Computational Optimization and Applications, 29, 263–287, 2004

Computing Eigenelements of Real Symmetric Matrices via Optimization

M. MONGEAU mongeau@mip.ups-tlse.fr
M. TORKI* torki@iup.univ-avignon.fr
Labo MIP, Université Paul Sabatier, 31062 Toulouse Cedex 04, France

Received September 10, 2002; Revised September 12, 2003

Abstract. In certain circumstances, it is advantageous to use an optimization approach in order to solve the generalized eigenproblem, $Ax = \lambda Bx$, where A and B are real symmetric matrices and B is positive definite. In particular, this is the case when the matrices A and B are very large *and* the computational cost, prohibitive, of solving, with high accuracy, systems of equations involving these matrices. Usually, the optimization approach involves optimizing the Rayleigh quotient.

We first propose alternative objective functions to solve the (generalized) eigenproblem via (unconstrained) optimization, and we describe the variational properties of these functions.

We then introduce some optimization algorithms (based on one of these formulations) designed to compute the largest eigenpair. According to preliminary numerical experiments, this work could lead the way to practical methods for computing the largest eigenpair of a (very) large symmetric matrix (pair).

Keywords: generalized eigenproblem, variational principles, Newton-type algorithms, Lanczos method, limited-memory algorithms

AMS Subject Classifications: 65F15, 49M37, 65K10, 49M10

1. Introduction

In certain circumstances, it is advantageous to use an optimization approach in order to solve the generalized eigenproblem, $Ax = \lambda Bx$, where A and B are real symmetric matrices and B is positive definite. In particular, this is the case when the matrices A and B are very large *and* the computational cost, prohibitive, of solving, with high accuracy, systems of equations involving these matrices. We shall assume that the problem is to compute one extreme eigenvalue (the largest or smallest) and an associated eigenvector. The reader is referred to [12] and [16] for the practical motivation of such assumptions and the reasons why problems of this type may prevent the direct use of standard techniques such as power and Lanczos methods (see [8]) which require solving, with high accuracy, a system involving the matrix B at each iteration. However, the optimization approach, which usually involves optimizing the Rayleigh quotient (for instance, with the conjugate gradient method) does apply under the above assumptions, since it only involves matrix-vector multiplications.

*Present address: Département de Mathématiques, Université d'Avignon et des Pays du Vaucluse, 84000 Avignon, France.

Therefore, we are considering a class of problems for which no factorization is practicable. See [10] for a complete bibliographical review of conjugate-gradient-like methods for eigen-like problems. For recent surveys of the state of the art on large eigenvalue problems in the general (nonsymmetric) case, see [3] and [17].

In order to simplify the presentation, we shall restrict our attention to the problem of computing the largest (or smallest) eigenvalue of a symmetric matrix A (i.e. we consider the case $B = I$). However, as we shall see, these alternative objective functions can easily be adapted for solving the *generalized* eigenproblem $Ax = \lambda Bx$. Moreover, we shall assume A to be positive definite. The method can be adapted to deal with the case where the matrix has non-positive eigenvalues (for μ large enough, the matrix $A + \mu I$ is positive definite).

We present in the next section alternative objective functions (to solve the eigenproblem via unconstrained optimization) which are not homogeneous. Some of these functions have been introduced by Auchmuty in [2] but remain apparently unknown by the numerical analysis community. We further introduce a new optimization formulation of the eigenproblem. One of our purposes is to review (and introduce one new) objective functions as good alternatives to the Rayleigh quotient for computing eigenelements via (unconstrained) optimization. We describe the variational properties of these functions. A nice property of these variational principles is the non-singularity of their Hessian at a minimum point (under a weak condition which is always satisfied in practice). This feature allows us, in Section 3, to specialize standard and recent optimization techniques for minimizing these objective functions into efficient methods for computing the largest eigenpair. Limited-memory approaches will prove to be especially well suited to our large-scale context. According to preliminary numerical experiments, this work leads the way to practical methods for computing the largest eigenpair of a (very) large symmetric matrix (pair). We conclude in Section 4.

The remaining of this introduction sets the terminology and notation required. Throughout the paper, $\|\cdot\|$ denotes the usual Euclidean norm, $\langle\cdot,\cdot\rangle$ is the usual inner product $\langle x, y\rangle = x^T y$, A denotes an n-by-n real symmetric matrix with eigenvalues

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_{n-1} \geq \lambda_n,$$

and $\mathcal{S}_n(\mathbb{R})$ represents the sets of real symmetric matrices. The spectrum of a matrix $A \in \mathcal{S}_n(\mathbb{R})$ is denoted by

$$\operatorname{spec}(A) := \{\lambda_1, \lambda_2, \ldots, \lambda_{n-1}, \lambda_n\},$$

E_λ represents the eigenspace associated with the eigenvalue λ, while S_λ denotes the set of *unitary* eigenvectors associated with the eigenvalue λ. We shall use ln to denote the natural logarithm. Note that in this paper, when we talk about a positive definite matrix, we assume that the matrix is symmetric. When talking about critical points at which a function is nonsmooth, we mean critical point in the generalized sense of nonsmooth analysis [6]. We shall say that a function $f(x)$ is *coercive* if it tends towards infinity with the norm of x.

2. Variational principles for the largest eigenvalue of a real symmetric matrix

Our first aim is to analyze some functions whose global minima are precisely the eigenvectors of A associated with the largest eigenvalue of A. We shall see in fact that the critical points of these functions give the eigenvectors of A. Undeniably, the best known function having such properties is the standard *Rayleigh quotient*, $R_A(x)$, defined by

$$R_A(x) := \frac{\langle Ax, x\rangle}{\|x\|^2}. \tag{1}$$

In fact, since for $x \neq 0$ the gradient of the Rayleigh quotient is given by

$$\nabla R_A(x) = \frac{2}{\|x\|^2}[Ax - R_A(x)x],$$

the (non-zero) critical points of R_A are precisely the eigenvectors of A, and hence the critical points corresponding to a given eigenvalue form a space whose dimension is equal to the multiplicity of this eigenvalue. Also, even in the case of a simple eigenvalue, R_A has an infinity of corresponding critical points (a subspace of dimension one). This is due to the *homogeneity* of the Rayleigh quotient, i.e. $R_A(\alpha x) = \alpha R_A(x)$, for any $x \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$. The critical points corresponding to the largest (respectively smallest) eigenvalue are global maxima (respectively minima) of R_A, any other critical point being a saddle point. It is an acknowledged fact that the presence of saddle points can make the optimization more difficult. The homogeneity of the Rayleigh quotient seems therefore to be an undesirable feature of R_A as an objective function in order to compute an extreme eigenvalue via optimization (if x is a saddle point of R_A, then so is any αx, $\alpha \in \mathbb{R}\backslash\{0\}$). Note that one could alternatively consider the constrained optimization problem

$$\max_{\|x\|=1} \langle Ax, x\rangle \tag{2}$$

in order to compute the largest eigenvalue of A. However, the nonlinearity of the constraint makes this optimization problem less attractive.

Auchmuty [1] proposed in the eighties some new unconstrained variational principles for finding various eigenvalues and eigenvectors of a real symmetric matrix. The functions he described have, as does the Rayleigh quotient, critical points related to specific eigenvalues and eigenvectors. Although these different formulations provide interesting alternatives to the standard Rayleigh quotient, they apparently remain unknown to the numerical analysis community.

We propose to recall these functions and their main properties. In the sequel, we will assume that A is positive definite. Firstly, consider $S_A : \mathbb{R}^n \to \mathbb{R}$ (S for Square root) defined by

$$S_A(x) := \|x\|^2 - 2\sqrt{\langle Ax, x\rangle}. \tag{3}$$

As for the Rayleigh quotient, R_A, this function is smooth on $\mathbb{R}^n\backslash\{0\}$. However, we shall further see that the non-smoothness at 0 causes no practical difficulty when minimizing S_A (or R_A) with a descent method. The next result is due to Auchmuty [1 ,Theorem 3].

Theorem 2.1. *Let A be a positive definite matrix, and S_A be defined in* (3). *Then*
(i) *S_A is coercive on $\mathbb{R}^n$ with*

$$\min_{x \in \mathbb{R}^n} S_A(x) = -\lambda_1.$$

The minimum is attained at any $\sqrt{\lambda_1} e_1$, where $e_1 \in S_{\lambda_1}$;
(ii) *The set of critical points of S_A is $\{0\} \cup \{\sqrt{\lambda_k} e_k \mid e_k \in S_{\lambda_k}, k = 1, \dots, n\}$. Moreover, if $\lambda_k \neq \lambda_1$, $\sqrt{\lambda_k} e_k$ is a saddle point.*

Note that S_A is not differentiable at $x = 0$. By computing the directional derivative of S_A at $x = 0$ in any direction $d \in \mathbb{R}^n \backslash \{0\}$: $S'_A(0, d) = -2\sqrt{\langle Ad, d \rangle}$, one can show in fact that the origin is a local maximum of S_A.

