Application of adjustment via *Mathematica* for deflection of the vertical determination

L. Völgyesi
Department of Geodesy and Surveying
Gy. Popper
Department of Structural Mechanics, Research Group for HAS-BUTE Computational Structural Mechanics
B. Paláncz
Department of Photogrammetry and Geoinformatics
Budapest University of Technology and Economics, H-1521 Budapest, Hungary, Müegyetem rkp. 3.

Abstract. In majority of real adjustment problems, the observation equations to be solved in least square sense, have more hundreds or thousands variables, the matrix of the system is sparse and often ill-conditioned. In this case study, the application of integrated, symbolic-numeric system, *Mathematica* to solve a practical adjustment problem for deflection of the vertical determination is demonstrated. Solving a real world adjustment problem, represented by more thousands variables and equations, with very high sparseness of the system matrix, we illustrate, how easy for practitioners to create and handle sparse matrices economically, compute condition numbers, pseudoinverse and carry out singular value decomposition and solve least square problems with *Mathematica* within seconds.

Keywords. Adjustment via *Mathematica*, sparse matrices, singular value decomposition, pseudoinverse solution, deflection of the vertical determination.

Introduction

It is necessary to treat large sparse matrices which may be ill conditioned depending on the geometry of interpolation network in the case of deflection of the vertical's determination based on torsion balance measurements Völgyesi (1993, 1995, 2001), Tóth, Völgyesi (2002).

However, MATLAB has also advanced numeric methods for sparse matrices, the authors select *Mathematica,* because of its functional programming ability and perfect interactivity make it ideal for engineers without professional programming knowledge. The interested readers can found a comparison of numeric performances of different mathematical programs for data analysis in Steinhaus (2002) as well as a very good introduction to *Mathematica* in Ruskeepaa (2004).

1 Mathematical background

Consider the system of linear algebraic equations

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{1}$$

where $\mathbf{A}$ is a real $m \times n$ matrix, $m > n$ and $\mathbf{b}$ is a real vector of m elements. If $m > n$, then the equation (1) is *overdetermined* and in general its solution does not exist. In similar cases it is advantageous generalize the concept of solution. The vector $\mathbf{x}$ of n elements, which minimizes the Euclidian norm of the error-vector $\mathbf{b} - \mathbf{A}\mathbf{x}$, i.e.

$$\|\mathbf{b} - \mathbf{A}\mathbf{x}\|_E \equiv \left\|(\mathbf{b} - \mathbf{A}\mathbf{x})^T (\mathbf{b} - \mathbf{A}\mathbf{x})\right\|^{1/2} = \min \tag{2}$$

is called *the least-squares solution of equation (1).* If the rank of $\mathbf{A}$ is smaller than n, then the minimizing problem (2) has not unique solution. The solution can be made unique if from among all solutions of problem (2) we select that vector $\mathbf{x}^*$ which Euclidian norm is minimal, i.e. $\|\mathbf{x}^*\|_E = \min$ holds.

This unique least-squares solution of equation (1) can be computed using the formula

$$\mathbf{x}^* = \mathbf{A}^+ \mathbf{b}$$

where $\mathbf{A}^+$ is the generalized- (or Moore-Penrose-) inverse of $\mathbf{A}$. In *Mathematica* $\mathbf{A}^+$ is implemented using the function *Pseudoinverse.*

The problem (2) is equivalent to solving so-called *normal equations*

$$\mathbf{A}^T \mathbf{A}\mathbf{x} = \mathbf{A}^T \mathbf{b}.$$

The solution of a least-squares problem directly from the normal equations is rather susceptible to roundoff error.

The most powerful method for computing the generalized- (or Moore-Penrose-) inverse and consequently for least-squares solution of equation (1) too, is based on the singular value decomposition or SVD of matrix **A**.

