ZEROS OF DETERMINANTS OF λ-MATRICES

WALTER GANDER*

Abstract. Jim Wilkinson discovered that the computation of zeros of polynomials is ill conditioned when the polynomial is given by its coefficients. For many problems we need to compute zeros of polynomials, but we do not necessarily need to represent the polynomial with its coefficients. We develop algorithms that avoid the coefficients. They turn out to be stable, however, the drawback is often heavily increased computational effort. Modern processors on the other hand are mostly idle and wait for crunching numbers so it may pay to accept more computations in order to increase stability and also to exploit parallelism. We apply the method for nonlinear eigenvalue problems.

Key words. Nonlinear eigenvalue problems, Gaussian Elimination, Determinants, Algorithmic Differentiation.

AMS subject classifications. 35P30 , 65F15 , 65F05 , 65F40 ,

1. Introduction. The classical textbook approach to solve an eigenvalue problem $A\mathbf{x} = \lambda\mathbf{x}$ is to first compute the coefficients of the characteristic polynomial $P_n(\lambda) = \det(\lambda I - A)$ by expanding the determinant

$$P_n(\lambda) = c_0 + c_1\lambda + \cdots + c_{n-1}\lambda^{n-1} + \lambda^n.$$

Then second apply some iterative method like e.g. Newton's method to compute the zeros of P_n which are the eigenvalues of the matrix A.

In the beginning of the area of numerical analysis a research focus was to develop reliable solvers for zeros of polynomials. A typical example is e.g. [4]. However, the crucial discovery by Jim Wilkinson [6] was that the zeros of a polynomial can be very sensitive to small changes of the coefficients of the polynomial. Thus the determination of the zeros from the coefficients is ill conditioned. It is easy today to repeat the experiment using a computer algebra system. Executing the following Maple statements

```
p :=1:
for i from 1 by 1 to 20 do  p := p*(x-i) od:
PP := expand(p);
Digits := 7
PPP := evalf(PP)
Digits := 30
Z := fsolve(PPP, x, complex, maxsols = 20)
```

we can simulate what Jim Wilkinson experienced. We first expand the product

$$\prod_{i=1}^{20}(x-i) = x^{20} - 210x^{19} \pm \cdots + 20!$$

then round the coefficients to floating point numbers with 7 decimal digits.

$$x^{20} - 210.0\,x^{19} + 2.432902 \times 10^{18} \mp \cdots - 8.752948 \times 10^{18}\,x + 20615.0\,x^{18}$$

Continuing now the computation with 30 decimal digits to determine the exact zeros of the polynomial with truncated coefficients we note that we do not obtain the numbers $1, 2, \ldots, 20$. Instead many zeros are complex such as e.g. $17.175 \pm 9.397i$. Thus truncating the coefficients to 7 decimal digits has a very large effect on the zeros. The problem is ill conditioned.

*Computational Science, ETH, CH-8092 Zurich, Switzerland (gander@inf.ethz.ch).

2. Matlab Reverses Computing. Instead of expanding the determinant to obtain the coefficients of the characteristic polynomial the command `P = poly(A)` in Matlab computes the eigenvalues of A by the QR-Algorithm and expands the linear factors

$$P_n(\lambda) = (\lambda - \lambda_1)(\lambda - \lambda_2)\cdots(\lambda - \lambda_n) = \lambda^n + c_{n-1}\lambda^{n-1} + \cdot + c_0$$

to compute the coefficients.

Given on the other hand the coefficients c_k of a polynomial, the command `lambda = roots(P)` forms the companion matrix

$$A = \begin{pmatrix} -c_{n-1} & -c_{n-2} & \cdots & -c_1 & -c_0 \\ 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

and uses again the QR-Algorithm to find the eigenvalues which are the zeros of the polynomial.

3. Evaluating the Characteristic Polynomial. How can we evaluate the characteristic polynomial without first computing its coefficients? One way is to use Gaussian elimination and the fact that it is easy to compute the determinant of a triangular matrix. Assume that we have computed the decomposition

$$C = LU$$

with L a lower unit triangular and U an upper triangular matrix. Then

$$\det(C) = \det(L)\det(U) = u_{11}u_{22}\cdots u_{nn}$$

since $\det(L) = 1$. Using partial pivoting for the decomposition we have to change the sign of the determinant each time that we interchange two rows. The program then becomes:

```
function f = determinant(C)
n = length(C);
f = 1;
for i = 1:n
   [cmax,kmax]= max(abs(C(i:n,i)));
   if cmax == 0  %  Matrix singular
      f = 0; return
   end
   kmax = kmax+i-1;
   if kmax ~= i
      h = C(i,:); C(i,:) = C(kmax,:); C(kmax,:) = h;
      f = -f;
   end
   f = f*C(i,i);
  % elimination step
   C(i+1:n,i) = C(i+1:n,i)/C(i,i);
   C(i+1:n,i+1:n) = C(i+1:n,i+1:n) - C(i+1:n,i)*C(i,i+1:n);
end
```