In order to compute an extreme eigenpair of the *generalized eigenproblem*, $Ax = \lambda Bx$, one rather optimizes

$$S_{A,B}(x) := \langle Bx, x \rangle - 2\sqrt{\langle Ax, x \rangle}.$$

In [2], Auchmuty suggested another function whose differential properties are related to the spectral properties of a positive definite matrix. He considered the function $P_A : \mathbb{R}^n \to \mathbb{R}$ (P for Polynomial) defined by

$$P_A(x) := \|x\|^4 - 2\langle Ax, x \rangle. \tag{4}$$

This function is smooth on $\mathbb{R}^n$, since it is a polynomial. The variational properties of P_A are summarized in the following theorem [2, Theorem 1]:

Theorem 2.2. *Let A be a positive definite matrix, and P_A be defined in* (4). *Then*
(i) *P_A is coercive on $\mathbb{R}^n$ with*

$$\min_{x \in \mathbb{R}^n} P_A(x) = -\lambda_1^2.$$

The minimum is attained at any $\sqrt{\lambda_1} e_1$, where $e_1 \in S_{\lambda_1}$;
(ii) *The set of critical points of P_A is $\{0\} \cup \{\sqrt{\lambda_k} e_k \mid e_k \in S_{\lambda_k}, k = 1, \dots, n\}$. Moreover, if $\lambda_k \neq \lambda_1$, $\sqrt{\lambda_k} e_k$ is a saddle point of P_A.*

Note that $x = 0$ is a local maximum of P_A.

We now propose a new variational principle for finding the largest eigenvalue of a positive definite matrix. Let $L_A : \mathbb{R}^n \backslash \{0\} \to \mathbb{R}$ (L for Logarithm) be the function defined by

$$L_A(x) := \|x\|^2 - \ln(\langle Ax, x \rangle). \tag{5}$$

Note that, contrary to S_A and P_A, L_A is not defined everywhere on $\mathbb{R}^n$. However, it is smooth at any $x \in \mathbb{R}^n \backslash \{0\}$. It is worth noting that $L_A(x)$ increases to $+\infty$ as x goes to zero. Then, any reasonable descent algorithm for minimizing this function should avoid the non-differentiability. We summarize the results concerning L_A in the following theorem.

Theorem 2.3. *Let A be a positive definite matrix, and L_A be defined in* (5). *Then*
(i) *L_A is coercive on $\mathbb{R}^n$ with*

$$\min_{x\in\mathbb{R}^n\backslash\{0\}} L_A(x) = 1 - \ln\lambda_1.$$

The minimum is attained at any $x \in S_{\lambda_1}$;
(ii) *The set of critical points of L_A is $\bigcup_{1\leq k\leq n} S_{\lambda_k}$. Moreover, the elements of $\bigcup_{1<k\leq n} S_{\lambda_k}$ are saddle points of L_A.*

Proof: For any x, we have $\langle Ax, x\rangle \leq \lambda_1\|x\|^2$. Hence

$$L_A(x) \geq \|x\|^2 - \ln(\|x\|^2) - \ln\lambda_1.$$

This proves the coercivity of L_A.

Since L_A is continuous and coercive on $\mathbb{R}^n\backslash\{0\}$, and since $\lim_{x\to 0} L_A(x) = +\infty$, a (finite) minimum is attained for L_A. The gradient of L_A is given by

$$\nabla L_A(x) = 2\left(x - \frac{Ax}{\langle Ax, x\rangle}\right). \tag{6}$$

Thus, the critical points of L_A satisfy the equation

$$Ax = \langle Ax, x\rangle x. \tag{7}$$

This shows that a critical point x is an eigenvector corresponding to the eigenvalue $\lambda := \langle Ax, x\rangle$. Taking the inner product of (7) with x, we then get

$$\langle Ax, x\rangle = \langle Ax, x\rangle\|x\|^2.$$

Thus $\|x\| = 1$, i.e. x is a *normalized* eigenvector. Moreover, the critical value corresponding to such an x is

$$L_A(x) = 1 - \ln(\langle Ax, x\rangle) = 1 - \ln\lambda.$$

By differentiating (6), one obtains the Hessian of L_A,

$$\nabla^2 L_A(x) = 2\left[I_n - \frac{\langle Ax, x\rangle A - 2(Ax)(Ax)^T}{\langle Ax, x\rangle^2}\right].$$

Let e_k be a critical point of L_A corresponding to λ_k. We complete e_k with vectors e_i, $1 \leq i \leq n, i \neq k$, into an orthonormal set of eigenvectors of A, $\{e_1, \ldots, e_n\}$, corresponding respectively to $\lambda_1, \ldots, \lambda_n$. Thus,

$$\nabla^2 L_A(e_k)e_i = \begin{cases} 2\left(1 - \dfrac{\lambda_i}{\lambda_k}\right)e_i, & \text{if } i \neq k; \\ 4e_k, & \text{else.} \end{cases}$$

Thus, $\nabla^2 L_A(e_k)$ is positive semidefinite if and only if $\lambda_k = \lambda_1$. Moreover, $\nabla^2 L_A(e_1)$ is positive definite if and only if λ_1 is a simple eigenvalue. □

Contrary to the function P_A, which is well defined for any symmetric matrix A (irrespective of whether A is positive definite or not), both functions L_A and S_A require A to be positive definite. Assume now that A is *not* positive definite. What about the variational properties of P_A in this case? The results are almost the same as the ones given in Theorem 2.2. The only difference is that the critical points reduce to the eigenvectors corresponding to *positive* eigenvalues. When A has no positive eigenvalue (A is semidefinite negative), then the only critical point of P_A is zero, which is then also the global minimizer. Auchmuty thus suggested the following function which depends on a parameter $\mu \in \mathbb{R}$:

$$P_{A,\mu}(x) := \|x\|^4 - 2\langle (A - \mu I_n)x, x\rangle.$$

From this variational principle, on can find the eigenvalues of A greater than μ and the associated eigenvectors. The properties of $P_{A,\mu}$ are the same as those of P_A, except that the critical points are now (zero and) the eigenvectors associated with the eigenvalues larger than μ. If there are no such eigenvalues, zero is the unique critical point and also the global minimizer. Such a principle may be interesting if one knows a value μ between λ_1 and λ_2. Thus, any minimization algorithm applied to $P_{A,\mu}$ will "easily" converge to an eigenvector corresponding to λ_1 (in the sense that all saddle points—but zero— are discarded).

Note finally that although we only described above how one can obtain the *largest* eigenvalue (and corresponding eigenvector) of a given real symmetric matrix, one can straightforwardly adapt the formulations for computing the *smallest* eigenvalue. For instance, using the L_A formulation, one first subtracts a sufficiently large multiple of the identity from the given matrix in order to obtain a negative definite matrix A. Then the smallest eigenvalue of the given matrix will be obtained through the minimization of L_{-A}. One can similarly use the other formulations in order to compute the smallest eigenvalue (for the Rayleigh quotient, it clearly simply suffices to *minimize* R_A instead of maximizing it).

We compared numerically the performance of the three formulations, involving the objective functions S_A, P_A, and L_A, which we presented above, for the computation of the largest eigenvalue, with that involving the Rayleigh quotient, R_A. The set of test matrices consisted of the 19 non-diagonal positive-definite matrices of dimension up to 6000 from the Matrix Market [13]. We used a limited-memory BFGS algorithm (software L-BFGS [11]) in order to solve the optimization problems. All formulations performed similarly with a slight advantage for formulations S_A and R_A (more details can be found in [14]). Homogeneity does not appear to be a serious shortcoming of the Rayleigh quotient function. In the next subsection, however, we shall show that our non-homogeneous variational principles have some special features so that powerful algorithms from optimization can be used in order to design efficient methods for computing the largest eigenpair of a real symmetric matrix.

3. Optimization algorithms for computing the largest eigenpair of a real symmetric matrix

This section is devoted to the design of numerical algorithms for estimating the *largest eigenpair* (i.e. the largest eigenvalue and a corresponding eigenvector) of a (very) large real symmetric matrix. In a first step, our approach consists of applying classical algorithms from numerical optimization to one of the non-standard variational formulations presented in Section 2. Next, we exploit the particular structure of our cost function in order to improve the performance of these algorithms. As we shall see, this leads to well-known methods in numerical analysis such as the *power method* and the *Rayleigh quotient iteration*, as well as new (to our knowledge) techniques which are competitive with (in our large-scale context) methods such as the Lanczos algorithm (see [8] for an introduction to these methods).

