## ALGORITHM 401
## AN IMPROVED ALGORITHM TO PRODUCE COMPLEX PRIMES [A1]

PAUL BRATLEY (Recd. 25 Feb. 1970)
Département d'informatique, Université de Montréal, C.P. 6128, Montréal 101, Quebec, Canada

KEY WORDS AND PHRASES: number theory, prime numbers, complex numbers
*CR* CATEGORIES: 5.39

```
integer procedure cprimes(m, PR, PI);
  value m; integer m; integer array PR, PI;
comment The procedure generates the complex prime numbers
  located in the one-eighth plane defined by 0 ≤ y < x. Any prime
  found in that area has seven more associated primes: −x + yi,
  ± x − yi, ± y ± xi. These associated primes must be generated
  externally to cprimes. The first complex prime generated by
  cprimes is 1 + i, which exceptionally lies on x = y and has only
  three associated primes.
  The algorithm generates a list of complex primes in order of
  increasing modulus: the parameter m of the call is the highest
  modulus to be included in the list and should satisfy m > 2.
  PR and PI will contain respectively the real and imaginary
  parts of the generated list, with PR ≥ PI ≥ 0 for each prime.
  The value of the procedure is the number of primes generated.
  Algorithm 311 [1], sieve 2, is used to generate the rational
  primes less than m². Then it is known (see, for instance [2])
  that a rational prime p of the form p = 4n + 1 can be expressed
  as p = a² + b², and factorized as (a+bi)(a−bi) in the complex
  plane, where a + bi and a − bi are complex primes. For our
  present purpose we choose a > b and include only a + bi in the
  list. A rational prime p of the form p = 4n + 3 remains prime
  in the complex plane, so we include p + 0i in the list if p < m.
  Finally, the complex prime 1 + i may be thought of as one of
  the factors of the remaining rational prime 2 = (1+i)(1−i).
  Although this algorithm and Algorithm 372 [3] are not directly
  comparable, since they produce the list of complex primes in a
  different order, the accompanying remark suggests that the
  present algorithm is often to be preferred.
  REFERENCES:
  1. CHARTRES, B. A. Algorithm 311, Prime number generator 2.
     Comm. ACM 10 (Sept. 1967), 570.
  2. HARDY, G. H., AND E. M. WRIGHT. An Introduction to the
     Theory of Numbers, 4th ed. Clarendon Press, Oxford, 1965,
     Chs XII and XV.
  3. DUNHAM, K. B. Algorithm 372, An Algorithm to produce
     complex primes, CSIEVE. Comm. ACM 13 (Jan. 1970),
     52–53;
begin
  integer a, b, c, d, e, i, j, p, q;
  integer array P2[1:0.7×m↑2/ln(m)],
    P3[1:1.4×m/ln(m)];
  e := sieve 2(m↑2, P2);
  PR[1] := PI[1] := a := c := 1;
  b := 0;
  for d := 2 step 1 until e do
  begin
    p := P2[d]; q := p − 1;
    if (q÷4) × 4 ≠ q then
    begin
      if p ≤ m then
        begin b := b + 1; P3[b] := p end
    end
    else
    begin
L1:
      if a ≤ b then
      begin
        if P3[a] ↑ 2 < p then
        begin
          c := c + 1; PR[c] := P3[a];
          a := a + 1; PI[c] := 0;
          go to L1
        end
      end;
      q := entier(sqrt(p/2)+1);
      for i := q step 1 until p do
      begin
        j := sqrt(p−i↑2);
        if i ↑ 2 + j ↑ 2 = p then go to L2
      end
      comment Note that the jump to L2 is always made before
        the cycle is terminated;
L2:
      c := c + 1; PR[c] := i; PI[c] := j
    end
  end;
L3:
  if a ≤ b then
  begin
    c := c + 1; PR[c] := P3[a];
    a := a + 1; PI[c] := 0;
    go to L3
  end;
  cprimes := c
end cprimes
```

## ALGORITHM 402
## INCREASING THE EFFICIENCY OF QUICKSORT* [M1]

M. H. VAN EMDEN (Recd. 15 Dec. 1969 and 7 July 1970)
Mathematical Centre, Amsterdam, The Netherlands

* The algorithm is related to a paper with the same title and by the same author, which was published in *Comm. ACM 13* (Sept. 1970), 563–567.

