let $q_1, q_2, \ldots, q_n$ be the sequence of quotients obtained when the Euclidean algorithm is applied to a and b, $a \ge b > 0$. Let $c = \gcd(a, b)$. Then $q_n \prod_{i=1}^{n-1}(q_i + 1) < ab/c^2$.

Proof. Let $a_1 = a$, $a_2 = b$, $a_i = a_{i+1} q_i + a_{i+2}$ with $0 \le a_{i+2} < a_{i+1}$ for $1 \le i \le n$, and $a_{n+2} = 0$, so that $c = a_{n+1}$. Then $a_i = a_{i+1} q_i + a_{i+2} > a_{i+2} q_i + a_{i+2} = a_{i+2}(q_i + 1)$ for $1 \le i \le n$. Taking the product of these inequalities for $1 \le i \le n-1$ we have $\prod_{i=1}^{n-1} a_i > \prod_{i=1}^{n-1} a_{i+2}(q_i + 1)$. Cancelling a_i's on both sides, $ab = a_1 a_2 > a_n c \prod_{i=1}^{n-1}(q_i + 1)$. But $a_n = q_n a_{n+1} = q_n c$, so $ab > q_n c^2 \prod_{i=1}^{n-1}(q_i + 1)$.

As an interesting sidelight, notice that each $q_i \geq 1$ and $q_n \geq 2$ except when $a = b$. Hence $2^n \leq q_n \prod_{i=1}^{n-1} (q_i + 1) < ab/c^2 \leq ab < a^2$ so that $n < 2 \log_2 a$. This bound for the number of divisions compares with a bound of about $1.44 \log_2 a$ obtained from Lamé's theorem.

Theorem 2.5. Let $t(a, b)$ be the computing time for the Euclidean algorithm. Let $T(d, e) = \max \{t(a, b): a > b > 0 \quad \& \quad b \leq d \quad \& \quad a/\gcd(a, b) \leq e\}$. Then $T(d, e) = O((\ln d)(\ln e))$.

Proof. Let $c = \gcd(a, b)$. By Theorem 2.3, there exist constants C_1 and C_2 such that the time for the i-th division in Euclid's algorithm is $\leq C_1(\ln q_i)(\ln a_{i+1}) + C_2$. Hence the time for all divisions is $\leq C_1 \sum_{i=1}^{n} (\ln q_i)(\ln a_{i+1}) + C_2 n \leq C_1 (\ln b) \sum_{i=1}^{n} \ln q_i + C_2 n = C_1 (\ln b)(\ln \prod_{i=1}^{n} q_i) + C_2 n < C_1 (\ln b)(\ln (ab/c^2)) + 2 C_2 \log_2 a < C_1 (\ln b)(\ln (a^2/c^2)) + 3 C_2 \ln a \leq 2 C_1 (\ln d)(\ln e) + 3 C_2 \ln ce \leq 2 C_1 (\ln d)(\ln e) + 3 C_2 \ln d + 3 C_2 (\ln e) = O((\ln d)(\ln e))$.

In analyzing the computing time for the Chinese remainder theorem algorithm below, we shall also need a bound for the computing time for the extended Euclidean algorithm which, given a and b, $a \geq b > 0$, computes not only $c = \gcd(a, b)$ but also, simultaneously, integers x and y such that $ax + by = c$. The extended Euclidean algorithm may be defined as follows (see [12] and [10]). Let the a_i and q_i be defined as above. Set $x_1 = 1$, $y_1 = 0$, $x_2 = 0$ and $y_2 = 1$. For $1 \leq i \leq n - 1$, let $x_{i+2} = x_i - q_i x_{i+1}$ and $y_{i+2} = y_i - q_i y_{i+1}$. Then $x = x_{n+1}$ and $y = y_{n+1}$.

Theorem 2.6. Let $t(a, b)$ be the computing time for the extended Euclidean algorithm. Let $T(d) = \max \{t(a, b): a > b > 0 \ \& \ a/\gcd(a, b) \leq d\}$. Then $T(d) = O((\ln d)^2)$.

Proof. The additional computing time for the extended Euclidean algorithm is that required to compute x_{i+2} and y_{i+2} for $1 \leq i \leq n - 1$. It was shown in [10] that $|x_i| \leq b/2c$ and $|y_i| \leq a/2c$ for all i. Hence the time for all the multiplications $q_i x_i$ is, by Theorem 2.2, $\leq C_1 \sum_{i=1}^{n-1} (\ln q_i)(\ln x_{i+1}) + C_2 n \leq C_1 \ln(b/c) \sum_{i=1}^{n-1} \ln q_i + C_2 n < C_1 \ln(b/c) \ln(ab/c^2) + C_2 n = O((\ln d)^2)$, as in the proof of Theorem 2.5. It is also shown in [10] that the x_i (also the y_i) alternate in sign. Hence $|q_i x_{i+1}| \leq |x_{i+2}| \leq b/c$ and $|q_i y_{i+1}| \leq |y_{i+2}| \leq a/c$ for all i. Each subtraction $x_i - q_i x_{i+1}$ can therefore be performed in $C_1(\ln|x_{i+2}|) + C_2 \leq C_1 \ln d + C_2$ time units. Since $2^n \leq d^2$, the time for all such subtractions in $O((\ln d)^2)$. Likewise, the time to compute all y_i is $O((\ln d)^2)$.

In the application of the extended Euclidean algorithm in the Chinese remainder theorem, as in many other applications, a is a prime number p, so that $c = \gcd(a, b) = 1$. We then have $px + by = 1$, i.e., $by \equiv 1 \pmod p$. Since $|y| \leq p/2$, the inverse of b in $GF(p)$ is y or $y + p$ according as $y > 0$ or $y < 0$.

Our next theorem bounds the computing time for the Chinese remainder theorem algorithm. We summarize below the computations performed in carrying out this algorithm in order to make the theorem precise; we shall also refer to these steps in the proof of the theorem. Input to the algorithm includes a

sequence $(p_1, p_2, \ldots, p_n)$ of pairwise relatively prime numbers. We shall assume each $p_i \geq 2$; in our application below the p_i will be prime numbers, but this need not be assumed in the theorem. Additional input to the algorithm is a corresponding sequence $(a_1, a_2, \ldots, a_n)$ such that $0 \leq a_i < p_i$ for all i. The output of the algorithm is the unique integer A such that $|A| \leq p_1 \cdot p_2 \cdots p_n/2$ and $A \equiv a_i \pmod{p_i}$ for all i.

Chinese Remainder Theorem Algorithm

(1) Set $Q_1 = P_1$ and compute $Q_i = Q_{i-1} \cdot p_i$ for $2 \leq i \leq n$. Set $P = Q_n$.

(2) Compute $P_i = P/p_i$ for $i = 1, 2, \ldots, n$.

(3) Compute q_i and r_i such that $P_i = p_i q_i + r_i$ and $0 \leq r_i < p_i$ for $1 \leq i \leq n$.

(4) Compute t_i such that $r_i t_i \equiv 1 \pmod{p_i}$ and $0 < t_i < p_i$ for $1 \leq i \leq n$.

(5) Compute $S = \Sigma_{i=1}^{n} P_i t_i a_i$.

(6) Compute Q and R such that $S = PQ + R$, $0 \leq R < P$.

(7) Compute $H = [P/2]$.

(8) Set $A = R - H$ if $R \geq H$; otherwise set $A = R$.

Theorem 2.7. Let $t(p_1, p_2, \ldots, p_n, a_1, a_2, \ldots, a_n)$ be the computing time for the Chinese remainder theorem algorithm. Let $T(d) = \max \{t(p_1, \ldots, p_n, a_1, \ldots, a_n)$: $\Pi_{i=1}^{n} p_i \leq d\}$. Then $T(d) = O((\ln d)^2)$.

Proof. It suffices to show that the time for each of the eight steps is $O((\ln d)^2)$. The time for Step (1) is $\leq \Sigma_{i=2}^{n} (C_1(\ln Q_{i-1})(\ln p_i) + C_2) \leq$

$C_1 (\ln P) \Sigma_{i=2}^{n} (\ln p_i) + C_2 n \le C_2 (\ln d)(\ln \Pi_{i=2}^{n} p_i) + C_2 n \le C_2 (\ln d)^2 + C_2 n = O((\ln d)^2)$ since $2^n \le P \le d$ so that $n = O(\ln d)$.

The time for Step (2) is $\le \Sigma_{i=1}^{n} (C_1 (\ln P_i)(\ln p_i) + C_2) \le C_1 (\ln d) \Sigma_{i=1}^{n} (\ln p_i) + C_2 n = O((\ln d)^2)$. The time for Step (3) is $\le \Sigma_{i=1}^{n} C_1 ((\ln p_i)(\ln q_i) + C_2) \le C_1 (\ln d) \Sigma_{i=1}^{n} (\ln p_i) + C_2 n = O((\ln d)^2)$. By Theorem 2.6, the time for Step (4) is $\le \Sigma_{i=1}^{n} (C_1 (\ln p_i)^2 + C_2) \le C_1 \Sigma_{i=1}^{n} (\ln p_i)^2 + C_2 n \le C_1 (\Sigma_{i=1}^{n} (\ln p_i))^2 + C_2 n \le C_1 (\ln d)^2 + C_2 n = O((\ln d)^2)$.

The time to compute all products in Step (5) is clearly $O(\ln d)^2)$ since $t_i < p_i$, $a_i < p_i$, $P_i < P$, $P_i t_i < P$ and $P < d$. Let $S_j = \Sigma_{i=1}^{j} P_i t_i a_i$. Then $S_j \le P \Sigma_{i=1}^{j} a_i < P \Sigma_{i=1}^{j} p_i \le P \Pi_{i=1}^{j} p_i \le P^2 \le d^2$. Hence the time for all additions is $\le n (C_1 (\ln d^2) + C_2) = n(2 C_1 (\ln d) + C_2) = O((\ln d)^2)$.

In Step (6), $S = S_n < P^2$, so $Q < P$ and hence the time is $O((\ln d)^2)$. The times for Steps (7) and (8) are clearly $O(\ln d)$.

3. Operations on Univariate Polynomials

The primary purpose of the present section is to give a relatively complete analysis of the computing time for computing the g.c.d. of two univariate polynomials using the reduced p.r.s. algorithm of [6]. Along the way, however, we shall also take the occasion to bound the computing time for various other operations on univariate polynomials with integer coefficients.

It turns out to be very useful, for the purpose of such analyses, to define the norm of such a polynomial, as follows.

Definition 3.1. If $P(x) = \Sigma_{i=1}^{n} a_i x^i$ is a polynomial with integer coefficients, the norm of P is defined to be $\Sigma_{i=0}^{n} |a_i|$.

Norm (P) is, in fact, a norm for the ring of polynomials over the integers, as shown by the following theorem.

Theorem 3.2. Norm $(P + Q) \leq$ norm (P) + norm (Q). norm $(P \cdot Q) \leq$ norm (P) $\cdot$ norm (Q).

Proof. The first part is trivial. Let $P(x) = \Sigma_{i=0}^{m} a_i x^i$, $Q(x) = \Sigma_{i=0}^{n} b_i x^i$ and $R(x) = \Sigma_{i=0}^{m+n} c_i x^i$, where $R = P \cdot Q$. Then norm $(R) = \Sigma_{k=0}^{m+n} |c_k| = \Sigma_{k=0}^{m+n} |\Sigma_{i\ j=k} a_i b_j| \leq \Sigma_{k=0}^{m+n} \Sigma_{i\ j=k} |a_i b_j| = \Sigma_{i=0}^{m} \Sigma_{j=0}^{n} |a_i b_j| = \Sigma_{i=0}^{m} |a_i| \Sigma_{j=0}^{n} |b_j|$ = norm (P) $\cdot$ norm (Q).

The norm has two other important properties which will be frequently used in the following: (1) $|a_i| \leq$ norm (P) for all i, and (2) norm $(P) \leq (\Sigma_{i=0}^{m} a_i^2)^{1/2}$.

The following notation is frequently useful for simultaneously bounding both norm (P) and deg (P).

Definition 3.3. $U(d, m) = \{P:$ norm $(P) \leq d$ & deg $(P) \leq n\}$.

The following three theorems bound the computing times for the sum, difference, product or quotient of two polynomials, using the classical algorithms. It is assumed here that the polynomials are represented by a canonical form as in [2], [5], [9] or [13].

Theorem 3.4. The time to compute $P + Q$ or $P - Q$ for $P, Q \in U(d, m)$ is $O((\ln d)n)$.

Here we have for the first time stated a thorem using a more elliptic phraseology. If stated in full the theorem would have the same form as our previous theorems: Let $t(P, Q)$ be the time to compute $P + Q$ (or $P - Q$). Let $T(d, m) = \max \{t(P, Q): P, Q \in U(d, m)\}$. Then $T(d, m) = O((\ln d)m)$.

Proof. At most $m + 1$ coefficient additions or subtractions are required, and each takes $\leq C_1 \ln d + C_2$ computing time. But $(m + 1)(C_1 \ln d + C_2) = O((\ln d)m)$.

Theorem 3.5. The time to compute $P \cdot Q$ for $P \in U(d, m)$ and $Q \in (e, n)$ is $O((\ln d)(\ln e)mn)$.

Proof. At most $(m + 1)(n + 1)$ coefficient multiplications are required, the time for each being $\leq C_1(\ln d)(\ln e) + C_2$. Also, at most $(m + 1)(n + 1)$ additions are required and, by the proof of Theorem 3.2, the time for each addition is $\leq C_3 \ln de + C_4 = C_3 (\ln d + \ln e) + C_4$. But $(m + 1)(n + 1)$ $(C_1(\ln d)(\ln e) + C_3 (\ln d + \ln e) + C_4) = O((\ln d)(\ln e)mn)$.

Theorem 3.6. The time to compute P/Q for $Q \in U(d, m)$ and $P/Q \in U(e, n)$ is $O((\ln d)(\ln e)mn)$.

Proof. At most $n + 1$ coefficient divisions are required. By Theorem 2.3, the time for each division is $\leq C_1(\ln d)(\ln e) + C_2$, and $(n + 1)$ $(C_1(\ln d)$ $(\ln e) + C_2) = O((\ln d)(\ln e)mn)$. The other required arithmetic is essentially the same as in multiplying P/Q by Q.

We now begin an analysis of univariate polynomial g.c.d. algorithms by considering the content and primitive part algorithms.

Theorem 3.7. The time to compute cont (P) for $P \in U(d, m)$ and norm (pp(E)) $\leq e$ is $O((\ln d)(\ln e)m)$.

Proof. Let $P(x) = \Sigma_{i=1}^{k} a_i x^{e_i}$ where $e_1 > e_2 \ldots > e_k$ and each $a_i \neq 0$. Let $d_1 = |a_1|$ and $d_{i+1} = \gcd(d_i, |a_{i+1}|)$ for $1 \leq i \leq k \leq 1$. Then cont (P) = d_k. $0 < d_i \leq |a_i| \leq$ norm (P) $\leq d$ for all i. Also, cont (P) $\leq d_i$ for all i. Hence $\max\{d_i, |a_{i+1}|\}/d_{i+1} \leq$ norm (P)/cont (P) = norm (pp(P)) $\leq e$ for $1 \leq i \leq k - 1$. By Theorem 2.5, the time for the $k - 1$ g.c.d.'s is $\leq (k - 1)(C_1(\ln d)(\ln e) + C_2) = O((\ln d)(\ln e)m)$ since $k - 1 \leq m$.

Theorem 3.8. The time to compute pp(P) for $P \in U(d, m)$ and norm (pp(P)) $\leq e$ is $O((\ln d)(\ln e)m)$.

Proof. To compute pp(P), we first compute cont (P), then divide P by cont (P). By Theorem 3.7 we need only show that the time for the division is $O((\ln d)(\ln e)m)$. The division requires at most $m + 1$ integer divisions, and in each of these the divisor is cont (P) $\leq d$ while the quotient is a coefficient of pp (P), hence bounded by e. Now apply Theorem 2.2.

Corollary 3.9. The time to compute either cont (P) or pp (P) for $P \in U(d, m)$ is $O((\ln d)^2 m)$.

Proof. Apply Theorems 3.7 and 3.8, noting that norm (pp(P)) $\leq$ norm (P).

Next we shall study the time required to compute a reduced polynomial reaminder sequence over the integers. For this purpose the following theorem on the standardized Euclidean remainder, $\overline{\mathfrak{R}}(P, Q)$ is helpful.

Theorem 3.10. Let $t(P, Q)$ be the time to compute $\bar{\mathfrak{R}}(P, Q)$. Let $T(m, n, e) = \max \{t(P, Q): \deg(P) = m \ \& \ \deg(Q) = n \ \& \ m \geq n > 0 \ \& \ \text{norm}(P) \leq e \ \& \ \text{norm}(Q) \leq e\}$. There is a constant C such that $T(m, n, e) \leq C(m + n)(m - n + 2)^2 (\ln e)^2$ for all sufficiently large e.

Proof. Let $P_1, P_2, \ldots, P_{m-n+2}$ be the sequence of polynomials such that $P_1 = P$, $P_{i+1} = \rho(P_i, Q)$ if $\deg(P_i) \geq n$, and $P_{i+1} = L(Q) \cdot P_i$ if $\deg(P) < n$. Then $\bar{\mathfrak{R}}(P, Q) = P_{m-n+2}$.