Henceforth, we shall concentrate our attention to the following function

$$P_A(x) := \frac{1}{4}\|x\|^4 - \frac{1}{2}\langle Ax, x\rangle. \tag{8}$$

The gradient and the Hessian of this function are given by

$$\begin{aligned}\nabla P_A(x) &= \|x\|^2 x - Ax,\\ \nabla^2 P_A(x) &= \|x\|^2 I_n + 2xx^T - A.\end{aligned}$$

Some differential properties of this function are described in Theorem 2.2. It should be noted that any of the formulations considered in Section 2 could be used in order to derive the numerical algorithms we are about to present. For the sake of simplicity, we illustrate our ideas solely on formulation (8).

3.1. Steepest descent for computing the largest eigenpair

The steepest descent method is certainly the simplest algorithm in optimization. It is divided into two steps:

Algorithm 1 (Steepest descent)
Choose $x_0 \in \mathbb{R}^n$. For $k \geq 0$ do
- *Compute the steepest descent direction $d_k := -\nabla f(x_k)$;*
- *(Line search) Compute an acceptable new iterate x_{k+1} from x_k and d_k.*

Despite its simplicity, this method can sometimes be very slow, and is rarely used in practice. When applied to P_A, this yields the following algorithm:

Algorithm 2 (Steepest descent for computing the largest eigenpair)
Choose $x_0 \in \mathbb{R}^n$. For $k \geq 0$ do
- *Compute $d_k := Ax_k - \|x_k\|^2 x_k$;*
- *(LS) Compute an acceptable new iterate x_{k+1} from x_k and d_k.*

We will discuss in detail the line-search problem in a subsequent subsection. We state a convergence result for this method.

Theorem 3.1. *If we use an exact line search, then for any $x_0 \in \mathbb{R}^n$, the sequence $\{x_k\}$ generated by the steepest descent algorithm applied to P_A converges linearly to an eigenvector of A. If, moreover, x_k converges to $\sqrt{\lambda_1}e_1$, and if λ_1 is simple, then the convergence ratio is given by*

$$\tau = \frac{1+c}{3-c} \quad \textit{where } c = \frac{\lambda_2}{\lambda_1}.$$

Proof: Noting that the spectrum of $\nabla^2 P_A(\sqrt{\lambda_1}e_1)$ is given by

$$2\lambda_1 > \lambda_1 - \lambda_n \geq \cdots \geq \lambda_1 - \lambda_2,$$

we refer the reader to [9] for a convergence result for the steepest descent method applied to a general smooth function f. □

In practice, under the condition that x_0 is not orthogonal to e_1, the above algorithm (with a satisfactory line-search procedure) converges to an eigenvector associated with the largest eigenvalue. Note also that this method is closely related to a classical algorithm in numerical analysis: the so-called *power method*, which is well known to converge linearly (with convergence ratio λ_2/λ_1) to an eigenvector corresponding to the largest eigenvalue (under the assumption that λ_1 is simple, and that the initial iterate has a non-null component in the e_1 direction).

3.2. Newton's method for computing an eigenpair

Contrary to the steepest descent method, Newton's algorithm is considered as a very fast algorithm because of its quadratic rate of convergence. However, it is much more expensive, since it requires solving, a possibly ill-conditioned linear system of equations at every iteration. Another disadvantage of Newton's method is that it is not globally convergent.

Let f be a twice continuously differentiable function. Newton's algorithm is divided into two steps:

Algorithm 3 (Newton's method)

```
Choose $x_0 \in \mathbb{R}^n$. For $k \geq 0$ do
• Solve $\nabla^2 f(x_k)d_k = -\nabla f(x_k)$;
• Set $x_{k+1} := x_k + d_k$.
```

A key condition to ensure the (local) quadratic convergence of the above algorithm is the non-singularity of the Hessian at the solution x^*. If $\nabla^2 f(x^*)$ is singular or ill-conditioned, the Newton equation cannot be reliably solved when x_k is closed to x^*.

The Hessian of P_A is non-singular at stationary points (precisely the eigenvectors of A, e_k, satisfying $\|e_k\|^2 = \lambda_k$) if and only if the corresponding eigenvalue is simple. Indeed, if

e_k is an eigenvector of A satisfying $\|e_k\|^2 = \lambda_k$, then

$$\nabla^2 P_A(e_k)e_i = \begin{cases} (\lambda_k - \lambda_i)e_i, & \text{if } i \neq k, \\ 2\lambda_k e_k, & \text{otherwise.} \end{cases}$$

Thus, Newton's method can be safely applied to P_A. This yields the following (local) quadratically convergent algorithm:

Algorithm 4 (Newton's method for computing an eigenpair)

```
Choose $x_0 \in \mathbb{R}^n$. For $k \geq 0$ do
• Solve $(\|x_k\|^2 I_n + 2x_k x_k^T - A)d_k = Ax_k - \|x_k\|^2 x_k$;
• Set $x_{k+1} := x_k + d_k$.
```

It is important to note that the Rayleigh quotient, because of its homogeneity, is systematically singular at critical points (even if the eigenvector is associated with a simple eigenvalue). This feature prevents the use of this formulation for any *Newton-like method*. This point is our main motivation in considering non-homogeneous formulations whose differential properties are related to the eigenpairs of a real symmetric matrix.

In order to improve the above algorithm, we can add an additional (free) step, namely a *radial minimization* which consists in:

Given any $x \in \mathbb{R}^n \backslash \{0\}$, find $\bar{r}$ solving the one-dimensional problem

$$\min_{r \geq 0} P_A(rx).$$

A straightforward computation yields $\bar{r} = \frac{\langle Ax, x\rangle^{1/2}}{\|x\|^2}$.

Thus, we can replace the second step of Algorithm 4 with

$$\begin{aligned} y_{k+1} &:= x_k + d_k; \\ x_{k+1} &:= \frac{\langle Ay_{k+1}, y_{k+1}\rangle^{1/2}}{\|y_{k+1}\|^2} y_{k+1}. \end{aligned}$$

The following theorem proves that, by adding the radial minimization step to the above Newton algorithm, we obtain a (local) *cubically* convergent algorithm.

Theorem 3.2. *The Newton algorithm for computing an eigenpair with the radial minimization is equivalent to the Rayleigh quotient iteration algorithm. As a consequence, it is cubically convergent.*

Proof: First, note that the radial minimization induces that $\|x\|^2 = R_A(x)$ (we drop the subscript k). Therefore, the Newton equation can be written as follows

$$d = -\left(A - R_A(x)I_n - 2R_A(x)\frac{xx^T}{\|x\|^2}\right)^{-1}(A - R_A(x)I_n)x.$$

Set $B := A - R_A(x)I_n$ (B is invertible if and only if x is not an eigenvector). Now, the Shermann-Morrison-Woodbury formula [9] implies

$$\left[B - 2R_A(x)\frac{xx^T}{\|x\|^2}\right]^{-1} = B^{-1} + \frac{2R_A(x)}{\sigma\|x\|^2}B^{-1}xx^TB^{-1},$$

where $\sigma = 1 - 2R_A(x)\frac{\langle B^{-1}x,x\rangle}{\|x\|^2}$, and thus,

$$d = -x + \alpha(A - R_A(x)I_n)^{-1}x, \quad \text{where } \alpha = -\frac{2R_A(x)}{\sigma}.$$

Finally,

$$y = x + d = \alpha(A - R_A(x)I_n)^{-1}x,$$

which corresponds to the Rayleigh quotient iteration algorithm [8]. $\square$

The above theorem tells us that, in theory, Newton's algorithm with radial minimization is equivalent to the Rayleigh quotient iteration algorithm (RQI). However, in practice both methods behave quite differently. Remember that at every step of RQI, we solve the following system of equations

$$y = (A - R_A(x)I_n)^{-1}x.$$

When x converges to an eigenvector, the matrix $A - R_A(x)I_n$ becomes more and more ill-conditioned; thus, solving the above system requires more and more computational work. Also, note that since (generally) $\lambda_1 < R_A(x) < \lambda_n$, the matrix $A - R_A(x)I_n$ is never definite (even near a solution). This prevents the use of cheap methods for solving the above equation (such as, for instance, the conjugate gradient algorithm).