The SVD methods are based on the following theorem: any real *m x n* matrix **A**, $m \geq n$ can be decomposed as

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathbf{T}}$$

where $\mathbf{U}^{\mathbf{T}}\mathbf{U} = \mathbf{V}^{\mathbf{T}}\mathbf{V} = \mathbf{I}_{\mathbf{r}}$, $\boldsymbol{\Sigma} = \langle \sigma_1, ..., \sigma_r \rangle$ is diagonal matrix, $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r > 0$ and *r* is the rank of **A** .

The *r* columns of **U** are the orthonormal eigenvectors of $\mathbf{A}\mathbf{A}^{\mathbf{T}}$ corresponding to its *r* largest eigenvalues. The *r* columns of **V** are the orthonormal eigenvectors of $\mathbf{A}^{\mathbf{T}}\mathbf{A}$ corresponding to its *r* largest eigenvalues. The diagonal elements of $\boldsymbol{\Sigma}$ are square roots of the *r* largest eigenvalues of $\mathbf{A}^{\mathbf{T}}\mathbf{A}$ and are called *singular values* of **A**.

In *Mathematica* the SVD is implemented using the function *SingularValues*.

Using the SVD factorization, the $\mathbf{A}^{+}$ generalized- (or Moore-Penrose-) inverse of **A** can be easily computed as follows:

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathbf{T}},$$

$$\boldsymbol{\Sigma}^{-1} = \langle 1/\sigma_1, ..., 1/\sigma_r \rangle ,$$

$$\mathbf{A}^{+} = \mathbf{V}\boldsymbol{\Sigma}^{-1}\mathbf{U}^{\mathbf{T}} .$$

The linear least-squares problem becomes more strongly ill-conditioned as the spectral condition number

$$cond_S(\mathbf{A}) = \sigma_1 / \sigma_n \geq 1$$

increases.

2 Data preparation

Computations of deflections of the vertical were performed in a test area in Hungary extending over some 3000 km^2 where torsion balance measurements and astrogeodetic data are available.

A simple relationship, based on potential theory, can be written for the changes of $\Delta\xi_{ik}$ and $\Delta\eta_{ik}$ between arbitrary points *i* and *k* of the deflection of the vertical components ξ and η as well as for curvature values $W_\Delta = W_{yy} - W_{xx}$ and $2W_{xy}$ measured by a torsion balance (Völgyesi 1993, 1995):

$$\begin{aligned} &\Delta\xi_{ik}\sin\alpha_{ki} + \Delta\eta_{ik}\cos\alpha_{ki} = \\ &\xi_k \sin\alpha_{ki} + \eta_k \cos\alpha_{ki} - \xi_i \sin\alpha_{ki} - \eta_i \cos\alpha_{ki} = \\ &\quad \frac{s_{ik}}{4g}\left\{ \left[\left(W_\Delta - U_\Delta\right)_i + \left(W_\Delta - U_\Delta\right)_k \right] \sin 2\alpha_{ki} \right. \\ &\quad \left. + \left[\left(W_{xy} - U_{xy}\right)_i + \left(W_{xy} - U_{xy}\right)_k \right] 2\cos 2\alpha_{ki} \right\} \end{aligned} \quad (3)$$

where s_{ik} is the distance between points *i* and *k*, *g* is the average value of gravity between them, $U_\Delta = U_{yy} - U_{xx}$ and U_{xy} are curvature values of the normal (reference) gravity field, whereas α_{ki} is the azimuth between the two points.

The computation being fundamentally integration, practically possible only by approximation; in deriving (3) it had to be assumed that the change of gravity gradients between points *i* and *k*, measurable by torsion balance, is linear (Völgyesi 1993). This means Eq. (3) results a large sparse linear system.

In order to illustrate the solution methods supported by *Mathematica*, first we show how to create a spare matrix object and how to set up the equation system from the input data (Popper, 2003).

