Software
V2

Neculai Andrei

Academy of Romanian Scientists,
Center for Advanced Modeling and Optimization,
Ilfov Street, No. 3, Bucharest 5, Romania,
E-mail: neculaiandrei1948@gmail.com

Technical Report
October 16, 2020

This Technical Report is the **second version** of the list of the software I have written and tested along the years. The software in this version of this paper is an improvement of the software I listed in the Technical Report from December 8, 2011, where I introduced some new software elaborated along the years. The programs and subroutines are organized on directories and subdirectories. In each directory I placed a number of programs I have elaborated in my research activity. Some other information concerning the algorithms or the results of the software is also included. The list of chapters and their contents is as follows:

	Support Programs	
1.	**ENORM**	Program for computing the Euclidian norm of a vector. (This is a variant of function enorm from MINPACK1)
2.	**PERFORM**	Program for Profile Performance Analysis. There are three subroutines: **PERF2N** for analysis of two algorithms (ALG1 versus ALG2) subject to the number of iterations, of function evaluations and CPU time in sense of Dolan and Morè. [See: Dolan, E.D., & Moré, J.J., (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming, 91*, 201-213.] **PERFNN** for analysis of at least 20 algorithms subject to the number of iterations, of function evaluations and CPU time in sense of Dolan and Morè. [See: Dolan, E.D., & Moré, J.J., (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming, 91*, 201-213.]

		PERLOG for performance analysis of two algorithms according to the index $$r_i = -\frac{\log(a^A)}{\log(a^B)},$$ where a^A and a^B referes to the number of iterations, or the number of functions evaluartions, or the CPU time respectively corresponding to the algorithm A or B. A complete description of these algorithms could be find in: **N. Andrei,** *Criticism of the Unconstrained Optimization Algorithms Reasoning,* Editura Academiei Române, Bucureşti, 2009. ISBN: 978-973-27-1669-4 (pp. 89-94) **December 30, 2004**
3.	**GRADIENT**	Approximation of derivatives of a function using forward or central finite difference. The step length is sqrt of epsilon machine. **April 1983**
4.	**JACOBIAN**	Program for computation of the sparse Jacobian with minimization of the number of function evaluations. The algorithm is described in the paper: **N. Andrei**, *RP - a package for efficient calculation of sparse jacobian matrix for nonlinear systems of equations using finite differences.* Technical Report, Bucharest, April 15, 1983. (Please see the file: JACOBIAN.DOC) **April 15, 1983**
5.	**LS**	A collection of subroutines for one-dimensional searching used in nonlinear optimization. The following subroutines belong to this collection: L1 Golden search L2 Fibonacci search L3 Quadratic interpolation of Powell technique L4 Dichotomous search **April, 1990**
6.	**LINE-SEARCH**	Line search: Backtracking versus Wolfe versus Moré – Thuente in context of Steepest Descent Method. The purpose of this program is to see 3 line search subroutines (backtracking, Wolfe and Strong Wolfe) in some particular Fortran implementation. ♦ Subroutine BACK (backtracking) is authored by Andrei. ♦ Subroutine WOLFE (standard Wolfe conditions) is coauthored by Shanno and Phua with some additional modifications by Andrei. ♦ Subroutine MTLINES (strong Wolfe conditions) is coauthored by Moré and Thuente. A description of these algorithms could be find in: **N. Andrei,** *Criticism of the Unconstrained Optimization Algorithms Reasoning,* Editura Academiei Române, Bucureşti, 2009. ISBN: 978-973-27-1669-4 (pp. 122-133)

		August 30, 2006
7.	**AORDER**	Subroutine for setting the elements of an array in ascending order. The elements of the array may be positive or negative. This subroutine is accompanied by an main program illustrating the calling of the subroutine. **Array to be ordered (increasingly)** **1 0.1000000000000E+02** **2 0.9000000000000E+01** **3 0.8000000000000E+01** **4 -0.6000000000000E+01** **5 0.6000000000000E+01** **6 0.5000000000000E+01** **7 0.4000000000000E+01** **8 0.3000000000000E+01** **9 0.2000000000000E+01** **10 0.1000000000000E+01** **Ordered array** **1 -0.6000000000000E+01** **2 0.1000000000000E+01** **3 0.2000000000000E+01** **4 0.3000000000000E+01** **5 0.4000000000000E+01** **6 0.5000000000000E+01** **7 0.6000000000000E+01** **8 0.8000000000000E+01** **9 0.9000000000000E+01** **10 0.1000000000000E+02** **August, 1995**

<><><><><><><><><><>

	Numerical Linear Algebra	
1.	**SIMEQ**	**<u>Directory SIMEQ</u>** **Simultaneous linear equations system solving** **SIMEQ** subroutine for solving linear algebraic systems of equations (the dense case). The matrix and its inverse are stored in (NxN) arrays. **SIMEQL** subroutine for solving linear algebrais systems of equations (the dense case). In this case the matrix and its inverse are stored in one-dimensional arrays on rows in natural order, i.e. [row1, row2, ... rowKC]. Both subroutines implements the Gauss method of ellimination with column pivoting to determine the solution vector and the inverse of the matrix. Calling sequence: CALL SIMEQ (A,RHS,KC,AINV,X,IERR) CALL SIMEQL(A,RHS,KC,AINV,X,IERR) where: A = The array of the system's matrix. RHS = The right-hand-side of the system. KC = The order of the system (The number of variables). AINV = The array of the inverse of the matrix. X = The solution vector of the system. IERR = Return code with the following values: 0 if the matrix is singular,

