Polar Decomposition

Thomas Seleiro[*]

December 15, 2020

2. Prove that the singular values of A are the eigenvalues of H.

We know that any matrix $A \in \mathbb{C}^{m\times n}$, $m \geq n$ has a thin singular value decomposition $A = P\Sigma Q^*$ where $P \in \mathbb{C}^{m\times n}$ has orthogonal columns, $Q \in \mathbb{C}^{n\times n}$ is unitary, and $\Sigma \in \mathbb{C}^{n\times n}$ is diagonal with $\Sigma = \operatorname{diag}(\sigma_1, \dots, \sigma_r)$ where $\operatorname{rank}(A) = r$. The coefficients $\sigma_1 \geq \cdots \geq \sigma_r \geq 0$ are the singular values of A.

We can write

$$A = (PQ^*)(Q\Sigma Q^*) =: UH, \tag{1}$$

where U and H satisfy the properties of a polar decomposition. In particular we have $H = Q\Sigma Q^*$ where Σ is diagonal with non-negative diagonal entries and Q is unitary. Thus the diagonal values of Σ are the eigenvalues of H.

3. Prove that A is normal ($A^*A = AA^*$) iff U and H commute.

We first suppose that U and H commute. Note that for the product HU to be well defined, we must have $m = n$, and hence $U \in \mathbb{C}^n \times n$ is unitary. Since $A = UH = HU$ we get

$$\begin{aligned} A^*A &= (UH)^*(UH) = H^*(U^*U)H = H^2 \\ AA^* &= (HU)(HU)^* = H(UU^*)H^* = H^2 \end{aligned} \tag{2}$$

so A is normal.

Now suppose A is normal. Since $A^*A \in \mathbb{C}^{n\times n}$ and $AA^* \in \mathbb{C}^{m\times m}$, normality of A requires $m = n$. Using the singular value decomposition of A, we have

$$AA^* = (P\Sigma Q^*)(Q\Sigma P^*) = P\Sigma^2 P^* \tag{3}$$

where $\Sigma^2 = \operatorname{diag}(\sigma_1^2, \dots, \sigma_r^2)$. Equating (2) and (3), we get $H^2 = P\Sigma^2 P^*$. From [1, p. 405], we know that there is a unique Hermitian positive semi-definite matrix $(AA^*)^{1/2}$ such that

$$(AA^*)^{1/2}(AA^*)^{1/2} = AA^* = H^2.$$

[*] Department of Mathematics, University of Manchester, Manchester, M13 9PL, UK (thomas.seleiro@postgrad.manchester.ac.uk)

It is clear that H is the unique Hermitian positive semi-definite matrix square root of A^*A. We also have that

$$(P\Sigma P^*)(P\Sigma P^*) = P\Sigma^2 P^* = AA^*.$$

Therefore $H = P\Sigma P^*$ and

$$HU = (P\Sigma P^*)(PQ^*) = P\Sigma Q^* = A = UH,$$

using $P^*P = I_n$. Therefore U and H commute.

4. Verify the formula

$$U = \frac{2}{\pi} A \int_0^\infty (t^2 I + A^*A)^{-1} dt \tag{*}$$

for full rank A by using the singular value decomposition (SVD) of A to diagonalize the formula.

Suppose A is full rank. Then the singular values of A are strictly positive. Since $A^*A = (Q\Sigma P^*)(P\Sigma Q^*) = Q\Sigma^2 Q^*$, we have

$$t^2 I + A^*A = Q(t^2 I)Q^* + Q\Sigma^2 Q^* = QDQ^*, \tag{4}$$

where $D := \operatorname{diag}(t^2 + \sigma_i)$ is an $n \times n$ matrix. Inverting (4) gives

$$(t^2 I + A^*A)^{-1} = QD^{-1}Q^*, \qquad D^{-1} = \operatorname{diag}\left(\frac{1}{\sigma_i^2 + t^2}\right).$$

Since Q and Q^* do not depend on t, they can be taken outside the integral, leaving the right hand side of (*) in the form

$$\frac{2}{\pi} A\, Q \left(\int_0^\infty D^{-1} dt\right) Q^*.$$

The integral is a diagonal matrix where the ith diagonal component is

$$\int_0^\infty \frac{1}{\sigma_i^2 + t^2}\, dt = \left[\frac{1}{\sigma_i} \arctan\left(\frac{t}{\sigma_i}\right)\right]_0^\infty = \frac{\pi}{2\sigma_i},$$

using [2, 4.2.4.4]. So the right-hand side of (*) is

$$\frac{2}{\pi} AQ\left(\frac{\pi}{2}\Sigma^{-1}\right) Q^* = P\Sigma Q^*\, Q\Sigma^{-1} Q^* = PQ^* = U.$$