Our Newton method does not suffer from these drawbacks. Indeed, the matrix $-(A - R_A(x)I_n - 2R_A(x)\frac{xx^T}{\|x\|^2})$ is always well conditioned in the neighbourhood of an eigenvector associated with a simple eigenvalue. If $x = e_k$, its condition number is equal to (we assume for the sake of simplicity that A is positive definite)

$$\begin{cases} \dfrac{\max\{2\lambda_k, \lambda_1 - \lambda_k\}}{\min\{\lambda_{k-1} - \lambda_k, \lambda_k - \lambda_{k+1}\}}, & \text{if } 1 < k < n, \\ \dfrac{2\lambda_1}{\lambda_1 - \lambda_2}, & \text{if } k = 1, \\ \dfrac{\max\{2\lambda_n, \lambda_1 - \lambda_n\}}{\min\{2\lambda_n, \lambda_{n-1} - \lambda_n\}}, & \text{if } k = n. \end{cases}$$

Moreover, if $x = \sqrt{\lambda_1}e_1$ with $\|e_1\| = 1$, i.e. if the algorithm converges to a minimizer of P_A, and if λ_1 is simple, the above Newton matrix is positive definite, and thus will remain positive definite in a neighbourhood of the minimizer. This feature is the key to all the algorithms we consider in the next subsections. However, there still remains fundamental drawbacks in our Newton algorithm. Despite its very fast convergence, it is not a globally convergent algorithm. Moreover, nothing guarantees that convergence will occur to a minimum point. Finally, each iteration requires solving with high accuracy, a linear system of equations with a different matrix. This is generally not desirable if the matrix is very large.

In the next subsection we describe some adaptations of the previous (basic) algorithm which avoid the two pitfalls mentioned above. The approaches we consider will lead to two globally convergent algorithms which are particularly well suited for estimating the largest eigenpair of a (very) large real symmetric matrix.

3.3. A quasi-Newton method for computing the largest eigenpair

As indicated in the previous subsection, Newton's method has two major disadvantages: it is a local method, and it requires solving a linear system of equations at every step. Quasi-Newton algorithms have been designed to circumvent these two difficulties. The main idea in these methods is to update at every iteration an (preferably symmetric positive definite) approximation of the Hessian. In general, a quasi-Newton algorithm is coupled with a line-search procedure (such as the Wolfe line-search) to ensure its global convergence. In this section, we do not discuss line-search strategies (this will be considered in detail below); we assume that we have at our disposal a procedure which produces an acceptable new iterate from the current iterate and the current direction.

Classically, a quasi-Newton algorithm builds a pair of vectors (y_k, s_k) at every iteration, where $y_k := \nabla f(x_{k+1}) - \nabla f(x_k)$ and $s_k := x_{k+1} - x_k$, and uses this pair to update a (inverse of the) Hessian approximation which is forced to satisfy the following *quasi-Newton* (sometimes called *secant*) equation: $M_{k+1}s_k = y_k$ (or $H_{k+1}y_k = s_k$, if one wants to approximate $(\nabla^2 f(x_{k+1}))^{-1}$).

This comes from the fact that it is generally not desirable (or possible) to compute the Hessian at every iteration. However, this is not true in our situation, since the Hessian only involves (in a simple way) the matrix A and the current iterate. This particular feature leads us to build a more adapted approximation of the inverse of the Hessian.

Quasi-newton methods usually use past and current information produced by the algorithm to build up a satisfactory approximation of the (inverse of the) Hessian. We denote by d_i, $i = 0, 1, \dots, k$, the search directions constructed by a minimization algorithm (we set $d_0 := x_0$). Moreover, we assume that we have also computed the vectors $Ad_i, i = 0, 1, \dots, k$ (this assumption is always satisfied). Thus, at iteration $k + 1$, we have at our disposal $k + 1$ pairs (d_i, Ad_i). A natural question, is how to build an “optimal” approximation of the (inverse of the) Hessian at x_{k+1}, i.e. exploiting as much information contained in these pairs as possible. Remember that

$$\nabla^2 f(x) = \|x\|^2 I_n + 2xx^T - A. \tag{9}$$

Thus, it suffices to provide an estimate for A. We could use, for instance, the BFGS or DFP updates or their multi-secant version [5] to approximate A. However, for convenience, we propose the following (simpler but efficient) choice:

$$B_{k+1} = \left(I_n - R_k R_k^T\right) B_0^{(k)} \left(I_n - R_k R_k^T\right) + R_k T_k R_k^T,$$

where $B_0^{(k)}$ is a (varying) initial approximation of A, R_k is a matrix whose columns form an orthonormal basis of the subspace spanned by the previous search directions (d_i, $i = 0, 1, \ldots, k$), $T_k := R_k^T A R_k$ is the projection of A onto the above-mentioned space. Thus, the matrix B_{k+1} is the sum of the "known part" of A (the term $R_k T_k R_k^T$), and the "unknown part" of A given by the first term. We only consider initial matrices of the form $B_0^{(k)} = \gamma_k I_n$, $\gamma_k \in \mathbb{R}$, and thus B_{k+1} can be written as follows

$$B_{k+1} = \gamma_k\big(I_n - R_k R_k^T\big) + R_k T_k R_k^T.$$

From (9), this approximation naturally induces the following estimate of $\nabla^2 f(x_{k+1})$:

$$M_{k+1} = \|x_{k+1}\|^2 I_n + 2x_{k+1}x_{k+1}^T - B_{k+1},$$

which yields in turn an estimate H_{k+1} of $(\nabla^2 f(x_{k+1}))^{-1}$. Indeed,

$$\begin{aligned} H_{k+1} = M_{k+1}^{-1} = \big(&[\|x_{k+1}\|^2 - \gamma_k]\big(I_n - R_k R_k^T\big) \\ &+ R_k[\|x_{k+1}\|^2 I - T_k]R_k^T + 2x_{k+1}x_{k+1}^T\big)^{-1}. \end{aligned}$$

Now, since x_{k+1} lies in the subspace spanned by the search directions, there exists (a unique) v_k such that $x_{k+1} = R_k v_k$, and thus, assuming that $\gamma_k \neq \|x_{k+1}\|^2$,

$$\begin{aligned} H_{k+1} &= \big([\|x_{k+1}\|^2 - \gamma_k]\big(I_n - R_k R_k^T\big) + R_k\big[\|x_{k+1}\|^2 I - T_k + 2v_k v_k^T\big]R_k^T\big)^{-1} \\ &= \frac{1}{\|x_{k+1}\|^2 - \gamma_k}\big(I_n - R_k R_k^T\big) + R_k\big[\|x_{k+1}\|^2 I - T_k + 2v_k v_k^T\big]^{-1} R_k^T, \qquad (10) \end{aligned}$$

since the columns of R_k form an orthonormal basis. We assumed above that the reduced matrix $\|x_{k+1}\|^2 I - T_k + 2v_k v_k^T$ is invertible, which is reasonable to expect in practice.

It is important to note that M_{k+1} (and thus H_{k+1}) is not guaranteed to be positive definite (since the reduced matrix $\|x_{k+1}\|^2 I - T_k + 2v_k v_k^T$ may not be positive definite). However, there is a simple way to render this approximation of the Hessian positive definite. Indeed, it suffices to consider the following perturbation of M_{k+1}:

$$\begin{aligned} \hat{M}_{k+1} &:= M_{k+1} + [c - \|x_{k+1}\|^2]I_n \\ &= [c - \gamma_k]\big(I_n - R_k R_k^T\big) + R_k\big[cI - T_k + 2v_k v_k^T\big]R_k^T, \end{aligned}$$

where $c > \lambda_{\max}(T_k)$, and to choose a scaling factor γ_k satisfying $\gamma_k < c$ (for instance, $\gamma_k := \lambda_{\min}(T_k)$). The inverse of $\hat{M}_{k+1}$ exists and is given by

$$\hat{H}_{k+1} := \hat{M}_{k+1}^{-1} = \frac{1}{c - \gamma_k}\big(I_n - R_k R_k^T\big) + R_k\big[cI - T_k + 2v_k v_k^T\big]^{-1} R_k^T.$$

However, contrary to M_{k+1}, $\hat{M}_{k+1}$ does not satisfy the curvature condition

$$R_k^T \hat{M}_{k+1} R_k = R_k^T \nabla^2 P_A(x_{k+1}) R_k.$$

Thus, by coercing $\hat{M}_{k+1}$ to be positive definite, we have lost the main thing: the approximation $\hat{M}_{k+1}$ does not reflect the local behaviour of the Hessian.