Let $C(\lambda) = \lambda I - A$. We would like to use Newton's method to compute zeros of $P(\lambda) = \det(C(\lambda)) = 0$. For this we need the derivative $P'(\lambda)$. It can be computed by *algorithmic differentiation*, that is by differentiating each statement of the program to compute $P(\lambda)$. For instance the statement to update the determinant `f = f*C(i,i);` will be preceded by the statement for the derivative, thus

```
fs =fs*C(i,i)+f*Cs(i,i) ; f = f*C(i,i);
```

We used the variable `Cs` for the matrix $C'(\lambda)$ and `ds` for the derivative of the determinant.

There is, however, for larger matrices the danger that the value of the determinant over- respectively underflows. Notice that for Newton's iteration we do not need both values $f = \det(A - \lambda I)$ and $fs = \frac{d}{d\,\lambda} \det(A - \lambda I)$. It is sufficient to compute the *ratio*

$$\frac{P(\lambda)}{P'(\lambda)} = \frac{f}{fs}.$$

Overflow can be reduced by computing the logarithm. Thus instead of computing `f = f*C(i,i)` we can compute `lf = lf + log(C(i,i))`. Even better is the derivative of the logarithm

$$lfs := \frac{d}{d\,\lambda} \log(f) = \frac{fs}{f}$$

which yields directly the *inverse Newton correction*.

Thus instead updating the logarithm $lf = lf + \log(c_{ii})$ we directly compute the derivative

$$lfs = lfs + \frac{cs_{ii}}{c_{ii}}.$$

This considerations lead to

```
function ffs = deta(C,Cs)
% DETA computes Newton correction ffs = f/fs
n = length(C);
lfs = 0;
for i = 1:n
   [cmax,kmax]= max(abs(C(i:n,i)));
   if cmax == 0  %  Matrix singular
      ffs = 0; return
   end
   kmax = kmax+i-1;
   if kmax ~= i
      h = C(i,:); C(i,:) = C(kmax,:); C(kmax,:) = h;
      h = Cs(kmax,:); Cs(kmax,:) = Cs(i,:); Cs(i,:) = h;
   end
   lfs = lfs + Cs(i,i)/C(i,i);
  % elimination step
   Cs(i+1:n,i) = (Cs(i+1:n,i)*C(i,i)-Cs(i,i)*C(i+1:n,i))/C(i,i)^2;
   C(i+1:n,i) = C(i+1:n,i)/C(i,i);
   Cs(i+1:n,i+1:n) = Cs(i+1:n,i+1:n) - Cs(i+1:n,i)*C(i,i+1:n)- ...
                     C(i+1:n,i)*Cs(i,i+1:n);
   C(i+1:n,i+1:n) = C(i+1:n,i+1:n) - C(i+1:n,i)*C(i,i+1:n);
end
ffs = 1/lfs;
```

Note that as an alternative to the algorithmic differentiation presented here one could use the *Formula of Jacobi*

$$\frac{d}{d\,\lambda}\det(C(\lambda)) = \det(C(\lambda))\ \mathrm{trace}\left(C^{-1}(\lambda)C'(\lambda)\right)$$

which gives an explicit expression for the derivative of the determinant.

4. Suppression instead Deflation. If $x_1, \ldots, x_k$ are already computed zeros then we would like to continue working with the deflated polynomial

$$P_{n-k}(x) := \frac{P_n(x)}{(x-x_1)\cdots(x-x_k)} \tag{4.1}$$

of degree $n-k$. However, we cannot explicitly deflate the zeros since we are working with $P(\lambda) = \det(\lambda I - A)$. Differentiating Equation (4.1) we obtain

$$P'_{n-k}(x) = \frac{P'_n(x)}{(x-x_1)\cdots(x-x_k)} - \frac{P_n(x)}{(x-x_1)\cdots(x-x_k)}\sum_{i=1}^{k}\frac{1}{x-x_i}.$$

Thus the Newton-iteration becomes

$$x_{new} = x - \frac{P_{n-k}(x)}{P'_{n-k}(x)} = x - \frac{P_n(x)}{P'_n(x)}\frac{1}{1-\dfrac{P_n(x)}{P'_n(x)}\displaystyle\sum_{i=1}^{k}\frac{1}{x-x_i}}$$

This variant of Newton's Iteration is called *Newton-Maehly Iteration* [2, 3].