5. Derive Newton's method for computing U by considering equations $(X + E)^*(X + E) = I$, where E is a "small perturbation". (Newton's method is $X_{k+1} = (X_k + X_k^{-*})/2, X_0 = A$)

We know that U is the closest unitary matrix to A, and since $U^*U = I$, we want to find a solution to the equation

$$F(X) = 0, \qquad F(X) \coloneqq X^*X - I$$

using a Newton method starting at A. The general form of the Newton method [3, p.2] is

$$F(X_{k+1}) + DF_{X_k}\left[X_{k+1} - X_k\right] = 0, \tag{5}$$

where DF_{X_k} is the Fréchet derivative and, is the first order E term in

$$F(X + E) - F(X) = X^*E + E^*X + E^*E.$$

So $DF_{X_k}[E] = X^*E + E^*X$. Substituting in (5),

$$X_k^*X_k - I + X_k^*\left(X_{k+1} - X_k\right) + \left(X_{k+1}^* - X_k^*\right)X_k = 0.$$

Eliminating $X_k^*X_k$ terms,

$$X_k^*X_{k+1} + X_{k+1}^*X_k = X_k^*X_k + I.$$

We know that for any matrix we can write $B = 1/2(B + B^*) + 1/2(B - B^*)$, where the terms on the right-hand side are the Hermitian and skew Hermitian components respectively [1, p.170]. Setting the skew Hermitian part to zero, and taking $B = X_k^*X_{k+1}$ gives

$$X_k^*X_{k+1} = \frac{1}{2}\left(X_k^*X_k + I\right).$$

Assuming that X_k (and by extension A) is non-singular, left-multiplication by $X_k^{-*} = (X_k^{-1})^*$ yields the recursive iteration equation for the desired Newton method:

$$X_{k+1} = \frac{1}{2}\left(X_k + X_k^{-*}\right). \tag{6}$$

6. Prove that Newton's method converges, and at a quadratic rate, by using the SVD of A.

For the Newton iteration to be well defined, we require that A and the iterates X_k be invertible.

We have the SVD of A given by $A = P\Sigma Q^*$, and $U = PQ^*$. The iterates X_k also have a singular value decomposition, which we write $X_k = P_k\Sigma_kQ_k^*$. Using this in eq. (6) gives

$$\begin{aligned} X_{k+1} = (X_k + X_k^{-*})/2 &= \frac{1}{2}(P_k\Sigma_kQ_k^* + P_k\Sigma_k^{-1}Q_k^*) \\ &= P_k\left[\frac{1}{2}(\Sigma_k + \Sigma_k^{-1})\right]Q_k^* \end{aligned}$$

So we can identify the factors in the SVD of X_{k+1} with those of X_k (up to reordering of rows) and write

$$X_k = P\Sigma_k Q^*,$$

where Σ_k is defined by the recurrence relation

$$\Sigma_{k+1} = \frac{1}{2}\left(\Sigma_k + \Sigma_k^{-1}\right).$$

We now have

$$\begin{aligned} U - X_{k+1} &= PQ^* - P\left[\frac{1}{2}\left(\Sigma_k + \Sigma_k^{-1}\right)\right]Q^* \\ &= \frac{1}{2}P\left[(I - \Sigma_k) + \left(I - \Sigma_k^{-1}\right)\right]Q^*. \end{aligned}$$

Since

$$-\Sigma_k^{-1}(I - \Sigma_k)^2 = 2I - \Sigma_k - \Sigma_k^{-1},$$

we are left with

$$U - X_{k+1} = -\frac{1}{2}P\Sigma_k^{-1}\left(I - \Sigma_k\right)^2 Q^*.$$

Taking the 2-norm on both sides and exploiting the fact that P and Q are orthogonal,

$$\begin{aligned} \|U - X_{k+1}\|_2 &\leq \frac{1}{2}\left\|P\Sigma_k^{-1}\right\|_2 \left\|(I - \Sigma_k)^2 Q^*\right\|_2 \\ &= \frac{1}{2}\left\|\Sigma_k^{-1}\right\|_2 \left\|(I - \Sigma_k)^2\right\|_2 \\ &\leq \frac{1}{2}\|X_k\|_2\|I - \Sigma_k\|_2^2 \\ &= \frac{1}{2}\|X_k\|_2\|U - X_k\|_2^2 \end{aligned}$$