Is it essential in our situation to approximate $\nabla^2 P_A(x_{k+1})$ with a positive definite matrix when far from the solution? The key argument in unconstrained optimization is that positive definite estimates of the Hessian ensure the direction $-H_k \nabla f(x_k)$ to be a descent direction. This is important when using a line search requiring an approximate minimization of the original cost function on a *half line*. However, as we shall see in Section 4.5, we find it preferable to use a line-search procedure based on the minimization of P_A on a whole subspace. Thus, imposing the search direction to be a descent direction is not crucial in our situation, and hence, it is not essential to force the approximation of the Hessian to be positive definite.

Every iteration of our quasi-Newton algorithm requires the following operations:

- Update the orthonormal basis of the search subspace, R_k, as well as $S_k := AR_k$;
- Update the projection of A onto R_k, i.e. update $T_k := R_k^T A R_k$;
- Compute the inverse of the reduced matrix $\|x_{k+1}\|^2 I - T_k + 2v_k v_k^T$.

These different updates should be done as follows:

At step $k+1$, let R_k, d_k, Ad_k, S_k, and T_k be given;

Update of R_k:

- Compute $\hat{r}^{k+1} := d_k - R_k R_k^T d_k$; if $\|\hat{r}^{k+1}\| \neq 0$, set $r_{k+1} := \hat{r}^{k+1}/\|\hat{r}^{k+1}\|$, and $R_{k+1} := [R_k \; r_{k+1}]$; otherwise, set $R_{k+1} := R_k$.

Update of $S_k := AR_k$:

- if $\|\hat{r}^{k+1}\| \neq 0$, set $S_{k+1} := [S_k \; s_{k+1}]$, where $s_{k+1} = \frac{Ad_k - S_k(R_k^T d_k)}{\|\hat{r}^{k+1}\|}$; otherwise, set $S_{k+1} := S_k$.

Update of $T_k := R_k^T A R_k$: if $\|\hat{r}^{k+1}\| \neq 0$, set

$$T_{k+1} := \begin{pmatrix} T_k & R_k^T A r_{k+1} \\ r_{k+1}^T A R_k & r_{k+1}^T A r_{k+1} \end{pmatrix} = \begin{pmatrix} T_k & S_k^T r_{k+1} \\ r_{k+1}^T S_k & r_{k+1}^T s_{k+1} \end{pmatrix},$$

where s_{k+1} denotes the last column of S_{k+1}; otherwise set $T_{k+1} := T_k$.

In order to state the resulting algorithm, let us rewrite (10) as

$$\begin{aligned} H_{k+1} &= \frac{1}{\|x_{k+1}\|^2 - \gamma_k}\left(I_n - R_k R_k^T\right) + R_k \Lambda_k R_k^T, \quad \text{where} \\ \Lambda_k &:= \left[\|x_{k+1}\|^2 I - T_k + 2v_k v_k^T\right]^{-1}. \end{aligned} \tag{11}$$

Remember that radial minimization induces $\|x_k\|^2 = R_A(x_k)$.

Algorithm 5 (QN method for computing the largest eigenpair)

```
Choose $x_0 \in \mathbb{R}^n\backslash\{0\}$, set $R_{-1} := x_0/\|x_0\|$, $S_{-1} := Ax_0/\|x_0\|$, $T_{-1} := R_A(x_0)$, and $\Lambda_{-1} := 1/2$.
For $k \geq 0$ do
  • Compute $d_k := -H_k[R_A(x_k)x_k - Ax_k]$ (using formula (11)), and update
    $R_{k-1} \to R_k$ (as indicated above);
  • Compute $Ad_k$, and update $S_{k-1} \to S_k$ and $T_{k-1} \to T_k$ (as indicated above);
  • Line search: Compute an acceptable new iterate $x_{k+1}$ from $x_k$ and $d_k$;
  • Compute $Ax_{k+1}$, $R_A(x_{k+1})$, $v_k := R_k^T x_{k+1}$, and $\Lambda_k$ (defined in (11)).
```

Note that we have not specified the value of the scaling factor γ_k in the above algorithm. We could use, for instance, scaling strategies proposed in [11], but we do not go into broader detail since this is beyond the scope of the present work.

It is important to observe that every iteration of the above algorithm requires only one matrix-vector multiplication involving A. At step k, the storage requirements are:

– storage of two k-by-k matrices (T_k and Λ_k);
– storage of two n-by-k matrices (R_k and S_k);
– storage of four n-vectors (x_k, Ax_k, d_k, and Ad_k).

Some computational savings could certainly be made but, again, we do not go into further detail.

Unfortunately, except for standard quasi-Newton matrix updates, such as BFGS and DFP, there are very few convergence results, and the existing ones are quite poor (see for instance [9] for convergence results for quasi-Newton algorithms). However, all our experiments show that the above algorithm always converges to the largest eigenpair with a superlinear rate of convergence (this is a characteristic of quasi-Newton methods).

3.4. A truncated Newton algorithm for computing the largest eigenpair

Truncated Newton methods [7] were introduced (in the eighties) as an alternative to quasi-Newton methods for solving (very) large optimization problems. The basic idea is as follows. For the computation of the search direction, it is not necessary (and generally not desirable or, even, not possible for large problems) to solve the Newton equation with high accuracy, especially when far from the solution. In fact, any descent direction will suffice when the objective function is not well approximated by its convex quadratic model. However, in order to ensure fast local convergence, more and more effort is required for solving the Newton equation as we approach the solution.

Now, the question is: how to solve (approximatively) the Newton equation? Keep in mind that we are interested in solving very large equations, where direct methods (generally involving the factorization of the Newton matrix) cannot be applied. Thus, we must rely on iterative methods for linear systems requiring only matrix-vector multiplications. Remember that our Newton matrix is guaranteed to be positive definite only in a neighbourhood of a minimum point (under the assumption that λ_1 is simple). This may preclude the use of the conjugate gradient algorithm (CG), which requires a positive definite matrix. However, according to the above discussion on the truncated Newton method, we may use CG even in the first steps of the algorithm and stop the CG algorithm whenever a breakdown occurs (since we only require a descent direction, when far from the solution).

From now on, the (truncated) Newton iteration will be referred to as the *outer iteration*, whereas the CG iteration (for solving the Newton equation) will be referred to as the *inner iteration*.

An essential question that remains is the choice of stopping criteria for CG. The CG process should be stopped if any of the following situations occur:

(i) the residual is small enough;
(ii) the (fixed) maximal number of iterations allowed is attained;
(iii) a direction of negative curvature is encountered.

The last two criteria are trivial to implement (although they require a careful choice of the parameters). The first criterion is more awkward, since it should incorporate information from the outer iterations. For our numerical experiments, we chose the following implementation due to [7].

At the k-th outer iteration, exit inner loop if the residual r satisfies

$$\|r\| \leq \min\{c/k, \|g_k\|\}\|g_k\|,$$

where g_k denotes the right-hand side of the Newton equation at step k, and c is a real number around 0.5. Note that this criterion becomes more restrictive as k increases.

We describe below the outer and inner iterations of the truncated Newton algorithm.

Algorithm 6 (TN algorithm for computing the largest eigenpair)

```
Outer iteration:
    Choose $x_0 \in \mathbb{R}^n \backslash \{0\}$. For $k \geq 0$, do
• Solve (approximatively)
        $\left(R_A(x_k)I_n + 2R_A(x_k)\frac{x_k x_k^T}{\|x_k\|^2} - A\right)d_k = Ax_k - R_A(x_k)x_k;$
• Line search: Compute an acceptable new iterate $x_{k+1}$ from $x_k$ and $d_k$.
Inner iteration:
    For the sake of simplicity, we denote by $H_k$ the matrix in the Newton equation, and by $g_k$ the right-hand side.
Set $p_0 := 0$, $r_0 := -g_k$, and $s_0 := r_0$.    For $i \geq 0$ do
• $\alpha_i := \|r_i\|^2/\langle H_k s_i, s_i\rangle$;        (step length)
• $p_{i+1} := p_i + \alpha_i s_i$;                   (approximate solution)
• $r_{i+1} := r_i - \alpha_i H_k s_i$;               (residual)
• Termination test:
    if $R_{H_k}(s_i) < 0$, exit with $d_k := p_{i+1}$;
    if $i >$ maximal number of iterations, exit with $d_k := p_{i+1}$;
    if $\|r_{i+1}\| \leq \min\{c/k, \|g_k\|\}\|g_k\|$, exit with $d_k := p_{i+1}$;
• Else
    set $\beta_i := \|r_{i+1}\|^2/\|r_i\|^2$, and        (improvement at this step)
    $s_{i+1} := r_{i+1} + \beta_i s_i$.                  (search direction)
```