		1 if the matrix is non-singular. **January 21, 1995**
2.	**BT**	**<u>Directory BT</u>** **Program for Block Triangularization of a large-scale sparse matrix.** The following subroutines are invokated: RP02A, RP02B, RP02D, RP02C, RP02E. **RP02A** subroutine, for a given large-scale sparse matrix A, row packed, if it is possible; permute its rows and columns to the lower-block-triangular form. The matrix is permuted to the form PAQ, according to P and Q permutation matrices, so that the non-zero elements in the off-diagonal blocks preceed those in the diagonal blocks, which are in order. If the user introduce his matrix by columns, then RP02A subroutine will produce the upper-block-triangular form and the corresponding permutation matrices. **RP02B** subroutine, for a given pattern of non-zeros of a sparse matrix, finds a row permutation that makes the matrix have a maximum number of non-zero elements on its diagonal, using a depth first search with look ahead technique. **RP02C** This subroutine, for a given pattern of non-zeros of a sparse matrix, finds a symmetric permutation that makes the matrix lower block triangular, using the Tarjan's depth first search algorithm. To obtain the best results, the user must first permute the structure of the matrix so that it has a zero-free diagonal. This can be done using RP02B subroutine. **November 2, 1992**
3.	**RF**	**<u>Directory RF</u>** **Program for solving large-scale linear algebraic systems** A*X=B or AT*X=B **taking into account the sparsity of the A matrix. (AT is the transpose of the matrix A)** This is a package of subroutines dedicated to compute the Product Form of Inverse (PFI) with a preassigned pivot procedure, to solve corresponding large-scale systems of linear equations, exploiting sparsity in all cases. The calling sequence is as follows: MAIN ---- RF02A RF02C ---- MRM03A MFTRAN ---- RM02A MBTRAN ---- RM02A where: RF02A determine a permutation of rows and columns of the matrix to the "*bump and spike*" structure. RF02C compute the Product Form of Inverse of the matrix permuted to this form "bump and spike" MRM03A compress the arrays **a** and **indl** in order to create more room for PFI generation. MFTRAN is for forward transformation of the RHS term of the system to compute the solution of A*X = B. X = A**(-1)*B = QP*(Tn*.....(T1*B)). MBTRAN is for backward transformation of the RHS term of the system to compute the solution of AT*X = B.

		X = A**(-T)*B = B*(((QP)*Tn)*.....T1). RM02A is for insitu permutation of a vector according to a permutation vector. The theory of Product Form of Inverse (PFI) factorization is described in: **N. Andrei,** *Criticism of the linear programming algorithms reasoning.* Romanian Academy Publishing House, Bucharest, 2011. ISBN: 978-973-27-2076-9 (pages: 1239-273) **December 4, 1995**
4.	**LU**	**<u>Directory LU</u>** **Package for Solving Large-Scale Linear Algebraic Systems with LU Factorization and Markowitz Procedure for Pivots Selection** The calling sequence is as follows: LU ----- RF01A --- RM01A, --- RM04A. ----- RS01A where: LU Main program for solving linear algebraic systems of equations Ax=b or ATx=b using LU factorization of the matrix with Markowitz's pivot selection strategy. RF01A Subroutine for LU factorization of the matrix A. RS01A Subroutine for solving large-scale systems of linear equations computing: A**(-1)*b or A**(-T)*b, for a given vector b, using the LU factorization of the matrix A (given by RF01A subroutine), exploiting the sparsity in all cases. RM01A Subroutine for sorting the non-zeros of a sparse matrix from arbitrary order to column order, but unordered within each column. RM04A Subroutine for compressing the column / row array of U factor (from the LU factorization of a sparse matrix) in order to eliminate the spaces between columns / rows. ♦ The file **LU.DOC** contains a number of 15 examples of linear algebraic systems solved by LU package. The theory of LU factorization is described in: **N. Andrei,** *Criticism of the linear programming algorithms reasoning.* Romanian Academy Publishing House, Bucharest, 2011. ISBN: 978-973-27-2076-9 (pages: 287-316) **May 3, 1995**

<><><><><><><><><><>

	Zero of nonlinear functions	
1.	**ZERO1**	In sub-directory ZERO, directory: SOFT-ANDREI-TOTAL: A simple algorithm for computing zero of a nonlinear function of a variable in a given interval [a,b].

The Fortran program and 5 examples are presented in file **zero1.doc** (see directory: LUCRARI-MANUSTRISE-ANDREI (total).

For function: $f(x) = x^2 - 4,$ one zero in interval [0,3] is:

```
  a= 0.0000000000000E+00      fa=-0.4000000000000E+01
  b= 0.3000000000000E+01      fb= 0.5000000000000E+01
 Solution:  0.2000000000015E+01    Function value:  0.5820766091347E-10
 Number of iterations;      36
```

March 28, 1975

2. **ZERO2**

Fortran program in subdirectory ZERO of the directory SOFT-ANDREI-TOTAL.
A simple algorithm for computing all the zeros of a nonlinear function of a variable in a given interval [a,b]. b>a. The program include 14 examples of nonlinear functions for which the zeros (roots) are presented.

For example, the function: $f(x) = x^4 - 5x^2 + 4$ in interval $[-3,+3]$, has four zeros:

```
 Zero # 1
 ZERO: -0.2000000000001E+01    Function value:  0.6977529665164E-11
 Number of iterations:                52
 Number of function evaluations:      53

 Zero # 2
 ZERO: -0.1000000000003E+01    Function value: -0.1745270594711E-10
 Number of iterations:                50
 Number of function evaluations:      51

 Zero # 3
 ZERO:  0.9999999999994E+00    Function value:  0.3477662602336E-11
 Number of iterations:                66
 Number of function evaluations:      67

 Zero # 4
 ZERO:  0.1999999999997E+01    Function value: -0.3488764832582E-10
 Number of iterations:                50
 Number of function evaluations:      51

 ========================================
 TOTAL # OF iterations:               218
 TOTAL # of function evaluations:     222
```

The graphical representation of this function is given in Figure 1. Observe it has exactly four zeros (roots).

