ROMANIAN JOURNAL OF MATHEMATICS AND COMPUTER SCIENCE, 2011, VOLUME 1, p.29-36

# On the influence of rounding errors in the C++ implementation of simple iteration algorithm

*George Daniel Mateescu*[1]

**Abstract**. *Rounding errors can completely change the outcome of an algorithm. Moreover, mathematical equivalent implementations may generate different results. We analyze this assertion, in C++ implementation, by using the algorithm for determining the maximum eigenvalue. We proved that certain statements in the bibliography are not true. It can happen that instead of the maximum eigenvalue, one other eigenvalue to be determined. It is also possible that the algorithm for determining all eigenvalues to generate the eigenvalues in a different order.*

**AMS Classification**: 65F15, 03-04

## I. Introduction

Let $A=\left(a_{ij}\right)_{i,j=1..m}$ be a real, symmetrical matrix. The simple vector iteration algorithm consists into the following iterative sequence:

$$x_0 \in \mathbb{R}^m,\ x_{n+1}=\frac{Ax_n}{\|x_n\|}, n \geq 0, \tag{1}$$

thus $\|x_n\| \to \left|\overline{\lambda}\right|$, where $\left|\overline{\lambda}\right| = \max_{\lambda \in \sigma(A)} |\lambda|$

This is because $x_n = \frac{A^n x_0}{\left\|A^{n-1}x_0\right\|}$, and if we consider the orthonormal basis

$$u_1,\ u_2,\ \ldots,\ u_m,\ \left\langle u_i, u_j \right\rangle = \delta_{ij}, i, j = 1..m, \tag{2}$$

[1] University of Civil Engineering, Department of Mathematics and Computer Science
e-mail: dan@mateescu.ro

consisting of eigenvectors of the *A* matrix, it follows:

$$\|x_n\| = \frac{\left\|\lambda_1^n \alpha_1 u_1 + \lambda_2^n \alpha_2 u_2 + ... + \lambda_m^n \alpha_m u_m\right\|}{\left\|\lambda_1^{n-1} \alpha_1 u_1 + \lambda_2^{n-1} \alpha_2 u_2 + ... + \lambda_m^{n-1} \alpha_m u_m\right\|} \tag{3}$$

where $x_0 = \alpha_1 u_1 + \alpha_2 u_2 + ... + \alpha_m u_m$

If we consider

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq ... \geq |\lambda_m|$$

it results

$$\|x_n\| = \frac{|\lambda_1^n| \left\|\alpha_1 u_1 + \left(\frac{\lambda_2}{\lambda_1}\right)^n \alpha_2 u_2 + ... + \left(\frac{\lambda_m}{\lambda_1}\right)^n \alpha_m u_m\right\|}{|\lambda_1^{n-1}| \left\|\alpha_1 u_1 + \left(\frac{\lambda_2}{\lambda_1}\right)^{n-1} \alpha_2 u_2 + ... + \left(\frac{\lambda_m}{\lambda_1}\right)^{n-1} \alpha_m u_m\right\|} \to |\lambda_1| \tag{4}$$

Moreover, if $\lambda_1 > 0$ then

$$x_n = \frac{\lambda_1^n \left(\alpha_1 u_1 + \left(\frac{\lambda_2}{\lambda_1}\right)^n \alpha_2 u_2 + ... + \left(\frac{\lambda_m}{\lambda_1}\right)^n \alpha_m u_m\right)}{\lambda_1^{n-1} \left\|\alpha_1 u_1 + \left(\frac{\lambda_2}{\lambda_1}\right)^{n-1} \alpha_2 u_2 + ... + \left(\frac{\lambda_m}{\lambda_1}\right)^{n-1} \alpha_m u_m\right\|} \to \frac{\lambda_1 \alpha_1}{|\alpha_1| \|u_1\|} u_1 \in \mathbb{R} u_1,$$

i.e. $x_n$ converges to an eigenvector which corresponds to the eigenvalue $\lambda_1$.

We observe that if $\lambda_1 = \lambda_2$, the algorithm is still convergent.

The above result has many demonstrations, as it can be seen in [1], [2], [3]. But, we prefer this demonstration (see [2]) because it reveals a particular situation, in that concern the initial vector, $x_0$

**II. C++ implementation of the algorithm**

It is clear that we cannot use formula (3) because the eigenvectors of the matrix are not known; thus, we have to use formula (1). On the other hand, formula

(3) holds if $\alpha_1$ is nonzero, otherwise the formula (4) is false. Because the eigenvectors are unknown, there is no way to be sure that the initial vector is orthogonal (or not) to $u_1$. Some authors suggest that in such a case, *the rounding errors* will generate a nonzero component for $u_1$ and the algorithm still determines $|\lambda_1|$ ([1], [2], [3]).

