NEW ALGORITHMS FOR SOLVING NONLINEAR EIGENVALUE PROBLEMS

WALTER GANDER*

Abstract. To solve a nonlinear eigenvalue problem we develop algorithms which compute zeros of $\det A(\lambda) = 0$. We show how to apply third order iteration methods for that purpose. The necessary derivatives of the determinant are computed by algorithmic differentiation. Since many nonlinear eigenvalue problems have banded matrices we also present an algorithm which makes use of their structure.

Key words. Nonlinear Eigenvalue Problem, Third Order Methods, Algorithmic Differentiation.

AMS subject classifications. 11C20, 30C15, 35P30.

1. Introduction.

Let $A : \lambda \mapsto \mathbb{C}^{n\times n}$ be analytic on a open set $\{\lambda\} \subset \mathbb{C}$. We consider the problem to find λ such that $f(\lambda) = \det A(\lambda) = 0$.

To compute a zero of f one can consider Newton's method

$$\lambda_{k+1} = \lambda_k - \frac{f(\lambda_k)}{f'(\lambda_k)},$$

which needs the derivative of the determinant. The *formula of Jacobi*, well known and discussed in linear algebra textbooks, gives an explicit expression for it:

$$f'(\lambda) = \det A(\lambda)\,\mathrm{trace}(A^{-1}(\lambda)A'(\lambda)).$$

The Newton correction becomes

$$\frac{f(\lambda_k)}{f'(\lambda_k)} = \frac{1}{\mathrm{trace}(A^{-1}(\lambda_k)A'(\lambda_k))}.$$

An alternative way to compute the derivative, respectively the Newton correction, is by *algorithmic differentiation* (see [1], [11], [4])

2. Computing Determinants and the Newton Correction.

Numerically a good method to compute a determinant is applying Gaussian elimination to compute the LU-decomposition. If $PA = LU$ where P is the permutation matrix resulting from partial pivoting, L is the lower unit triangular matrix and U is the upper triangular factor, then

$$\det(A) = \pm u_{11}u_{22}\cdots u_{nn}.$$

When overwriting the matrix A by the LU-decomposition, the value of the determinant is updated in the kth elimination step by multiplying with the pivot element $f := f \times a_{kk}$. Determinants tend soon to over- or underflow, therefore it is better to compute their logarithm: $\mathrm{logf} := \mathrm{logf} + \log(a_{kk})$.

Note that the derivative of the logarithm

$$\frac{d}{d\lambda}\log f(\lambda) = \frac{f'(\lambda)}{f(\lambda)}$$

is the inverse Newton correction. Thus if we compute the derivative of the logarithm by algorithmic differentiation,

$$\mathrm{logf} := \mathrm{logf} + \log(a_{kk}) \implies \mathrm{logfp} := \mathrm{logfp} + \frac{a'_{kk}}{a_{kk}},$$

*ETH Zurich and HKBU Hong Kong (gander@inf.ethz.ch, http://people.inf.ethz.ch/gander/)

the inverse ffp $= 1/$logfp $= f(\lambda)/f'(\lambda)$ is the Newton correction we need. This has been used in the following program in [11]:

```
function ffp=deta(A,Ap)
% DETA compute determinant of A  and derivative
% Given A=A(lambda) and Ap=A'(lambda), DETA(A,Ap)
% computes Newton correction ffp=f/f' where f=det(A).
n=length(A); logfp=0;
for j=1:n
   [amax,kmax]= max(abs(A(j:n,j)));          % partial pivoting
   if amax == 0,ffp=0; return, end
   kmax=kmax+j-1;
   if kmax ~= j                              % interchange rows
      h=Ap(kmax,:); Ap(kmax,:)=Ap(j,:); Ap(j,:)=h;
      h=A(j,:); A(j,:)=A(kmax,:); A(kmax,:)=h;
   end
   logfp=logfp + Ap(j,j)/A(j,j);
   Ap(j+1:n,j)=(Ap(j+1:n,j)*A(j,j)-A(j+1:n,j)*Ap(j,j))/A(j,j)^2;
   A(j+1:n,j)=A(j+1:n,j)/A(j,j);
   Ap(j+1:n,j+1:n)=Ap(j+1:n,j+1:n) - Ap(j+1:n,j)*A(j,j+1:n)- ...
                         A(j+1:n,j)*Ap(j,j+1:n);
   A(j+1:n,j+1:n)=A(j+1:n,j+1:n) - A(j+1:n,j)*A(j,j+1:n);
end
ffp=1/logfp;
```