KEY WORDS AND PHRASES: sorting, quicksort
*CR* CATEGORIES: 5.31, 3.73, 5.6, 4.49

```
procedure qsort(a, l1, u1);
  value l1, u1; integer l1, u1; array a;
comment This procedure sorts the elements a[l1], a[l1+1], ··· ,
  a[u1] into nondescending order. It is based on the idea described
  in [1]. A comparison of this procedure with another procedure,
  called sortvec, obtained by combining C. A. R. Hoare's quicksort
  [2] and R. S. Scowen's quickersort [3], in such a way as to be
  optimal for the Algol 60 system in use on the Electrologica X-8
  computer at the Mathematical Centre is shown below. Here
```

"repetitions" denotes the number of times the sorting of a sequence of that "length" is repeated; "average time" is the time in seconds averaged over the repetitions; "gain" is the difference in time relative to time taken by *sortvec*.

| *procedure* | *length* | *repetitions* | *average time* | *gain* |
|---|---|---|---|---|
| *sortvec* | 30 | 23 | .09 | |
| *qsort* | 30 | 23 | .06 | +.37 |
| *sortvec* | 300 | 16 | 1.25 | |
| *qsort* | 300 | 16 | 1.03 | +.17 |
| *sortvec* | 3000 | 9 | 17.43 | |
| *qsort* | 3000 | 9 | 15.25 | +.13 |
| *sortvec* | 30000 | 2 | 232.46 | |
| *qsort* | 30000 | 2 | 197.96 | +.15 |


REFERENCES:

1. VAN EMDEN, M. H. Increasing the efficiency of quicksort. *Comm. ACM 13* (Sept. 1970), 563–567.
2. HOARE, C. A. R. Algorithm 64, quicksort. *Comm. ACM 4* (July 1961), 321–322.
3. SCOWEN, R. S. Algorithm 271, quickersort. *Comm. ACM 8* (Nov. 1965), 669;


```
begin
  integer p, q, ix, iz;
  real x, xx, y, zz, z;
  procedure sort;
  begin
    integer l, u;
    l := l1;  u := u1;
part:
    p := l;  q := u;  x := a[p];  z := a[q];
    if x > z then
    begin y := x;  a[p] := x := z;  a[q] := z := y end;
    if u - l > 1 then
    begin
      xx := x;  ix := p;  zz := z;  iz := q;
left:
      for p := p + 1 while p < q do
      begin
        x := a[p];
        if x ≥ xx then go to right
      end;
      p := q - 1; go to out;
right:
      for q := q - 1 while q > p do
      begin
        z := a[q];
        if z ≤ zz then go to dist
      end;
      q := p;  p := p - 1;  z := x;  x := a[p];
dist:
      if x > z then
      begin
        y := x;  a[p] := x := z;
        a[q] := z := y
      end;
      if x > xx then
      begin xx := x;  ix := p end;
      if z < zz then
      begin zz := z;  iz := q end;
      go to left;
out:
      if p ≠ ix ∧ x ≠ xx then
      begin a[p] := xx;  a[ix] := x end;
      if q ≠ iz ∧ z ≠ zz then
      begin a[q] := zz;  a[iz] := z end;
      if u - q > p - l then
      begin l1 := l;  u1 := p - 1;  l := q + 1 end
      else
      begin u1 := u;  l1 := q + 1;  u := p - 1 end;
      if u1 > l1 then sort;
      if u > l then go to part
    end
  end of sort;
  if u1 > l1 then sort
end of qsort
```

## REMARK ON ALGORITHM 343 [F1] EIGENVALUES AND EIGENVECTORS OF A REAL GENERAL MATRIX [J. Grad and M. A. Brebner. *Comm. ACM 11* (Dec. 1968), 820–826]