To achieve convergence, we need to bound $\|X_k\|$ by a constant. We do so by observing that

$$\|X_{k+1}\|_2 = \max_{i=1:n}\frac{\sigma_i + \sigma_i^{-1}}{2} \leq \max\left\{\|X_k\|_2, \left\|X_k^{-1}\right\|_2\right\}.$$

So for all k, $\|X_k\|_2 \leq M \coloneqq \max\{\|A\|_2, \|A^{-1}\|_2\}$. Thus we conclude that the Newton method converges quadratically.

7. Use the SVD to analyse the convergence of the Newton-Schulz iteration for computing U:

$$X_{k+1} = \frac{1}{2}X_k(3I - X_k^* X_k), \qquad X_0 = A$$

We assume $A \in \mathbb{C}^{m\times n}$ and $m \geq n$. We replace A in the expression of X_1 with the thin SVD $A = P\Sigma Q^*$ and get

$$X_1 = \frac{1}{2}P\Sigma Q^*\left(3I - (Q\Sigma P^*)(P\Sigma Q^*)\right) = \frac{1}{2}P\Sigma(Q^*Q)(3I-\Sigma^2)Q^*$$
$$= P\left[\frac{1}{2}(3\Sigma - \Sigma^3)\right]Q^*.$$

Thus we can write $X_1 = P\Sigma_1 Q^*$ where $\Sigma_1 = (3\Sigma - \Sigma^3)/2$. Applying the same method recursively we can write $X_k = P\Sigma_k Q^*$ with

$$\Sigma_k = \mathrm{diag}(\sigma_i^{(k)}), \qquad \Sigma_{k+1} = \mathrm{diag}(3\sigma_i^{(k)} - (\sigma_i^{(k)})^3).$$

In order for the method to converge, we require every diagonal element of $(\Sigma_k)_{k\in\mathbb{N}}$ to converge. Since we want X_k to converge to $U = PQ^*$, we want each diagonal element to converge to 1.

We consider the real sequence $(x_k)_{k\in\mathbb{N}}$ defined by the recurrence relation

$$x_{k+1} = p(x_k), \qquad x_0 \geq 0, \qquad p(x) := \frac{1}{2}(3x - x^3),$$

where the condition $x_0 \geq 0$ is motivated by fact that the singular values of A are positive.

We first note that p is an odd function with roots at 0, $\pm\sqrt{3}$ and local maxima at ± 1 ($p(\pm 1) = \pm 1$). Since the leading coefficient of p is negative, p is positive on $[0, \sqrt{3}]$ and negative on $[\sqrt{3}, \infty)$.

We can study the convergence of (x_k) for different values of x_0.

- For $x_0 = 0$ or $\sqrt{3}$, x_0 is a root of p so $x_k = 0$ for $k \geq 1$.
- For $0 < x_0 \leq 1$, $p(0,1) = (0,1)$ and $p(x) > x$ on $(0,1)$ so $x_k \to 1$ as $k \to \infty$.
- For $1 < x_0 < \sqrt{3}$, $p(x_0) \in (0,1)$ so $x_n = p^n(x_0) = p^{n-1}(p(x_0)) \to 1$ as $k \to \infty$.
- For $x_0 > \sqrt{3}$, $p(x_0)$ is negative so we cannot guarantee convergence to 1. It is easy to show that for $x_0 \geq \sqrt{5}$ the iteration diverges, since $|p(x)| \geq x$ and p is unbounded for $|x| \geq \sqrt{5}$.

Thus, every diagonal sequence converges to 1 if $0 < \sigma_i < \sqrt{3}$ for $i = 1 : \mathrm{rank}(A)$, or equivalently $\|A\|_2 < \sqrt{3}$ and A is full rank.