3. Suppression Instead of Deflation. With the function `deta` we can compute a solution of $\det A(\lambda) = 0$ by Newton's method. In order to avoid recomputing already computed zeros $\lambda_1, \ldots, \lambda_k$, we suppress them by working with the function

$$f_k(\lambda) := \frac{f(\lambda)}{p(\lambda)},$$

where $p(\lambda) = (\lambda - \lambda_1) \cdots (\lambda - \lambda_k)$. Then

$$p'(\lambda) = \sum_{j=1}^{k} \prod_{\substack{i=1\\ i\neq j}}^{k} (\lambda - \lambda_i) = p(\lambda)s(\lambda), \quad \text{where} \quad s(\lambda) = \sum_{j=1}^{k} \frac{1}{\lambda - \lambda_j}.$$

The derivative of f_k is (we omit the argument λ):

$$f_k' = \frac{pf' - psf}{p^2} = \frac{f' - sf}{p}.$$

The Newton correction f_k/f_k' expressed in terms of f and f' becomes

$$\frac{f_k}{f_k'} = \frac{f/p}{(f' - sf)/p} = \frac{f}{f' - sf} = \frac{f}{f'} \frac{1}{1 - \frac{f}{f'}s}. \tag{3.1}$$

The resulting iteration

$$\lambda_{j+1} = \lambda_j - \frac{f_k(\lambda_j)}{f_k'(\lambda_j)} = \lambda_j - \frac{f(\lambda_j)}{f'(\lambda_j)} \frac{1}{1 - \frac{f(\lambda_j)}{f'(\lambda_j)} \sum_{j=1}^{k} \frac{1}{\lambda - \lambda_j}}$$

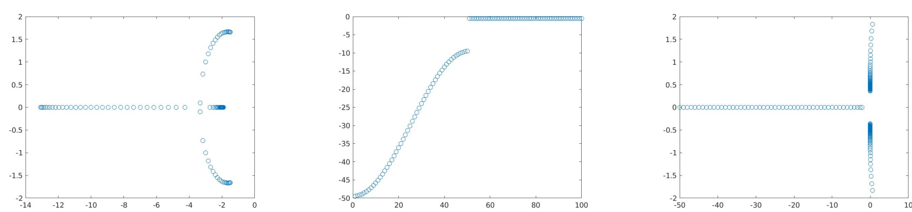
is called *Newton-Maehly* iteration [7].

In [11] we computed two mass-spring examples from [2] and the cubic example in [1]. The MATLAB program to compute the first mass-spring example is

```
n=50, tau=3, kappa=5,               % nonoverdamped
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; lamb=[];            % start
for k=1:2*n
  ffp=1;
  while abs(ffp)>1e-14
    Qp=2*lam*eye(n)+C; Q=lam*(lam*eye(n)+ C)+K;
    ffp=deta(Q,Qp);
    s=sum(1./(lam-lamb(1:k-1)));
    lam=lam-ffp/(1-ffp*s);         % Newton step
  end
  lamb(k)=lam;
  lam=lam*(1+0.01*i);              % start for next eigenvalue
end
plot(lamb,'o')
```

We start the iteration for the first eigenvalue with some random complex number, here $\lambda_0 = -0.5 + 0.1\,i$. As initial value for the following eigenvalues we choose the last computed and suppressed eigenvalue λ_k with some small perturbation: $\lambda_0 = \lambda_k(1 + i/100)$.

Similarly we compute the overdamped mass-spring example and the cubic eigenvalue problem for $n = 50$. The resulting eigenvalues are plotted in Figure 1.



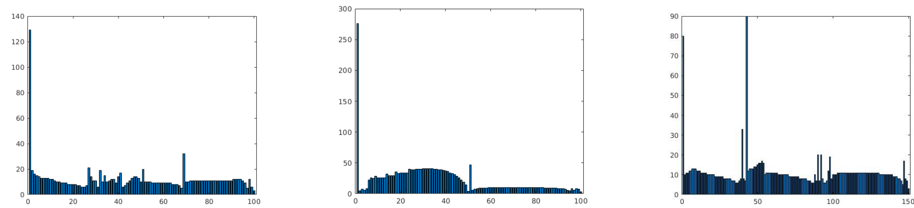
nonoveredamped   overdamped   cubic eigenvalue problem

FIG. 1. *Examples solved with Newton Iteration.*

It is interesting to display the number of iterations needed for each eigenvalue. The bar plots and the mean number of iterations are shown in Figure 2. The initial value for the iteration for the first eigenvalue is obviously not well chosen – a large number of iterations is needed to converge. For the cubic eigenvalue problem large numbers of iterations also occur in between for some other eigenvalues.