WILLIAM KNIGHT AND WILLIAM MERSEREAU (Recd. 7 Apr. 1970)

Computing Center, University of New Brunswick, Fredericton, New Brunswick, Canada



KEY WORDS AND PHRASES: eigenvalues, eigenvectors, latent roots, Householder's method, QR algorithm, inverse iteration

*CR* CATEGORIES: 5.14


This remark reports certain failures of Algorithm 343 when applied to pathological matrices. The smallest example is a $4 \times 4$ matrix for which 16 guard bits (5+ digits) proved insufficient; all computed eigenvalues were incorrect in the most significant digit.

The algorithm was implemented on an IBM System/360 model 50 using Fortran IV-G. The program was not modified to operate completely in double precision as was done for Knoble's certification [2]. Satisfactory agreement was obtained for the three sample matrices given with the algorithm.

*Example A*

| | | | |
|---|---|---|---|
| −50 | 53 | 52 | 51 |
| −52 | 1 | 53 | 52 |
| −53 | 0 | 1 | 53 |
| −51 | 53 | 52 | 52 |

The exact eigenvalues are all 1. The computed eigenvalues follow. (Computed eigenvalues are reported rounded to 2 places after the decimal point, any further figures being, rather obviously, pointless.)

$$2.35$$
$$1.03 \pm 1.38\,i$$
$$-0.41$$

The maximum error in a computed eigenvalue exceeds 2 percent of the largest element of the matrix.

*Example B*

| | | | | | |
|---|---|---|---|---|---|
| −41 | 55 | 4 | 3 | 2 | 51 |
| − 2 | 10 | 55 | 4 | 3 | 2 |
| − 3 | 0 | 10 | 55 | 4 | 3 |
| − 4 | 0 | 0 | 10 | 55 | 4 |
| −55 | 0 | 0 | 0 | 10 | 55 |
| −51 | 55 | 4 | 3 | 2 | 61 |

The exact eigenvalues are all 10. The computed eigenvalues:

$$14.76 \pm 2.92\,i$$
$$9.70 \pm 5.33\,i$$
$$5.54 \pm 2.39\,i$$

The maximum error in a computed eigenvalue exceeds 9% of the largest element in the matrix.

*Example C*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| −91 | −94 | 0 | 0 | 0 | 0 | 0 | 0 |
| 95 | 98 | 0 | 0 | 0 | 0 | 0 | 0 |
| 90 | 99 | 5 | 0 | 0 | 0 | 0 | 0 |
| 90 | 0 | 99 | 6 | 0 | 0 | 0 | 0 |
| 90 | 0 | 0 | 99 | 7 | 0 | 0 | 0 |
| 90 | 0 | 0 | 0 | 99 | 8 | 0 | 0 |
| 90 | 0 | 0 | 0 | 0 | 99 | 9 | 0 |
| 99 | 99 | 0 | 0 | 0 | 0 | 99 | 10 |

The exact eigenvalues are 3, 4, 5, 6, 7, 8, 9, 10. The computed eigenvalues are:

$$12.68$$
$$10.96 \pm 3.73\,i$$
$$6.47 \pm 5.38\,i$$
$$2.09 \pm 3.73\,i$$
$$0.27$$

Although all eigenvalues are real, the imaginary part of one pair of computed eigenvalues exceeds 5 percent of the largest element of the matrix. This matrix, like the other two, was maliciously devised to take advantage of the program; it is indicative of this that the transpose, being already in lower Hessenberg form, fares much better, all computed eigenvalues being correct to within ±0.05.

Although, in view of the known sensitivity of multiple eigenvalues to small changes of certain elements of certain matrices, such counter examples are to be expected, it is probably worth putting a few examples on record as the casual and unsophisticated user is more apt to take warning of the dangers of eigenvalue computations in single precision from a concrete case.


REFERENCES:

[1] GRAD, J., AND M. A. BREBNER. Algorithm 343, Eigenvalues and eigenvectors of a real general matrix. *Comm. ACM 11* (Dec. 1968), 820–826.