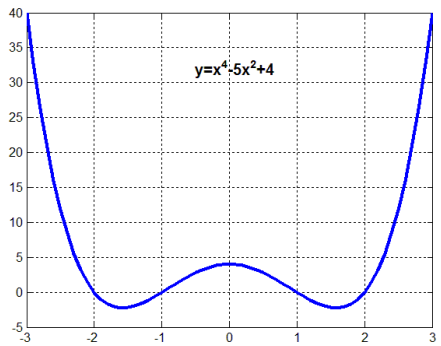


Fig. 1. Function $f(x) = x^4 - 5x^2 + 4$

April 16, 1975

<><><><><><><><><><>

	Solving Nonlinear Algebraic Systems F(x) = 0	
1.	**NEWTON**	Newton method without line search. The method is described in: **N. Andrei**, *Criticism of the Unconstrained Optimization Algorithms Reasoning*, Academiei Publishing House, Bucharest, 2009, Chapter 6, (pages: 243-255). The Newton system is solved by means of the LA05** package by J.K. Read with minor modifications by N. Andrei. In directory NEWTON there are the following nonlinear algebraic systems: 1) CANAL – Flow in a chanel problem, 2) CAVITATE – Flow in a driven cavity problem, 3) CIRCUIT – Circuit design problem, 4) E1 – Calculul temperaturii stationare într-un reactor. 5) E2 – Calculul fractiei de conversie a unei substante într-un reactor Chimic, 6) PROPAN - Propan combustion in aer - Reduced Formulation, 7) REACTOR - Stationar solution of a chemical reactor, 8) ROBOT - Robot kinematics problem, 9) SOLID - Solid Fuel Ignition. **June 1, 2006**
2.	**GRFLOW**	Gradient Flow Algorithm for solving nonlinear algebraic systems $F(x)=0$, where $F(x)=[f_1(x),\ldots,f_m(x)]$. The algorithm is as follows: $$x_{k+1}=x_k+d_k,$$ where d_k is computed as solution ofthe following system of linear algebraic equations: $$\left[I+h_k\theta\left(\nabla F(x_k)^T\nabla F(x_k)+\sum_{i=1}^{m}f_i(x_k)\nabla^2 f_i(x_k)\right)\right]d_k=-h_k\nabla F(x_k)^T F(x_k$$ If $f_i(x)$ are convex and positive for all $i=1,\ldots,m$; $rank(\nabla F(x))=n$; $\theta=1$ and $h_k\to\infty$, then the algorithm is quadratically convergent to a local solution of the system. FLOW.FOR is the main program for the Gradient Flow Algorithm for solving F(x) = 0. Some variants of this Fortran program can be found in FLOWC.FOR and FLOWZ.FOR. In directory ACAD there are the following problems (please see MINPACK-2 collection): GFLOWS1 – Circuit design problem,

		GFLOWS2 – Propan combustion problem, GFLOWS3 – Stationar solution of a chemical reactor, GFLOWS4 – Robot kinematics problem, GFLOWS5 – Solid fuel ignition problem, GFLOWS6 – Flow in a driven cavity problem, GFLOWS7 – Flow in a chanel problem, GFLOWS8 – Human heart dipole problem. The mathematical description of these problems, as well as their solution can be studied in: **N. Andrei**, *Criticism of the Unconstrained Optimization Algorithms Reasoning*, Academiei Publishing House, Bucharest, 2009, Chapter 6, (pages: 243-255). The theoretical aspects of the gradient flow method are presented in: **N. Andrei**, *Criticism of the Unconstrained Optimization Algorithms Reasoning*, Academiei Publishing House, Bucharest, 2009, Chapter 15. The performances of the gradient flow method are shown in chapter 15 of this monography, (pages: 675-679). Please, see the paper: **Gradient Flow Algorithm for Systems of Nonlinear Equations,** (ZEROF.PDF) (ZEROS.RTF). **April 4, 2004**
3.	**LEV-MARQ**	Levenberg-Marquardt Algorithm for solving nonlinear algebraic systems $F(x)=0$. This directory contains the following problems (please see MINPACK-2 collection): LMS1 – Circuit design problem, LMS2 – Propan combustion problem, LM3 – Stationar solution of a chemical reactor, LMS4 – Robot kinematics problem, LMS5 – Solid fuel ignition problem, LMS6 – Flow in a driven cavity problem, LMS7 – Flow in a chanel problem, A description of this algorithm together with the applications are presented in: **N. Andrei**, *Criticism of the Unconstrained Optimization Algorithms Reasoning*, Academiei Publishing House, Bucharest, 2009, Chapter 14.3 (pages: 636-655) **November 12, 2004** **January 10, 2007 (modified)**

<><><><><><><><><><>

	Linear programming

1.	**AFFINE-SCALING**	Main program **Affine Scaling with Rows Partitioning** for solving Linear Programming Problems: $$\text{Min } c^T x \text{ subject to } Ax \le b,\ x \ge 0.$$ The program implements an algorithm described into the book: **N. Andrei,** *Programarea Matematică - Metode de punct interior*, Editura Tehnică, 1999, chapter 5, section 5.4. (pages: 125-157) The program uses two dimensional arrays and doesn't takes the advantage of sparsity of the matrix A. **May 21, 1998**
2.	**SPLIT**	**Splitting the dense columns of a Linear Programming Problem**. Please see: Chapter 15 of the book: **N. Andrei,** "*Criticism of the Linear Programming Algorithms Reasoning*". Romanian Academy Publishing - Bucharest, Romania. 2010. (pages: 605-612) **June 9, 2010**
3.	**BCR**	**Balance Rows Reduction in linear programming.** The idea of this program is to eliminate the balance constraints, i.e. the constraints with zero RHS term. The package has two main components. The first one eliminate the balance constraints and solve the reduced problem. The second one recover the solution from the solution of the reduced problem. Please see: Chapter 16, Section 2, pp. 591-605 of the book: **N. Andrei,** *Criticism of the Linear Programming Algorithms Reasoning.* Romanian Academy Publishing – Bucharest, 2011. (pages: 591-612) **January 21, 1993**
4.	**ISLO**	Interactive System for maintaing Linear Programming Problems. This is an interactive package for solving linear programming problems using PFI of the basis having the possibility to establish the optrimization conditions at the very beginning of the process. **January 12, 1995**
5.	**ASLO**	Advanced System for Linear Optimization. The LU factorization of the basis (subroutines LA05AD, LA05BD, LA05CD, LA05ED and MC20AD) is used to implement the primal simplex method. The input of the problem is in MPS format. **December 16, 1992**
6.	**ASLONEW**	Advanced System for Linear Optimization – New version.
7.	**CALP**	A collection of Linear Programming Applications in ALLO Language. Se prezintă 10 prototupuri de modele de programare linară în limbajul ALLO, direct utilizate în context industrial. See: **N. Andrei,** *O colecţie de aplicaţii de programare liniară în limbajul ALLO.* Technical Report No.4/2007, September 3, 2007. (88 pagini cu CD) (see files: FRONT-RT3-2007.DOC & RT3-2007.DOC in directory CALP) Please, see the directory CALP in LINEAR-PROGRAMMING. Please,

		see the Technical Report RT3-2007.doc. The mathematical models in ALLO language are placed in the directory MODELS. **September 3, 2007**