4. Improving Convergence. The reason for many iteration steps needed for computing the eigenvalues in the last three examples is because of the rather poor global convergence of Newton's method. Locally, Newton's method converges quadratically to a simple zero, thus after 3 to 4 iterations one should obtain a result to machine precision.

Let $f(z) = 0$ and λ_k be an approximation near z. Newton's iteration replaces f at λ_k by a *linear function* g such that $f(\lambda_k) = g(\lambda_k)$, $f'(\lambda_k) = g'(\lambda_k)$. Thus g is the Taylor-polynomial $g(\lambda) = f(\lambda_k) + f'(\lambda_k)(\lambda - \lambda_k)$ and the next iterate λ_{k+1} is the zero of g.



mean = 12.0 nonoveredamped | mean = 22.3 overdamped | mean= 11.3 cubic eigenvalue problem

FIG. 2. *Iterations needed.*

Halley's Iteration

$$\lambda_{k+1} = \lambda_k - \frac{f(\lambda_k)}{f'(\lambda_k)} \frac{1}{1 - \frac{1}{2}\frac{f(\lambda_k) f''(\lambda_k)}{f'(\lambda_k)^2}} \tag{4.1}$$

replaces f locally by a *hyperbolic function*

$$g(\lambda) = \frac{a}{\lambda + b} + c$$

such that $f(\lambda_k) = g(\lambda_k)$, $f'(\lambda_k) = g'(\lambda_k)$ and $f''(\lambda_k) = g''(\lambda_k)$ and the next iterate λ_{k+1} is the zero of g. Halley's iteration is a third order method which means that it converges cubically to a simple zero [10]. Possibly this hyperbolic approximation of f leads to better global convergence.

5. Implementing Halley's Iteration. We need the *second derivative* of the determinant, more precisely, we need to compute the function

$$t(\lambda) = \frac{f(\lambda) f''(\lambda)}{f'^2(\lambda)}.$$

Note that the derivative of Newton's correction is

$$\frac{d}{dx}\left(\frac{f}{f'}\right) = \frac{f'^2 - f f''}{f'^2} = 1 - \frac{f f''}{f'^2}.$$

Thus

$$t = \frac{f f''}{f'^2} = 1 - \frac{d}{dx}\left(\frac{f}{f'}\right),$$

and we only need to compute the derivative of the Newton correction in our function `deta` to get $t(\lambda)$. This can be done by algorithmic differentiation of the function `deta`.

The following function `det2p` needs as input the matrices A, A' and A'' and computes the Newton correction f/f' and its derivative.

```
function [ffp,dffp] = det2p(A,Ap,App)
% DET2P computes Newton correction ffp = f/f'
%       and its derivative  dffp = (f/f')'
n=length(A);
logfpp=0;                             % logfpp = log(f)''
logfp=0;                              % log(f)'
```

```
for k=1:n
  [amax,kmax]=max(abs(A(k:n,k)));        % partial pivoting
  if amax==0                             % matrix singular
    ffp=0; dffp=0;return
  end
  kmax=kmax+k-1;
  if kmax~=k                             % interchange rows
    h=App(k,:); App(k,:)=App(kmax,:); App(kmax,:)=h;
    h=Ap(k,:); Ap(k,:)=Ap(kmax,:); Ap(kmax,:)=h;
    h=A(k,:); A(k,:)=A(kmax,:); A(kmax,:)=h;
  end
  logfpp=logfpp+(A(k,k)*App(k,k)-Ap(k,k)^2)/A(k,k)^2;
  logfp=logfp+Ap(k,k)/A(k,k);
  App(k+1:n,k)=(A(k,k)*App(k+1:n,k)-Ap(k+1:n,k)*Ap(k,k))/A(k,k)^2-...
       (Ap(k+1:n,k)*Ap(k,k)/A(k,k)^2+ ...
       A(k+1:n,k)*App(k,k)/A(k,k)^2-...
       2* A(k+1:n,k)*Ap(k,k)^2/A(k,k)^3);

  Ap(k+1:n,k)=Ap(k+1:n,k)/A(k,k)-A(k+1:n,k)*Ap(k,k)/A(k,k)^2;
  A(k+1:n,k)=A(k+1:n,k)/A(k,k);         % elimination step

  App(k+1:n,k+1:n)=App(k+1:n,k+1:n) -...
      (App(k+1:n,k)*A(k,k+1:n) + Ap(k+1:n,k)*Ap(k,k+1:n) )  - ...
      (Ap(k+1:n,k)*Ap(k,k+1:n) + A(k+1:n,k)*App(k,k+1:n));

  Ap(k+1:n,k+1:n)=Ap(k+1:n,k+1:n) - Ap(k+1:n,k)*A(k,k+1:n)-...
      A(k+1:n,k)*Ap(k,k+1:n);
  A(k+1:n,k+1:n)=A(k+1:n,k+1:n) - A(k+1:n,k)*A(k,k+1:n);
end
dffp=-logfpp/logfp^2; ffp=1/logfp;
```