Let M_i be the $i + 1$ by $m + 1$ matrix

$$M_i = \begin{pmatrix} b_n & b_{n-1} & b_{n-2} & \cdots & b_0 & 0 & 0 & \cdots & 0 \\ 0 & b_n & b_{n-1} & \cdots & b_1 & b_0 & 0 & \cdots & 0 \\ 0 & 0 & b_n & \cdots & b_2 & b_1 & b_0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_m & a_{m-1} & a_{m-2} & \cdots & \cdots & \cdots & \cdots & \cdots & a_0 \end{pmatrix}.$$

It is easy to see that P_{i+1} is the associated polynomial of M_i. Since the Euclidean norm of each row of M_i is at most e, Hadamard's theorem implies that the coefficients of P_{i+1} are bounded by e^{i+1}. At most $m + n$ multiplications are required to compute P_{i+1} from P_i and the time for each multiplication is $\leq C_1(\ln e)(\ln e^{i+1}) + C_2 = C_1(i + 1)(\ln e)^2 + C_2 \leq C_1(m - n + 2)(\ln e)^2 + C_2$. The time for all multiplications is therefore bounded by $(m - n + 1)(m + n)(C_1(m - n + 2)(\ln e)^2 + C_2)$. At most m additions are required to compute P_{i+1} from P_i, and the time for each is $\leq C_3(\ln e^{i+1}) + C_4 \leq C_3(m - n + 2)$

$(\ln e) + C_4$. So the time for all additions is $\leq (m - n + 1)m\,(C_3(m - n + 2)(\ln e) + C_4)$. The validity of the theorem is now evident.

In [6] we defined a p.r.s. $P_1, P_2, \ldots, P_k$ to be normal in case $n_i - n_{i+1} = 1$ for $2 \leq i \leq k - 1$, where $n_i = \deg(P_i)$. It was shown that, among other nice properties, a normal reduced p.r.s. agrees, to within signs, with the associated subresultant p.r.s. It was stated there that, empirically, almost all p.r.s.'s are normal. And, as Knuth observes in [13], this is also true in a definite mathematical sense. Notice that if P_1 and P_2 have a g.c.d. of degree greater than one and if $P_1, P_2, \ldots, P_k$ is the complete p.r.s., then P_{k-1} is an associate of the g.c.d. and $P_k = 0$. Hence $n_{k-1} - n_k > 1$, and $P_1, P_2, \ldots, P_k$ is not normal.

Since, in practice, g.c.d.'s of degree greater than one will occur frequently, our computing time analysis, to be useful, must not be restricted to a normal p.r.s. So we now define a p.r.s. $P_1, P_2, \ldots, P_k$ to be <u>weakly normal</u> in case $\deg(P_i) - \deg(P_{i+1}) = 1$ for $2 \leq i \leq k - 2$. We can now make the stronger assertion that, for any r, almost all complete p.r.s.'s $P_1, P_2, \ldots, P_k$ for which $\deg(\gcd(P_1, P_2)) = r$ are weakly normal.

In the following we will bound the time to compute a complete weakly normal reduced p.r.s. For convenience, we will also say that (P, Q) is <u>weakly normal</u> in case any complete p.r.s. $P_1, P_2, \ldots, P_k$ for which $P_1 = P$ and $P_2 = Q$ is weakly normal.

By Theorem 1, part (b), of [6], we have the following theorem.

Theorem 3.11. Let $P_1, P_2, \ldots, P_r$ be a complete weakly normal reduced p.r.s. Let $P_1, P_2, S_3, \ldots, S_r$ be the associated complete subresultant p.r.s. Then $P_i = S_i$ if i is even or $n_1 - n_2$ is odd, and $P_i = -S_i$ otherwise, for $3 \leq i \leq r$.

The next theorem bounds the coefficients of a weakly normal complete reduced p.r.s.

Theorem 3.12. Let $P_1, P_2, \ldots, P_r$ be a complete weakly normal reduced p.r.s. such that $\deg(P_1) = m$, $\deg(P_2) = n$, $\text{norm}(P_1) \leq e$ and $\text{norm}(P_2) \leq e$. Then the coefficients of P_i are bounded by $e^{m-n+2i-4}$ for $3 \leq i \leq r$.

Proof. By weak normality, $\deg(P_i) = n - i + 2$ for $2 \leq i \leq r - 1$ and, by Theorem 3.11, $P_i = \pm S_i$ for $3 \leq i \leq r$. The coefficients of S_i are, by definition, determinants of order $(m + n) - 2(\deg(P_{i-1}) - 1) = m - n + 2i - 4$ of submatrices of the Sylvester matrix of P_1 and P_2. By Hadamard's theorem they are therefore bounded by $e^{m+n-2i+4}$.

Theorem 3.13. The time to compute the complete reduced p.r.s. for P and Q such that $\deg(P) = m$, $\deg(Q) = n$, $m \geq n > 0$, $\deg(\gcd(P, Q)) = k$, $\text{norm}(P) \leq e$, $\text{norm}(Q) \leq e$, and (P, Q) is weakly normal is $O(((m + n)(m - n + 1)^2 + (m + n - 2k + 2)^2 (n^2 - (k - 1)^2))(\ln e)^2)$.

Proof. Let $P_1, P_2, \ldots, P_r$ be the complete reduced p.r.s.

By Theorem 3.10, the time to compute $P_3 = \overline{\mathcal{R}}(P_1, P_2)$ is $O((m + n)(m - n + 1)^2 (\ln e)^2)$. This completes the proof if $r = 3$. Suppose $r \geq 4$.

For $i \geq 2$, let $P'_i = \rho(P_i, P_{i+1})$. Then $\bar{\mathcal{R}}(P_i, P_{i+1})$ is either $\rho(P'_i, P_{i+1})$ or $L(P_{i+1}) \cdot P'_i$. $\deg(P_{i+1}) < \deg(P_i) = n - i + 2$ so at most $2(n - i + 2)$ multiplications are required to compute P'_i. By Theorem 3.12, the computing time for each is $\leq C_1(\ln e^{m-n+2i-4})(\ln e^{m-n+2i-2}) + C_2 \leq C_1(m - n + 2i - 2)^2 (\ln e)^2 + C_2$. The coefficients of P'_i are bounded by $2e^{2m-2n+4i-6} < e^{2(m-n+2i-2)}$. It follows that the time for all multiplications in computing $\bar{\mathcal{R}}(P_i, P_{i+1})$ has a bound of the form $(n - i + 2)(C_1(m - n + 2i - 2)^2 (\ln e)^2 + C_2)$. Such a bound continues to hold when additions are also considered.

Now $P_4 = \bar{\mathcal{R}}(P_2, P_3)/(L(P_2))^{m-n+1}$. The successive powers $(L(P_2))^j$, $2 \leq j \leq m - n$ can be computed in $(m - n + 1)(C_3(m - n + 1)(\ln e)^2 + C_4)$. The coefficients of P_4 are bounded by e^{m-n+4} and $|(L(P_2))^{m-n+1}| \leq e^{m-n+1}$. Hence $\bar{\mathcal{R}}(P_2, P_3)$ can be divided by $(L(P_2))^{m-n+1}$ in $(n - 2)(C_5(m - n + 1)(m - n + 4)(\ln e)^2 + C_6)$. The total time to compute P_4 from $\bar{\mathcal{R}}(P_2, P_3)$ is therefore $\leq C_7(n - 2)(m - n + 1)(m - n + 4)(\ln e)^2 + C_8 m$. Adding to this the time above to compute $\bar{\mathcal{R}}(P_2, P_3)$ gives a bound of the form $(m + n)(C_5(m - n + 1)(m - n + 4)(\ln e)^2 + C_6)$, which is $O((m + n)(m - n + 1)^2(\ln e)^2)$. This proves the theorem for $r = 4$. Suppose $r > 4$.

For $i \geq 3$, $P_{i+2} = \bar{\mathcal{R}}(P_i, P_{i+1})/(L(P_i))^2$. Since $\deg(P_{i+2}) = n - i$, P_{i+2} can be computed from $\bar{\mathcal{R}}(P_i, P_{i+1})$ in $(n - i)(C_3(m - n + 2i)^2 (\ln e)^2 + C_4)$. Altogether, the time to compute P_{i+2} from P_i and P_{i+1} has a bound of the form $(n - i + 2)(C_1(m - n + 2i)^2(\ln e)^2 + C_2)$. Now $r = n - k + 3$ if $k > 0$ and $r = n + 2$ if $k = 0$, so $r \leq n - k + 3$. The total time to compute P_5, P_6, ..., P_r is therefore bounded by $(C_3(m - n + 2(n - k + 1))^2(\ln e)^2 + C_4)$

$\Sigma_{i=3}^{n-k+1} (n - i) \leq (C_3(m + n - 2k + 2)^2(\ln e)^2 + C_4) \Sigma_{i=3}^{n-k+1} (n - i) = O((m + n - 2k + 2)^2(n^2 - (k - 1)^2)(\ln e)^2)$ since $\Sigma_{i=3}^{n-k+1} (n - i) \leq \frac{1}{2}((n - 2)(n - 3) - (k - 1)^2) = O(n^2 - (k - 1)^2)$.

Theorem 3.14. The time to compute gcd (P, Q) by the reduced p.r.s. algorithm such that deg (P) = m, deg (Q) = n, $m \geq n > 0$, deg (gcd (P, Q)) = k, norm (P) $\leq$ e, norm (Q) $\leq$ e, and (P, Q) is weakly normal is $O(((m + n)(m - n + 1)^2 + (m + n - 2k + 2)^2 (n^2 - (k - 1)^2))(\ln e)^2)$.

Proof. The required computations are as follows:

(1) a = cont (P), b = cont (Q), P_1 = pp(P), P_2 = pp(Q), c = gcd (a, b).

(2) Compute the complete reduced p.r.s. $P_1, P_2, \ldots, P_r$.

(3) If $P_r = 0$, compute $R = c \cdot pp(P_{r-1})$.

By Theorem 2.6 and Corollary 3.9, the computations (1) can be performed in $O((m + n)(\ln e)^2)$. The time for (2) is $O(((m + n)(m - n + 1)^2 + (m + n - 2k + 2)^2 (n^2 - (k - 1)^2))(\ln e)^2)$ by Theorem 3.13. If $P_r = 0$, then $k > 0$ and $r - 1 = n - k + 2$. By Theorem 3.12, the coefficients of P_{r-1} are bounded by e^{m+n-2k}. Next we notice that Corollary 3.9 would still hold under the weaker assumptions that deg (P) $\leq$ m and that the coefficients of P are bounded by d. Since $\deg(P_{r-1}) = k$, $pp(P_{r-1})$ can be computed in $O(k(m + n - 2k)^2(\ln e)^2)$. The multiplication of $pp(P_{r-1})$ by c can be done in $O(k(m + n - 2k)(\ln e)^2)$. So the time for (3) is $O(k(m + n - 2k)^2(\ln e)^2)$. But $k \leq n^2 - (k - 1)^2$ since $k \leq n$.

By eliminating the variables n and k, singly or together, we obtain the following three corollaries of Theorem 3.14.

Corollary 3.15. The time to compute $\gcd(P, Q)$ by the reduced p.r.s. algorithm such that $\deg(P) = m$, $\deg(Q) = n$, $m \geq n > 0$, $\text{norm}(P) \leq e$, $\text{norm}(Q) \leq e$, and (P, Q) is weakly normal is $O(((m + n)(m - n + 1)^2 + n^2 (m + n)^2)(\ln e)^2)$.

Proof. Use Theorem 3.14, noting that $(m + n - 2k + 2)^2 \leq (m + n + 2)^2 = O((m + n)^2)$ and $n^2 - (k - 1)^2 \leq n^2$.

Corollary 3.16. The time to compute $\gcd(P, Q)$ by the reduced p.r.s. algorithm such that $P, Q \in U(e, m)$, $\deg(\gcd(P, Q)) = k$, and (P, Q) is weakly mornal is $O((m^3 + (m - k + 1)^3(m + k)(\ln e)^2)$.

Proof. Use Theorem 3.14, noting that $(m + n)(m - n + 1)^2 \leq 2m(m + 1)^2 = O(m^3)$ and $(m + n - 2k + 2)^2 (n^2 - (k - 1)^2) \leq 4(m - (k - 1))^2(m^2 - (k - 1)^2) = 4(m - k + 1)^3(m + k - 1) = O((m - k + 1)^3 (m + k))$.

Corollary 3.17. The time to compute $\gcd(P, Q)$ by the reduced p.r.s. algorithm such that $P, Q \in U(e, m)$ and (P, Q) is weakly normal is $O(m^4(\ln e)^2)$.

Proof. Use Corollary 3.16, noting that $m^3 + (m - k + 1)^3 (m + k) \leq m^3 + 2m(m + 1)^3 = O(m^4)$.

Having now analyzed the computing time for the reduced p.r.s. algorithm, let us observe that Theorems 3.12, 3.13 and 3.14, and Corollaries 3.15, 3.16 and 3.17 still hold if we replace "reduced p.r.s." everywhere by "primitive p.r.s.". Let $Q_1, Q_2, \ldots, Q_r$ be the complete primitive p.r.s. Then Q_i is a divisor

and associate of P_i for all i. Hence 3.12 still holds. In 3.13 we now have to compute $pp(\overline{\mathfrak{R}}(Q_i, Q_{i+1}))$ in place of dividing $\overline{\mathfrak{R}}(P_i, P_{i+1})$ by $(L(P_i))^{\delta_{i-1}+1}$. The computing time bounds are of the same order for the two operations. In Theorem 3.14, the computation $pp(Q_{r-1})$ is omitted, but this does not affect the bound of the theorem.

It would be interesting to know whether Theorems 3.13 and 3.14, stated for a primitive p.r.s. without the assumption of weak normality, would still hold. It seems likely that they would, but I have not attempted a proof. It also seems quite unlikely that they hold for a reduced p.r.s. without the weak normality assumption.

4. A Congruence Arithmetic G.C.D. Algorithm for Univariate Polynomials

In this section we describe a congruence arithmetic algorithm for computing the g.c.d. of two polynomials with integer coefficients. Several theorems are then proved to show that the algorithm does what it is supposed to do. In Section 5 we will analyze the computing time for the algorithm, showing that, on the average, it is much faster than previous algorithms.

For any prime number p, let GF(p) be the finite field with p elements $0, 1, \ldots, p-1$ and let φ_p be the unique homomorphism of the integers, I, onto GF(p), so that $\varphi_p(i) = i$ for $0 \le i < p$. Let φ_p^* be the homomorphism from $I[x]$ onto $GF(p)[x]$ induced by φ_p. That is, $\varphi_p^*(\Sigma_{i=0}^{n} a_i x^i) = \Sigma_{i=0}^{n} \varphi_p(a_i) \cdot x^i$. We shall usually just write φ_p in place of φ_p^*.

Given two polynomials with integer coefficients, P_1 and P_2, $\deg(P_1) \geq \deg(P_2) > 0$, the algorithm to be described does the following: (a) decides whether $\deg(\gcd(P_1, P_2)) = 0$, and (b) if not, computes the last non-zero term, S, of the complete subresultant p.r.s. for P_1 and P_2 (this being an associate of $\gcd(P_1, P_2)$). A complete g.c.d algorithm is constucted from this algorithm in an obvious way.

Let $P_1, P_2, S_3, \ldots, S_r$ be the complete subresultant p.r.s. and let S be the last non-zero subresultant. The general idea is to compute $\varphi_p(S)$ for a sufficiently large number of primes p that S can then be computed by the Chinese remainder theorem algorithm. The required number of primes is determined by Hadamard's theorem as a function of $\deg(P_i)$ and $\text{norm}(P_i)$ for $i = 1$ and 2.

In addition to P_1 and P_2, the algorithm requires as input an infinite sequence of distinct prime numbers, $p_1, p_2, p_3, \ldots$, and an integer h such that $p_i \geq 2^h$ for all i. In practice the p_i would probably be the first few hundred primes greater than 2^h, where 2^h is about half the largest integer which can be stored in one computer word.

Following is a complete description of the algorithm.

Congruence Arithmetic Subresultant Algorithm

(1) Compute $d = \text{norm}(P_1)$, $e = \text{norm}(P_2)$, $m = \deg(P_1)$ and $n = \deg(P_2)$. Compute the least integer r such that $2^r \geq d$ and the least integer s such that $2^s \geq e$. Compute $t = ms + nr$ and $u = [t/h] + 1$.

(2) Set $I = 0$ and $M = \mathcal{S} = \mathcal{P} = ()$.

(3) Select the next prime, p.

(4) Compute $P_i^* = \varphi_p(P_i)$ for $i = 1, 2$. If $\deg(P_1^*) < m$ or $\deg(P_2^*) < n$, go to (3).

(5) Compute the complete reduced p.r.s. $P_1^*, P_2^*, \ldots, P_k^*$ of P_1^* and P_2^* over GF(p).

(6) If $P_k^* \neq 0$, terminate with indication that $\deg(\gcd(P_1, P_2)) = 0$.

(7) Set $N = (n_1^*, n_2^*, \ldots, n_k^*)$, where $n_i^* = \deg(P_i^*)$.

(8) Compute the subresultant associate, S, of P_{k-1}^*, using Theorem 1 of [6].

(9) If $N < M$, go to (3).