<><><><><><><><><><>

		SAMO Technology **Advanced System for Linear Optimization**
1.	**ALLO**	**<u>Directory ALLO_LANGUAGE</u>** **ALLO is a mathematical programming language having the ability to write <u>linear programming models</u>.** It is fully algebraic, being notable for its syntax which is very simple but comprehensive, with a small number of reserved keywords. The compiler associated to ALLO language translate the model into the equivalent MPS form, directly admited by any professional linear optimizer. The directory ALLO_LANGUAGE contains the following subdirectories: ♦ **ALLO-source**: - Subdirectory **ALLO** contains three executable programs: ALLOPAS1 .EXE and ALLOPAS2.EXE for compiling an ALLO textsource and DSOLVER.EXE for solving the generated linear programming problem. - Subdirectories **PAS1** and **PAS2** contains Fortran and C programs which implement the **ALLO** compiler. The list of subroutines is described in **LIST-OF-SUBROUTINES.DOC** file**.** **-** Subdirectory **MODELE** contains 11 prototypes of linear programming models in ALLO language. -ALLO language is also described in **ALLO.RTF** file (17 pages, September 10, 2004) ♦ **APPLIC:** includes 25 prototypes of linear programming models in ALLO language. ♦ **DOC**: contains some files describing the ALLO language. - ALLO language is explained in **LANGALLO.RTF** file (english version, 16 pages, September 10, 2004) - The documents **ALLO1.DOC** and **ALLO2.DOC** present a collection of linear programming models including: allocation models, assembling models, desassembling, desassembling-assembling, transportation models, diet models, production modelswith optimal selection of technologies, systems with stocks, combined models of production with stocks, discrete production models. ♦ **EXE**: contains an old version of ALLO compiler (executables), SAMO executable, DSOLVER for solving a linear programming problem and HOPD which is HOPDM package by Gondzio. The language ALLO is described in a number of papers as:

		- **N. Andrei, Gh. Borcan**, *ALLO: Algebraic Language for Linear Optimization.* Technical Report, LSSO-2-95, Research Institute for Informatics, Bucharest, September 1995. - **N. Andrei, Gh. Borcan,** *ALLO – Limbaj algebric pentru optimizare liniară.* Revista Română de Informatică şi Automatică, vol.8, nr. 3, 1998, pp.55-67. - **N. Andrei,** *Pachete de Programe, Modele şi Probleme de Test pentru Programarea Matematică*, Editura MATRIXROM, Bucureşti, 2001. - **N. Andrei,** *Critica Raţiunii Algoritmilor de Programare Liniară,* Editura Academiei Române, Bucureşti, 2011. (pages: 815-830) **Martie 8, 2007**
2.	**SAMO**	**<u>Directory INSTAL_SAMO</u>** **Advanced informatic technology for linear programming modeling and optimization.** **SAMO** – is an advanced informatic technology for linear programming modeling and optimization at an industrial level. **SAMO** permits conceptualization, elaboration, maintainance, modification and solving large-scale linear programming models. The system **SAMO** is based of the language ALLO which is a dialect of the natural language used by the user to conceptualize, to build up, to modify linear programming models in algebraic format, as well as on the ALLO compiler which translate the algebraic format of the model in MPS format. **SAMO** uses a professional optimizer able to solve large-scale linear programming problems. **SAMO** allows generation and solving of linear programming prototypes. **SAMO is described in:** **TR18.DOC** file**:** SAMO - tehnologie informatică avansată pentru modelare şi optimizare. (Advanced informatioc technology for modeling and optimization.) Description of SAMO. Illustration of an ecran capture of this technology. (Martie 8, 2007) **SAMO.MSI** is windows installer package. To install SAMO, the serial number is: **CB48T H668K C9W64** A description of SAMO, as well as some prototypes of industrial models in ALLO language, working under SAMO technology are presented in: **N. Andrei,** *Critica Raţiunii Algoritmilor de Programare Liniară,* Editura Academiei Române, Bucureşti, 2011. (pages: 642-752) **N. Andrei,** *Pachete de Programe, Modele şi Probleme de Test pentru Programarea Matematică*, Editura MATRIXROM, Bucureşti, 2001. (Lucrarea conţine 13 prototipuri de modele de programare liniară exprimate în limbajul ALLO.) **Martie 8, 2007** **New Version: June 8, 2011**

<><><><><><><><><><>

	Unconstrained optimization	
Direct Search Methods		
1.	**UNO**	**<u>Directory UNO</u>** **UNCONSTRAINT OPTIMIZATION METHODS** **using** **DIRECT SEARCHING TECHNIQUES** The following techniques are implemented: - Hook-Jeeves - form searching, (HOOKJ.FOR - Rosenbrook - rotation of coordinates, (ROSE.FOR) - Powell - conjugate directions, (POWEL.FOR) - Nelder-Mead - Simplex, (NELMED.FOR) - Parallel with Axes Searching. (CPA.FOR) These methods are implemented with different onedimensional optimization methods like: L1 - golden section, L2 – Fibonacci search, L3 - Quadratic fitting of Powell, L4 - Simple lambda =1.0. The theory behind all these direct search methods is presented in: **N. Andrei,** *Criticism of the Unconstrained Optimization Algorithms Reasoning,* Academy Publishing House, Bucharest, 2009, Chapter 16. **April 1991** **New version: August 8, 2007**
2.	**FIBO**	**<u>Directory FIBONACCI</u>** **O subrutină de calcul a minimului unei funcţii neliniare de o variabilă, pe un interval dat, bazată pe metoda de căutare directă Fibonacci.** The Fibonacci search method is presented in: fibonacci.doc file. **Februarie 4, 1980**
3.	**MAXFUN**	**<u>Directory MAXFUN</u>** **O subrutină de calcul a maximului unei funcţii neliniare de o variabilă, bazată pe metoda de interpolare pătratică Powell.** The maxfun search method is described in: maxfun.doc file. **Septembrie 23, 1980**
4.	**PSO-UO**	**<u>Directory PSO-UO</u>** **Particle Swarm Optimization for unconstrained optimization** In this directory I included three Fortran programs for minimizing the Rosenbrock function (extended and generalized) and the Wood function using the particle swarm optimization method. **May 21, 2014**
5.	**DEEPS**	**<u>Directory DEEPS-TOTAL</u>** This directory contains a number of 13 sub-directories as follows: **<u>DEEPS1</u>**