6. Halley-Maehly. As before we want to suppress already computed eigenvalues and consider again

$$f_k(\lambda) := \frac{f(\lambda)}{p(\lambda)}, \quad p(\lambda) = (\lambda - \lambda_1)\cdots(\lambda - \lambda_k).$$

We now apply Halley's iteration to f_k

$$\lambda_{\text{new}} = \lambda - \frac{f_k}{f_k'} \frac{1}{1 - \frac{1}{2}\frac{f_k f_k''}{{f_k'}^2}},$$

and express the iteration in terms of f, f' and f''. For the Newton correction f_k/f_k' we use Equation (3.1). For f_k''/f_k' we compute first

$$\begin{aligned} f_k'' &= \frac{d}{d\lambda}\left(\frac{f' - sf}{p}\right) = \frac{p(f'' - s'f - sf') - p'(f' - sf)}{p^2} \\ &= \frac{f'' - s'f - sf' - sf' + s^2 f}{p}, \quad p' = ps, \quad s = \sum_{j=1}^{k} \frac{1}{\lambda - \lambda_j}. \end{aligned}$$

Then dividing with f_k' we get

$$\frac{f_k''}{f_k'} = \frac{f'' - s'f - 2sf' + s^2 f}{f' - sf} = \frac{\frac{f''}{f'} - s'\frac{f}{f'} - 2s + s^2\frac{f}{f'}}{1 - s\frac{f}{f'}},$$

		nonoverdamped	overdamped	cubic EVP
Newton				
	max iterations	128	275	90
	mean iterations	11.4	20.9	11.3
Halley				
	max iterations	67	140	46
	mean iterations	7	12.1	7.1

TABLE 1
Comparing Newton and Halley

and by multiplying with f_k/f_k' we obtain

$$t = \frac{f_k f_k''}{{f_k'}^2} = \frac{\frac{ff''}{f'^2} + (s^2 - s')\left(\frac{f}{f'}\right)^2 - 2s\frac{f}{f'}}{\left(1 - s\frac{f}{f'}\right)^2}. \tag{6.1}$$

Summarizing we compute a Halley-Maehly iteration step as follows:

1. Compute Newton's correction for f_k:
$$\frac{f_k}{f_k'} = \frac{f}{f'}\frac{1}{1 - \frac{f}{f'}s}.$$
2. Compute $t(\lambda)$ for f_k according to Equation (6.1).
3. Iterate
$$\lambda_{\text{new}} = \lambda - \frac{f_k}{f_k'}\frac{1}{1 - \frac{1}{2}t}.$$

We solve the three NEV-problems with Halley and compare the results with those of Newton's iteration, see Table 1. Indeed global convergence has improved, we need fewer iterations with Halley.

7. Laguerre and Ostrowski. Another third order method which is designed for zeros of polynomials is *Laguerre's Method*. This method uses as approximation for a polynomial function f of degree n the polynomial $g(\lambda) = a(\lambda - \lambda_1)(\lambda - \lambda_2)^{n-1}$. The parameters a, λ_1 and λ_2 are determined such that g interpolates f and its derivatives $f(\lambda_k) = g(\lambda_k)$, $f'(\lambda_k) = g'(\lambda_k)$, $f''(\lambda_k) = g''(\lambda_k)$. The next iteration is the zero of g closer to λ_k:

$$\lambda_{k+1} = \lambda_k - \frac{f(\lambda_k)}{f'(\lambda_k)}\frac{n}{1 + \sqrt{(n-1)^2 - n(n-1)\frac{f(\lambda_k)f''(\lambda_k)}{f'(\lambda_k)^2}}}. \tag{7.1}$$

The degree n is a parameter of Laguerre's method. If we let $n \to \infty$ in Equation (7.1) then we get the iteration

$$\lambda_{k+1} = \lambda_k - \frac{f(\lambda_k)}{f'(\lambda_k)}\frac{1}{\sqrt{1 - \frac{f(\lambda_k)f''(\lambda_k)}{f'(\lambda_k)^2}}} \tag{7.2}$$

which is *Ostrowski's Square Root Iteration*. Note that for both iterations Laguerre and Ostrowski we need as for Halley only the two expressions

$$\frac{f}{f'} \quad \text{and} \quad t = \frac{ff''}{f'^2}.$$

	nonoverdamped	overdamped	cubic EVP
Halley			
max iterations	67	140	46
mean iterations	7	12.1	7.1
Laguerre			
max iterations	18	36	16
mean iterations	5.3	6.6	5.2
Ostrowski			
max iterations	23	43	18
mean iterations	5.5	7.1	5.2

TABLE 2
Halley, Laguerre and Ostrowski

Since Laguerre's iteration is designed for zeros of polynomials we can expect a good performance on our three examples. Indeed comparing the three methods in Table 2 for the three examples shows that this is the case.