(10) If $N > M$, set $\mathcal{S} = (S)$, $\mathcal{P} = (p)$, $I = 1$ and $M = N$, then go to (3).

(11) Adjoin S to $\mathcal{S}$ and p to $\mathcal{P}$. Set $I = I + 1$. If $I < u$, go to (3).

(12) Let $\mathcal{P} = (p_1, p_2, \ldots, p_u)$, $\mathcal{S} = (S_1, S_2, \ldots, S_u)$. Each S_i is a polynomial of degree $k > 0$. By $k + 1$ applications of the Chinese remainder theorem, compute the unique polynomial S of degree k such that $\varphi_{p_i}(S) = S_i$ for $1 \leq i \leq u$ such that the coefficients of S are bounded by $\frac{1}{2} p_1, p_2 \cdots p_u$.

Here are some remarks explaining the above algorithm. In (1) we are applying Hadamard's theorem to obtain an upper bound, u, for the number of primes p for which $\varphi_p(S)$ will be needed in order to determine S. By Hadamard's theorem, the coefficients are bounded by $d^n e^m \leq 2^{ms+nr} = 2^t < 2^{hu} < p_1, p_2 \cdots p_u$ for any u primes greater than 2^h.

$\mathcal{P}$ is a list of primes $(p_1, p_2, \ldots, p_k)$ which have been used but not discarded. $\mathcal{S}$ is a corresponding list of polynomials $(S_1, S_2, \ldots, S_k)$ for which, hopefully, $S_i = \varphi_{p_i}(S)$. If it later turns out that $S_i \neq \varphi_{p_i}(S)$, both p_i and S_i are discarded. The value of I is always k, the number of primes in the list $\mathcal{P}$.

For each prime p the sequence of degrees $(n^*_1, n^*_2, \ldots, n^*_k)$ is computed. All such sequences are ordered lexicographically. The value of the variable M is always the maximum of all degree sequences which have been computed. Since (), the null sequence, is least among all sequences, M is initialized to (). Any prime whose degree sequence proves to be non-maximal is discarded. It will be proved below that if any u distinct primes all have the same degree sequence, then their common degree sequence is the degree sequence of the complete reduced p.r.s. for P_1 and P_2 over the integers, and hence is maximal. It will also be shown that if p has a maximal degree sequence then the complete reduced p.r.s. over GF(p) is the homomorphic image under φ_p of the complete reduced p.r.s. over the integers.

The following theorem and its corollary justify Step (6).

Theorem 4.1. Let P and Q be non-zero polynomials over I, p a prime such that $\varphi_p(\mathfrak{L}(P)) \neq 0$ and $\varphi_p(\mathfrak{L}(Q)) \neq 0$. Then $\deg(\gcd(\varphi_p(P), \varphi_p(Q))) \geq \deg(\gcd(P, Q))$.

Proof. Let $R = \gcd(P, Q)$. Then $P = R \cdot P_1$ and $Q = R \cdot Q_1$. Since φ_p is a homomorphism, $\varphi_p(P) = \varphi_p(R) \cdot \varphi_p(P_1)$ and $\varphi_p(Q) = \varphi_p(R) \cdot \varphi_p(Q_1)$. Hence $\varphi_p(R)$ is a common divisor in $GF(p)[x]$ of $\varphi_p(P)$ and $\varphi_p(Q)$. Also, $\mathfrak{L}(P) = \mathfrak{L}(R) \cdot \mathfrak{L}(P_1)$ so $0 \neq \varphi_p(\mathfrak{L}(P)) = \varphi_p(\mathfrak{L}(R_1)) \cdot \varphi_p(\mathfrak{L}(P_1)$ and $\varphi_p(\mathfrak{L}(R)) \neq 0$. So $\deg(\gcd(\varphi_p(P), \varphi_p(Q))) \geq \deg(\varphi_p(R)) = \deg(R)$.

Corollary 4.2. Let P and Q be polynomials over I, $\deg(P) \geq \deg(Q) > 0$. Let p be a prime such that $\varphi_p(\mathfrak{L}(P)) \neq 0$ and $\varphi_p(\mathfrak{L}(Q)) \neq 0$. Let $P_1, P_2, \ldots, P_k$ be a complete p.r.s. over $GF(p)$ such that $P_1 = \varphi_p(P)$ and $P_2 = \varphi_p(Q)$. If $P_k \neq 0$, then $\deg(\gcd(P, Q)) = 0$.

Proof. If $P_k \neq 0$, then $\deg(\gcd(P_1, P_2)) = 0$. Use Theorem 4.1.

Lemma 4.3. Let P and Q be polynomials over I, $\deg(P) \geq \deg(Q) > 0$. Let p be a prime such that $\varphi_p(\mathfrak{L}(P)) \neq 0$ and $\varphi_p(\mathfrak{L}(Q)) \neq 0$. Then $\varphi_p(\bar{\mathfrak{R}}(P, Q)) = \bar{\mathfrak{R}}(\varphi_p(P), \varphi_p(Q))$.

Proof. Let $m = \deg(P)$, $n = \deg(Q)$; $R = \bar{\mathfrak{R}}(P, Q)$. R is uniquely determined by the condition that $(\mathfrak{L}(Q))^{m-n+1} \cdot P = Q \cdot S + R$ for some S, with $\deg(R) < n$. Since φ_p is a homomorphism, $(\varphi_p(\mathfrak{L}(Q)))^{m-n+1} \cdot \varphi_p(P) = \varphi_p(Q) \cdot \varphi_p(S) + \varphi_p(R)$. But $\varphi_p(\mathfrak{L}(Q)) = \mathfrak{L}(\varphi_p)Q))$, $\deg(\varphi_p(P)) = m$, and $\deg(\varphi_p(Q)) = n$. Also, $\deg(\varphi_p(R)) \leq \deg(R) < n$. So $\varphi_p(R) = \bar{\mathfrak{R}}(\varphi_p(P), \varphi_p(Q))$.

Theorem 4.4. Let $P_1, P_2, \ldots, P_k$ be a reduced p.r.s. over I, p a prime such that $\deg(\varphi_p(p_i)) = \deg(P_i)$ for $1 \le i \le k-1$. Let $P^*_1, P^*_2, \ldots, P^*_k$ be a reduced p.r.s. over GF(p) such that $P^*_1 = \varphi_p(P_1)$ and $P^*_2 = \varphi_p(P_2)$. Then $P^*_i = \varphi_p(P_i)$ for $1 \le i \le k$.

Proof. $P_3 = \overline{\mathcal{R}}(P_1, P_2)$ and $P^*_3 = \overline{\mathcal{R}}(P^*_1, P^*_2) = \overline{R}(\varphi_p(P_1), \varphi_p(P_2))$, so $P^*_3 = \varphi_p(P_3)$ by Lemma 4.3, since $\deg(\varphi_p(P_i)) = \deg(P_i)$ implies $\varphi_p(\mathcal{L}(P_i)) \neq 0$. Let $n_i = \deg(P_i)$ and $\delta_i = n_i - n_{i+1}$ for all i. Assume $P^*_i = \varphi_p(P_i)$ and $P^*_{i+1} = \varphi_p(P_{i+1})$ where $2 \le i \le k-2$. Then $P^*_{i+2} = \overline{\mathcal{R}}(P^*_i, P^*_{i+1})/(\mathcal{L}(P^*_{i-1}))^{\delta_{i-1}+1} = \varphi_p(\overline{\mathcal{R}}(P_i, P_{i+1}))/(\varphi_p(\mathcal{L}(P_{i-1})))^{\delta_{i-1}+1} = \varphi_p(\overline{\mathcal{R}}(P_i, P_{i+1})/(\mathcal{L}(P_{i-1}))^{\delta_{i-1}+1}) = \varphi_p(P_{i+2})$ by Lemma 4.3. By induction, $P^*_i = \varphi_p(P_i)$ for all i.

Theorem 4.5. Let $P_1, P_2, \ldots, P_k$ be a reduced p.r.s. over I. Let p be a prime. Let $P^*_1, P^*_2, \ldots, P^*_k$ be a reduced p.r.s. over GF(p) such that $P^*_1 = \varphi_p(P_1)$, $P^*_2 = \varphi_p(P_2)$ and $\deg(P_i) = \deg(P^*_i)$ for $1 \le i \le k-1$. Then $P^*_i = \varphi_p(P_i)$ for $1 \le i \le k$.

Proof. By induction on k. For $k = 3$ the theorem is an immediate consequence of Theorem 4.4 applied with $k = 3$. Assume Theorem 4.5 holds for $k = j$, and assume its hypotheses for $k = j + 1$. Then $P^*_i = \varphi_p(P_i)$ for $1 \le i \le j$ by induction hypothesis. Hence $\deg(P_i) = \deg(\varphi_p(P_i))$ for $1 \le i \le j$. Hence $P^*_i = \varphi_p(P_i)$ for $1 \le i \le j+1$ by Theorem 4.4. applied with $k = j + 1$.

The next theorem shows that if u primes all produce the same degree sequence, then that common degree sequence is the degree sequence over the integers. By the previous theorem, therefore, each prime produces a homomorphic image of the reduced p.r.s. over the integers.

Theorem 4.6. Let $P_1, P_2, \ldots, P_k$ be a complete reduced p.r.s. over the integers. Let $\deg(P_1) = m$, $\deg(P_2) = n$, norm $(P_1) \leq d$, norm $(P_2) \leq e$. Let $n_i = \deg(P_i)$. Let $p_1, p_2, \ldots, p_u$ be distinct primes such that $\Pi_{i=1}^{u} p_i > d^n e^m$. Let $\nu_1, \nu_2, \ldots, \nu_r$ be such that for every i, $1 \leq i \leq u$, the complete reduced p.r.s. over $GF(p_i)$ for $\varphi_{p_i}(P_1)$ and $\varphi_{p_i}(P_2)$ is a sequence $P_1^{(i)}, P_2^{(i)}, \ldots, P_r^{(i)}$ such that $\nu_j = \deg(P_j^{(i)})$ for $1 \leq j \leq r$, $\nu_1 = n_1$ and $\nu_2 = n_2$. Then $r = k$ and $\nu_i = n_i$ for $1 \leq i \leq k$.

Proof. Assume $r \geq t$ and $\nu_j = n_j$ for $1 \leq j \leq t$. This holds by assumption for $t = 2$. We will show that if it holds for t and $t < k$, then it holds for $t + 1$.

Let $S_1, S_2, \ldots, S_k$ be the complete subresultant p.r.s. over the integers such that $S_1 = P_1$ and $S_2 = P_2$. By Hadamard's theorem we know that the coefficients of all S_i are bounded by $d^n e^m$. Now $\nu_t = n_t > n_k = 0$, so $r \geq t + 1$. $\Pi_{i=1}^{u} p_i$ is not a divisor of $\mathfrak{L}(S_{t+1})$ so, for some j, p_j is not a divisor of $\mathfrak{L}(S_{t+1})$. By Theorem 1 of [6], $P_{t+1} = \Pi_{i=2}^{t-1} c_i^{\delta_{i-1}(\delta_i^{-1})} S_{t+1}$, where $\delta_i = n_i - n_{i+1}$ and $c_i = \mathfrak{L}(P_i)$ for $2 \leq i \leq t - 1$. By induction hypothesis, $\nu_i = n_i$ for $1 \leq i \leq t$. Hence $n_i = \deg(P_i^{(j)})$ for $1 \leq i \leq t$. By Theorem 4.5, $P_i^{(j)} = \varphi_{p_j}(P_i)$ for $1 \leq i \leq t + 1$. So $\deg(P_i) = \deg(\varphi_{p_j}(P_i))$ for $1 \leq i \leq t$. Hence $\varphi_{p_j}(\mathfrak{L}(P_i)) = \varphi_{p_j}(c_i) \neq 0$ for $1 \leq i \leq t$. So p_j is not a divisor of $\Pi_{i=2}^{t-1} c_i^{\delta_{i-1}(\delta_i^{-1})}$. So p_j is not a divisor of $\mathfrak{L}(P_{t+1})$, i.e., $\nu_{t+1} = n_{t+1}$.

By induction we now have $r \geq k$ and $\nu_i = n_i$ for $1 \leq i \leq k$. This implies $\nu_k = n_k = 0$ and therefore $r = k$.

Let $P_1, P_2, \ldots, P_k$ be the complete reduced p.r.s. over I for the two polynomials, P_1 and P_2, which are inputs to the algorithm. Let S_{k-1} be the subresultant associate of P_{k-1}. The next theorem now shows that if Step (12) of the algorithm is ever reached then, for each p_i in $\mathscr{P}$, $S_i = \varphi_{p_i}(S_{k-1})$.

Theorem 4.7. Let $P_1, P_2, \ldots, P_k$ be a complete reduced p.r.s. over I. Let $P_i^* = \varphi_p(P_i)$ and assume $\deg(P_i^*) = \deg(P_i)$ for $1 \leq i \leq k$. Let $S_1, S_2, \ldots, S_k$ be the subresultant p.r.s. over I such that $S = P_1$ and $S_2 = P_2$. Let $S_{k-1}^* = (-1)^{\tau_{k-1}} P_{k-1}^* / \Pi_{i=2}^{k-3} L(P_i^*)^{\delta_{i-1}(\delta_i - 1)}$, where $n_i = \deg(P_i^*)$, $\delta_i = n_i - n_{i+1}$ and $\tau_k = \Sigma_{i=1}^{k-3} n_i n_{i+1} + (n_1 + k - 1)(n_{k-2} + 1)$. Then $S_{k-1}^* = \varphi_p(S_{k-1})$.

Proof. Since $\deg(P_i) = \deg(P_i^*)$, $\mathfrak{L}(P_i^*) = \mathfrak{L}(\varphi_p(P_i)) = \varphi_p(\mathfrak{L}(P_i))$. Now apply Theorem 1 of [6], and use the homomorphism property of φ_p.

We still have to show that if Step (12) is reached, then S_{k-1} is an associate of $\gcd(P_1, P_2)$, i.e. that $S_k = 0$. But this follows easily from the proof of Theorem 4.6.

Finally, we must show that the algorithm will eventually terminate. This is equivalent to showing that only a finite number of primes can ever be discarded by the algorithm in Steps (4), (9) and (10). But a prime is only discarded in case its degree sequence is non-maximal, i.e. in case it divides $\Pi_{i=1}^{k-1} c_i$ where $P_1, P_2, \ldots, P_k$ is the complete reduced p.r.s. and $c_i = \mathfrak{L}(P_i)$. In the next section we will take a closer look at the number of primes which can be discarded.

5. Analysis of the Congruence Arithmetic Algorithm

In analyzing computing times for the congruence arithmetic g.c.d. algorithm we will consider h to be a constant. h will ordinarily be in a range between 30 and 60, depending on the computer word length. Since there will then be a minimum of about $2^{h-1}/h \geq 10^7$ primes in the interval from 2^h to 2^{h+1}, we can safely ignore the size of the primes in our analysis and assume they are all single-precision. There will then be a fixed bound for the time required to perform any arithmetic operation in $GF(p_i)$ for all primes, p_i, encountered. Likewise, we may safely assume m, n, r, s, t and u are all single-precision integers.

Theorem 5.1. The time to compute norm (P) such that $P \in U(d, m)$ is $O(m(\ln d))$.

Proof. Obvious.

Theorem 5.2. The time to compute the least r such that $2^r \geq d$ is $O((\ln d)^2)$.

Proof. Let $d_0 = d - 1$, $d_{i+1} = [d_i/2]$. Let k be least such that $d_k = 0$. Then $r = k$. r divisions are required and the time for each is $O(\ln d)$. But $r = O(\ln d)$.

Theorem 5.3. The time to compute $\varphi_p(d)$ is $O(\ln d)$.

Theorem 5.4. The time to compute $\varphi_p(P)$ for $P \in U(d, m)$ is $O(m(\ln d))$.

Theorem 5.5. The time to compute the complete reduced p.r.s. $P_1^*, P_2^*, \ldots, P_k^*$ over GF(p) from P_1^* and P_2^* such that $\deg(P_1^*) = m$ and $\deg(P_2^*) = n$, $m \geq n > 0$, is $O(m^2)$.

Proof. Let $\deg(P_i^*) = n_i$, $\mathcal{L}(P_i^*) = c_i^*$, $\delta_i = n_i - n_{i+1}$ for $1 \leq i \leq k$, $\delta_0 = -1$. Then $P_{i+2}^* = \overline{\mathcal{R}}(P_i^*, P_{i+1}^*)/c_i^{*\,\delta_{i-1}+1}$ for $1 \leq i \leq k-2$. Clearly at most $3n_i(\delta_i + 1)$ operations in GF(p) are required to compute $\overline{\mathcal{R}}(P_i^*, P_{i+1}^*)$. $(c_i^{*\delta_{i-1}+1})^{-1}$ can be computed in $\delta_{i-1}+1$ operations and P_{i+2}^* can then be computed in at most $n_{i+2} + 1 \leq n_{i+1}$ operations. But $\Sigma_{i=1}^{k-2} 3n_i(\delta_i+1) \leq 3m \Sigma_{i=1}^{k-2}(\delta_i+1) \leq 6m \Sigma_{i=1}^{k-2} \delta_i \leq 6m^2$, $\Sigma_{i=1}^{k-2}(\delta_{i-1}+1) \leq 1 + 2\Sigma_{i=1}^{k-3} \delta_i \leq 1 + 2(m - n_{k-2}) \leq 2m$, and $\Sigma_{i=1}^{k-2} n_{i+2} \leq (k-2)m \leq m^2$. So the total number of operations is at most $7m^2 + 2m < 8m^2$.