This assertion may be false, as we can see in the following. Moreover, the result may depend on the implementation, as it can be seen from the following two sequences. We used the matrix $A=\begin{pmatrix} 12 & -2 & 4 \\ -2 & 8 & 5 \\ 4 & 5 & 9 \end{pmatrix}$, implemented by `a[][]`; the sequence $x_n$ is implemented by `x[]`. The eigenvalues are $\lambda_1 = 15.1311783$, $\lambda_2 = 12.1245619$, $\lambda_3 = 1.7442598$ and the corresponding eigenvectors are:

$$u_1 = (-0.6820291 \quad -0.2818689 \quad -0.6748231)$$

$$u_2 = (0.6229796 \quad -0.7072428 \quad -0.3342215)$$

$$u_3 = (-0.3830572 \quad -0.6483498 \quad 0.6579588)$$

I)

```
{ norm=0;
 for(i=1;i<=m;i++)
     norm=norm+x[i]*x[i];
 norm=sqrt(norm);
 for(i=1;i<=m;i++)
     x[i]=x[i]/norm;
 for(i=1;i<=m;i++)
    {s=0;
     for(j=1;j<=m;j++)
        s=s+a[i][j]*x[j];
     y[i]=s;}
 for(i=1;i<=m;i++)x[i]=y[i];
}
```

II)

```
{ norm=0;
 for(i=1;i<=m;i++)
     norm=norm+x[i]*x[i];
 norm=sqrt(norm);
 for(i=1;i<=m;i++)
    {s=0;
     for(j=1;j<=m;j++)
        s=s+a[i][j]*x[j];
     y[i]=s/norm;}
 for(i=1;i<=m;i++)x[i]=y[i];
}
```

We used the starting vector $x_0 = u_2 + u_3$, which is orthogonal to $u_1$. Both sequences are executed repeatedly, until the 7-digit accuracy is achieved, i.e. until `norm` keeps the first 7 digits unchanged. The first block corresponds to the iteration: $x_{n+1} = A\left(\frac{x_n}{\|x_n\|}\right)$, while the second corresponds to the iteration $x_{n+1} = \frac{Ax_n}{\|x_n\|}$. Because $A$ is a linear operator (matrix) the two formulas are equivalent. But, due to rounding errors, with the first block we found the eigenvalue $\lambda_1 = 15.1311783$ and with the second block, we found the eigenvalue $\lambda_2 = 12.1245619$. The results differ, even we used the same starting vector, implemented by:

```
x[1]=0.2399224; x[2]=-1.3555927; x[3]=0.3237372;
```

which is orthogonal to $u_1$.

We conclude that the rounding errors cannot assure the convergence of the algorithm, in the second implementation, unlike [1]. In the first sequence, the rounding errors will generate a nonzero coefficient $\alpha_1$ having as result the determination of $\lambda_1$.

**III. Determine all the eigenvalues**

If we have determined the maximum eigenvalue and the corresponding eigenvector, we examine the possibility of finding other eigenvalues. More specifically, we assume that the eigenvalues are:

$$\lambda_1 \geq \lambda_2 ... \geq \lambda_m > 0$$

We suppose that $\lambda_1$ and $u_1$ are known, and let $\overline{x}$ be the vector $\overline{x} = x_0 - \langle x_0, u_1 \rangle u_1$, which is orthogonal to $u_1$. Using the initial vector $x_1 = \overline{x}$, the sequence $x_{n+1} = \frac{Ax_n}{\|x_n\|}$ will be also orthogonal to $u_1$. Indeed, while $\overline{x}$ is orthogonal to $u_1$, by using the orthonormal basis (2) it results that:

$$\overline{x} = \overline{x}_2 u_2 + \overline{x}_3 u_3 + ... + \overline{x}_m u_m \ ,$$

and consequently $A\overline{x} = \lambda_2 \overline{x}_2 u_2 + \lambda_3 \overline{x}_3 u_3 + ... + \lambda_m \overline{x}_m u_m$ thus $\frac{A\overline{x}}{\|\overline{x}\|}$ is also orthogonal to $u_1$

Apparently, the algorithm consist into the initialization $x \leftarrow \overline{x}$ and the execution of the following block, until a suitable condition of convergence is fulfilled.

*Repeat*:

$$\{\ y \leftarrow \frac{Ax}{\|x\|}$$

$$x \leftarrow y\ \}$$

But, as in the previous, the rounding errors will introduce a perturbation such that the coefficient of $u_1$ will be nonzero. **There is no way to avoid rounding errors thus the algorithm will fail**. In fact, we will obtain the maximum eigenvalue $\lambda_1$ instead of $\lambda_2$! The right solution, in order to correct the errors, is to ortogonalize the vector at each step, i.e. to implement the repeatedly execution of the block:

$$\{\ y \leftarrow \frac{Ax}{\|x\|}$$

$$y \leftarrow y - \langle y, u_1 \rangle u_1$$

$$x \leftarrow y\ \}$$

Next, assuming that we determined the eigenvalues $\lambda_1$, $\lambda_2$, …, $\lambda_{k-1}$ and the eigenvectors $u_1$, $u_2$, …, $u_{k-1}$ it is possible to find the next eigenvalue and the next eigenvector by using the following algorithm:

$$\overline{x} = x_0 - \langle x_0, u_1 \rangle u_1 - \langle x_0, u_2 \rangle u_2 ... - \langle x_0, u_{k-1} \rangle u_{k-1}$$