		A simple deep random search method for unconstrained optimization. Preliminary computational results This sub-directory includes: a) Thechnical Report No. 1/2020: R2020T1.DOC describing the method, b) The Fortran package DEEPS.FOR which implements the algorithm and c) The file FUNCNAME.TXT with the name of the minimizing functions. DEEPS implements an algorithm based on direct search without using derivatives. The numerical experiments include 115 unconstrained optimization problems. See: **N. Andrei**, *A simple deep random search method for unconstrained optimization. Preliminary computational results*. Technical Report No. 1/2020, February 29, 2020, Bucharest. **<u>DEEPS2</u>** **Comparison of DEEPS algorithm using a simple deep random search method versus Steepest Descent for solving an unconstrained optimization problem with a narrow positive cone.** This sub-directory includes the Technical Report No. 2/2020: R2020T2.DOC describing the method. **<u>DEEPS3</u>** **Influence of local bounds "lobndc" and "upbndc" defining the size of the local domains on performances of the DEEPS algorithm.** This sub-directory includes: a) Technical Report No.3/2020: R2020T3.DOC describing the influence of bounds b) The Fortran package DEEPS2.FOR which implements the algorithm and c) The file FUNC115.TXT with the name of the minimizing functions. **<u>DEEPS4</u>** **Performances of DEEPS algorithm for solving large-scale unconstrained optimization problems.** The sub-directory includes: a) Technical Report No.4/2020: R2020T4.DOC describing the performances of DEEPS for solving large-scale problems b) The Fortran package DEEPS4.FOR which implements the algorithm and c) The file FUNC115.TXT with the name of the minimizing functions. The package DEEPS4.FOR can solve large-scale minimization problems up to 500 variables. For example for solving the problem DIXMAANA (CUTE) with 500 variables, DEEPS4 gives a local optimal solution in 1685 iterations, 94162 evaluations of the minimizing function and 27.32 seconds. **<u>DEEPS5</u>** **Performances of DEEPS for solving 16 real applications of unconstrained optimization.** This sub-directory contains: a) Technical Report No.5/2020: R2020T5.DOC, b) The Fortran package DEEPS5.FOR which implements the algorithm and c) The file FUNC16.TXT with the name of the minimizing functions included in this numerical experiments. **<u>DEEPS6</u>**

		A new simple deep random search method for unconstrained optimization This subdirectory contains: a) Technical Report No.6/2020: R2020T6.DOC b) The Fortran package DEEPS6.FOR which implements the algorithm and c) The file FUNCNAME.TXT with the name of the minimizing functions included in this numerical experiments. **<u>DEEPS7</u>** **Solution of Human Heart Dipole unconstrained optimization problem by means of DEESP6 and Nelder-Mead methods** This subdirectory contains: a) Technical Report No.7/2020: R2020T7.DOC b) The Fortran package DEEPS7.FOR which implements the algorithm and c) The file FUNCNAME.TXT with the name of the minimizing functions included in this numerical experiments. **<u>DEEPS8</u>** **A simple deep random search method for unconstrained optimization** This subdirectory contains: a) Technical Report No.8/2020: R2020T8.DOC b) The Fortran packages DEEPS2.FOR (for distance among the trial points), DEEPS4.FOR (for large-scale optimization), DEEPS5.FOR (with 16 applications) which implements the algorithm and c) The file FUNC116.TXT and FUNC16.TXT with the name of the minimizing functions included in these numerical experiments. **<u>DEEPS9</u>** **A two level random search method for unconstrained optimization** This subdirectory contains: a) Subdirectory APRIL: - PAS (steepest descent method) - REZ (rezults for solving 16 applications) - DEEP8L.FOR (for solving large-scale problems – Final) - DEEPS8A.FOR (for solving 16 applications – Final) - DEEPS8Z.FOR (For computing the maximum distance – Final) - FUNC16.TXT (with name of the applications - Final) - FUNC115.TXT (with name of the 115 problems - Final) b) Subdirectory NELMEAD (Nelder – Mead method) with: - FUNCNEL.TXT (with name of the functions) - NELMEAD.FOR (Fortran code for Nelder-Mead method in implementation of R, Oneill and modified by John Burkardt) c) Technical Report No. 9/2020: R2020T9 (April 19, 2020) DEEP8L.FOR is taylored for solving large-scale unconstarined optimization problems up to 500 variables. The numerical experiments include solving the problems: VARDIM, EG3, DIXMAANA, Broyden Tridiagonal, Broyden Pentadiagonal and DESSCHNF. DEEPS8A.FOR is designed for solving 16 applications of unconstrained optimization: Weber(1), Weber(2), Weber(3), Enzyme reaction, Solution of a chemical reactor, Robot kinematics problem, Solar Spectroscopy, Estimation of parameters, Propan combustion in air, Gear train with

minimum inertia, Human Heart Dipole, Neurophysiology, Combustion application, Circuit design, Termistor and Optimal design of a Gear Train. The performances of DEEPS8A is presented in Table 5 below.

Table 5. Performances of DEEPS for solving 16 unconstrained optimization applications

Nr.	n	N	M	*iter*	*nfunc*	*cpu*	$f(x^*)$	$f(x_0)$
1.	2	2	5	118	1473	0.01	-264.453135	-37.473137
2.	2	3	5	67	1318	0.0	9.56074405	78.594324
3.	2	5	10	167	9160	0.0	8.74984910	78.602864
4.	4	30	50	28	42903	0.02	0.308765E-03	0.531317E-02
5.	6	50	100	1375	6989983	2.76	0.747292E-04	0.196173E+08
6.	8	5	10	43	2459	0.0	0.182801E-07	5.334258
7.	4	5	10	4	233	0.0	8.3163822	9.958700
8.	4	100	500	54	2705727	1.73	0.3185759-01	2.905300
9.	5	9900	100	10	10013014	4.83	0.2467602E-04	0.331226E+08
10.	2	5	3	15	342	0.0	1.7441520	2563.3250
11.	8	50	300	2404	36265585	26.67	0. 996608E-04	0.190569
12.	6	1000	6	1044	7999853	3.87	0.854501E-04	23.917600
13.	10	3	10	25	856	0.0	0.406198E-08	121.998899
14.	9	50	50	3731	9654682	23.62	0.103618E-03	2964.578187
15.	3	100	500	15	752179	4.73	175.565438	0.233591E+10
16.	4	10	50	6	3090	0.0	0.3886716E-13	0.737081E-03
Total				9106	74442856	68.24		

NELMEAD.FOR is designed to solve the above 16 applications by Nelder-Mead method in implementation of R. Oneill and modified by John Burkard and Neculai Andrei. Table 16 below shows the performances of Nelder-Mead method.