The matrix of the equation system can be stored in a file, element by element, in the following way:

row number	column number	value

Command reading data from a file into a list object is,

```
MatrixInput = ReadList["D:\Data\Matrix.dat",
              {Number,Number,Number}];
```

This list object can be converted into a sparse array object as,

```
Converter[{row_,column_,value_}]:=
              {row,column}→value
```

where *value* is the value of the matrix element assigned to the *row*-th row and *column*-th column. Converting `MatrixInput` list into a sparse array, we get

```
A=SparseArray[Map[Converter[#]&,MatrixInput]]
SparseArray[<8218>,{2068.1462}]
```

`SparseArray` with rules involving patterns uses cylindrical algebraic decomposition to find connected array components. Sparse arrays are stored

internally using compressed sparse row formats, generalized for tensors of arbitrary rank.

The echo of this command shows us, that **A** is a sparse matrix object with 2068 rows and 1462 columns. The number of the nonzero elements are 8218. These can be computed as,

```
NumberOf NonzeroElements =
        Length[Select[Flatten[A], # ≠ 0 &]]
NumberOf AllElements =
        Apply[Times, Dimensions[A]]
```

Consequently the sparseness of the matrix is,

```
Sparseness = 1. - NumberOf NonzeroElements/
        NumberOf AllElements
```

which is: 0.997282.

The right hand side vector of the system can be read similarly, from a different file,

```
RightHandSideVector =
  ReadList["D:\Data\Rside.dat",{Number}]//Flatten;
```

which is a list of 2068 elements.

3 Condition Number

Looking for a solution in sense of least squares, the pseudo matrix is,

```
PseudoMatrix = Traspose[A].A;
```

its condition number based on p-norm, $p = 2$, is

```
MatrixConditionNumber[PseudoMatrix, 2]
```

which is 2107.78. This means that the `PseudoMatrix` matrix is not an ill conditioned matrix, therefore pseudoinverse solution can be employed.

4 Displaying the structure

Matrix structure can be graphically displayed via the `MatrixPlot` functionality. The array is shown as a grid of black and white cells, by default representing zero-valued cells as white, and non-zero values as black.

`MatrixPlot` accept the usual range of options for a graphics function. In addition, it takes the `MaxMatrixSize` option, specifying a maximum display size to use for the array. By default, this is set to 512. If the matrix dimensions are larger than this size, the matrix is downsampled to this size or less so that the output graphic size can be controlled. In this case, a darkened cell indicates that at least one of the covered cells has a non-central value. The `MaxMatrixSize` may be set to `Infnity` to prevent downsampling before display,

```
<< LinearAlgebra`MatrixManipulation`
MatrixPlot[A, MatrixSize→Infinity];
```

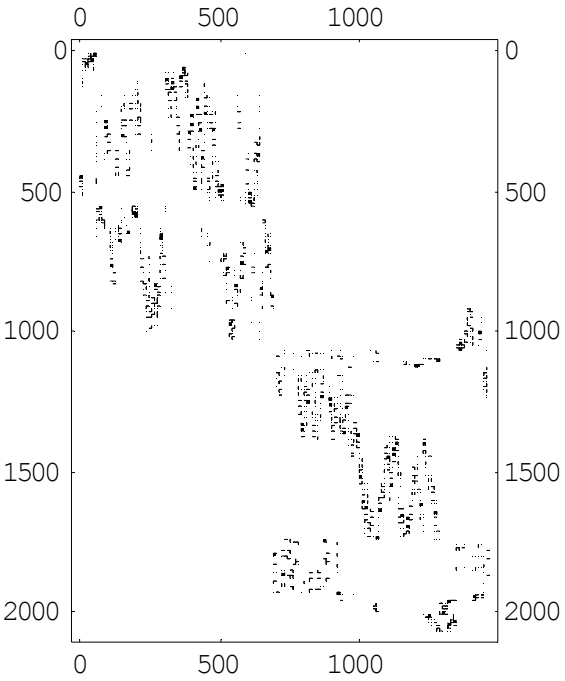

Fig. 1 The structure of the sparse matrix

5 LinearSolve solution

First we try the `LinearSolve` function works on both numerical and symbolic matrices, as well as `SparseArray` objects. For sparse arrays, `LinearSolve` uses UMFPACK multifrontal direct solver methods and with `Method->"Krylov"` uses Krylov iterative methods preconditioned by an incomplete LU factorization only for numeric matrices.

```
Timing[(LinearSolve[PseudoMatrix,
      Transpose[A].RightHandSideVektor]);]
{0. Second, Null}
```