Theorem 5.6. Given a complete reduced p.r.s. $P_1^*, P_2^*, \ldots, P_k^*$ over GF(p) such that $\deg(P_1^*) = m$, the time to compute $P_{k-1}^* / \Pi_{i=2}^{k-3} \mathcal{L}(P_i^*)^{\delta_{i-1}(\delta_i-1)}$ is $O(m^2)$.

Proof. $\Sigma_{i=2}^{k-3} \delta_{i-1}(\delta_i - 1) \leq \Sigma_{i=2}^{k-3} \delta_{i-1}\delta_i \leq (\Sigma_{i=2}^{k-3} \delta_i)^2 = (m - n_{k-2})^2$. So $d_{k-1} = \Pi_{i=2}^{k-3} \mathcal{L}(P_i^*)^{\delta_{i-1}(\delta_i-1)}$ can be computed in at most $(m - n_{k-2})^2$ operations. d_{k-1}^{-1} and $d_{k-1}^{-1} P_{k-1}^*$ can then be computed in $n_{k-1} + 2$ additional operations. But $(m - n_{k-2})^2 + n_{k-1} + 2 \leq (m^2 - mn_{k-2}) + (n_{k-2}+1) \leq m^2 - 2n_{k-2} + 1 < m^2$.

In Steps (9) and (10) of the algorithm, we have to compare two degree sequences M and N. The maximum length for such a degree sequence is $m + 2$, so the time for this operation is $O(m)$. This observation together with Theorems 5.4, 5.5 and 5.6 gives us the following theorem.

Theorem 5.7. For each prime p_i selected by the congruence arithmetic subresultant algorithm, the computing time is $O(m^2 + m(\ln d + \ln e))$.

We will show next that the number of primes selected by the algorithm is $O(n(n \ln d + m \ln e)$.

Theorem 5.8. Let $P_1, P_2, \ldots, P_k$ be a reduced p.r.s. over I. Let $S_1, S_2, \ldots, S_k$ be a subresultant p.r.s. over I such that $S_1 = P_1$ and $S_2 = P_2$. Let p be a prime and $1 \leq i \leq k$. If p divides $\mathcal{L}(P_i)$ then, for some $j \leq i$, p divides $\mathcal{L}(S_j)$.

Proof. The theorem clearly holds for $1 \leq i \leq 3$. Assume it holds for t where $3 \leq t < k$. Let $c_i = \mathcal{L}(P_i)$ and $d_i = \mathcal{L}(S_i)$. By Theorem 1 of [6], $c_{t+1} = \pm \Pi_{i=2}^{t-1} c_i^{\delta_{i-1}(\delta_i - 1)} d_{t+1}$. Suppose p divides c_{t+1} but not d_{t+1}. Then it divides c_j for some $j \leq t - 1$ and hence, by induction hypothesis, d_j. So the theorem holds for $t + 1$.

Theorem 5.9. Let $S_1, S_2, S_3, \ldots, S_k$ be a complete reduced p.r.s. over I with $\deg(S_1) = m$, $\deg(S_2) = n$, norm $(S_1) \leq d$ and norm $(S_2) \leq e$. Let $d_i = \mathcal{L}(S_i)$. Then $|\Pi_{i=1}^{k-1} d_i| \leq d^{n^2} e^{mn}$.

Proof. By Hadamard's theorem, $|d_i| < d^n e^m$ for all i, so $|\Pi_{i=3}^{k-1} d_i| \leq (d^n e^m)^{k-3}$. But $|d_1| \leq d$ and $|d_2| \leq e$, so $|\Pi_{i=1}^{k-1} d_i| \leq (d^n e^m)^{k-2} \leq (d^n e^m)^n$.

Theorem 5.10. The number of primes selected by the congruence arithmetic subresultant algorithm is $O(n(n \ln d + m \ln e))$.

Proof. Every prime discarded by the algorithm is a divisor of some $\mathfrak{L}(P_i)$ and hence, by Theorem 5.8, some $\mathfrak{L}(S_i) = d_i$, $1 \le i \le k - 1$. By Theorem 5.9, the product of all discarded primes is at most $d^{n^2} e^{mn}$. Since each prime is greater than 2^h, if N is the number of discarded primes we have $2^{Nh} < d^{n^2} e^{mn}$, hence $N < (n^2 \log_2 d + mn \log_2 e)/h$. So $N = O(n(n \ln d + m \ln e))$. The number of primes selected but not discarded is at most $u \le t/h + 1 \le (ms + nr)/h + 1 \le (m(s - 1) + n(r - 1) + m + n)/h + 1 < (m(\log_2 e + 1) + n(\log_2 d + 1))/h + 1 = O(n \ln d + m \ln e)$.

Putting all the pieces together, we have the following computing time bound for the entire process.

Theorem 5.11. The computing time for the congruence arithmetic subresultant g.c.d. algorithm such that $\deg(P_1) = m$, $\deg(P_2) = n$, $\text{norm}(P_1) \le d$ and $\text{norm}(P_2) \le e$ is $O(mn(m + \ln d + \ln e)(n \ln d + m \ln e))$.

Proof. By Theorems 5.7 and 5.10, the time to process all primes is $O(mn(m + \ln d + \ln e)(n \ln d + m \ln e))$. It therefore remains only to show that the same is true of the other parts of the algorithm. By Theorems 5.1 and 5.2 the computing time for Step (1) is $O(m \ln d + n \ln e + (\ln d)^2 + (\ln e)^2)$. In accordance with the remarks at the beginning of this section we will assume that each prime is $\le 2^{h+1}$. Hence $\ln(p_1 p_2 \cdots p_u) \le (h + 1)u = O(n \ln d + m \ln e)$ as in the proof of Theorem 5.10. By Theorem 2.7, the time for each of the $k + 1$ applications of the Chinese remainder theorem algorithm is $O((n \ln d + m \ln e)^2)$. Since $k + 1 = O(n)$, the computing time for Step (12) is $O(n(n \ln d + m \ln e)^2)$. Incorporating this algorithm into a complete g.c.d

algorithm, we must consider the times to compute $pp(P_1)$, $pp(P_2)$, $pp(S)$ and $c \cdot pp(S)$, where $c = gcd(cont(P_1), cont(P_2))$. The times to compute $pp(P_1)$ and $pp(P_2)$ are $O(m(\ln d)^2)$ and $O(n(\ln e)^2)$ by Corollary 3.9. Since the coefficients of S are bounded by $p_1 p_2 \cdots p_u$, Corollary 3.9 shows that the time to compute $pp(S)$ is $O(n(n \ln d + n \ln e)^2)$, the same as Step (12), and the time to multiply $pp(S)$ by c is then $O(n(\ln d + \ln e)(n \ln d + m \ln e)$.

The following two corollaries follow easily.

Corollary 5.12. The computing time for the congruence arithmetic subresultant g.c.d. algorithm such that $deg(P_1) = m$, $deg(P_2) = n$, $norm(P_1) \leq e$, and $norm(P_2) \leq e$ is $O(mn(m + n)(m + \ln e)(\ln e))$.

Corollary 5.13. The computing time for the congruence arithmetic subresultant g.c.d. algorithm such that $P_1, P_2 \in U(e, m)$ is $O(m^3(m + \ln e)(\ln e))$.

Comparing Corollaries 3.17 and 5.13, we see that the computing time bound for the reduced p.r.s. algorithm is larger than that for the congruence arithmetic algorithm by a ratio of $m(\ln e)/(m + \ln e)$, a ratio which grows indefinitely with m and e. Actually, the superiority of the congruence algorithm is much greater than this ratio indicates. We found that the number of primes discarded is $O(n(n \ln d + m \ln e))$ while the number of primes retained is only $O(n \ln d + n \ln e))$. This suggests that most primes are discarded whereas it is intuitively clear that, on the average, a prime will be discarded only on very rare occasions. If one chooses an integer N at random, the

probability that $\varphi_p(N) = 0$ is only $1/p$. Hence if the leading coefficients of the subresultant p.r.s. are random in an appropriate sense, the probability that p will be discarded is at most $1 - (1 - 1/p)^n \cong n/p$ since n is much smaller than p in all cases of interest. This reasoning suggests that the average number of primes selected is $O(n \ln d + m \ln e)$. Going back over the proof of Theorem 5.11, one can easily show that, under this hypothesis, the average computing time is $O(k(m + n)^2(\ln e)^2 + m^2 (m + n)(\ln e))$, where k is the degree of the g.c.d., hence $O(km^2(\ln e)^2 + m^3 (\ln e))$, hence $O(m^3(\ln e))$. If P_1 and P_2 are relatively prime there is a very high probability that p will prove it via Corollary 4.2 if n is much smaller than p. If we assume that the average number of primes selected in the relatively prime case is less than two (or any other fixed number) then we easily conclude that the average computing time for relatively prime polynomials is $O(m^2 + (m + n)(\ln e)^2)$, hence $O(m^2 + m (\ln e)^2)$.

The above algorithm can be made faster in two ways. The first way requires only a simple modification of Step (12). In Step (12), if $M = (n_1, n_2, \ldots, n_a)$, then $n_a = 0$ and n_{a-1} is the degree k of the g.c.d. If $a = 3$, then P_2 is the primitive associate of the g.c.d. and the Chinese remainder theorem is not needed. If $a > 3$, then S, the subresultant associate of the g.c.d., has coefficients bounded by $d^{(n-n_{a-2}+1)} e^{(m-n_{a-2}+1)}$, by Theorem 1 of [6] and Hadamard's theorem. Hence only $v = [((m - n_{a-2} + 1)s + (n - n_{a-2} + 1)r)/h] + 1$ primes are needed to determine S by the Chinese

remainder theorem. The last $u - v$ elements of $\mathscr{P}$ and $\mathscr{\delta}$ may be ignored. If, for example, $m = n = 2k$, then $n_{a-2} - 1 \geq k$ and $v = u/2$ approximately. By Theorem 2.7, the computing time for Step (12) will therefore be reduced by a factor of about 4.

The algorithm as thus revised computes an a priori bound, u, for the number of primes needed. After having processed u undiscarded primes it then determines k and adjusts downwards its original estimate for the number of primes needed. The second modification avoids this waste by determining the degrees $n_1, n_2, n_3, \ldots, n_a$ sequentially. $n_1 = m$ and $n_2 = n$ are initially given. Let $S_1, S_2, \ldots, S_a$ be the complete subresultant p.r.s. over the integers. The coefficients of S_3 are bounded by $d^{(n-n_2+1)}e^{(m-n_2+1)}$. Hence at most $u_3 = [((m - n_2 + 1)s + (n - n_2 + 1)r)/h] + 1$ primes are needed to determine S_3. Having accumulated u_3 undiscarded primes, n_3 is determined as m_3, where $M = (m_1, m_2, m_3, \ldots)$. If $n_3 = 0$, then $S_3 = 0$ and Step (12) is undertaken. Otherwise, $u_4 = [((m - n_3 + 1)s + (n - n_3 + 1)r)/h] + 1$ primes are needed to determine S_4. Continuing in this way one eventually goes to Step (12) with u_a primes in $\mathscr{P}$. The Chinese remainder theorem is then applied using u_{a-1} of these primes. If, for example, $m = n = 2k$, this modification will reduce by a factor of about 2 the time required to compute reduced p.r.s.'s and subresultant associates over fields $GF(p_i)$.

The univariate congruence arithmetic g.c.d. algorithm can be advantageously used as part of any multivariate g.c.d. algorithm since the calculation of a

multivariate g.c.d. requires the calculation of numerous univariate g.c.d.'s. Also, the methods employed in the univariate algorithm can be extended to multivariate algorithms. Viewed abstractly we wish to compute the g.c.d. of two polynomials, P_1 and P_2, over an integral domain $\mathfrak{I}$ (where $\mathfrak{I}$ itself may be a polynomial domain). We have at our disposal a sequence $\psi_1, \psi_2, \ldots$ of homomorphisms from $\mathfrak{I}$ to some integral domains J_i. Instead of computing a complete p.r.s. $P_1, P_2, P_3, \ldots, P_k$ over $\mathfrak{I}$, we compute a complete p.r.s. $P_1^{(i)}, P_2^{(i)}, P_3^{(i)}, \ldots$ over J_i such that $P_1^{(i)} = \psi_i^*(P_1)$ and $P_2^{(i)} = \psi_i^*(P_2)$ for $i = 1, 2, 3, \ldots, \psi_i^*$ being the homomorphism from $\mathfrak{I}[x]$ to $J_i[x]$ induced by ψ_i. If ψ_i^* preserves the degrees of $P_1, P_2, P_3 \ldots$, we have $P_j^{(i)} = \psi_i^*(P_j)$. One must then have a mechanism for discarding those ψ_i^* which do not preserve degrees. One must also be able to compute a bound u such that P_{k-1} can be determined from $\psi_1^*(P_{k-1})$, $\psi_2^*(P_{k-1}), \ldots, \psi_u^*(P_{k-1})$. If, for example, $\mathfrak{I} = I[y]$ one can let ψ_i be the evaluation homomorphism $\psi_i(P) = P(i)$ from $I[y]$ to I. In this case a bound u can be computed as a function of the degrees of the coefficients of P_1 and P_2, and interpolation replaces the Chinese remainder theorem for computing P_{k-1} from the $\psi_i^*(P_{k-1})$. Those ψ_i^* which produce non-maximal degree sequences are discarded as in the univariate algorithm above. One advantage of this choice of the ψ_i is that it leads to an algorithm which is recursive in the number of variables. To compute the complete reduced p.r.s. of $\psi_i^*(P_1)$ and $\psi_i^*(P_2)$ over I, the univariate congruence algorithm above can be used (since

only the degree sequence and the subresultant associate of the last non-zero term are needed). From an algorithm for polynomials in n variables, an algorithm for polynomials in $n + 1$ variables is thus obtained using the evaluation homomorphisms from $I[x_1, \ldots, x_{n+1}]$ to $I[x_1, \ldots, x_n]$.

ACKNOWLEDGEMENT

This research was supported in part by the Wisconsin Alumni Research Foundation and the University of Wisconsin Computing Center.

REFERENCES

1. Borosh, I. and Fraenkel, A. S. Exact Solutions of Linear Equations by Congruence Techniques. Math. Comp., Vol. 20, No. 93 (Jan. 1966), pp. 107-112.

2. Brown, W. S. The ALPAK System for Non-Numerical Algebra on a Digital Computer -- I: Polynomials in Several Variabels and Truncated Power Series with Polynomial Coefficients. B.S.T.J., Vol. 42, No. 3 (Sept. 1963), pp. 2081-2119.

3. Brown, W. S., Hyde, J. P., and Tague, B. A. The ALPAK System for Non-Numerical Algebra on a Digital Computer -- II: Rational Functions of Several Variables and Truncated Power Series with Rational Function Coefficients. B.S.T.J., Vol. 43, No. 1 (March 1964), pp. 785-804.

4. Hyde, J. P. The ALPAK System for Non-Numerical Algebra on a Digital Computer -- III: Systems of Linear Equations and a Class of Side Relations. B.S.T.J., Vol. 43, No. 4 (July 1964), pp. 1547-1562.

5. Collins, G. E. PM, A System for Polynomial Manipulation. C.A.C.M., Vol. 9, No. 8 (Aug. 1966), pp. 575-589.

6. Collins, G. E. Subresultants and Reduced Polynomial Remainder Sequences. J.A.C.M., Vol. 14, No. 1 (Jan. 1967), pp. 128-142.

7. Collins, G. E. The SAC-1 List Processing System. Univ. of Wisconsin Computing Center Report (34 pages), July, 1967.

8. Collins, G.E. The SAC-1 Integer Arithmetic System. Univ. of Wisconsin Computing Center Report (31 pages), Sept. 1967.

9. Collins, G. E. The SAC-1 Polynomial System. Univ. of Wisconsin Computing Center Technical Reference 2:1968 (68 pages), Jan. 1968.

10. Collins, G. E. Computing Multiplicative Inverses in GF(p). Univ. of Wisconsin Computer Sciences Technical Report No. 22 (8 pages), May 1968.

11. Cook, S. A. On the Minimum Computation Time of Functions. Harvard University Computation Lab. Report BL-41, May 1966.

12. Knuth, D. E. The Art of Computer Programming, Vol. I: Fundamental Algorithms. Addison-Wesley, 1968.

13. Knuth, D. E. The Art of Computer Programming, Vol. II: Seminumerical Algorithms. Addison-Wesley, to be published.

14. Takahasi, H. and Ishibashi, Y. A New Method for "Exact Calculation" by a Digital Computer. Information Processing in Japan, Vol. 1 (1961), pp. 28-42.