8. Third Order Methods. Halley, Laguerre and Ostrowski are special cases of the following theorem

THEOREM 8.1 (Third Order Methods [10]). *If s is a simple zero of f, G any function with $G(0) = 1$, $G'(0) = \frac{1}{2}$ and $|G''(0)| < \infty$, then*

$$x_{\text{new}} = x - \frac{f(x)}{f'(x)}\, G(t(x)), \quad t(x) = \frac{f(x)f''(x)}{f'(x)^2}$$

converges at least cubically *to s.*

Examples:
- Halley's formula: $G(t) = \frac{1}{1-\frac{1}{2}t} = 1 + \frac{1}{2}t + \frac{1}{4}t^2 + \frac{1}{8}t^3 + \dots$
- Euler's formula: $G(t) = \frac{2}{1+\sqrt{1-2t}} = 1 + \frac{1}{2}t + \frac{1}{2}t^2 + \frac{5}{8}t^3 + \dots$
- Quadratic inverse interpolation: $G(t) = 1 + \frac{1}{2}t$
- Ostrowski's square root iteration: $G(t) = \frac{1}{\sqrt{1-t}} = 1 + \frac{1}{2}t + \frac{3}{8}t^2 + \dots$
- Laguerre: $G(t) = \frac{n}{1+\sqrt{(n-1)^2 - n(n-1)t}} = 1 + \frac{1}{2}t + \frac{1}{8}\,\frac{3\,n-2}{n-1}t^2 + \dots$
- Hansen-Patrick family [5]: $G(t) = \frac{\alpha+1}{\alpha+\sqrt{1-(\alpha+1)t}} = 1 + \frac{1}{2}t + \frac{\alpha+3}{8}t^2 + \dots$

We note that in order to apply these iteration formulas we need only to compute the Newton-correction and $t = ff''/f'^2$.

9. NLEVP – Resources for Nonlinear EV Problems. T. Betcke, N. J. Higham, V. Mehrmann, C. Schröder, and F. Tisseur have assembled a remarkable collection of nonlinear eigenvalue problems[1] (see [8] and [9]). The examples include all sorts of matrices.

Using Laguerre's method we computed the two quadratic eigenvalue problems `sign1` and `sign2` (dense matrices, $n = 81$). The results are given in Table 3. `sign1` has the $2n = 162$ eigenvalues on the unit circle with two accumulation points at ± 1. Convergence to zeros of these two clusters is slow as we can see from the bar plot. Convergence for `sign2` is much better as the eigenvalues are more separated. Using

[1] http://www.maths.manchester.ac.uk/our-research/research-groups/numerical-analysis-and-scientific-computing/numerical-analysis/software/nlevp/

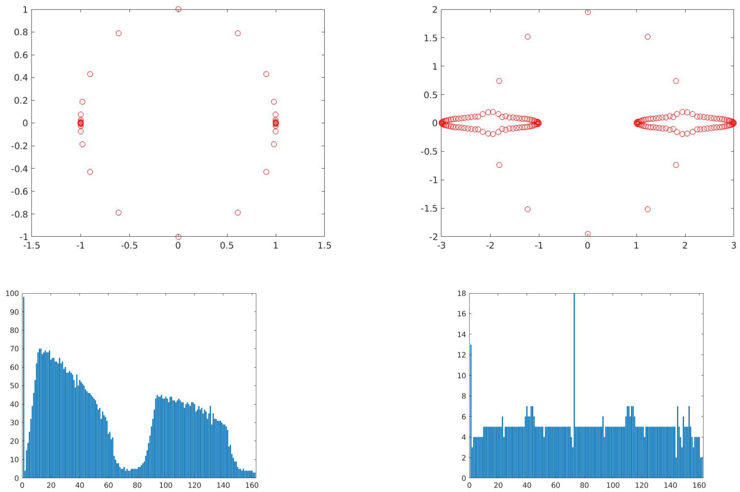


TABLE 3
left: Sign1 – right: Sign2

MATLAB's time measurement `tic,toc` we needed for `sign1` 63.92 seconds and for `sign2` 10.82 seconds on my laptop.