5. Example. We generate a random symmetric matrix A with eigenvalues $1, 2, \ldots, n$:

```
x = [1:n]'; Q = rand(n);
Q = orth(Q); A = Q*diag(x)*Q';
```

respectively a non symmetric matrix with

```
x = [1:n]'; Q = rand(n);
A = Q*diag(x)*inv(Q);
```

Then we compute the solutions of $\det(C(\lambda)) = 0$ with $C(\lambda) = \lambda I - A$ using the Newton-Maehly iteration. We compare the results with the ones obtained by the QR-

n	$roots(poly(A))$	$eig(A)$	$\det(A-\lambda I)=0$
50	1.3598e+02	3.9436e−13	4.7243e−14
100	9.5089e+02	1.1426e−12	1.4355e−13
150	2.8470e+03	2.1442e−12	3.4472e−13
200	− − −	3.8820e−12	6.5194e−13

TABLE 5.1
Norm of difference of the computed to the exact eigenvalues for a symmetric matrix

Algorithm *eig(A)* and with the zeros of the characteristic polynomial *roots(poly(A))*. In Tables 5.1 and 5.2 the norm of the difference of the computed eigenvalues to the exact ones is printed. Notice that due to ill-conditioning the roots of the characteristic polynomial differ very much and that for $n = 200$ the coefficients of the characteristic polynomial overflow and the zeros cannot be computed any more. On the other hand we can see that the our method competes in accuracy very well with the standard QR-algorithm.

n	$roots(poly(A))$	$eig(A)$	$\det(A-\lambda I)=0$
50	$1.3638e{+}02$	$3.7404e{-}12$	$2.7285e{-}12$
100	$9.7802e{+}02$	$3.1602e{-}11$	$3.5954e{-}11$
150	$2.7763e{+}03$	$6.8892e{-}11$	$3.0060e{-}11$
200	$---$	$1.5600e{-}10$	$6.1495e{-}11$

TABLE 5.2

Norm of difference of the computed to the exact eigenvalues for a non-symmetric matrix

6. Generalization to λ-matrices. Consider a quadratic eigenvalue problem

$$\det(C(\lambda))=0, \quad \text{with} \quad C(\lambda)=\lambda^2 M+\lambda C+K.$$

If $\det(M)\neq 0$ then one way to "linearize" the problem is to consider the equivalent general eigenvalue-problem with dimension $2n$:

$$\det\left(\begin{bmatrix} M & 0 \\ 0 & K \end{bmatrix}-\lambda\begin{bmatrix} 0 & M \\ -M & -C \end{bmatrix}\right)=0$$

Alternatively with our approach we can compute the zeros of $\det(C(\lambda))$ with Newton's iteration. Take the mass-spring system example from [5]. For the nonoverdamped case the matrix is $C(\lambda)=\lambda^2 M+\lambda C+K$ with

$$M=I,\ C=\tau\,\mathrm{tridiag}(-1,3,-1),\ K=\kappa\,\mathrm{tridiag}(-1,3,-1)$$

and with $\kappa=5$, $\tau=3$ and $n=50$. The Matlab program to compute the eigenvalues is

```
% Figure 3.3 in Tisseur-Meerbergen
clear, format compact
n=50
tau = 3, kappa = 5,
e = -ones(n-1,1);
C = (diag(e,-1)+ diag(e,1)+ 3*eye(n));
K = kappa*C;
C = tau*C;
lam = -0.5+0.1*i;
tic
for k=1:2*n
  ffs = 1; q=0;
  while abs(ffs)>1e-14
    Q = lam*(lam*eye(n)+ C)+K;
    Qs = 2*lam*eye(n)+C;
    ffs = deta(Q,Qs);
    s = 0;
    if k>1
       s = sum(1./(lam-lamb(1:k-1)));
    end
    lam = lam-ffs/(1-ffs*s); q=q+1;
  end
  clc
  k, lam, q, ffs, lamb(k) = lam;
  lam = lam*(1+0.01*i);
```

```
end
toc
clf
 plot(real(lamb),imag(lamb),'o')
```

and produces Figure 6.1. The computation in Matlab needed 13.9 seconds on a

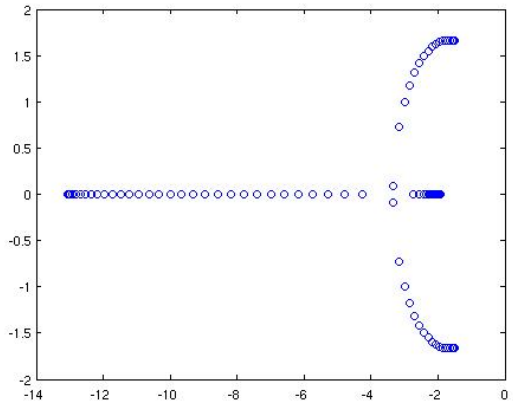

FIG. 6.1. *Eigenvalues in the complex plane for the nonoverdamped case*

IBM X41 laptop. As starting values for the iteration we used the complex number $\lambda(1 + 0.01i)$ near the last computed eigenvalue λ. In the second "overdamped" case