It follows that for a starting matrix A with full rank and $\|A\|_2 < \sqrt{3}$,

$$\|U - X_{k+1}\|_2 = \left\|PQ^* - P\left[\frac{1}{2}(3\Sigma_k - \Sigma_k^3)\right]Q^*\right\|_2 = \left\|I - \frac{1}{2}(3\Sigma_k - \Sigma_k^3)\right\|_2$$
$$= \max_{i=1:n}\left|1 - \frac{1}{2}\left(3\sigma_i^{(k)} - (\sigma_i^{(k)})^3\right)\right|,$$

which tends to 0 as $k \to \infty$.

8. Evaluate the operation count for one step of Newton's method and one step of the Newton-Schulz iteration (taking account of symmetry). Ignoring operation counts, how much faster does matrix multiplication have to be than matrix inversion for Newton-Schulz to be faster than Newton (assuming both take the same number of iterations)?

We first consider the kth step $X_{k+1} = (X_k + X_k^{-*})/2$ for the Newton iteration of a matrix $A \in \mathbb{C}^{n\times n}$. Counting the number of complex flops, we identify 3 operations:

- One matrix inversion of $X_k \in \mathbb{C}^{n\times n}$: $2n^3 + O(n^2)$ flops
- One matrix addition in $\mathbb{C}^{n\times n}$: n^2 flops
- One element-wise division in $\mathbb{C}^{n\times n}$: n^2 flops

So the total number of operations for the Newton method is $2n^3 + O(n^2)$.

We now consider a step of the Newton-Schulz iteration

$$\frac{1}{2}X_k(3I - X_k^* X_K), \qquad X_0 = A \in \mathbb{C}^{m\times n}.$$

We first note that $X_k^* X_k$, and by extension $(3I - X_K^* X_k)/2$, is Hermitian in $\mathbb{C}^{n\times n}$. Therefore only the upper triangular elements of these matrices need to be calculated, ie we only have to compute $\sum_{k=1}^{n} k = (n^2+n)/2$ components. To calculate $X_k^* X_k$, we use Algorithm 1, using a total of $mn^2 + mn$ flops.

Algorithm 1: Algorithm to compute the top diagonal elements of $X_k^* X_k$

```
$b_{ij} = (X_k^* X_k)_{ij}$ ;
for $i = 1 : n$ do
    for $j = i : n$ do
        for $r = 1 : m$ do
            $b_{ij} = b_{ij} + \overline{x_{ri}} x_{rj}$ ;
        end
    end
end
```

Forming $3I - X_k^* X_k$ and dividing the result by 2 adds $n^2 + n$ operations. Finally multiplying by X_k takes $2mn^2$ flops for a total of $3mn^2 + O(n^2) + O(mn)$ flops per step.

We compare both methods, assuming both require the same number of iterations to converge. One step of the Newton iteration requires one matrix inversion, whereas one step of the Newton-Schulz iteration requires two matrix multiplications. Therefore, in order for the Newton-Schulz iterations to be quicker to compute, we need matrix multiplication to be twice as fast as matrix inversion. Note that these considerations do not account for symmetry in the Newton-Schulz iteration, which we consider to be implementation dependent.

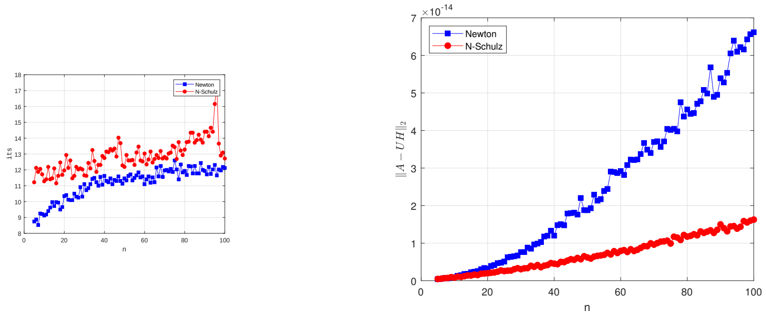
9. Write a MATLAB M-file `poldec` that computes the polar decomposition of a nonsingular $A \in \mathbb{C}^{n \times n}$.