or with the Krylov method

```
Timing[(LinearSolve[PseudoMatrix,
      Transpose[A].RightHandSideVektor,
      Method→Krylov]);]
{0.031 Second, Null}
```

The solution vector

```
x=LinearSolve[PseudoMatrix,
      Transpose[A].RightHandSideVektor]);]
ListPlot[x, PlotJoined→True, Frame→True,
      PlotRange→{-3,+3}, FrameLabel→
      {"Number of variables","Value of variable"}];
```

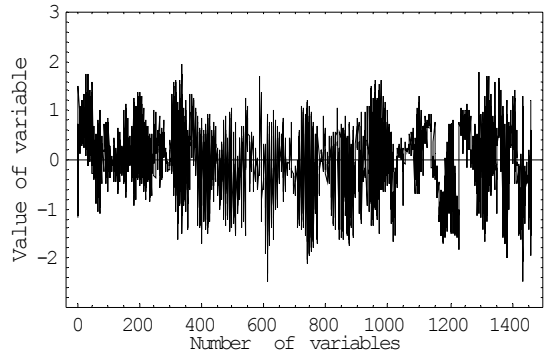

Fig. 2 The structure of the solution

We can check the quality of the pseudo solution

```
Norm[PseudoMatrix.x-
Transpose[A].RightHandSideVektor]
1.12688×10^-11
```

The residium of the solution is

```
Norm[A.x- RightHandSideVektor]
6.68605
```

The distribution of the residium among the equations

```
ListPlot[Map[Abs[#]&, (RightHandSideVektor-A.x],
      PlotJoined→True, PlotRange→All,
      Frame→True, FrameLabel→
      {"Number of equations","Value of
residium"}];
```

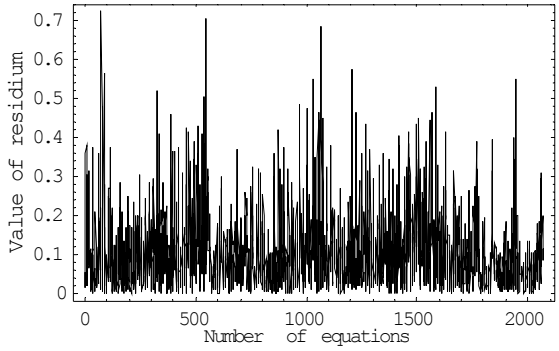

Fig. 3 Distribution of the residuum

6 SVD Solution

The `SingularValueDecomposition` function gives the singular value decomposition for a numerical matrix **A**. The result is a list of matrices $\{u, \sigma, v\}$, where σ is a diagonal matrix, and **A** can be written as $u \cdot \sigma \cdot$ `Conjugate [Transpose[`v`]]`.

`SingularValueDecomposition` uses the QR decomposition algorithm of Francis with Given's rotations.

```
Timing[SingularValues[A];]
{42.781 Second, Null}
```

which is greater with two magnitude than running time of `LinearSolve`.

```
{u,σ,v} = SingularValues[A];
```

The diagonal matrix is

```
Ξr = DiagonalMatrix[σ];
```

The $p = 2$ norm of a matrix is the largest principal axis of the ellipsoid, equal to the largest singular value of the matrix.

```
Norm[A] == Max[σ]
True
```

The spectral condition number gives a better estimation for the ill-conditioning, than condition number based on $p = 2$ norm

```
σ[[1]]/Last[σ]
45.9466
```