Generally, after detection of a negative curvature direction, s_i, the CG process returns $d_k := p_i$. However, for our purposes (remember that we wish to compute the largest eigenpair), the negative curvature direction is an interesting direction, and is then incorporated in the (outer) descent direction. Indeed, assume that s_i satifies $R_{H_k}(s_i) < 0$, this implies

$$R_A(x_k)\|s_i\|^2 + 2R_A(x_k)\frac{\langle x_k, s_i\rangle^2}{\|x_k\|^2} - \langle As_i, s_i\rangle < 0,$$

and thus,

$$R_A(x_k) + 2R_A(x_k)\left[\frac{\langle x_k, s_i\rangle}{\|x_k\|\|s_i\|}\right]^2 < R_A(s_i).$$

A first straightforward implication of the above inequality is that $R_A(s_i) > R_A(x_k)$. Hence, s_i has larger components in the space spanned by the eigenvectors associated with the largest eigenvalues than x_k. Consequently, s_i contains useful information relative to the largest eigenvalues. Moreover, this inequality yields

$$\frac{|\langle x_k, s_i\rangle|}{\|x_k\|\|s_i\|} < \left[\frac{R_A(s_i) - R_A(x_k)}{2R_A(x_k)}\right]^{1/2},$$

and thus the angle between x_k and s_i is monitored by the right-hand side, and cannot be too small. To conclude this subsection, we simply mention that, contrary to the previous methods, the truncated Newton method allows the use of a preconditioner to accelerate the convergence in the inner iterations. In practice, such a feature should be used to improve the convergence of the whole algorithm.

3.5. The line-search procedure

The line search is an essential step in any (globally convergent) optimization algorithm. It generally ensures the convergence of the algorithm to a stationary point (in general, a local minimizer) from any starting point. There exists various (inexact) line-search strategies, such as the Wolfe line-search, or the backtracking strategy [4]. Here, however, we concentrate on an *exact* line search. Such a strategy can rarely be used for minimizing general nonlinear functions, since it requires the exact (expensive) solution of a one-dimensional minimization problem.

As we shall see, an exact line search can be implemented for free in our situation. Given $x, d \in \mathbb{R}^n$, we wish to solve the following one-dimensional problem

$$\min_{t \in \mathbb{R}} P_A(x + td)$$

(note that we do not restrict t to be positive, since this constraint yields no improvement in practice, and would complicate our exposition). When coupled with a radial minimization step (cf. Section 3.2), this provides an improved iterate $x_+ := \bar{r}(x + \bar{t}d)$, where $\bar{t}$ is a global minimizer of $P_A(x + td)$, and $\bar{r}$ is the minimum point of $P_A(r(x + t\bar{d}))$. Note that the new iterate lies in the subspace spanned by x and d.

What is the result of the exact line-search procedure when applied to the Rayleigh quotient? Again, we look for the solution of

$$\max_{t \in \mathbb{R}} R_A(x + td),$$

(remember that minimization of P_A is related to maximization of R_A). Because of the homogeneity of R_A, the above problem is equivalent to maximizing R_A on the subspace spanned by x and d. However, the analogue is not true for P_A, since it is not homogeneous. Clearly, the coupled exact line search/radial minimization provides a new iterate which is worst than the one produced by the minimization of P_A on the subspace spanned by x and d (henceforth, we denote this subspace by $V(x, d)$).

Moreover, the computational (and storage) requirements for both approaches are similar. Thus, it seems natural to opt for the exact minimization of P_A on $V(x, d)$, rather than for the exact line search/radial minimization procedure.

How does one compute x_+ (the new iterate) in practice? Let R be a 2-by-n matrix whose columns form an orthonormal basis of $V(x, d)$. Then, minimizing P_A on $V(x, d)$ reduces to the minimization of $P_{R^T AR}$ on $\mathbb{R}^2$. If we denote by v the solution of the latter problem, then $x_+ = Rv$. Note that v is the eigenvector of the reduced matrix $T := R^T AR$ corresponding to the largest eigenvalue, and of norm $\|v\| = \sqrt{\lambda_{\max}(T)}$ (see Theorem 2.2).

Thus, our special "line"-search procedure can be summarized as follows:

- Given x, d, compute R such that $R^T R = I_2$ and $Range(R) = V(x, d)$;
- Compute $T := R^T AR$, and v, an eigenvector associated with the largest eigenvalue and of norm $\|v\| = \sqrt{\lambda_{\max}(T)}$;
- Set $x_+ := Rv$.

3.6. A multi-dimensional search procedure

At step k of any descent algorithm, the current iterate is of the form

$$x_{k+1} = x_0 + t_1 d_1 + \cdots + t_k d_k,$$

where x_0 is the initial iterate, d_i, $i = 1, 2, \ldots, k$, are the previous search directions, and t_i, $i = 1, 2, \ldots, k$, are positive step lengths computed by a line-search procedure. A natural extension of the approach presented in the previous subsection would be to replace the k one-dimensional minimizations required to compute x_k, with one multi-dimensional minimization. That is, to compute at every step

$$x_{k+1} := \operatorname{argmin}\{P_A(x) \mid x \in V(x_0, d_1, \ldots, d_k)\},$$

where $V(x_0, d_1, \ldots, d_k)$ denotes the $k + 1$-dimensional subspace spanned by $x_0, d_1 \ldots, d_k$. It is obvious that such a step would result in a much faster (in terms of number of outer iterations) minimization algorithm than one using a standard line search. The cost of such a multi-dimensional search procedure is clearly prohibitive in standard minimization algorithms for general nonlinear functions.

Nevertheless, the special structure of our objective function allows the implementation of such a strategy at a reasonable cost. Indeed, let R be a $(k + 1)$-by-n matrix whose columns form an orthonormal basis of $V(x_0, d_1, \ldots, d_k)$. Then, $x_{k+1} = Rv$, where v is the minimum of the reduced cost function P_{R^TAR}. According to Theorem 2.2, v is an eigenvector of $T := R^TAR$ associated with the largest eigenvalue and of norm $\|v\| = \sqrt{\lambda_{\max}(T)}$.

Thus, the whole procedure requires (assuming that R^TAR is available) the computation of the largest eigenpair of a reduced $(k + 1)$-by-$(k + 1)$ matrix (keep in mind that $k \ll n$), which can be done in $O(k)$ flops (by using for instance a coupled bisection/inverse iteration algorithm [8]), and the matrix-vector multiplication Rv, which requires $2kn$ flops.

Of course, it is not desirable to extract an orthonormal basis of the search space, and to form the reduced matrix R^TAR at every iteration (the former requires $O(k^2n)$ flops, using the modified Gram-Schmidt algorithm, and the latter $O(k^2n)$ flops and k matrix-vector multiplications involving A). Then, an “updating strategy” should be implemented to make this search procedure affordable.

Such a strategy has already been implemented for the quasi-Newton method (cf. Section 3.3). Therefore, the multi-dimensional search (MDS) procedure can be summarized as follows:

At step $k + 1$ of a minimization algorithm, let d_k, Ad_k, R_k, $S_k(:= AR_k)$, and T_k be given;

- Update $R_k \to R_{k+1}$, $S_k \to S_{k+1}$, and $T_k \to T_{k+1}$, as indicated in Section 3.3;
- Compute (σ_{k+1}, v_{k+1}), the largest eigenpair of the reduced matrix T_{k+1}, with $\|v_{k+1}\|^2 = \sigma_{k+1}$;
- Set $x_{k+1} := R_{k+1}v_{k+1}$, and $R_A(x_{k+1}) := \sigma_{k+1}$;

Note that the radial minimization step is useless when using MDS, since MDS provides a radially-minimized solution. It is interesting to observe that the above procedure coincides with the well-known *Ritz procedure* commonly used in Krylov subspace methods such as Lanczos' or Arnoldi's algorithms [8].

In the next subsections, we replace the line-search step with the MDS step in the different algorithms we considered in the previous subsections.

3.6.1. The steepest descent algorithm with MDS (SD-MDS)

Algorithm 7

```
Choose $x_0 \in \mathbb{R}^n \backslash \{0\}$, and set $\sigma_0 := R_A(x_0)$, $R_0 := x_0/\|x_0\|$, $S_0 := Ax_0/\|x_0\|$, $T_0 := \sigma_0$;
For $k \geq 0$, do
• Compute $d_k := Ax_k - \sigma_k x_k$;
• Use MDS to update $R_k$, $S_k$, $T_k$, and compute $x_{k+1}$, $\sigma_{k+1}$, and $Ax_{k+1}$.
```

This algorithm is equivalent to the well-known *Lanczos algorithm*, which is one of the most efficient, robust, and adapted method for computing a few largest eigenpairs of a (very) large symmetric matrix. We compared it with the steepest descent method with the exact (bi-dimensional) “line”-search procedure described in Section 3.5. The MDS step leads to a very significant improvement, as we shall see in the Numerical experiments subsection (Section 3.8).