15. Uspensky, J. V. and Heaslett, M. A. Elementary Number Theory, McGraw-Hill, 1939.

NUMERICAL STUDIES OF PROTOTYPE CAVITY FLOW PROBLEMS

by Donald Greenspan

APPENDIX: CDC 3600 FORTRAN PROGRAM FOR CAVITY FLOW PROBLEMS

by D. Schultz

Computer Sciences Technical Report #37

July 1968

NUMERICAL STUDIES OF PROTOTYPE CAVITY FLOW PROBLEMS*

by

Donald Greenspan

1. Introduction.

The flow of a gas or of a liquid in a closed cavity has long been of interest in applied science (see, e.g., references [1, 2, 4, 7-12, 14] and the additional references contained therein). In this paper we will apply the power of the high speed digital computer to study prototype, steady state, two dimensional problems for such flows. The numerical methods to be developed will be finite difference methods and will be described in sufficient generality so as to be applicable to nonlinear coupled systems similar in structure to the Navier-Stokes equations.

2. The Eddy Problem in a Rectangle.

The class of problems to be studied, called eddy problems in a rectangle, can be formulated as follows. For $d > 0$, let the points $(0, 0)$, $(1, 0)$, $(1, d)$ and $(0, d)$ be denoted by A, B, C and D, respectively (see Figure 2.1). Let S be the rectangle whose vertices are A, B, C, D and denote its interior by R. On R the equations of motion to be satisfied are the Navier-Stokes equations, that is

*Funds for the computations described in this paper were made available by the Research Committee of the Graduate School of the University of Wisconsin.

$$\Delta\psi = -\omega \tag{2.1}$$

$$\Delta\omega + \mathcal{R}\left(\frac{\partial\psi}{\partial x}\frac{\partial\omega}{\partial y} - \frac{\partial\psi}{\partial y}\frac{\partial\omega}{\partial x}\right) = 0 \ , \tag{2.2}$$

where ψ is the stream function, ω is the vorticity and $\mathcal{R}$ is the Reynolds number. On S the boundary conditions to be satisfied are

$$\psi = 0, \ \frac{\partial\psi}{\partial x} = 0 \ , \quad \text{on AD} \tag{2.3}$$

$$\psi = 0, \ \frac{\partial\psi}{\partial y} = 0 \ , \quad \text{on AB} \tag{2.4}$$

$$\psi = 0, \ \frac{\partial\psi}{\partial x} = 0 \ , \quad \text{on BC} \tag{2.5}$$

$$\psi = 0, \ \frac{\partial\psi}{\partial y} = -1 \ , \quad \text{on CD} \ . \tag{2.6}$$

The analytical problem is defined on R + S by (2.1)-(2.6) and is shown diagrammatically in Figure 2.1.

In general, boundary value problem (2.1)-(2.6) cannot be solved by means of existing analytical techniques. Physical solutions have been produced in the laboratory by Pan and Acrivos [9], while numerical methods which "converge", but only for small $\mathcal{R}$, have been developed by Burggraf [4] and Runchal, Spalding and Wolfshtein [12]. A numerical method which converges for all $\mathcal{R}$, but which has been run only for relatively large values of the grid size, has been developed by the writer [7].

We shall describe next a modified, somewhat faster form of the method developed in [7] and apply it to a selection of difficult problems which are

of wide interest. Among our major objectives will be the construction of secondary vortices and the study of vorticity for large Reynolds number.

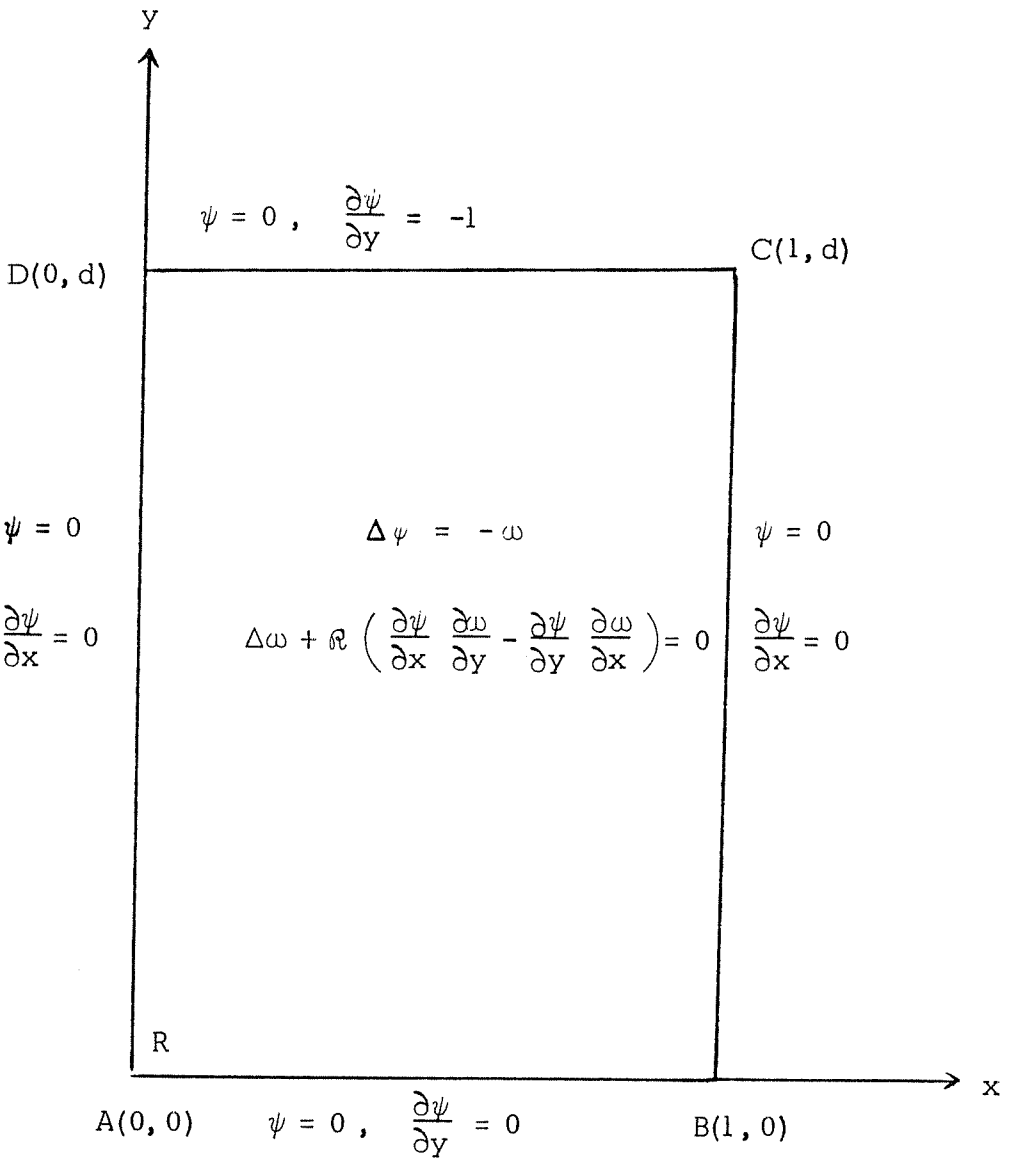


Figure 2.1

3. The General Numerical Method.

For a fixed positive integer n, set $h = \frac{1}{n}$. Assume, for simplicity, that d is an integral multiple of h. (If d is not an integral multiple of h, the method is easily modified as shown in [7].) Starting at $(0, 0)$ with grid size h, construct and number in the usual way [7] the set of interior grid points R_h and the set of boundary grid points S_h.

For given tolerances ε_1 and ε_2, we will show first how to construct on R_h a sequence of discrete stream functions

$$\psi^{(0)}, \psi^{(1)}, \psi^{(2)}, \cdots \tag{3.1}$$

and on $R_h + S_h$ a sequence of discrete vorticity functions

$$\omega^{(0)}, \omega^{(1)}, \omega^{(2)}, \cdots , \tag{3.2}$$

such that for some integer k both the following are valid:

$$|\psi^{(k)} - \psi^{(k+1)}| < \varepsilon_1 , \quad \text{on } R_h \tag{3.3}$$

$$|\omega^{(k)} - \omega^{(k+1)}| < \varepsilon_2 , \quad \text{on } R_h + S_h . \tag{3.4}$$

Initially, set

$$\psi^{(0)} = C_1 , \quad \text{on } R_h \tag{3.5}$$

$$\omega^{(0)} = C_2 , \quad \text{on } R_h + S_h , \tag{3.6}$$

where C_1 and C_2 are constants.

To produce the second iterate $\psi^{(1)}$ of sequence (3.1) proceed as follows. At each point of R_h of the form (h, ih), $i = 2, \cdots, n-2$, approximate (2.3) by

$$\psi(h, ih) = \frac{\psi(2h, ih)}{4} \quad . \tag{3.7}$$

At each point of R_h of the form (ih, h), $i = 1, 2, \cdots, n-1$, approximate (2.4) by

$$\psi(ih, h) = \frac{\psi(ih, 2h)}{4} \quad . \tag{3.8}$$

At each point of R_h of the form $(1-h, ih)$, $i = 2, 3, \cdots, n-2$, approximate (2.5) by

$$\psi(1-h, ih) = \frac{\psi(1-2h, ih)}{4} \quad . \tag{3.9}$$

At each point of R_h of the form $(ih, 1-h)$, $i = 1, 2, \cdots, n-1$, approximate (2.6) by

$$\psi(ih, 1-h) = \frac{h}{2} + \frac{\psi(ih, 1-2h)}{4} \quad . \tag{3.10}$$

And at each remaining point of R_h write down the difference analogue

$$-4\psi(x, y) + \psi(x+h, y) + \psi(x, y+h) + \psi(x-h, y) + \psi(x, y-h) = -h^2 \omega^{(0)}(x, y) \tag{3.11}$$

of (2.1). Solve the linear algebraic system generated by (3.7)-(3.11) by the generalized Newton's method [7] with over-relaxation factor r_ψ and denote this solution by $\bar{\psi}^{(1)}$. Then, on R_h, $\psi^{(1)}$ is defined by the smoothing formula

(3.12) $$\psi^{(1)} = \rho\,\psi^{(0)} + (1-\rho)\,\bar{\psi}^{(1)}\,, \qquad 0 \le \rho \le 1\,.$$

To produce the second iterate $\omega^{(1)}$ of sequence (3.2) proceed as follows. At each point of S_h of the form $(ih, 0)$, $i = 0, 1, 2, \cdots, n$, set

(3.13) $$\bar{\omega}^{(1)}(ih, 0) = -\frac{2\psi^{(1)}(ih, h)}{h^2}\,;$$

at each point of S_h of the form $(0, ih)$, $i = 1, 2, \cdots, n-1$, set

(3.14) $$\bar{\omega}^{(1)}(0, ih) = -\frac{2\psi^{(1)}(h, ih)}{h^2}\,;$$

at each point of S_h of the form $(1, ih)$, $i = 1, 2, \cdots, n-1$, set

(3.15) $$\bar{\omega}^{(1)}(1, ih) = -\frac{2\psi^{(1)}(1-h, ih)}{h^2}\,;$$

and at each point of S_h of the form $(ih, 1)$, $i = 0, 1, 2, \cdots, n$, set

(3.16) $$\bar{\omega}^{(1)}(ih, 1) = \frac{2}{h} - \frac{2\psi^{(1)}(ih, 1-h)}{h^2}\,.$$

Next, at each point (x, y) in R_h set

$$\alpha = \psi^{(1)}(x+h, y) - \psi^{(1)}(x-h, y)$$
$$\beta = \psi^{(1)}(x, y+h) - \psi^{(1)}(x, y-h)$$

and approximate (2.2), appropriately, by

(3.17) $$\left(-4 - \frac{\alpha\mathcal{R}}{2} - \frac{\beta\mathcal{R}}{2}\right)\omega(x, y) + \omega(x+h, y) + \left(1 + \frac{\alpha\mathcal{R}}{2}\right)\omega(x, y+h)$$
$$+ \left(1 + \frac{\beta\mathcal{R}}{2}\right)\omega(x-h, y) + \omega(x, y-h) = 0\,; \quad \text{if} \quad \alpha \ge 0, \quad \beta \ge 0\,,$$

$$(3.18)\quad (-4 - \frac{\alpha\Re}{2} + \frac{\beta\Re}{2})\omega(x, y) + (1 - \frac{\beta\Re}{2})\omega(x+h, y) + (1 + \frac{\alpha\Re}{2})\omega(x, y+h)$$

$$+ \omega(x-h, y) + \omega(x, y-h) = 0 ; \quad \text{if } \alpha \geq 0, \quad \beta < 0 ,$$

$$(3.19)\quad (-4 + \frac{\alpha\Re}{2} - \frac{\beta\Re}{2})\omega(x, y) + \omega(x+h, y) + \omega(x, y+h) + (1 + \frac{\beta\Re}{2})\omega(x-h, y)$$

$$+ (1 - \frac{\alpha\Re}{2})\omega(x, y-h) = 0 ; \quad \text{if } \alpha < 0, \quad \beta \geq 0 ,$$

$$(3.20)\quad (-4 + \frac{\alpha\Re}{2} + \frac{\beta\Re}{2})\omega(x, y) + (1 - \frac{\beta\Re}{2})\omega(x+h, y) + \omega(x, y+h)$$

$$+ \omega(x-h, y) + (1 - \frac{\alpha\Re}{2})\omega(x, y-h) = 0 ; \quad \text{if } \alpha < 0, \quad \beta < 0 .$$

Solve the linear algebraic system generated by (3.17)-(3.20) by the generalized Newton's method with over-relaxation factor r_ω and denote the solution by $\overline{\omega}^{(1)}$. Finally, on all of $R_h + S_h$ define $\omega^{(1)}$ by the smoothing formula

$$\omega^{(1)} = \mu\omega^{(0)} + (1 - \mu)\overline{\omega}^{(1)} , \quad 0 \leq \mu \leq 1 .$$

Proceed next to determine $\psi^{(2)}$ on R_h from $\omega^{(1)}$ and $\psi^{(1)}$ in the same fashion as $\psi^{(1)}$ was determined from $\omega^{(0)}$ and $\psi^{(0)}$. Then construct $\omega^{(2)}$ on $R_h + S_h$ from $\omega^{(1)}$ and $\psi^{(2)}$ in the same fashion as $\omega^{(1)}$ was determined from $\omega^{(0)}$ and $\psi^{(1)}$. In the indicated fashion, construct the sequences (3.1) and (3.2). Terminate the computation when (3.3) and (3.4) are valid.

Finally, when $\psi^{(k)}$ and $\omega^{(k)}$ are verified to be solutions of the difference analogues of (2.1) and (2.2), they are taken to be the numerical approximations of $\psi(x, y)$ and $\omega(x, y)$, respectively.

4. Examples.

Consider first the boundary value problem defined by (2.1)-(2.6) with $d = 1$. This problem was solved by the method of Section 3 for $\mathcal{R} = 200$ with $h = \frac{1}{20}$, $\varepsilon_1 = 1$, $\varepsilon_2 = 10^{-4}$, $\rho = 0.1$, $\mu = 0.7$, $r_\psi = 1.8$, $r_\omega = 1.0$, $C_1 = C_2 = 0$, and also for $\mathcal{R} = 500$, 2000 and 15000 with the same parameter values except for $\varepsilon_2 = 10^{-3}$. Convergence was achieved for $\mathcal{R} = 200$ in 14 minutes with 341 outer iterations, for $\mathcal{R} = 500$ in 11 minutes with 96 outer iterations, for $\mathcal{R} = 2000$ in 4 minutes with 80 outer iterations, and for $\mathcal{R} = 15000$ in $3\frac{1}{2}$ minutes with 40 outer iterations. The resulting stream curves exhibited only primary vortices and are shown in Figure 4.1. The resulting equivorticity curves exhibited the double spiral development shown in [7] and are given in Figure 4.2.

With an aim toward producing secondary vortices and toward studying vorticity for large Reynolds numbers, boundary value problems (2.1)-(2.6) was considered again with $d = 1$. The problem was solved for $\mathcal{R} = 50$, 10000, and 100000 with $h = \frac{1}{40}$. For $\mathcal{R} = 50$ the remaining input parameters were chosen to be $\varepsilon_1 = 10^{-4}$, $\varepsilon_2 = 10^{-3}$, $\rho = .03$, $\mu = .90$, $r_\psi = 1.8$, $r_\omega = 1.8$, $C_1 = C_2 = 0$. Convergence was achieved in 60 minutes with 100 outer iterations. The resulting flow with the secondary vortices is shown in Figure 4.3. For $\mathcal{R} = 10000$ the remaining input parameters were chosen to be $\varepsilon_1 = .004$, $\varepsilon_2 = .03$, $\rho = .03$, $\mu = .95$, $r_\psi = 1.8$, $r_\omega = 1$, $C_1 = C_2 = 0$. After 183 outer iterations, μ was changed to .85. Convergence was achieved in 260 minutes with a total of 226 outer iterations. The resulting flow with a

single secondary vortex is shown in Figure 4.4. For $\mathcal{R} = 100000$, the remaining input parameters were chosen to be $\varepsilon_1 = 10^{-4}$, $\varepsilon_2 = .005$, $\rho = .03$, $\mu = .95$, $r_\psi = 1.8$, $r_\omega = 1$, but $\psi^{(0)}$ and $\omega^{(0)}$ were taken to be the 57^{th} outer iterates of the run for $\mathcal{R} = 10000$. Convergence was achieved in 135 minutes with 386 outer iterations. The flow is shown in Figure 4.5 and contains no secondary vortices. The equivorticity curve $\omega = 1.630$, with its double-spiral, space filling characteristics is shown in Figure 4.6. Numerical evidence of Batchelor's result that the vorticity in a large subregion of R converges to a constant as $R \to \infty$ is exhibited in Figure 4.6 by setting crosses on those points at which the vorticity is between 1.6 and 1.7 .