We wrote the following MATLAB function `poldec`[1] that computes the polar decomposition of a matrix A.

```
function [U, H, its] = poldec(A)
%POLDEC Polar Decomposition
%   [U, H, ITS] = poldec(A) computes the polar decomposition
%   A = U*H of the square, nonsingular matrix A. ITS is the
%   number ofiterations for convergence.
    n = size(A,1);
    X    = A;
    Xnew = zeros(n);
    its  = 0;
    newtSchulz = false;
    converged = false;
    fprintf("k   \t|X_k-X_{k-1}|/|X_{k}|\t|I - X_k*X_k|\n");
    fprintf("====\t===================\t==============\n");
    while(not(converged) && its < 100)
        if(not(newtSchulz))
            %We use the Newton method until either the
            %   convergence conditionfor the Newton-Schulz
            %   iterations is fulfilled, or convergence is
            %   acheived.
            Xnew = (X + inv(X)')/2;
        else
            %We use the Newton-Schulz method, having guaranteed
            %   it will converge from this point onwards.
            Xnew = X/2 * (3*eye(n) - X' * X);
        end

        iterDist = norm(Xnew - X, inf)/norm(Xnew, inf);
        unitDist = norm(eye(n) - Xnew' * Xnew, inf);

        newtSchulz = norm(Xnew, 2) < sqrt(3);
        converged = (unitDist <= 1e-16*n) || (iterDist <= 1e-16*n);

        X = Xnew;
        its = its + 1;
        fprintf("%4d\t%19.8e\t%13.8e\n", its, iterDist, unitDist);
    end
    U = Xnew;
    H = U' * A;
end
```

The two variables `newtSchulz` and `converged` control the function's operation. The function starts by computing U using the Newton's method, until

[1] All functions discussed can be found in the GitHub repository *https://github.com/ThomasSeleiro/PolarDecompProj*



(b)

(a)

Figure 1: Figure showing (a) the number of iterations, (b) the accuracy of the polar decomposition of `rand(n)` computed using the Newton method only, and the `poldec` function. Note in (a) that the Newton only method quickly reaches the maximum allowed number of iterations at $n = 11$. These results were obtained by running `randnTest.m` .

the condition $\|X_k\|_2 < \sqrt{3}$ stored in `newtSchulz`, turns true. From this point onwards, the function begins using the Newton-Schulz iteration to calculate U.

The function stops iterating once the variable

```
converged = (unitDist <= 1e-16 * n) || (iterDist <= 1e-16 * n);
```

becomes true. Here `unitDist` stores the value $\|I - X_k^* X_k\|_\infty$ and `iterDist` stores $\|X_k - X_{k-1}\|_\infty / \|X_k\|_\infty$. We motivate this choice for convergence by noting that if either distance is of the order of the unit roundoff $u = 10^{-16}$, the iterates stop gaining precision in double floating point arithmetic. For example, if `unitDist` $< u$, the matrix X_k is unitary to machine precision. Similarly, if `iterDist` $< u$, the iterates X_k and X_{k-1} are sufficiently close in double precision floating point arithmetic, and any subsequent iterates will be close to X_k. Note that a maximum number of 100 iterates per computation was set to avoid long computation times if the algorithm doesn't manage to reduce `unitDist` or `iterDist` enough.

Once U is calculated, we simply form $H = U^*A$ to output the full polar decomposition.

We first tested the implementation of `poldec` by using random matrices of size n using the MATLAB function `rand(n)`. The results of these experiments are summarised in Fig. 1. We note that `poldec` converges significantly quicker

		`its`		$\|A - UH\|_2$	
A	$\kappa_2(A)$	Newton only	`poldec`	Newton only	`poldec`
`eye(8)`	1	1	1	0	0
`hilb(6)`	1.5e07	28	30	0	1.9230e-16
`magic(6)`	4.7e16	58	59	2.7954e-14	2.3013e-14
`hadamard(8)`	1	7	10	2.4369e-15	6.8717e-16

Table 1: Results of experiments applying `poldec` to various matrices. A variant of the function that only uses the Newton method was also used for comparison. Results obtained by running `otherTest.m`.

and produces a more accurate result. Quicker convergence for the Newton method could be achieved by using more lenient convergence criteria, but this would likely lead to decreased accuracy of the computed decomposition.

Table 1 shows metrics related to computing the polar decomposition of a few specific matrices using using only the Newton method, and using `poldec`.

We see that only one iteration is needed for the identity matrix `eye(8)` since it is already unitary.

Using `hilb(6)` as input illustrates the case when A is Hermitian positive definite. It is clear in this case that $U = I$ since H is a Hermitian positive definite matrix. However, both algorithms take a long time to compute U since they must calculate U iteratively. Since $U = I$ however, the result is accurate to machine precision.