Table 6. Performances of Nelder-Mead for solving 16 applications (FORTRAN77 version by R. ONeill [73], modifications by John Burkardt)

Nr.	*n*	*iter*	*nfunc*	*cpu*	$f(x^*)$	$f(x_0)$
1.	2	42611	141379	0.03	-264.45314	-37.473137
2.	2	5749	17111	0.01	9.560739	78.594324
3.	2	485	1515	0	8.749843	78.602864
4.	4	50589	257709	0.10	0.307599E-03	0.531317E-02
5.	6	4853773	40285964	9.37	0.472465E-05	0.196173E+08
6.	8	18824	131208	0.05	0.682946E-08	5.334258
7.	4	3553	19323	0.05	6.872370	9.958700
8.	4	1420090	7962328	3.61	0.318572E-01	2.905300
9.	5	218408	1304415	0.34	0.738972E-05	0.331226E+08
10.	2	50	151	0	1.744152	2563.3250
11.	8	12179330	116416638	39.66	0.109955E-03	0.190569
12.	6	749358	5880897	1.29	0.166672E-06	23.917600
13.	10	180224	1478094	0.53	0.606737E-07	121.998899
14.	9	995980	9939092	18.61	0.853510E-03	2964.578187
15.	3	3063875	10945869	70.68	175.091316	0.233591E+10
16.	4	8026086	36117397	5.58	0.751550E-12	0.737081E-03
Total		31808985	230899090	149.94		

The Technical Report R2020T9.DOC contains an Appendix with the mathematical expression of the applications considered in these numerical experiments.

<u>DEEPS10</u>

A two level random search method for solving the Elastic Plastic Torsion from MINPACK2

This directory contains:

A1EPT.FOR – Fortran program for solving the Elastic Plastic Torsion application from MINPACK2 with 2500 variables

R2020T10.DOC – Technical Report with results of optimization by DEEPS.

Figures 1 and 2 show the solution of this application with 2500 variables

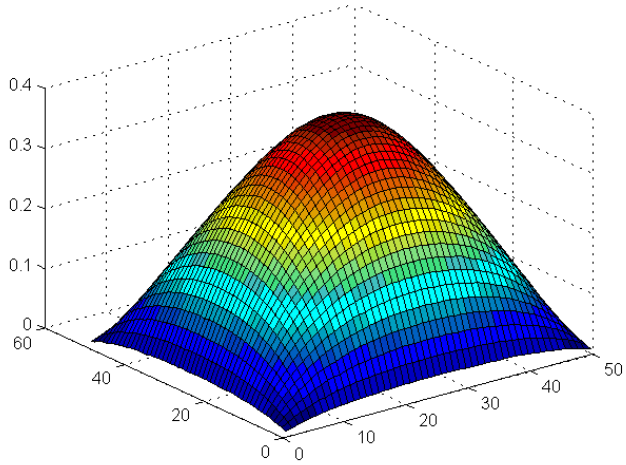


Fig. 1. Solution of Elastic Plastic Torsion application. *nx=50*, *ny=50*. Surface

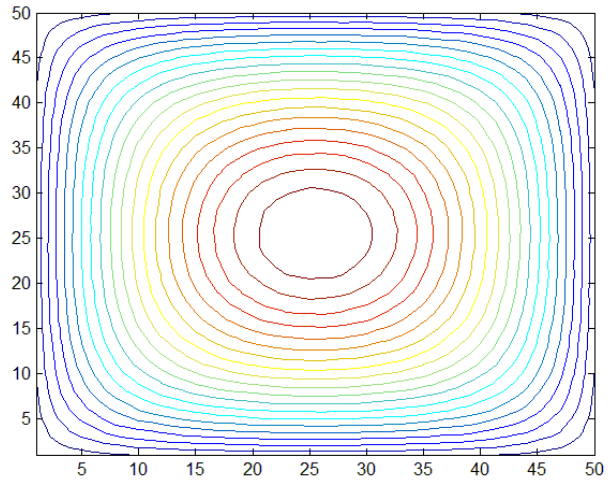


Fig. 2. Solution of Elastic Plastic Torsion application. *nx=50*, *ny=50*. Contour.

<u>DEEPS11</u>

Performances of DEEPS for solving 120 Unconstrained Optimization Problems

This directory contains:

- DEEPSPR.FOR
- DEEPS.OUT
- DEEPS.REZ
- FUNC120.TXT
- R2020T11.DOC

For solving 120 problems (16 applications and 104 test problems) with the number of variables in the range [2-40] the following results was obtained by DEEPSPR.

Table 1. Performances of DEEPS

2	118	1473	0	-0.2644531350428E+03	1. Weber Function (1) (Andrei, U71)
2	67	1318	0	0.9560744054913E+01	2. Weber Function (2) (Kelly, pp. 119)
2	167	9160	0	0.8749849108484E+01	3. Weber Function (3) (Kelly, pp. 119)
4	28	42903	2	0.3087657632221E-03	4. Enzyme reaction (Andrei, U79) (A)
6	1375	6989983	270	0.7472925581554E-04	5. Solution of a chemical reactor (A)
8	43	2459	1	0.1828017526180E-07	6. Robot kinematics problem (A)
4	4	233	0	0.8316382216967E+01	7. Solar Spectroscopy (A)
4	54	2705727	127	0.3185724691657E-01	8. Estimation of parameters (A)