Finally, consider boundary value problem (2.1)-(2.6) with $d = 2$ and $R = 10$. This problem was solved with $h = \frac{1}{40}$, $\varepsilon_1 = 10^{-4}$, $\varepsilon_2 = 10^{-3}$, $\rho = .05$, $\mu = .85$, $r_\psi = 1.8$, $r_\omega = 1.25$, $C_1 = C_2 = 0$. Convergence was achieved in 32 minutes with 102 outer iterations. The resulting flow, with its two primary and two secondary vortices, is shown in Figure 4.7.

5. Remarks.

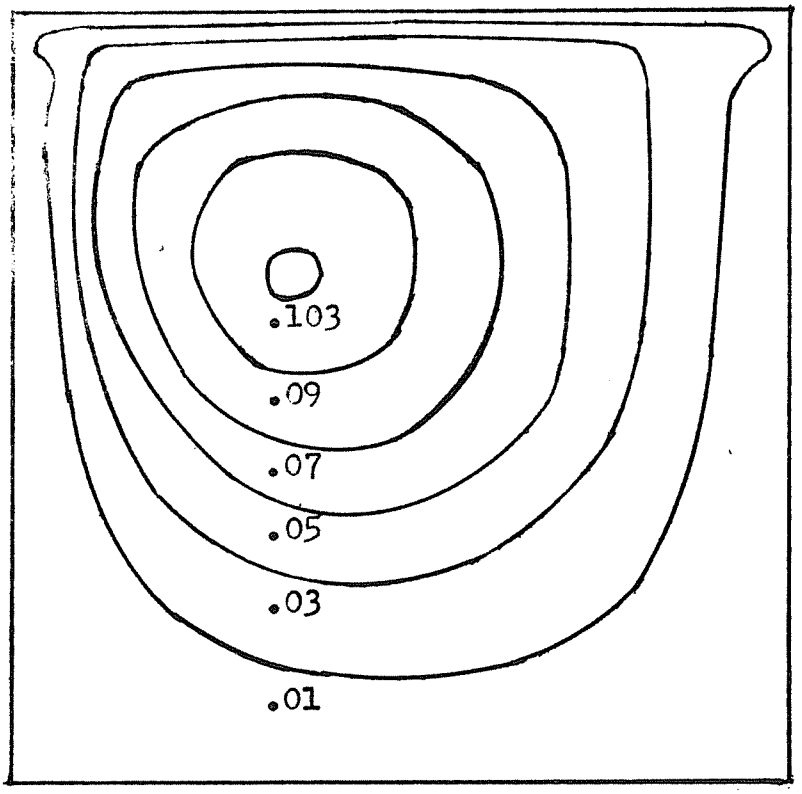
From the many examples run in addition to those described in Section 4, the following observations and heuristic conclusions resulted. Divergence or exceptionally slow convergence usually followed if any one of the following choices were made: $.4 \le \rho \le 1$, $0 \le \mu \le .6$, $r_\psi < 1$, $r_\omega << 1$. The choice $\rho = \mu = 0$ yields convergence only for large grid sizes and small Reynolds num-

bers. The choice $r_\psi = 1.8$ was consistently good. For grid sizes larger than or equal to $\frac{1}{20}$, sequence (3.1) converged so much faster than (3.2) that very little attention had to be directed toward the choice of ε_1, but for grids smaller than $\frac{1}{20}$ this was not the case and attention had to be directed to the choices of both ε_1 and ε_2. Deletion of all or even of some of the special formulas (3.7)-(3.10) and substitution with (3.11) always led to divergence for large Reynolds numbers ($\mathcal{R} \sim 10000$), but often did yield secondary vortices for $h = \frac{1}{20}$ for small Reynolds numbers ($\mathcal{R} \sim 50$). The difference equations for $\psi^{(k)}$ and $\omega^{(k)}$ were always satisfied to much smaller tolerances than those imposed in (3.3) and (3.4), respectively.

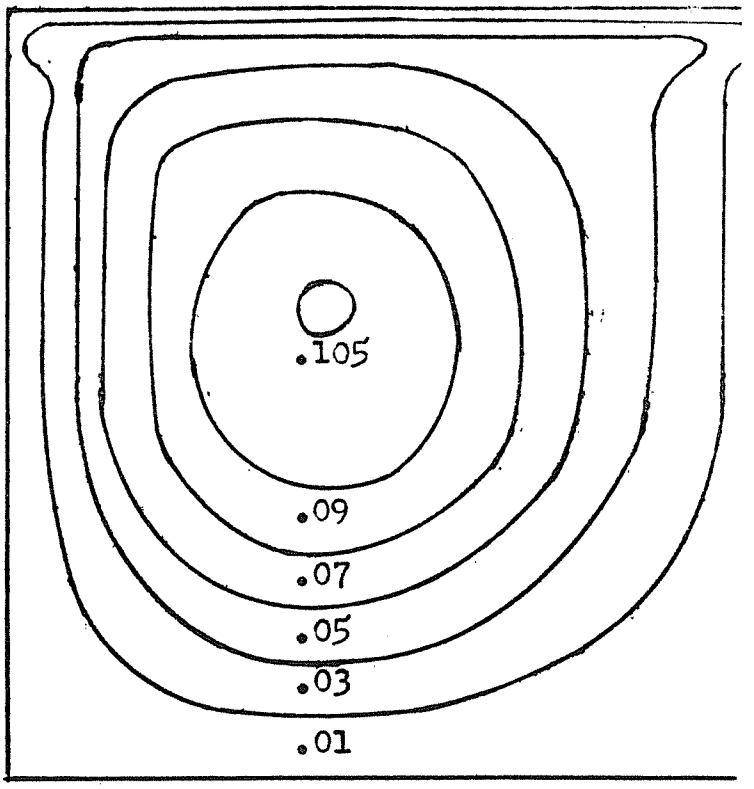
Several possible modifications of the method of this paper which should be explored if one wishes to speed up the convergence include allowing some or all of ρ, μ, r_ψ and r_ω to be variable [6], using line over relaxation [15], and choosing $\psi^{(0)}$ and $\omega^{(0)}$ in a more judicious manner than that prescribed in (3.5)-(3.6).

Observe also that the method of Section 3 applies directly to biharmonic problems (i.e., to the case $\mathcal{R} = 0$) and initial computations verify that it extends in a natural way to free convection problems [1].

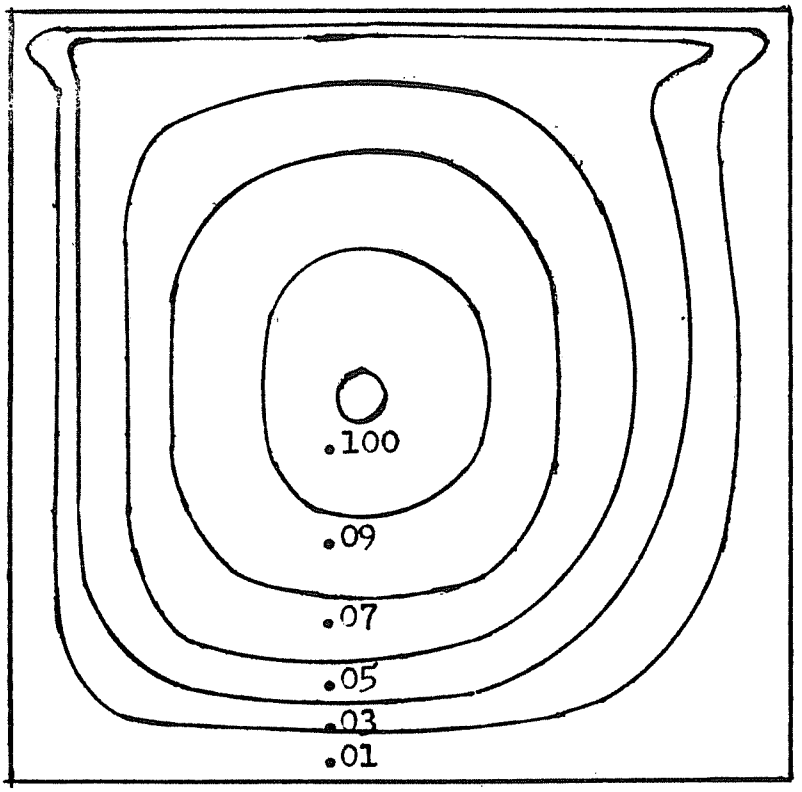
Finally, note that theoretical support for the method of this paper is now beginning to appear for very special cases [3, 5, 13].



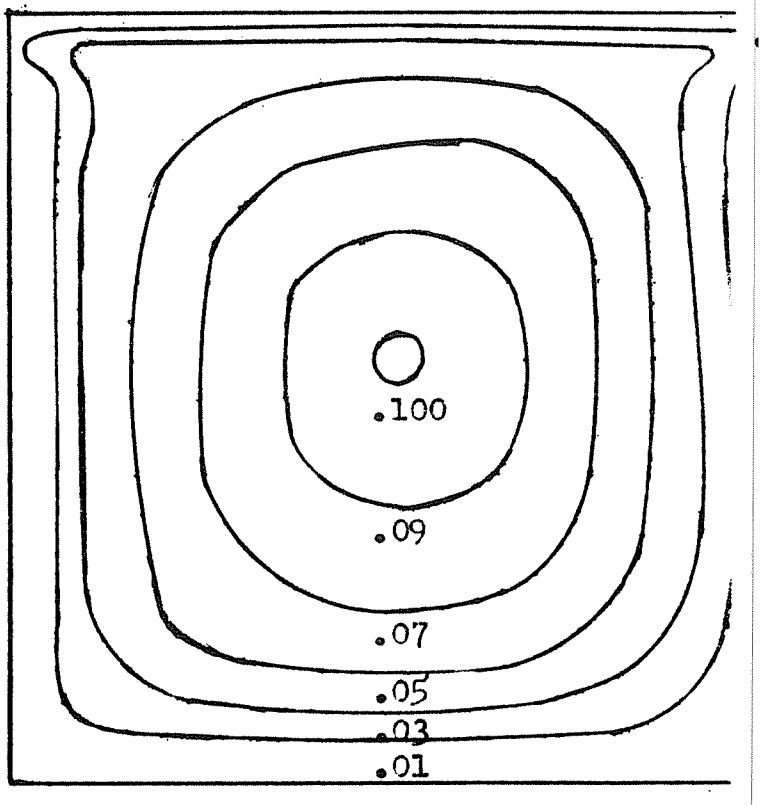
R = 200



R = 500

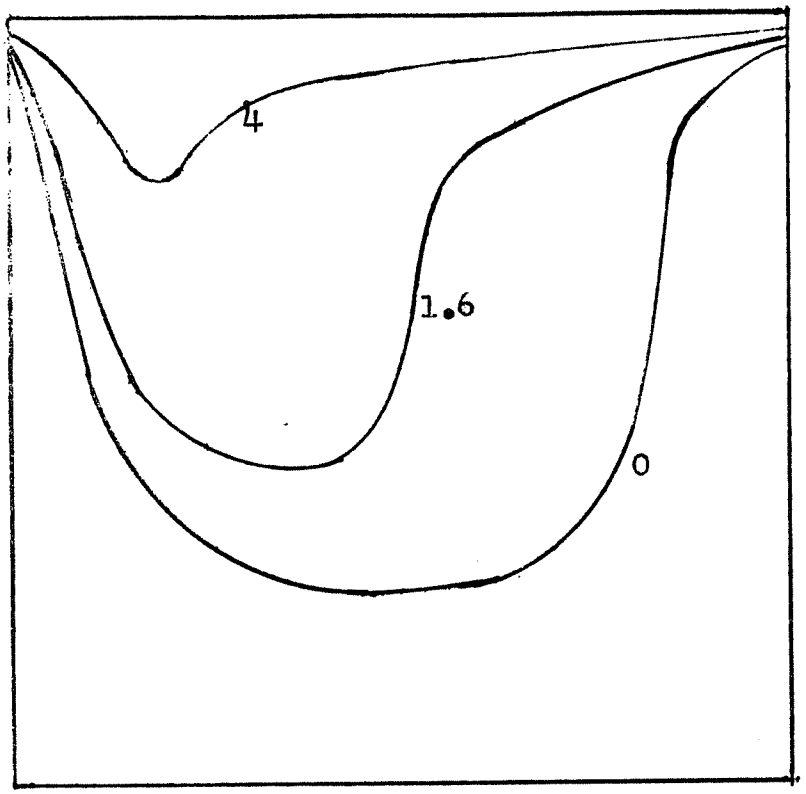


R = 2000

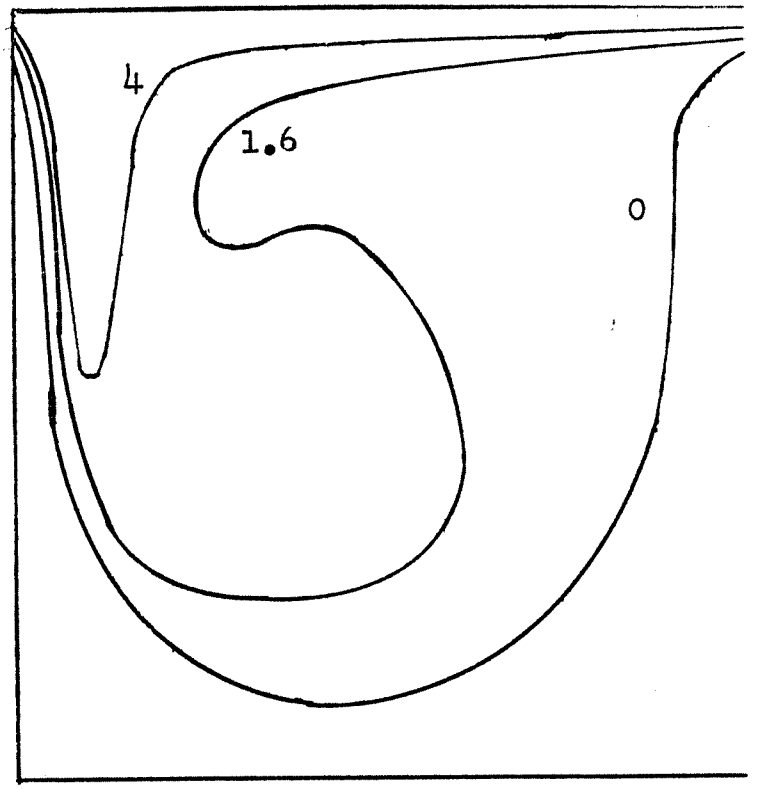


R = 15000

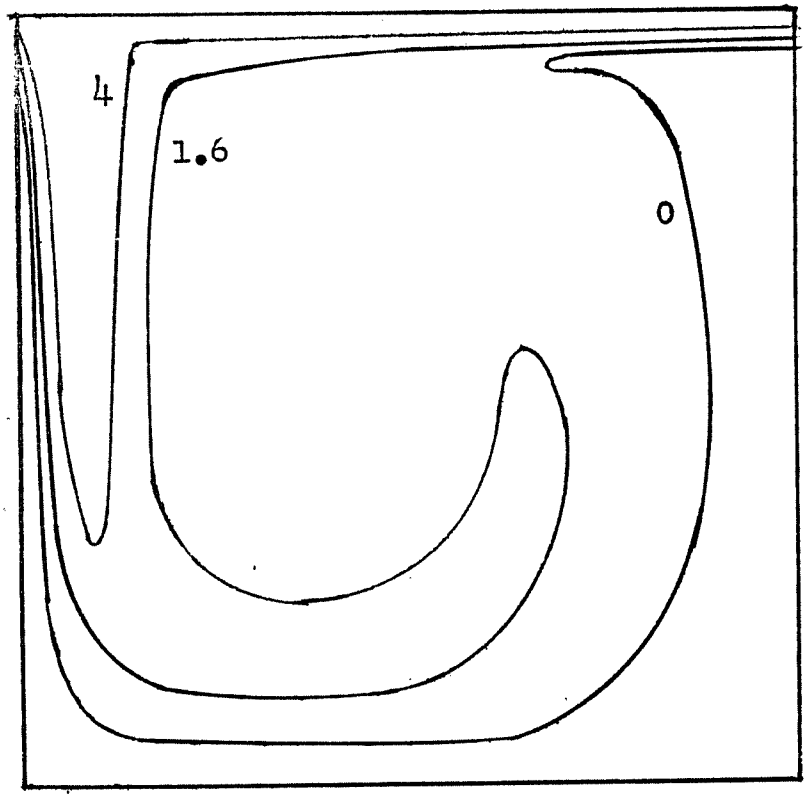
FIGURE 4.1 Typical streamlines for h = 1/20.