*Repeat*:

$$\{\ y \leftarrow \frac{Ax}{\|x\|},$$

$$y \leftarrow y - \langle y, u_1 \rangle u_1 - \langle y, u_2 \rangle u_2 ... - \langle y, u_{k-1} \rangle u_{k-1} \qquad (5)$$

$$x \leftarrow y\ \}$$

Formula (5) is designed to restore orthogonality at every step, because the orthogonality may be lost due to rounding errors. Mathematically speaking, the formula (5) is not necessary because other assignments of algorithm does not change the orthogonality. But, if we do not use the formula (5), then each step will generate the maximum eigenvalue, because of rounding errors can lead to nonzero components, corresponding to $u_1$.

The C++ implementation will contain the following sequence:

```
// normalise the vector x
      norm=0;
      for(i=1;i<=m;i++)norm=norm+x[i]*x[i];
      norm=sqrt(norm);
      for(i=1;i<=m;i++)x[i]=x[i]/norm;
// orthogonalise the vector x, with respect to the eigenvectors
// allready found
// vec index reprezent the numer of eigenvectors allready found
// the matrix u[ ][ ] implements the colum reprezentation of
// the eigenvectors
      for(j=1;j<=m;j++){
            s=0;
            for(i=1;i<vec;i++)
                  {s1=0;
                  for(l=1;l<=m;l++)s1=s1+x[l]*u[i][l];
                  s=s+s1*u[i][j];}
            y[j]=x[j]-s;}
            for(i=1;i<=m;i++)x[i]=y[i];
// calculate the new iteration
      for(i=1;i<=m;i++){
            s=0;
            for(j=1;j<=m;j++)s=s+a[i][j]*x[j];
            y[i]=s;}
      for(i=1;i<=m;i++)x[i]=y[i];
```

**Although the first eigenvalue may not be found at the first step, however, the first step will determine one eigenvalue.** Next, every step will determine one eigenvalue and one eigenvector, orthogonal to all the previous. Finally we will get *m* vectors, so *m* eigenvalues, even if the order of determination will not be decreasing.

We used the matrix $A=\begin{pmatrix} 12 & -2 & 3 \\ -2 & 8 & 5 \\ 3 & 5 & 9 \end{pmatrix}$, implemented by `a[][]`; the sequence $x_n$ is implemented by `x[]`. The eigenvalues are $\lambda_1 = 14.2339683$, $\lambda_2 = 12.5490733$, $\lambda_3 = 2.2169583$ and the corresponding eigenvectors are:

$$u_1 = (-0.6138071 \quad -0.3646656 \quad -0.7001856)$$

$$u_2 = (0.7132033 \quad -0.6364342 \quad -0.2937558)$$

$$u_3 = (-0.3384994 \quad -0.6796841 \quad 0.6507285)$$

By using the starting vector `x[1]=0.3747039; x[2]=-1.3161183; x[3]=0.3569727;` we found $\lambda_2 = 12.5490733$. This result is obtained independent of the implementation style, I or II. But, the algorithm will bring all the eigenvalues, in order: $\lambda_2 = 12.5490733$, $\lambda_1 = 14.2339683$, $\lambda_3 = 2.2169583$.

### IV. The non symmetric case

This case may be analyzed in the same way ([1]) and we used the matrix $A=\begin{pmatrix} 12 & -3 & 12 \\ -3 & 18 & 5 \\ 4 & 5 & 9 \end{pmatrix}$, implemented by `a[][]`; and the sequence $x_n$ implemented by `x[]`. The eigenvalues are $\lambda_1 = 19.290752$, $\lambda_2 = 18.4173239$, $\lambda_3 = 1.2919241$ and the corresponding eigenvectors are:

$$u_1 = (0.6242935 \quad 0.5792752 \quad 0.5241162)$$

$$u_2 = (0.8022669 \quad 0.3142054 \quad 0.5075852)$$

$$u_3 = (0.7465349 \quad 0.3101901 \quad -0.5886151)$$

By using `x[1]=1.5488015; x[2]=0.6243958; x[3]=-0.081025;` which is orthogonal, with 7 digits, to $u_1$ we have found the first eigenvalue $\lambda_2 = 18.4173239$, instead of $\lambda_1 = 19.290752$.

## V. Bibliography


1. Arbenz, P., "Lecture Notes on Solving Large Scale Eigenvalue Problems", http://people.inf.ethz.ch/arbenz/ewp/Lnotes/, 2010
2. Bakhvalov, N., "Methodes Numeriques", Editions Mir, 1976
3. Stoer, J., Bulirsch, R., "Introduction to Numerical Analysis", Springer Verlag, 1992


ROMANIAN JOURNAL OF MATHEMATICS AND COMPUTER SCIENCE, 2011, VOLUME 1, p.29-36