10. Non-polynomial Eigenvalue Problem. `TimeDelay` is a non-polynomial non-linear eigenvalue problem from the NLEVP collection with a 3×3 matrix $A(\lambda)$:

$$A(\lambda) = -\lambda I + A_0 + A_1 e^{-\lambda}$$

Tisseur et al. write for this problem:

> "*. . . characteristic equation of a time-delay system with a single delay and constant coefficients. The problem has a* double non-semisimple eigenvalue $\lambda = 3\pi i$"

The nonlinear equation $\det A(\lambda) = 0$ has infinitely many solutions. Using Ostrowski's iteration we can e.g. compute the first 20 solutions and get the plot in Table 4. On the imaginary axis we get the double eigenvalues (λ_2 and λ_3 in Table 4) mentioned above and also a single eigenvalue $\lambda_4 = 4.5\pi i$. The eigenvalue $\lambda_1 = 0.705244109106679 + 2.741466762205487i$ has a positive real part, all the others have negative real parts. The double eigenvalue is computed to the precision one can expect with IEEE floating point arithmetic.

11. Gaussian Elimination for Banded Matrices. Many problems in the NLEVP collection are banded (e.g. `beamsensitivity` with 7 or `pdde-stabiity` with 32 diagonals). It makes sense to develop an algorithm to compute determinants for banded matrices. A MATLAB-function for Gaussian elimination for banded matrices with partial pivoting is given in [4]. If A has q lower and p upper diagonals we store the diagonals as columns in the matrix B. For partial pivoting we add q zero columns to B, see Figure 3.

By adapting the function `det2p` to this banded structure we obtain the function `det2pband`.

```
function [ffp,dffp]=det2pband(p,q,B,Bp,Bpp);
% DET2PBAND computes Newton-correction and derivative for a banded matrix
```

$$
\begin{aligned}
\lambda_1 &= 0.705244109106679 + 2.741466762205487i \\
\lambda_2 &= 0.000000005149180 + 9.424777943433675i \\
\lambda_3 &= -0.000000007679198 + 9.424777969836999i \\
\lambda_4 &= -0.000000000000001 + 14.137166941154069i \\
\lambda_5 &= -0.422996397305027 + 20.485362607960255i \\
\lambda_6 &= -0.693701244038287 + 26.758000106609209i
\end{aligned}
$$

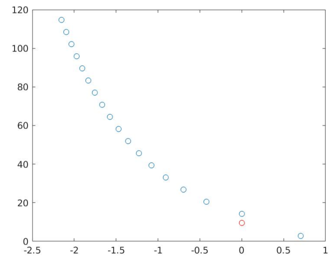


TABLE 4
Non-Polynomial EVP: Time Delay Example

$$
A = \begin{bmatrix}
x & x & 0 & 0 & 0 & 0 & 0 & 0 \\
x & x & x & 0 & 0 & 0 & 0 & 0 \\
x & x & x & x & 0 & 0 & 0 & 0 \\
0 & x & x & x & x & 0 & 0 & 0 \\
0 & 0 & x & x & x & x & 0 & 0 \\
0 & 0 & 0 & x & x & x & x & 0 \\
0 & 0 & 0 & 0 & x & x & x & x \\
0 & 0 & 0 & 0 & 0 & x & x & x
\end{bmatrix}
\qquad
B = \left[\begin{array}{cccc|cc}
0 & 0 & x & x & 0 & 0 \\
0 & x & x & x & 0 & 0 \\
x & x & x & x & 0 & 0 \\
x & x & x & x & 0 & 0 \\
x & x & x & x & 0 & 0 \\
x & x & x & x & 0 & 0 \\
x & x & x & x & 0 & 0 \\
x & x & x & 0 & 0 & 0
\end{array}\right]
$$