The matrices of u and v are

```
{Ur, Vr} = Map[Transpose, {u, v}];
```

The matrix **A** can be expressed with these matrices and with the diagonal matrix, consequently

```
Norm[A- Ur . Ξr .Transpose[Vr]]
1.98418×10^-14
```

The inverse matrix is

```
A+ = Vr . Inverse[Ξr] .Transpose[Ur];
```

To check round of errors we can express **A** with the inverse, too

```
Norm[A-A .A+.A]
1.80475×10^-14
```

The solution vector is

xx = A^{+}. RightHandSideVektor;

The difference between the first and second solution method is

Norm[x-xx]
9.7108×10^{-11}

7 Pseudoinverse solution

The third solution method computes the pseudoinverse of **A** directly, indeed

Norm[PseudoInverse[A] -A^{+}]
2.11709×10^{-14}

For numerical matrices, `PseudoInverse` is based on `SingularValueDecomposition`, therefore the time of this computation is very close to that of the SVD method,

Timing[(PseudoInverse[A] . RightHandSideVektor);]
`{43.563 Second, Null}`

which is greater again with two magnitude than running time of `LinearSolve`. The solution vector is

xxx = PseudoInverse[A] . RightHandSideVektor;

The total residium of the third solution is

Norm[A . xxx – RightHandSideVektor];
`6.68605`

Comparing the first and third solutions, the norm of the difference of the solution vectors is

Norm[x – xxx]
9.71054×10^{-11}

However, the execution time of SVD type methods are significantly greater than that of the `LinearSolve`, one can use them in case of ill conditioned problem, too.

8 Complexity Study

It is important to know, how the computation time increases with the increase of *n*. According to our computation carried out on PC Compaq Evo P4 2.8 MHz with *Mathematica* Version 5, the Table 1 shows the results of two characteristic runs where the computation time is in seconds.

Table 1. Comparison of methods in case of different problem sizes

Matrix **A**	Sparseness	Linear Solve	Pseudo Inverse	SVD
559×400	0.990	0.031	1.188	1.203
2068×1462	0.997	0.172	43.563	42.781

9 Interpolated deflections of vertical

Interpolated N-S (ξ) and E-W (η) components of deflections of the vertical that resulted from the computation visualized on isoline map in Figure 6 and 7. Isoline interval is 0.2”. The interpolation network has 738 torsion balance stations (marking by dots in figures) and 731 of these are points with unknown deflection of the vertical. Since there are two unknown components of deflection of the vertical at each point there are 1462 unknowns for which 2068 equations can be written. From these 738 torsion balance stations there were 11 astrogeodetic and astrogravimetric points where ξ, η values were known referring to the *GRS80* system. 7 astrogeodetic points were used as initial (fixed) points of interpolation and 4 points were used for checking of computations.

Standard deviations $m_{\xi} = \pm 0.60''$ and $m_{\eta} = \pm 0.65''$, computed at checkpoints confirm the fact that ξ, η values of acceptable accuracy can be computed from torsion balance measurements and *Mathematica* can be efficiently applied for solving this adjustment problem.