[2] KNOBLE, H. D. Certification of Algorithm 343. Eigenvalues and eigenvectors of a real general matrix. *Comm. ACM 13* (Feb. 1970), 122–124.


## REMARKS ON ALGORITHM 372 [A1] AN ALGORITHM TO PRODUCE COMPLEX PRIMES, CSIEVE [K. B. Dunham. *Comm. ACM 13* (Jan. 1970), 52–53] ALGORITHM 401 [A1] AN IMPROVED ALGORITHM TO PRODUCE COMPLEX PRIMES [P. Bratley. *Comm. ACM 13* (Nov. 1970), 693]


PAUL BRATLEY (Recd. 25 Feb. 1970)

Département d'informatiqué, Universite de Montréal, C.P. 6128, Montréal 101, Quebec, Canada



KEY WORDS AND PHRASES: number theory, prime numbers, complex numbers

*CR* CATEGORIES: 5.39


Algorithm 372 was run on the CDC 6400 at the University of Montreal. The variable $i$ is undefined if the **for**-loop at label $A$ is completed. The statement

$$i := j + 1;$$

should be added immediately before label $B$. Algol purists may also care to remove redundant semicolons after **go to** $A$ and **go to** $B$, and the redundant parentheses in one **if**-statement. With these changes the algorithm produced correct results for several values of $m$.

The comment in Algorithm 372 is slightly inaccurate. The first prime generated by the algorithm is $1 + i$, which does not have $PR > PI$, and which has not seven but three associated primes.

It is not possible to compare the speeds of Algorithm 372 and Algorithm 401 directly since they generate primes in a different order. However, the following test was run. A value of $m$ was chosen, and Algorithm 401 was used to list all the complex primes with modulus less than $m$. The time taken and the number of primes produced were noted. Then Algorithm 372 was used to produce an equal number of primes, the time taken again being noted. Times observed are shown in Table I.

TABLE I

| *Limit on modulus* | *Algorithm 401 produced this number of primes* | *Time taken (secs)* | *Time taken by Algorithm 372 to produce the same number of primes (secs)* | *Ratio of times taken* |
|---|---|---|---|---|
| 25 | 60 | 0.278 | 0.331 | 1.2 |
| 50 | 189 | 1.577 | 2.140 | 1.4 |
| 75 | 373 | 4.217 | 7.602 | 1.8 |
| 100 | 623 | 8.618 | 20.214 | 2.4 |
| 150 | 1266 | 23.732 | 79.481 | 3.4 |

The conclusion from the figures in Table I is that if the speed with which the complex primes are generated is of paramount importance then Algorithm 401 should be preferred to Algorithm 372.

As written Algorithm 401 will use more memory than Algorithm 372 since it is convenient and perspicuous to use *sieve*2 in an unmodified form, which makes it necessary to store temporarily all the rational primes less than $m^2$. However, if space is tight then *sieve*2 can easily be modified so as to generate rational primes one at a time on successive calls, and in this way the use of the long array $P2$ can be avoided. If this modification is made Algorithm 401 will in fact use less store than Algorithm 372, which wastefully stores many useless values in $PM$. It is also to be noticed that the factors 0.7 and 1.4 occurring in the declarations of $P2$ and $P3$ may be diminished for large $m$: all that is necessary is that $P2$ should be long enough to hold the rational primes less than $m^2$, and that $P3$ should be long enough to hold the rational primes which are not greater than $m$ and which are of the form $4n + 3$. Some space may be saved similarly in *sieve*2, which is called from Algorithm 401.

The policy concerning the contributions of algorithms to *Communications of the ACM* has been revised and was published in the August 1970 issue, page 513. Copies of "Algorithm Policy/Revised August 1970" will be mailed upon request.

Important new features in the revised policy statement are: the acceptance of translations of algorithms; a change in policy on PL/I algorithms; clarification of conditions for acceptance, particularly with reference to providing evidence of a correct algorithm.