FIG. 3. *storing banded matrices*

```
n=length(B); logfpp=0; logfp=0;
Bpp=[Bpp,zeros(n,q)];
Bp=[Bp,zeros(n,q)]; B=[B,zeros(n,q)];   % augment B with q columns
normb=norm(B,1);
for j=1:n
  maximum=0; kmax=j;                     % search pivot
  for k=j:min(j+q,n)
    if abs(B(k,j-k+q+1))>maximum,
      kmax=k; maximum=abs(B(k,j-k+q+1));
    end
  end
  if maximum<1e-14*normb;               % only small pivots
    ffp=0; dffp=0; return               % consider det=0
  end
  if j~=kmax                            % interchange rows
    ind1=j-kmax+q+1:min(n,j+2*q+p-kmax+1);
    ind2=q+1:min(n,2*q+p+1);
    h=Bpp(kmax,ind1); Bpp(kmax,ind1)=Bpp(j,ind2); Bpp(j,ind2)=h;
    h=Bp(kmax,ind1); Bp(kmax,ind1)=Bp(j,ind2); Bp(j,ind2)=h;
    h=B(kmax,ind1); B(kmax,ind1)=B(j,ind2); B(j,ind2)=h;
  end

  logfpp=logfpp+(B(j,q+1)*Bpp(j,q+1)-Bp(j,q+1)^2)/B(j,q+1)^2;
  logfp=logfp+Bp(j,q+1)/B(j,q+1);
  for k=j+1:min(n,j+q)                  % elimination step
    ind3=j-k+q+1;
```

```
    Bpp(k,ind3)=(Bpp(k,ind3)*B(j,q+1)-Bp(j,q+1)*Bp(k,ind3))/B(j,q+1)^2 ...
               -(Bp(k,ind3)*Bp(j,q+1)+B(k,ind3)*Bpp(j,q+1))/B(j,q+1)^2 ...
                +2*Bp(j,q+1)^2*B(k,ind3)/B(j,q+1)^3;
    Bp(k,ind3)=(B(j,q+1)*Bp(k,ind3)-B(k,ind3)*Bp(j,q+1))/B(j,q+1)^2;;
    B(k,ind3)=B(k,ind3)/B(j,q+1);
  end
  for k=j+1:min(n,j+q)
    for l=j+1:min(n,j+p+q)
      ind4=l-k+q+1; ind5=j-k+q+1; ind6=l-j+q+1;
      Bpp(k,ind4)=Bpp(k,ind4)-Bpp(k,ind5)*B(j,ind6)-Bp(k,ind5)*Bp(j,ind6)...
                 -Bp(k,ind5)*Bp(j,ind6)-B(k,ind5)*Bpp(j,ind6);
      Bp(k,ind4)=Bp(k,ind4)-Bp(k,ind5)*B(j,ind6)-B(k,ind5)*Bp(j,ind6);
      B(k,ind4)=B(k,ind4)-B(k,ind5)*B(j,ind6);
    end
  end
end
dffp=-logfpp/logfp^2; ffp=1/logfp;
```

The `Beamsensitivity` example is a quadratic eigenvalue problem with a banded matrix with 7 diagonals. For $n = 200$, that is for 400 eigenvalues, using Laguerre, measured with MATLAB's `tic,toc`, we need for the full-matrix algorithm: *83.85 seconds*. Using the banded algorithm the computation time drops to *10.39 seconds*.

12. Damped Beam Example. In [6] Higham et al. write: *The standard approach to the numerical solution of the QEP is to convert the quadratic $Q(\lambda) = \lambda^2 M + \lambda D + K$ into a linear polynomial $L(\lambda) = \lambda X + Y$ of twice the dimension of Q but with the same spectrum. The resulting generalized eigenproblem $L(\lambda)z = 0$ is usually solved by the QZ algorithm for small- to medium-size problems or by a Krylov method for large sparse problems.*

A common choice of L in practice is the first companion form, given by

$$C_1(\lambda) = \lambda \begin{pmatrix} M & 0 \\ 0 & I \end{pmatrix} + \begin{pmatrix} D & K \\ -I & 0 \end{pmatrix}$$

When K and M, respectively, are nonsingular the two pencils

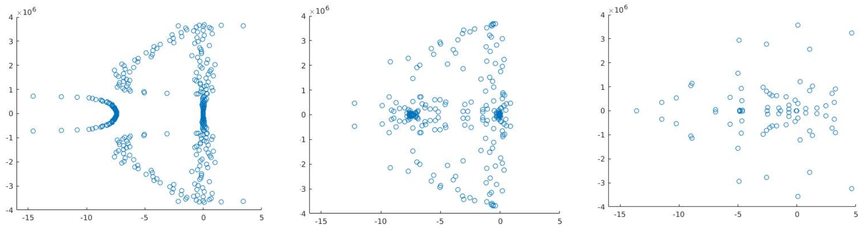
$$L_1(\lambda) = \lambda \begin{pmatrix} M & 0 \\ 0 & -K \end{pmatrix} + \begin{pmatrix} D & K \\ K & 0 \end{pmatrix}, \quad L_2(\lambda) = \lambda \begin{pmatrix} 0 & M \\ M & D \end{pmatrix} + \begin{pmatrix} -M & 0 \\ 0 & K \end{pmatrix}$$

are other possible linearizations.

Using these linearizations Higham et al. compute the eigenvalues of the QEP using MATLAB's function `eig` for the generalized eigenvalue problem. The results are rather disappointing, see Figure 4.