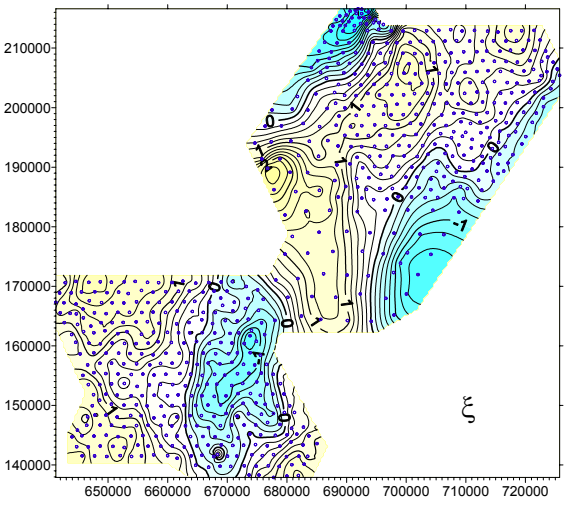

Fig. 6 Computed N-S (ξ) component of deflections of the vertical

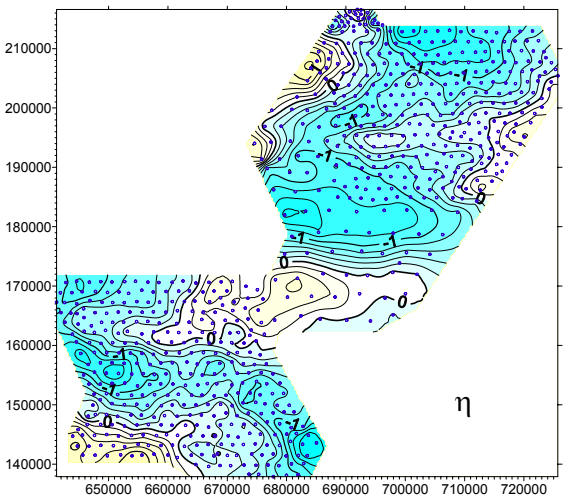

Fig. 7 Computed E-W (η) component of deflections of the vertical

10 Conclusions

`LinearSolve` based on UMFPACK multifrontal direct solver methods and with `Method->"Krylov"` uses Krylov iterative methods preconditioned by an incomplete LU factorization as well as the `PseudoInverse` uses SVD method, give the same result, however the running time of the latest is greater with about two magnitudes. In case of bad conditioned pseudo matrix, `PseudoInverse` function is recommended. Using a standard PC like we did, we get solution under realistic time (approx. 40 seconds) in case of a not bad conditioned system matrix, for n = 1462 variables. It goes without saying that these limits could be doubled with PC employing 64 bits processors.

Acknowledgements

We should thank for the funding of the above investigations to the National Scientific Research Fund (OTKA T-037929 and T-37880), and for the assistance provided by the Physical Geodesy and Geodynamic Research Group of the Hungarian Academy of Sciences.

References

Steinhaus, S. (2002) Comparison of mathematical programs for data analysis available: http://www.scientificweb.de/ncrunch/

Popper, G. (2003) Numerical methods with *Mathematica.* Müegyetemi Kiadó, Budapest. /in Hungarian/

Ruskeepaa, H. (2004) *Matematica* Navigator, Academic Press, 2nd edition

Tóth Gy, Völgyesi L. (2002) Comparison of interpolation and collocation techniques using torsion balance data. Reports on Geodesy, Warsaw University of Technology, Vol. 61, Nr.1, pp. 171-182.

Völgyesi L. (1993) Interpolation of Deflection of the Vertical Based on Gravity Gradients. Periodica Polytechnica Civ.Eng., Vo1. 37. Nr. 2, pp. 137-166.

Völgyesi L. (1995) Test Interpolation of Deflection of the Vertical in Hungary Based on Gravity Gradients. Periodica Polytechnica Civ.Eng., Vo1. 39, Nr. 1, pp. 37-75.

Völgyesi L. (2001) Geodetic applications of torsion balance measurements in Hungary. Reports on Geodesy, Warsaw University of Technology, Vol. 57, Nr. 2, pp. 203-212.

* * *

Völgyesi L, Popper Gy, Paláncz B (2004): Application of adjustment via *Mathematica* for deflection of the vertical determination. IAG International Symposium, Gravity, Geoid and Space Missions. Porto, Portugal August 30 - September 3, 2004.

Dr. Lajos VÖLGYESI, Department of Geodesy and Surveying, Budapest University of Technology and Economics, H-1521 Budapest, Hungary, Műegyetem rkp. 3.
Web: http://sci.fgt.bme.hu/volgyesi E-mail: volgyesi@eik.bme.hu