3.6.2. The quasi-Newton algorithm with MDS (QN-MDS). Despite the fact that the approximation (10) of the Hessian, M_{k+1}, is positive definite in the MDS situation (since $\|x_{k+1}\|^2 = \|v_{k+1}\|^2 = \lambda_{\max}(T_k)$), the quasi-Newton method coupled with the MDS strategy gives rise to a disappointing algorithm. Indeed, we tested it on many instances, and it systematically gave results which are identical (i.e. identical number of iterations, identical final solution) to results obtained with the steepest descent with MDS. Thus, it seems that the quasi-Newton matrix yields no improvement compared with the identity matrix in this situation. We also tested a multi-secant BFGS approximation of A (which is not guaranteed to produce a positive definite approximation of the Hessian), and the results we obtained were not better.

This phenomenon may be explained, by the fact that the information contained in the quasi-Newton matrix, has already been exploited by the MDS. Thus, the quasi-Newton matrix offers no additional information compared with the identity matrix.

However, we see next how the quasi-Newton matrix may be exploited in order to improve the truncated Newton algorithm.

3.6.3. The truncated Newton algorithm with MDS (TN-MDS). As for the steepest descent, the MDS procedure leads to a significant improvement compared with the “line”-search strategy (considered in Section 3.5) when implemented within the truncated Newton algorithm, as we shall see in the Numerical experiments subsection (Section 3.8). This results in a promising algorithm which, in many situations, is competitive and even more efficient than the steepest descent algorithm with MDS (the latter being equivalent to the Lanczos algorithm). Our computational experiments will clearly illustrate that when comparing the number of outer iterations, TN-MDS converges much faster than SD-MDS. However, it is the number of inner (CG) iterations which corresponds to the number of matrix-vector multiplications. Hence, TN-MDS requires more matrix-vector products than SD-MDS in all our tests. This is due to the fact that SD-MDS stores much more information about A (by the way of MDS) than TN-MDS, which only retains information from outer iterations. Nevertheless, a feature of the TN method is that it allows the use of a preconditioner in order to reduce appreciably, the number of iterations in inner loops (for instance, one could use the quasi-Newton approximation of the inverse of the Hessian we proposed in (10)). Thus, a more reliable criterion of comparison would, in fact, not be the number of inner iterations, but rather the number of outer iterations.

3.7. Limited-memory strategies

As seen in the previous subsections, the multi-dimensional search procedure is very efficient when coupled with the steepest descent or the truncated Newton method. Unfortunately, it requires a large amount of storage: in particular, at step k, we have to store two k-by-n matrices. When the size of the matrix A is (very) large (this is precisely the situation in which we are interested), storage limitation may render it impossible to store the two previous matrices when k becomes large (which is likely to happen if the largest eigenvalues of A are poorly separated).

Of course, this shortcoming should not prevent the use of this efficient search strategy. However, we are forced to define strategies for selecting a predefined number (say m) of vectors (in fact pairs of vectors) from the whole search space. A strategy which is common in optimization (for quasi-Newton methods) consists in storing the most recent pairs $\{d_i, Ad_i\}$ (the d_i's are the previous search directions), since earlier pairs are less likely to be relevant to the actual behaviour of the Hessian at the current point. Are there better strategies for our particular cost function?

First, note that the first m iterations of a limited-memory algorithm are the same as those of the full-memory version. At the beginning of step $m+1$, the available storage is full. The first strategy we propose is the "classical one", that consists in removing the oldest pair, and saving the new one. This strategy will be referred to as the *FIFO* (First In First Out) *selection*.

An alternative approach is to control the selection of the pairs according to the Rayleigh quotients. Indeed, remember that we are concerned with the computation of the largest eigenpair of A. Therefore, it seems natural to keep, in the search subspace, directions with large components in the direction of the eigenvectors corresponding to the largest eigenvalues, that is, the directions with a large Rayleigh quotient. This method will be referred to as the *Rayleigh quotient selection*. Note that this approach requires very little additional computation, since we only have to compute the Rayleigh quotient of the new pair at each iteration (the matrix-vector multiplication involved being already computed).

The third method is an extension of the previous one. It is based on the following idea: why should we select m directions amongst the $m + 1$ given vectors, when it would be advantageous to select them in the subspace spanned by these $m + 1$ vectors? This new strategy is described in detail below:

- We have at our disposal m old pairs $\{d_i, Ad_i\}_{1\leq i\leq m}$, and a new pair $\{d_{m+1}, Ad_{m+1}\}$;
- Compute R, an orthonormal basis of the extended search subspace spanned by $\{d_i\}_{1\leq i\leq m+1}$;
- Compute the eigendecomposition of the $(m + 1)$-by-$(m + 1)$ matrix $T := R^T AR$;
- Let W be the $(m + 1)$-by-m matrix whose columns are the eigenvectors of T associated with the m largest eigenvalues;
- Compute $[\hat{d}_1, \ldots, \hat{d}_m] := RW$, and $[A\hat{d}_1, \ldots, A\hat{d}_m] := (AR)W$.

At the end of this procedure, we have m new pairs $\{\hat{d}_i, A\hat{d}_i\}_{1\le i\le m}$ which are linear combinations of the $m+1$ pairs $\{d_i, Ad_i\}_{1\le i\le m+1}$. By construction, the search space, spanned by the new directions, contains more information relative to the largest eigenpairs of A than the previous ones. It is also clear that it provides a better search space than the first two strategies we considered. However, it requires a more significant computational effort. The above algorithm will be referred to as the *Ritz value selection*.

An essential point is that it is necessary that the new search space contains the pair $\{x_k, Ax_k\}$ (x_k denotes the last iterate) when using any of the above strategies, since we have been forced to discard some older search directions.

Now, assume that we are implementing the MDS procedure as in Section 3.6. Therefore, we no longer store previously computed directions, but rather an orthonormal basis of the current search space. It is therefore out of the question to use the FIFO selection in such a situation. Moreover, in this case, the Rayleigh quotient selection and the Ritz value selection are equivalent, since the stored directions are related to the eigenvectors of the projection of A onto the search space. Note also that there is no reason to store the additional pair $\{x_k, Ax_k\}$, since x_k is already contained in the search space.

A feature of our limited-memory approaches is that, when coupled with the MDS procedure, they yield, for $m = 1$, the (bi-dimensional) "line" search analyzed in Section 3.5 (for $m = 1$, the three strategies are of course equivalent).

It may be useful to retain some information from discarded pairs. For that purpose, Nazareth et al. [15] proposed to use a quasi-Cauchy algorithm (a kind of quasi-Newton algorithm where the updated matrices are restricted to be diagonal). Such a method requires only the storage of an n-dimensional vector, and has been shown to be very efficient when implemented within the steepest descent algorithm. Thus, this diagonal updating strategy should be implemented with the limited-memory steepest descent with MDS. The resulting algorithm may provide an interesting alternative to the (full-memory) Lanczos method which requires too large an amount of storage for very large problems, especially when the largest eigenvalues are poorly separated.

Indeed, we performed some numerical experiments which clearly show that the efficiency of the steepest descent with MDS deteriorates significantly when the whole search space is not used in the MDS procedure. The limited-memory approach (with the efficient *Ritz value selection*) with small values for m leads to a slowly convergent algorithm, and using larger m greatly improves the convergence (see first column of Table 2).

However, in our opinion, the most promising algorithm is the limited-memory truncated Newton method with MDS. The numerical comparison (see Tables 2 and 3) clearly shows that it behaves much better in our large-scale context than the steepest descent with MDS. Moreover, certain features, such as its potential for the use of preconditioning techniques to improve the convergence in inner loops, make it very attractive.

3.8. Numerical experiments

We implemented on Matlab the limited-memory (with the *Ritz value selection*) version of the two most promising algorithms introduced in this paper: the truncated Newton method and the steepest descent method, both using the multi-dimensional search procedure, the

Table 1. Test matrices.

Matrix name	Dimension	Non-zero entries
NOS1	237	627
NOS2	957	2547
NOS3	960	8402
NOS4	100	347
NOS5	468	2820
NOS6	675	1965
NOS7	729	2673

Table 2. Impact of the storage on the number of iterations.