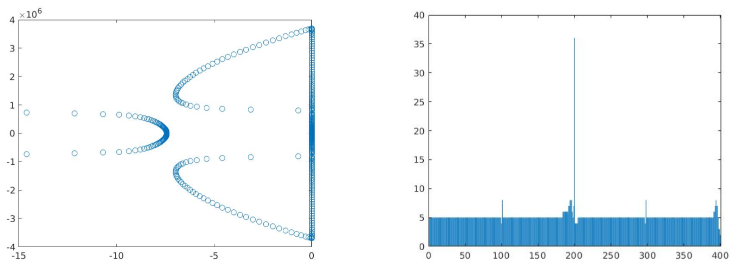
Fan, Lin and Van Dooren showed in [3] that by applying appropriate scalings the ill-conditioning of the linearized eigenvalue problems can be cured. Higham et al. show and explain in [6] why without the scaling the transformed systems are so ill-conditioned. The situation reminds me of an old problem of Jim Wilkinson. Transforming an eigenvalue problem by computing first the coefficients of the characteristic polynomial, then computing the zeros of the polynomial is also not a recommended way because the transformation may change the condition of the eigenvalues dramatically.

However, if we solve $\det A(\lambda) = 0$ directly with one of our methods using e.g. Laguerre's iteration then we get correct results without the necessity to scale (see Figure 5).



C_1 L_1 L_2

FIG. 4. *Linearization without scaling*



eigenvalues iterations

FIG. 5. *Damped Beam solved with Laguerre*

13. Conclusions. We have shown how to implement third order iteration methods for solving $f(\lambda) = \det A(\lambda) = 0$ using algorithmic differentiation. This technique produces the exact derivatives, since for computing determinants by Gaussian Elimination we only use the four basic arithmetic operations.

Since we work with the original problem the condition is not changed by transformations of the problem.

One could obtain cubic convergence by a multi-step iteration which avoids the second derivative. However, only by using one point iterations, we obtain the algorithm `det2p` which in an elegant way computes the Newton correction and t containing the necessary derivatives $t = ff''/f'^2$.

Computing the second derivative is expensive. For a full $n \times n$ matrix we need $\sim n^3$ operations per iteration. Since our computers have powerful processors, we can solve anyway medium size NEP. The situation is much more favorable for banded matrices for which one iteration needs only $\sim n$ operations.

Acknowledgments. The author would like to thank Zhong-Zhi Bai and Yu-Mei Huang, the two organizers of the 2019 Golub Memorial Workshop in Lanzhou. The invitation to participate at this conference was a special motivation for me to develop the algorithms discussed in this paper.

REFERENCES

[1] Gander Walter Arbenz Peter. Solving nonlinear eigenvalue problems by algorithmic differentiation. *Computing*, 36:205–215, 1986.

[2] Tisseur F. and Meerbergen K. The quadratic eigenvalue problem. *SIAM Review*, 43:234–286, 2001.

[3] Van Dooren P. Fan H-Y, Lin W-W. Normwise scaling of second order polynomial matrices. *SIAM Journal on Matrix Analysis and Application*, 26:252–256, 2004.

[4] Kwok Felix Gander Walter, Gander Martin J. *Scientific Computing, an Introduction Using* MAPLE *and* MATLAB. Springer, 2014.

[5] E. Hansen and M Patrick. A family of root finding methods. *Numer. Math.*, 27:257–269, 1977.

[6] Tisseur Françoise Garvey Seamus D. Higham Nicholas J., Mackey D. Steven. Scaling, sensitivity and stability in the numerical solution of quadratic eigenvalue problems. *Int. J. Numer. Meth. Engng*, 73:344–360, 2008.

[7] Maehly H. J. Zur iterativen Auflösung algebraischer Gleichungen. *ZAMP (Zeitschrift für angewandte Mathematik und Physik)*, pages 260–263, 1954.

[8] V. Mehrmann C. Schröder T. Betcke, N. J. Higham and F. Tisseur. Nlevp: A collection of nonlinear eigenvalue problems, 2011.

[9] V. Mehrmann C. Schröder T. Betcke, N. J. Higham and F. Tisseur. Nlevp: A collection of nonlinear eigenvalue problems, users guide, 2011.

[10] Gander Walter. On halley's iteration method. *The American Mathematical Monthly*, 92(2), February 1985.

[11] Gander Walter. Zeros of determinants of λ-matrices. In Vadim Olshevsky and Eugene Tyrtyshnikov, editors, *Matrix Methods: Theory, Algorithms and Applications, Dedicated to the Memory of Gene Golub*, pages 238–246. World Scientific Publishers, 2010.