5	10	10013013	369	0.2467602087429E-04	9.	Propan combustion in air (A)
2	15	342	0	0.1744152005590E+01	10.	Gear train with minimum inertia (A)
8	2404	36265585	2527	0.9966084682095E-04	11.	Human Heart Dipole. Andrei U84,
6	1044	7999853	380	0.8545018926146E-04	12.	Neurophysiology (A)
10	25	856	0	0.4061987800161E-08	13.	Combustion application (A)
9	3731	9654682	3297	0.1036184837525E-03	14.	Circuit design (A)
3	15	752179	844	0.1742216236340E+03	15.	Thermistor (A)
4	6	3090	0	0.3886716443010E-13	16.	Optimal design of a Gear Train (A)
2	50	5803	0	0.9835611823309E-10	17.	Rosenbrock - Valley of Banana
2	101	101022	2	0.4898425367948E+02	18.	Freudenstein-Roth
10	123	6261804	362	0.8062621526502E-07	19.	White & Holst Function
4	11	5558	1	0.3423435068941E-08	20.	Miele & Cantrell Function
2	20	40234	1	0.8214653544885E-07	21.	Himmelblau (F-P, pp. 326)
2	10	20123	0	-0.1031628453449E+01	22.	Three-hump camelback (1) (F-P, pp.
2	13	1448	0	0.9837324056667E-10	23.	Three-hump camelback (2) (F-P, pp.
4	71	365146	9	0.3554618463965E-07	24.	Wood Function (Andrei U13, pp.42)
2	66	35282	0	0.1100000000002E+02	25.	Sum of different power (x1=45,
2	34	171947	3	-0.1008600148064E+02	26.	Shekel function (F-P, pp. 111)
8	33	1664504	188	0.5587091822711E-08	27.	DENSCHNA function
2	13	112880	1	0.2472169120657E-12	28.	DENSCHNB function
4	124	6211517	374	0.6407636682735E-08	29.	DENSCHNC function
2	5	252020	7	0.3706190909725E-10	30.	Griewank function
2	19	9776	0	0.2392264124841E-10	31.	Brent function
2	4	201408	3	0.9875204934203E-11	32.	Booth function
2	3	151310	2	0.6418934775920E-12	33.	Matyas function
3	21	105187	1	0.2144472045122E-07	34.	Colville function (Andrei, U25)
2	6	3002326	166	-0.9999999999711E+00	35.	Easom function
8	280	19003	1	0.2866445718079E-04	36.	Beale function (Andrei, U16)
4	5	2502036	65	0.8730870071490E-09	37.	Powell function (Andrei, U62)
2	8	4010395	81	-0.1913222954963E+01	38.	McCormick function
2	4	2001074	27	0.3510557725703E-08	39.	Himmelblau function (-11,-7)
2	6	304420	3	0.2409465011726E-09	40.	Leon function
2	4	101964	1	0.1356415499635E-11	41.	Price4 function
2	6	151312	2	-0.3791237203937E-02	42.	Zettl function
8	63	3210964	141	0.4490426129186E-08	43.	Sphere function
8	21	1069087	46	0.5508803806450E-08	44.	Elipsoid function
2	4	414840	7	0.5922563192900E+01	45.	Himmelblau (Problem 29/428)
3	3	302066	14	0.2293552829184E-09	46.	Himmelblau (Problem 30/428)
2	4	8204	1	0.3013714813619E-09	47.	Himmelblau (Problem 33/430)
2	6	3012421	56	-0.3523860737488E+00	48.	Zirilli function
2	3	15519	1	-0.7833233140723E+02	49.	Styblinski function
2	3	15482	0	-0.1999999999992E+01	50.	Trid function
2	28	1404941	46	0.4913822073228E-08	51.	Scaled Quadratic function
3	6	122003	5	0.6981231133934E-08	52.	Schittkowski 241, pp. 65
6	46	933332	104	0.3535955270103E-07	53.	Schittkowski 271, pp. 95
2	2	4070	0	0.7731990565293E+00	54.	Schittkowski 308, pp. 131
5	4	4139	1	0.2541928088954E-08	55.	Brown's almost linear system
4	6	60194	4	0.9175380652822E-09	56.	Kelley function. Andrei U72
4	18	180402	13	0.7034029261395E-08	57.	A nonlinear system. Andrei U73
2	2	10363	1	-0.1819999999998E+02	58.	Zangwill function. Andrei U14
3	2	10392	0	-0.3923048452734E+00	59.	Circular function. Andrei U19
2	20	102710	11	0.3424486241119E-06	60.	Polexp function. Andrei U21
2	1	5200	1	-0.5000000000000E+00	61.	Dulce function. Andrei U20
4	3	154211	53	0.1291674224865E-13	62.	Cragg & Levy. Andrei U41, pp.49
5	84	6828	1	0.1285216666786E-06	63.	Broyden. Andrei U45, pp.50
8	501	2562184	392	0.9392975075304E+00	64.	Broyden (n=10). Andrei U45, pp.50
10	501	2571775	379	0.8789736953485E+00	65.	Broyden (n=20). Andrei U45, pp.50
5	33	170845	8	0.9648130225310E-12	66.	Full rank (n=5). Andrei U47, pp.51
8	16	80264	8	0.2290198352507E-05	67.	Full rank (n=10). Andrei U47, pp.51
4	34	170467	7	0.5306897356432E-05	68.	Full rank (n=20). Andrei U47, pp.51
5	8	41333	10	0.2246918386265E-08	69.	Trigonometric (n=5). Andrei U48,
9	12	601813	273	0.1026902990869E-07	70.	Trigonometric (n=10). Andrei U48,
10	10	1001508	514	0.1097167273012E-07	71.	Trigonometric (n=20). Andrei U48,
5	9	46626	8	0.7161813414106E-08	72.	Brown function. Andrei U75, pp.59
4	20	203504	243	0.8582220162694E+05	73.	Brown & Dennis, Andrei U32, pp.46
2	2	51661	2	-0.2345811576100E+01	74.	Hosaki function
5	48	5559	1	0.1920078161404E-08	75.	Cosmin function
10	50001	25603049	2209	0.1828116187903E+02	76.	BDQRTIC (CUTE)
10	10001	5121416	385	0.9090909868030E-01	77.	DIXON3DQ (CUTE)
4	56	282842	9	0.2495604369176E+01	78.	ENGVAL1 (CUTE)
5	75	76091	3	0.1522038725998E+01	79.	Extended Penalty Function
10	140	4549	1	0.2421900361697E-06	80.	Broyden pentadiagonal
2	6	69034	4	0.9000000000179E+00	81.	Teo function
2	5	25068	0	0.7415391589066E-02	82.	Coca function
2	29	15001	0	-0.1249999999982E+01	83.	Nec function. Andrei, U30, pp.46
4	1	5214	1	0.3420673059839E-18	84.	QuadraticPowerExp. Andrei, U51,
8	10	51586	2	0.9558267583740E-11	85.	NONDQUAR (CUTE)
40	191	297249	89	0.1515302199717E-05	86.	ARWHEAD (CUTE)
4	5227	31209850	988	0.9904978234274E-04	87.	CUBE (CUTE)
5	87	43635706	1832	0.6318587675248E-05	88.	NONSCOMP (CUTE)
10	148	50672	4	0.1456219360103E-05	89.	DENSCHNF (CUTE)
12	151	78087	6	0.1261454444921E-06	90.	BIGGSB1 (CUTE)
10	12	62424	4	0.1023966649364E-07	91.	Borsec6
2	21	1926879	13	0.3686757105536E+02	92.	Three terms all quadratics
3	18	1650274	24	0.1102124984312E-04	93.	Mishra9
2	17	1559944	13	0.2785637644975E-08	94.	Wayburn1
2	11	570063	5	0.1044643130973E-08	95.	Wayburn2
10	178	920355	70	0.6666667449697E+00	96.	Dixon & Price
15	165	855048	88	0.7426386427905E-06	97.	Qing
2	12	618990	7	-0.3873724182168E+04	98.	Quadratic 2 variables
2	20	100338	2	-0.6850076846409E+02	99.	Rump
4	60	311669	23	0.3995735589349E+00	100.	Extended Cliff (CUTE)
10	121	627317	44	0.9898969553451E+00	101.	NONDIA (CUTE)
4	16	817016	54	-0.3499997429359E+01	102.	EG2 (CUTE)
8	110	570563	47	0.1217995131007E-06	103.	LIARWHD (CUTE)
4	16	80331	12	0.1212871287929E+02	104.	Full Hessian (m=50)
2	39	1954856	143	0.2807057782276E-10	105.	A nonlinear algebraic system
4	14	72067	2	-0.3739004994563E+02	106.	ENGVAL8 (CUTE)
10	9	22568	11	0.1000000011710E+01	107.	DIXMAANA (CUTE)
10	7	36106	18	0.1000000011413E+01	108.	DIXMAANB (CUTE)
5	6	30831	7	0.1000000002733E+01	109.	DIXMAANC (CUTE)
5	113	582712	22	0.8897747881327E-07	110.	DIAG-AUP1
10	52	1710	0	-0.9499999882112E+01	111.	EG3 (COS)
10	62	63026	5	0.1885210564357E-07	112.	VARDIM (CUTE)
4	412	1068904	33	-0.9999606442786E+00	113.	A narrow positive cone
10	14	16328	3	-0.2718123003131E+01	114.	Ackley
10	20001	51044836	4250	-0.1942809040946E+02	115.	Modified Wolfe
2	460	1176773	1	-0.3530815425299E+01	116.	Peak function
3	79	800719	19	0.2492301853743E+02	117.	Function U18 (Andrei, pp. 43)
2	1	1025	0	-0.1400147716590E-05	118.	Function U23 (Andrei, pp. 44)
5	7	70128	3	0.3763764398013E-08	119.	Sum Squares
10	63	64020	6	0.5383942407341E-08	120.	VARDIM MODIFIED (**8)