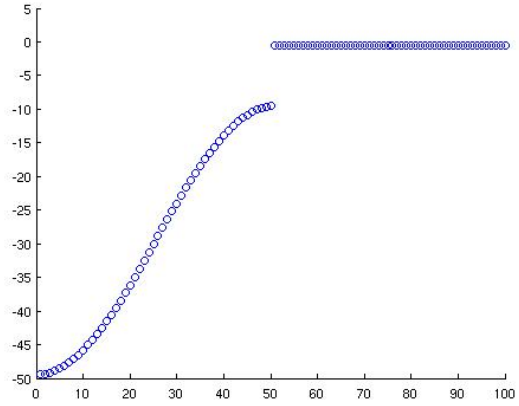

FIG. 6.2. *Real eigenvalues for the overdamped case*

we have $\kappa = 5$, $\tau = 10$. Since the eigenvalues are all real we can choose real starting values. We chose 1.01λ where again λ was the last eigenvalue found. Figure 6.2 shows the eigenvalues which are all real and computed with Matlab in 16.3 seconds.

Finally we recomputed a cubic eigenvalue problem from [1]. Here we have

$$C(\lambda) = \lambda^3 A_3 + \lambda^2 A_2 + \lambda A_1 + A_0$$

with

$$A_0 = \text{tridiag}(1, 8, 1)\ A_2 = \text{diag}(1, 2, \ldots, n) \text{ and } A_1 = A_3 = I.$$

In [1] the matrix dimension was $n = 20$ thus 60 eigenvalues had to be computed. Using our method we compute these in 1.9 seconds. Figure 6.3 shows the 150 eigenvalues for $n = 50$ which have been computed in 17.9 seconds.

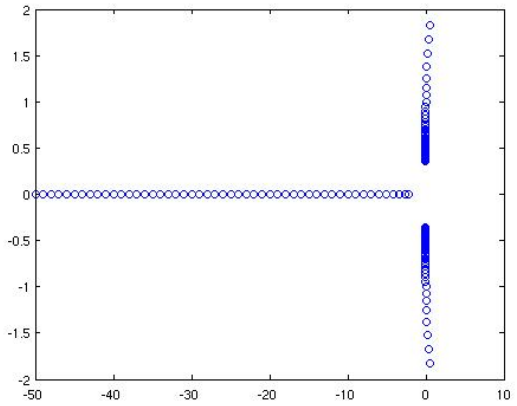

FIG. 6.3. *Cubic Eigenvalue Problem*

REFERENCES

[1] P. ARBENZ AND W. GANDER, *Solving nonlinear Eigenvalue Problems by Algorithmic Differentiation*, Computing 36, 205-215, 1986.
[2] H. J. MAEHLY, *Zur iterativen Auflösung algebraischer Gleichungen*, ZAMP (Zeitschrift für angewandte Mathematik und Physik), (1954), pp. 260–263.
[3] J. STOER AND R. BULIRSCH, *Introduction to Numerical Analysis*, Springer, 1991.
[4] W. KELLENBERGER, *Ein konvergentes Iterationsverfahren zur Berechnung der Wurzeln eines Polynoms*, Z. Angew. Math. Phys. 21 (1970) 647–651.
[5] F. TISSEUR AND K. MEERBERGEN, *The Quadratic Eigenvalue Problem*, SIAM. Rev., 43, pp. 234–286, 2001.
[6] J. H. WILKINSON, *Rounding errors in algebraic processes*, Dover Publications, 1994.

7. Conclusion. We have demonstrated that computing zeros of polynomials from their coefficients is *ill-conditioned*. However, direct evaluation of the characteristic polynomial is feasible. With this computational intensive method we have shown that medium size nonlinear eigenvalue problems may be solved with a simple program which computes determinants by Gaussian elimination and applies algorithmic differentiation and suppresses already computed zeros. We obtained results in reasonable time in spite that we did not compile the Matlab program and that we did not make use of the banded structure of the matrices. This algorithm, though computational expensive, maybe useful for its potential for parallelization on future multicore architectures.