The function takes a relatively large number of iterations to find the polar decomposition of `magic(6)`. We note that for any matrix A,

$$M := \max\left\{\|A\|_2, \|A^{-1}\|_2\right\} \geq \kappa_2(A)^{1/2}.$$

Since for the Newton method

$$\|U - X_{k+1}\|_2 \leq \frac{M}{2}\|U - X_k\|_2^2,$$

convergence will be slow for large values of M. This explains the larger number of iterations for both `hilb(6)` and `magic(6)`.

For the computation of the polar decomposition of `hadamard(8)`, we achieve good accuracy and a small number of iterations using the `poldec` function. For the Newton method only, while the iterations did not achieve the convergence criterion, both `iterDist` and `unitDist` rapidly decrease past 10^{-15} but keep fluctuating for a long time before `iterDist` eventually drops below 10^{-16}. The output of the Newton only script is shown below.

```
k     |X_k-X_{k-1}|/|X_k|   |I - X_k^*X_k|
===      ==============     ==============
                    [...]
5        1.46536674e-05     2.14730583e-10
```

		Runtime (in μs)	
A	`its`	Newton only	`poldec`
`rand(8)`	11	547.7	750.2
`rand(20)`	12	941.6	977.9
`hilb(6)`	28	1442	1325
`magic(6)`	59	6389	6510
`hadamard(8)`	10	718.9	564.3

Table 2: Runtimes when calculating the Polar Decomposition using Newton's method and the function `poldec`. Results obtained by running `speedTest.m`.

```
6         1.07365042e-10      9.33334681e-16
7         4.31775426e-16      1.54939455e-15
                    [...]
121       2.35513869e-16      6.09764714e-16
122       2.15887713e-16      6.08974934e-16
123       1.17756934e-16      5.84808662e-16
124       7.85046229e-17      5.59620674e-16
```

We also compared the runtime of `poldec` and a simple Newton's method over the same number of iterates (taken from the number of iterates required for the Newton only iteration to converge). Table 2 shows the times taken to compute the polar decompositions of some of the matrices. Overall the computation time remains lower in most cases for the `poldec` function. This suggests that the matrix multiplication implementation is much faster than the implementation of matrix inversion, as mentioned in Question 8.

10. Write another routine that computes the square root of a Hermitian positive definite matrix by doing a Cholesky decomposition and calling `poldec`.

For any Hermitian positive definite matrix $A \in \mathbb{C}^{n\times n}$, we can compute the unique Cholesky factorization $A = R^*R$, where $R \in \mathbb{C}^{n\times n}$ is an upper-triangular matrix with strictly positive diagonal values. Since the Cholesky factor is full rank, we can compute its polar decomposition $R = UH$ with U unitary and H Hermitian positive definite. Using this decomposition, we get

$$A = R^*R = (UH)^*UH = H^2.$$

Hence H is positive definite matrix square root of A.

We implemented the discussed method for calculating square roots in the function `poldecsqrt`.

```
function H = poldecsqrt(A)
%POLDECSQRT - Square-root using Cholesky & polar decomposition
%
```

```
%   H = poldecsqrt(A) computes the square root of the input
%   matrix (must be a Hermitian positive definite) using the
%   polar decomposition of the Cholesky factor (A = R*R,
%   R = UH,  A = H^2)

    % Compute the Cholesky factor of A
    [R, flag] = chol(A);
    if flag ~= 0
        error("The input is not Hermitian positive definite");
    end

    % Compute the Polar Decomposition of R, return H
    [U, H, its] = poldec(R);

end
```

We used the MATLAB function `chol` to calculate the Cholesky factor `R` of the input matrix `A`. `poldec` is then used to find the Hermitian factor `H` of `R`. Note the function raises an error if the input matrix is not Hermitian positive definite.

References

[1] Roger A Horn and Charles R Johnson. *Matrix analysis.* Cambridge University Press, Cambridge, 1985.

[2] Alan. Jeffrey and Hui-Hui Dai. *Handbook of mathematical formulas and integrals.* Academic Press/Elsevier, Amsterdam, 4th edition, 2008.

[3] C T Kelley. *Solving nonlinear equations with Newton's method.* Society for Industrial and Applied Mathematics, Philadelphia, 2003.