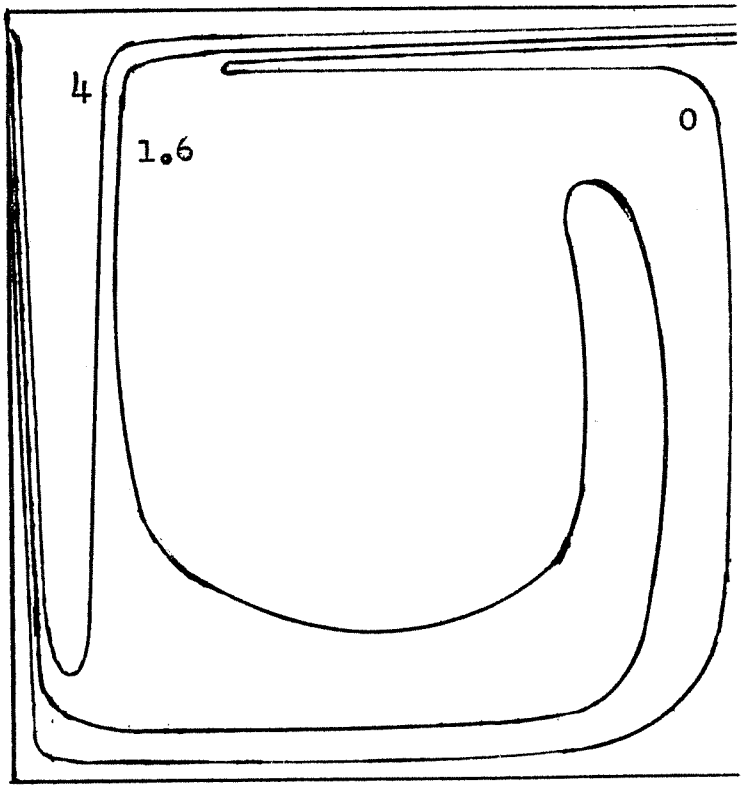
R = 200



R = 500



R = 2000



R = 15000

FIGURE 4.2 Selected equivorticity curves for h = 1/20.

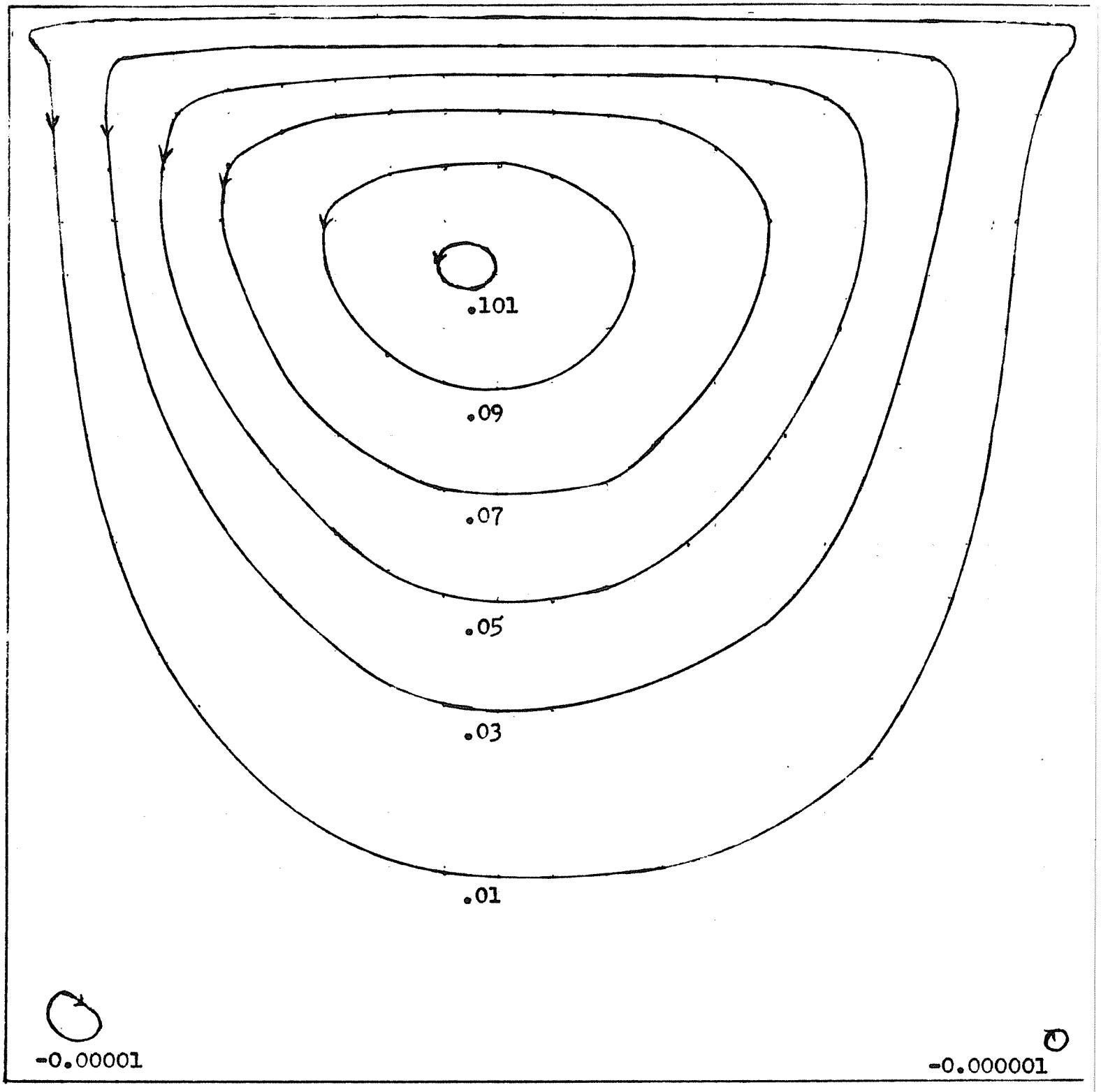


FIGURE 4.3 Streamlines for Reynolds number 50 with h = 1/40.

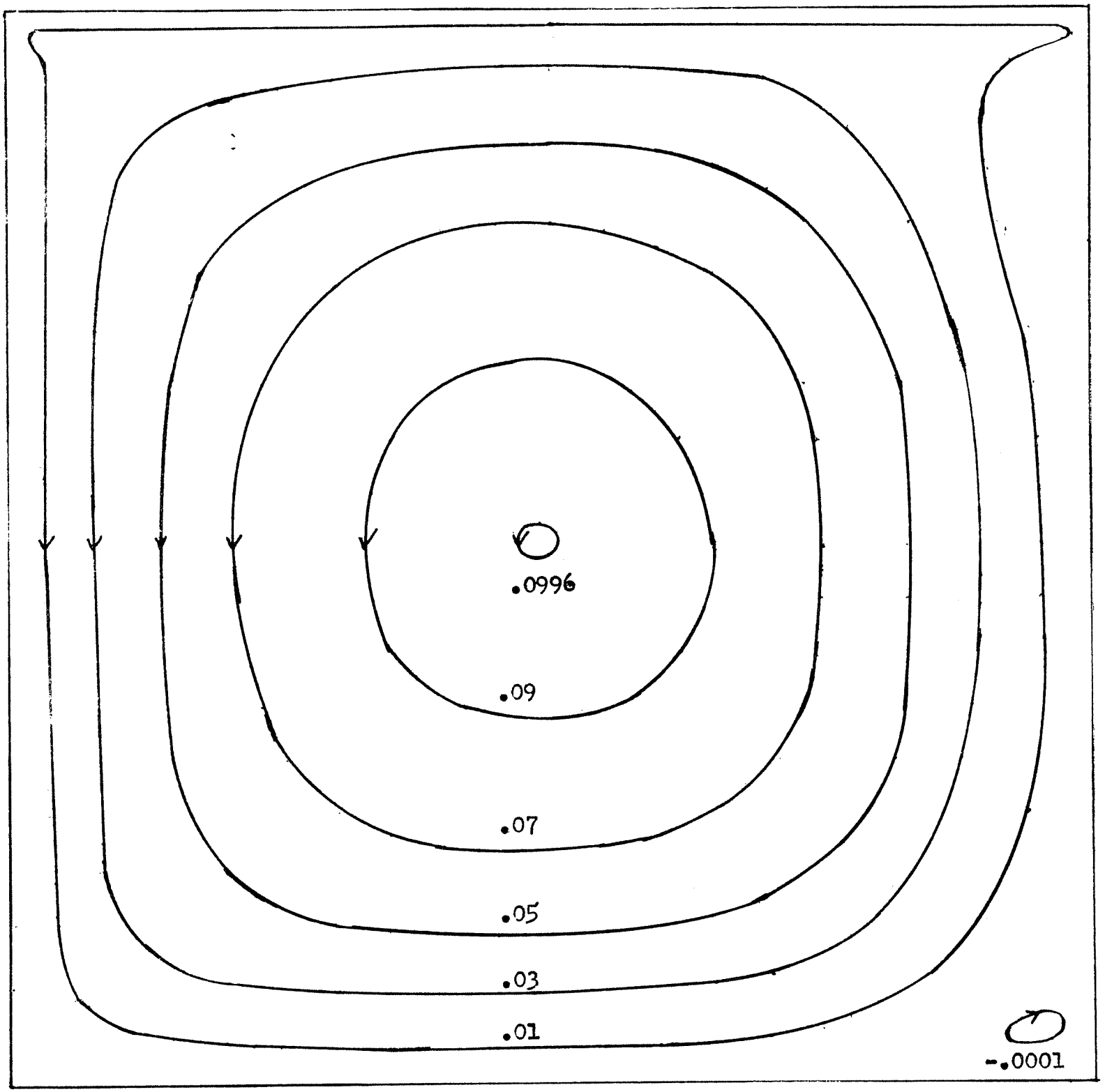


FIGURE 4.4 Streamlines for Reynolds number 10000 with $h = 1/40$.

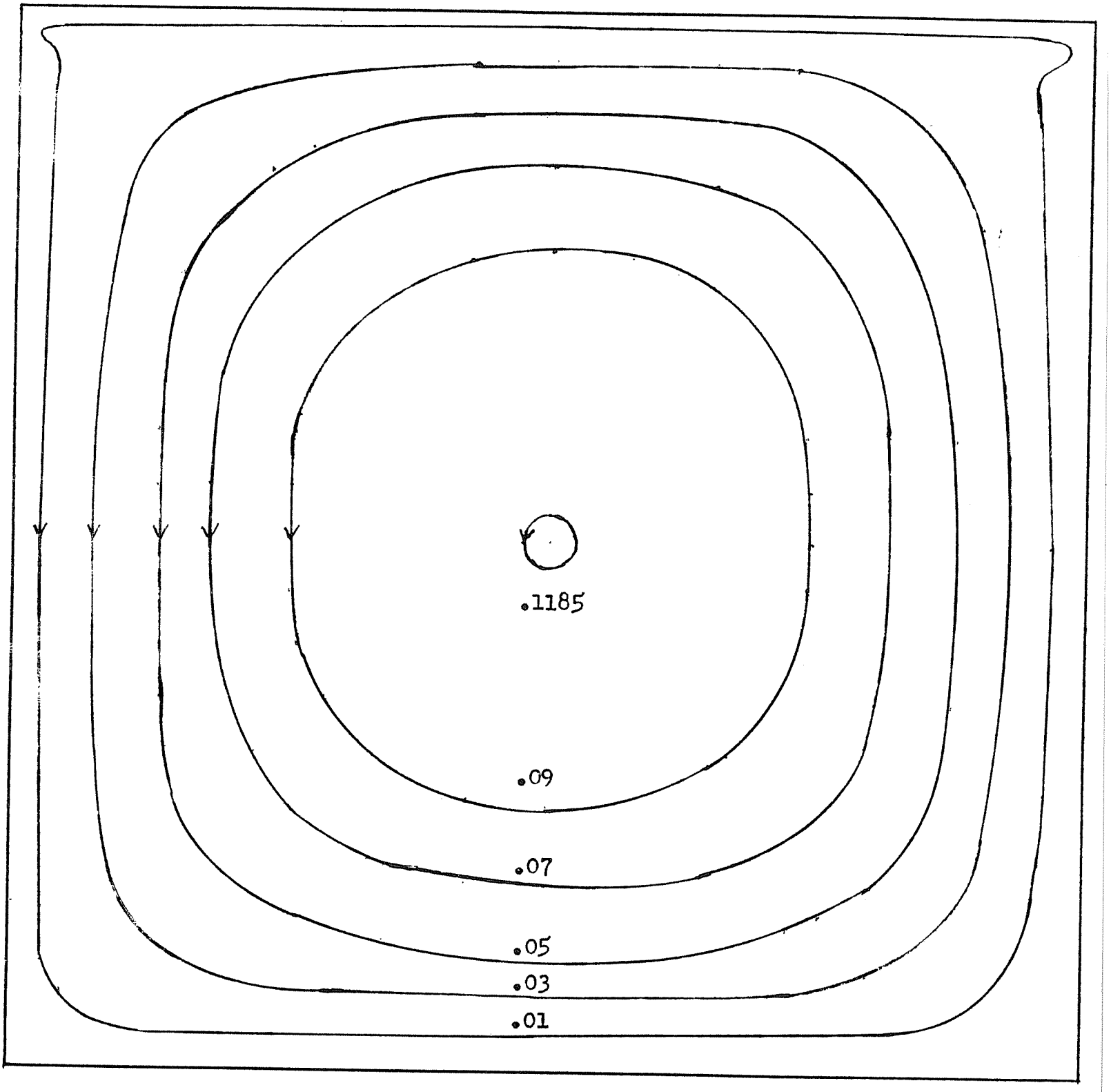


FIGURE 4.5 Streamlines for Reynolds number 100000 with h = 1/40.

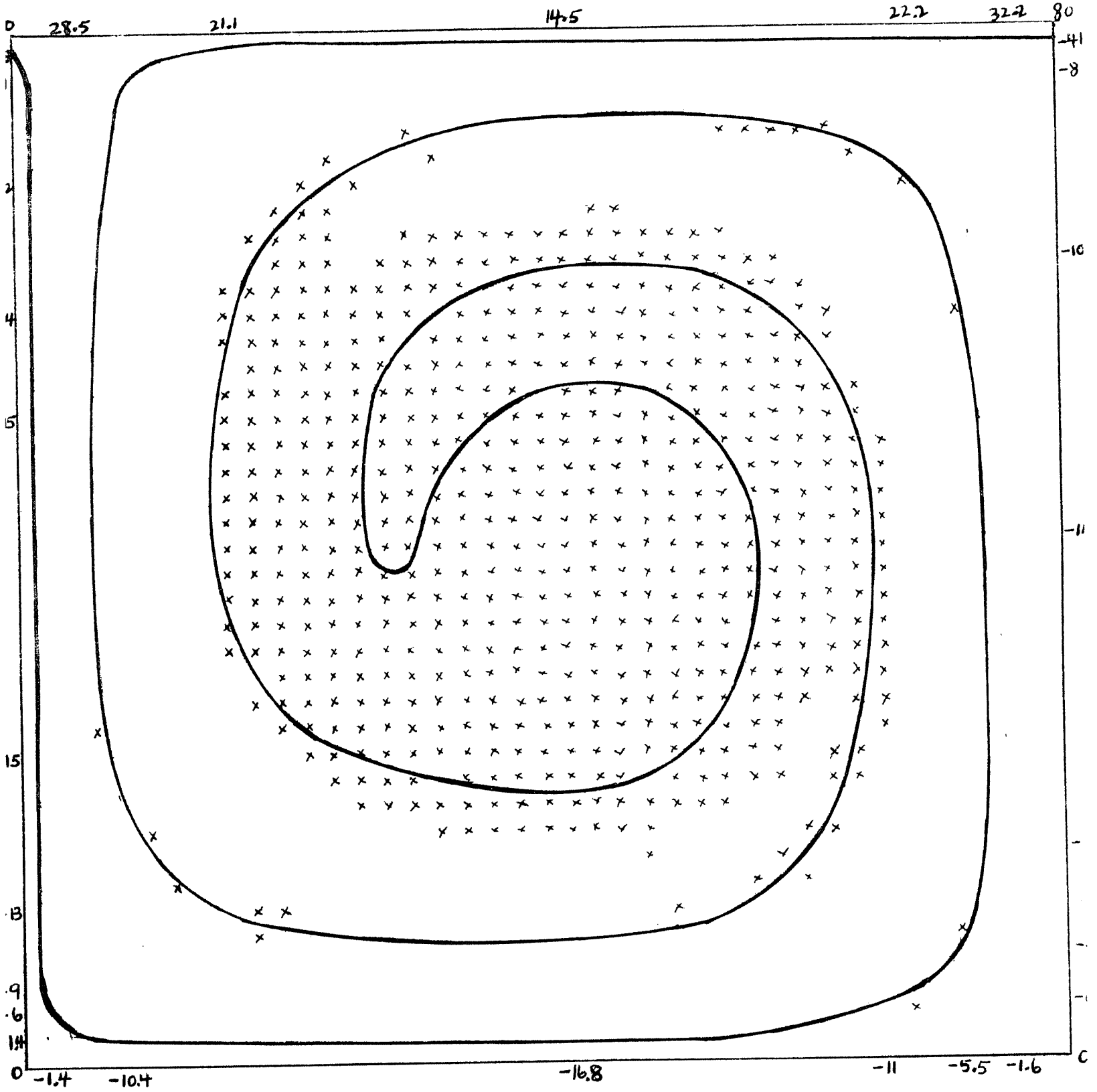


FIGURE 4.6 Equivorticity curve $\omega = 1.630$ for Reynolds number 100000 and $h = 1/40$. At crossed points vorticity is between 1.6 and 1.7.