		TN-MDS	
m	SD-MDS	Outer	Inner
1	5282	18	277
5	721	14	226
10	327	14	229
30	111	14	229
50	89	14	229
∞	82	14	229

latter being thus equivalent to Lanczos. We tested both algorithms on the matrices described in Table 1. These matrices come from the Matrix Market [13]: set LANPRO (Lanczos with partial reorthogonalization). We are concerned with the number of matrix-vector multiplications required to converge to the largest eigenpair. This corresponds to the number of iterations in the case of SD (and SD-MDS), and to the number of *inner* iterations in the case of TN (and TN-MDS).

In our first experiments, we analyze the impact of varying the value of the storage parameter, m, on the number of iterations, for both TN-MDS and SD-MDS algorithms. We used the matrix NOS1, and we used as a stopping rule, the following: the optimization stops when the current eigenvector estimate, x, and the current eigenvalue estimate, λ, are such that

$$\frac{\|Ax - \lambda x\|}{\lambda \|x\|} < \varepsilon, \tag{12}$$

where ε is some pre-specified tolerance. We set ε to 10^{-9}, and the initial point was $(1, \dots, 1)^T$ (the initial point for the CG subroutine in TN-MDS was always $(0, \dots, 0)^T$, although a practical implementation would rather use a “warm start”). The results are presented in Table 2, where $m = \infty$ refers to the full-memory algorithms, and $m = 1$ corresponds to algorithms SD and TN (using the “line”-search procedure described in Section 3.5).

Table 3. Comparison of the number of iterations on a sequence of test matrices.

	$m = 1$			$m = \infty$		
		TN		SD-MDS	TN-MDS	
Matrix	SD	Outer	Inner	($\equiv$ Lanczos)	Outer	Inner
NOS1	8502	20	368	83	16	280
NOS2	>10000	67	2775	321	23	783
NOS3	1541	20	198	93	17	190
NOS4	591	12	139	58	9	135
NOS5	351	12	88	43	12	86
NOS6	652	17	137	60	17	147
NOS7	87	14	39	19	14	36

These results clearly show that the performance of SD-MDS deteriorates significantly when we reduce the storage (the convergence becomes very slow for $m = 1$), whereas the TN-MDS algorithm does not suffer from storage limitation (this can be explained by the fact that fewer (outer) iterations are necessary for TN (compared to SD) to converge to a fixed accuracy). Note that naturally, once m is greater than the number of (outer) iterations, increasing m further has no effect. As previously mentioned, the use of a preconditioner for solving the CG (inner loop) subproblem, should bring down the number of inner iterations per outer iteration. That is why we expect that (a small multiple of) the number of *outer* iterations is a more reliable criterion for comparison. Thus, TN-MDS appears to have potential for improvement over, for instance, the Lanczos method (SD-MDS, $m = \infty$).

Similar conclusions can be drawn from Table 3, which reports analogous results (stopping criterion: $\varepsilon = 10^{-12}$) on several test matrices, and with the extreme values $m = 1$ and $m = \infty$ for the storage parameter.

In Table 4, we report the number of iterations required by TN-MDS to converge (tolerance: $\varepsilon = 10^{-11}$) with the test matrix NOS1. In these experiments, we vary the value of the limited-memory parameter, m, as well as the maximal number of inner (CG) iterations (remember

Table 4. Number of iterations of TN-MDS with respect to storage and maximal number of CG iterations.

	Maximal number of CG iterations									
	5		10		20		30		∞	
m	Outer	Inner	Outer	Inner	Outer	Inner	Outer	Inner	Outer	Inner
1	388	1547	59	505	38	595	24	439	20	368
5	65	255	24	191	18	249	16	268	15	263
10	41	159	22	173	18	247	16	252	15	250
∞	33	127	22	173	17	225	16	251	15	250

that this is only one of the three stopping rules used in the CG algorithm—cf. Section 3.4). When setting $m = 1$ (i.e. when using the (bi-dimensional) "line" search), it seems to be beneficial to use a large value for the maximal number of inner loops, that is to say to solve with high accuracy the Newton equation when close to the solution. For the full-memory version ($m = \infty$), the situation is completely different; limiting the number of inner loops results in faster convergence.

4. Conclusion

In this paper, we first reviewed non-homogeneous objective functions, as alternatives to the usual Rayleigh quotient, for computing eigenelements via (unconstrained) optimization. Two of these functions (S_A and P_A) were introduced by Auchmuty. We introduced a third one (L_A), and we described the variational properties of these functions. A feature of the proposed non-homogeneous formulations (namely the non-singularity of their Hessian at minimum points) allowed specialization, of standard and recent optimization techniques, into efficient methods for computing the largest eigenpair. According to the preliminary numerical experiments we presented, this work leads the way to improvement to the standard Lanczos method for computing the largest eigenpair of (very) large real symmetric matrices. Such benefits could be twofold. Firstly, in terms of speed of convergence (number of matrix-vector products) with our algorithm TN-MDS (a truncated Newton method with multi-dimensional search, specialized to one of the proposed non-homogeneous objective functions). Secondly, in terms of the size of problems that can be addressed, through the use of a limited-memory version of TN-MDS.

We saw that the alternative objective functions we presented, can easily be adapted for solving the generalized eigenproblem $Ax = \lambda Bx$. Indeed, even in this more general context, only matrix-vector products are involved (no factorization of B is required). This makes the optimization approach to the eigenproblem particularly well suited for practical problems involving very large matrices.

In order to obtain methods for computing the condition number and the width of the spectrum of a real symmetric matrix, using (unconstrained) optimization, we derived analogous non-homogeneous variational principles (details can be found in Section 5 of [14]).

Acknowledgment

The authors are indebted to J.-B. Hiriart-Urruty who suggested this study, and brought the work of Auchmuty to our attention. Moreover, we thank him for interesting discussions on the subject.

References

1. G. Auchmuty, "Unconstrained variational principles for eigenvalues of real symmetric matrices," SIAM Journal on Mathematical Analysis, vol. 20, no. 5, pp. 1186–1207, 1989.
2. G. Auchmuty, "Globally and rapidly convergent algorithms for symmetric eigenproblems," SIAM Journal on Matrix Analysis and Applications, vol. 12, pp. 690–706, 1991.

3. Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide. SIAM, 2000.
4. J.F. Bonnans, J.-C. Gilbert, C. Lemaréchal, and C. Sagastizábal, Optimisation Numérique, Springer, 1997.
5. R.H. Byrd, J. Nocedal, and R.B. Schnabel, "Representations of quasi-Newton matrices and their use in limited-memory," Mathematical Programming, vol. 63, pp. 129–156, 1994.
6. F.H. Clarke, Optimization and Nonsmooth Analysis, John Wiley, 1983.
7. R.S. Dembo and T. Steihaug, "Truncated-Newton algorithms for large-scale unconstrained optimization," Mathematical Programming, vol. 26, pp. 190–212, 1983.
8. J.W. Demmel, Applied Numerical Linear Algebra, SIAM, 1997.
9. J.E. Dennis and R.B. Schnabel, Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Prentice-Hall, 1983.
10. A. Edelman and S.T. Smith, "On conjugate gradient-like methods for eigen-like problems," BIT, vol. 36, no. 3, pp. 494–508, 1996.
11. D.C. Liu and J. Nocedal, "On the limited memory method for large scale optimization," Mathematical Programming, vol. 45, no. 3, pp. 503–528, 1989.
12. D.E. Longsine and S.F. McCormick, "Simultaneous Rayleigh-quotient minimization methods for $Ax = \lambda Bx$," Linear Algebra and its Applications, vol. 34, pp. 195–234, 1980.
13. Matrix Market, A visual repository of test data for use in comparative studies of algorithms for numerical linear algebra. See web site http://math.nist.gov/MatrixMarket/
14. M. Mongeau and M. Torki, "Computing eigenelements of real symmetric matrices via optimization," Technical Report MIP 99-54, Université Paul Sabatier, Toulouse, France, 1999. Available at web site http://mip.ups-tlse.fr.
15. J.L. Nazareth, H. Wolkowicz, and M. Zhu, "The quasi-Cauchy relation and diagonal updating," SIAM Journal on Optimization, vol. 9, no. 4, pp. 1192–1204, 1999.
16. A.H. Sameh and J.A. Wisniewski, "A trace minimization algorithm for the generalized eigenvalue problem," SIAM Journal on Numerical Analysis, vol. 19, no. 6, pp. 1243–1259, 1982.
17. D.C. Sorensen, "Numerical methods for large eigenvalue problems," Acta Numerica, vol. 11, pp. 519–584, 2002.