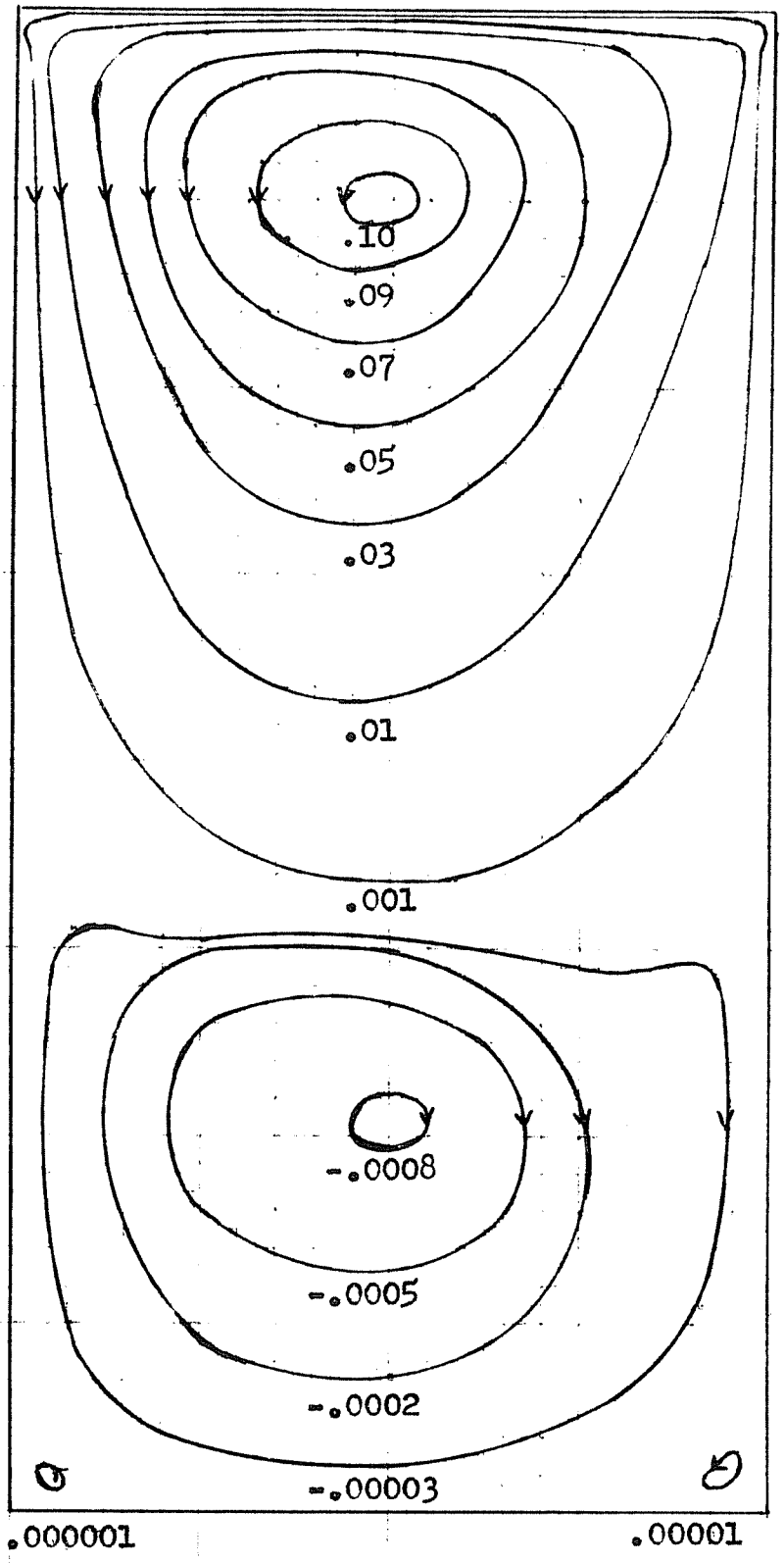


FIGURE 4.7 Streamlines for Reynolds number 10 with h = 1/40 for a 2 by 1 rectangular cavity.

REFERENCES

1. G. K. Batchelor, "Heat transfer by free convection across a closed cavity between vertical boundaries at different temperatures," Quart. Appl. Math., 3, 1954, 209-233.

2. _____, "On steady laminar flow with closed streamlines at large Reynolds number," Jour. Fluid Mech., 1, 1956, 177-190.

3. R. T. Boughner, "The discretization error in the finite difference solution of the linearized Navier-Stokes equations for incompressible fluid flow at large Reynolds number," T.M. - 2165, Oak Ridge Nat. Lab., 1968.

4. O. R. Burggraf, "Analytical and numerical studies of the structure of separated flows," Jour. Fluid Mech., 24, 1966, 113-151.

5. I. F. Burns, "A numerical solution to a non-self-adjoint elliptic partial differential equation," Tech. Rept. 8, Univ. Comp. Center, Univ. Tennessee, Knoxville, 1967.

6. B. A. Carre, "The determination of the optimum accelerating factor for successive over-relaxation," Comp. Jour., 4, 1961, 73-78.

7. D. Greenspan, Lectures on the Numerical Solution of Linear, Singular and Nonlinear Differential Equations, Prentice-Hall, Englewood Cliffs, 1968.

8. M. Kawaguti, "Numerical solution of the Navier-Stokes equations for the flow in a two-dimensional cavity," Jour. Phys. Soc. Japan, 16, 1961, 2307-2318.

9. F. Pan and A.Acrivos, "Steady flows in rectangular cavities," Journ. Fluid Mech., 28, 1967, 643-655.

10. G. Poots, "Heat transfer by laminar free convection in enclosed plane gas layers," QJMAM, 11, 1958, 257-273.

11. J. B. Rosen, "Approximate solution to transient Navier-Stokes cavity convection problems," Tech. Rept. #32, Dept. of Computer Sciences, University of Wisconsin, 1968.

12. A. K. Runchal, D. B. Spalding and M. Wolfshtein, "The numerical solution of the elliptic equations for transport of vorticity, heat and matter in two dimensional flows," Rept. SF/TN/14, Dept. Mech. Eng., Imperial College of Science and Tech., March, 1968.

13. J. Smith, "The coupled equation approach to the numerical solution of the biharmonic equation by finite differences," Mathematics Department Report, Univ. Tennessee, Knoxville, n.d.

14. H. B. Squire, "Note on the motion inside a region of recirculation," Jour. Roy. Aero. Soc., 60, 1956, 203-205.

15. R. S. Varga, Matrix Iterative Analysis, Prentice-Hall, Englewood Cliffs, 1962.

APPENDIX

CDC 3600 FORTRAN PROGRAM FOR CAVITY FLOW PROBLEMS

D. SCHULTZ

DEFINITIONS OF PROGRAM VARIABLES

OMA = VORTICITY VALUES

PSI = STREAM VALUES

N = NUMBER OF VERTICAL SPACES IN THE GRID

M = NUMBER OF HORIZONTAL SPACES IN THE GRID

R = REYNOLD'S NUMBER

H = GRID SIZE

EPS = TOLERANCE FOR INNER-AND OUTER-ITERATIONS

C1 = WEIGHTING FACTOR FOR OMA

F1 = WEIGHTING FACTOR FOR PSI

RV = RELAXATION FACTOR FOR OMA EQUATIONS

NM = NUMBER OF OUTER-ITERATIONS

NCOUNT = NUMBER OF INNER-ITERATIONS

W0,W1,W2,W3,W4 = COEFFICIENTS FOR THE VORTICITY EQUATION

ISTOP = SWITCH TO INDICATE CONVERGENCE

```
      PROGRAM SAVSTR
      DIMENSION PSI(50,50),OMA(50,50),SVPSI(50,50),SVOMA(50,50),SVOUT(5
     10,50)
      COMMON N,NPLUS1,M,MPLUS1,NZ,MP
      READ 300,N,M
300   FORMAT(2I2)
      MPLUS1=M+1
      MMESH=M-1
      JM=M
      NPLUS1=N+1
      NMESH=N-1
      H=1./N
      H2=H*H
      EPS=.001
C     INITIALIZE VECTORS
      NZ=0
      MP=6
      ISTOP=0
      R=50
      RW=1.
      C1=0
      F1=0
104   CONTINUE
      PRINT 2323,C1
2323  FORMAT(1H1,F8.2)
      DO 1 I=1,50
      DO 1 J=1,50
      SVOUT(I,J)=0
      SVPSI(I,J)=0
      SVOMA(I,J)=0
      PSI(I,J)=0
1     OMA(I,J)=0
      NM=0
      F2=1-F1
      C2=1-C1
C     BEGIN LOOP FOR OUTER ITERATIONS
C   SAVE VORTICITY FUNCTION FROM PREVIOUS OUTER ITERATION
23    DO 40 I=1,NPLUS1
      DO 40 J=1,MPLUS1
40    SVOUT(I,J)=OMA(I,J)
      NM=NM+1
      NCOUNT=0
C   BEGIN INNER ITERATION FOR STREAM FUNCTION
C     COMPUTE STREAM FUNCTION FOR INNER REGION
11    DO 2 I=3,NMESH
      DO 2 J=3,MMESH
      SVPSI(I,J)=PSI(I,J)
2     PSI(I,J)=(-.8*PSI(I,J))+.45*(PSI(I,J-1)+PSI(I,J+1)+PSI(I-1,J)+
     1PSI(I+1,J)+H2*OMA(I,J))
C     COMPUTE STREAM FUNCTION ON TOP AND BOTTOM INNER BOUNDARY LINES
      DO 3 I=2,N
      PSI(I,2)=(.25*PSI(I,3))
3     PSI(I,M)=.25*PSI(I,MMESH)+.5*H
C     COMPUTE STREAM FUNCTION ON LEFT AND RIGHT INNER BOUNDARY LINES
      DO 4 I=3,MMESH
```

```
      PSI(1,I)=    (.25*PSI(2,I))
      PSI(N,I)=    (.25*PSI(N-1,I))
C     TEST STREAM FUNCTION FOR CONVERGENCE
      DO 5 I=3,NMESH
      DO 5 J=3,MMESH
      DIFF=ABSF(SVPSI(I,J)-PSI(I,J))
      IF(DIFF .GT. EPS) GO TO 6
5     CONTINUE
C     RECALCULATE STREAM  FUNCTION USING WEIGHTING
      DO 222 I=3,NMESH
      DO 222 J=3,MMESH
222   PSI(I,J)=F1*SVPSI(I,J)+F2*PSI(I,J)
      DO 114 I=2,NMESH
      IF(PSI(I,M))28,114,114
114    CONTINUE
      GO TO 200
6     NCOUNT=NCOUNT+1
      IF(NCOUNT .GT. 100) GO TO 8
      GO TO 11
C     TEST STREAM FUNCTION FOR DIVERGENCE
8     IF(DIFF .GT. 10) GO TO 28
      PRINT 93
93    FORMAT(1H1,11H PSI VALUES)
      CALL PRNTLST(PSI)
10    FORMAT(10F11.6)
      NCOUNT=0
      GO TO 11
28     PRINT 81
81    FORMAT(13H PSI DIVERGED)
      CALL PRNTLST(PSI)
      CALL PRNTLST(OMA)
      GO TO 699
C     BEGIN INNER ITERATION FOR VORTICITY
200   NCOUNT=0
30     HCONST=C2*(-2./H2)
C     COMPUTE VORTICITY ON BOUNDARY LINES USING WEIGHTING
C     TOP AND BOTTOM BOUNDARY LINES
      DO 12 I=1,NPLUS1
      OMA(I,1)=C1*OMA(I,1)+HCONST*PSI(I,2)
12    OMA(I,M+1)=C1*OMA(I,M+1)+HCONST*(PSI(I,M)-H)
C        LEFT AND RIGHT BOUNDARY LINES
      DO 13 I=2,M
      OMA(1,I)=HCONST*PSI(2,I)+C1*OMA(1,I)
13     OMA(N+1,I)=HCONST*PSI(N,I)+C1*OMA(N+1,I)
90    CONTINUE
C     COMPUTE COEFFICIENTS FOR VORTICITY EQUATIONS
C     COMPLETE ONE SWEEP OF INTERIOR
      DO 14 I=2,N
      DO 14 J=2,M
      A1=PSI(I+1,J)-PSI(I-1,J)
      B1=PSI(I,J+1)-PSI(I,J-1)
      A=ABSF(A1)
      B=ABSF(B1)
      W0=4+(A+B)*(R/2)
      IF(A1.GE. 0)15,16
```

```
15      W2=1+(R/2)*A
        W4=1
        GO TO 17
16      W2=1
        W4=1+A*(R/2)
17      IF(R1.GE. 0)18,19
18      W1=1
        W3=1+B*(R/2)
        GO TO 20
19      W1=1+B*(R/2)
        W3=1
20      SVOMA(I,J)=OMA(I,J)
        IF(ISTOP .EQ. 1)GO TO 305
      OMA(I,J)=((W1/W0)*OMA(I+1,J)+(W2/W0)*OMA(I,J+1)+(W3/W0)*OMA(I-1,J)
     1+(W4/W0)*OMA(I,J-1))*RW+(1-RW)*OMA(I,J)
        GO TO 14
C       CHECK TO SEE IF DIFFERENCE EQUATIONS ARE SATISFIED TO .001
305     DIFF    =((W1/W0)*OMA(I+1,J)+(W2/W0)*OMA(I,J+1)+(W3/W0)*OMA(I-1,J)
     1+(W4/W0)*OMA(I,J-1))-OMA(I,J)
        DIF=ABSF(DIFF)
        IF(DIF .GT. EPS1)282,14
282     PRINT 183,I,J
        GO TO 700
14      CONTINUE
        IF (ISTOP .EQ. 1) GO TO 700
C     TEST   VORTICITY      FOR CONVERGENCE
        DO 21 I=2,N
       DO 21 J=2,M
        DIFF=ABSF(SVOMA(I,J)-OMA(I,J))
        IF(DIFF .GE. EPS) GO TO 22
21     CONTINUE
C       RECALCULATE VORTICITY      USING WEIGHTING
        DO 144 I=2,N
        DO 144 J=2,M
144      OMA(I,J)=C1*SVOMA(I,J)+ C2*OMA(I,J)
        JM=JM+1
C       PRINT OUT EVERY 4 OUTER ITERATES
        IF(JM .EQ. 4)89,59
89      JM=0
        PRINT 79,NM
79      FORMAT(1H1,12,17H OUTER ITERATIONS)
        PRINT 91
        CALL PRNTLST(PSI)
        PRINT 92
        CALL PRNTLST(OMA)
C     TEST OUTER ITERATIONS FOR CONVERGENCE
59      CONTINUE
        DO 45 I=1,NPLUS1
        DO 45 J=1,MPLUS1
        DIFF=ABSF(SVOUT(I,J)-OMA(I,J))
        IF(DIFF .GT. EPS) GO TO 7
45       CONTINUE
        NZ=0
        MD=8
         PRINT 99,NM
```

```
 99     FORMAT(1H1,22H PROBLEM CONVERGED IN ,I4)
        PRINT 91
 91     FORMAT(1X,11H PSI VALUES)
        CALL PRNTLST(PSI)
        PRINT 92
 92     FORMAT(1H1,14H OMEGA VALUES)
        CALL PRNTLST(OMA)
        EPS1=.001
        RMAX=0
        ISTOP=1
C       CHECK TO SEE IF DIFFERENCE EQUATIONS OR STREAM FUNCTION ARE SATISFIE
C      A TOLERANCE OF .001
        DO 181 II=3,NM1S
        DO 181 JJ=3,MM1SH
        RES=ABSF(PSI(II,JJ)-SVPSI(II,JJ))
        IF(RES .GT. RMAX)301,302
 301    RMAX=RES
 302    CONTINUE
        A=-4*PSI(II,JJ)+PSI(II+1,JJ)+PSI(II,JJ+1)+PSI(II-1,JJ)+PSI(II,JJ-
       11)
        B=-H*H*OMA(II,JJ)
        D=A-B
        IF(D .GT. EPS1) GO TO 182
 181     CONTINUE
        GO TO 90
 182    PRINT 183,II,JJ
 183    FORMAT(1H1,41H DIFFERENCE EQU. NOT SATISFIED AT POINT (,I2,1H,,I2
       1,1H))
        GO TO 699
C      TEST OUTER ITERATIONS FOR DIVERGENCE
 7      IF(DIFF .GT. 100)199,23
 22     NCOUNT=NCOUNT+1
        IF(NCOUNT .GT. 300) GO TO 24
        GO TO 90
C      TEST   VORTICITY      FOR DIVERGENCE
 24     IF(DIFF .GT. 10) GO TO 29
        PRINT 94
 94     FORMAT(1H1,14H OMAEGA VALUES)
        CALL PRNTLST(OMA)
        PRINT 91
        CALL PRNTLST(PSI)
 32     FORMAT(10F11.6)
        NCOUNT=0
        GO TO 90
 29     PRINT 82
 82      FORMAT(13H OMA DIVERGED)
        CALL PRNTLST(PSI)
        CALL PRNTLST(OMA)
        GO TO 699
 199    PRINT 189
 189    FORMAT(26H OUTER ITERATIONS DIVERGED)
 700     CONTINUE
        PRINT 303,RMAX
 303    FORMAT(1H1,17H PSI CONVERGED TO,E12.4)
 699    CONTINUE
```