Total number of iterations = 99913
Total number of function evaluations = 295010496
Total Elapsed time (centeseconds) = 21930

DEEPS12

Comparison Between the Performances of DEEPS and Nelder-Mead for solving 120 Unconstrained Optimization Problems

This directory contains:
- DEEPSPR.FOR (DEEPS algorithm)
- DEEPS.OUT
- DEEPS.REZ
- func120.txt
- f.bmp
- NMPF.FOR (Nelder-Mead algorithm)
- NELMIN.OUT
- NELMIN.REZ
- PERF2N.FOR
- R2020T12.DOC (Technical Report)

For solving 120 problems (16 applications and 104 test problems) with the number of variables in the range [2-40] the following results was obtained by Nelder-Mead.

Table 2. Performances of NELMEAD

n	iter	nfunc	cpu	f(x*)	Name
2	42611	141379	5	-0.2644531412951E+03	1. Weber Function (1) (Andrei, U71)
2	5749	17111	1	0.9560739834844E+01	2. Weber Function (2) (Kelly, pp. 119)
2	485	1515	0	0.8749843722120E+01	3. Weber Function (3) (Kelly, pp. 119)
4	50589	257709	26	0.3075992528786E-03	4. Enzyme reaction (Andrei, U79) (A)
6	4853773	40285964	1549	0.4724650698678E-05	5. Solution of a chemical reactor (A)
8	18824	131208	3	0.6829467973914E-08	6. Robot kinematics problem (A)
4	3553	19323	4	0.6872370208734E+01	7. Solar Spectroscopy (A)
4	1420090	7962328	281	0.3185723794137E-01	8. Estimation of parameters (A)
5	218408	1304415	24	0.7389721566503E-05	9. Propan combustion in air (A)
2	50	151	0	0.1744152013241E+01	10. Gear train with minimum inertia (A)
8	12179330	116416638	3181	0.1098385128966E-06	11. Human Heart Dipole. Andrei U84,
6	749358	5880897	135	0.1666727283741E-06	12. Neurophysiology (A)
10	180224	1478094	55	0.6067373743983E-07	13. Combustion application (A)
9	995980	9939092	1867	0.8535102372545E-03	14. Circuit design (A)
3	3063875	10945869	7870	0.1750913166362E+03	15. Thermistor (A)
4	8026086	36117397	408	0.7515500076120E-19	16. Optimal design of a Gear Train (A)
2	127791	414317	3	0.2734165291417E-08	17. Rosenbrock - Valley of Banana
2	7772	21242	0	0.4898425367977E+02	18. Freudenstein-Roth
10	4695026	36081186	899	0.2997209356058E-09	19. White & Holst Function
4	28366709	134400344	16145	0.2728619079879E-22	20. Miele & Cantrell Function
2	549	1801	0	0.2410661272377E-09	21. Himmelblau (F-P, pp. 326)
2	4138	12917	0	-0.1031628453487E+01	22. Three-hump camelback (1) (F-P, pp.
2	12681	40251	0	0.1791830653421E+01	23. Three-hump camelback (2) (F-P, pp.
4	9355302	47490787	544	0.9110670262491E-09	24. Wood Function (Andrei U13, pp.42)
2	110941	357658	3	0.1100000000000E+02	25. Sum of different power (x1=45,
2	13395	38020	0	-0.5065439735221E+01	26. Shekel function (F-P, pp. 111)
8	32893597	239932420	27981	0.1913163675383E-15	27. DENSCHNA function
2	147386	476192	5	0.1325420598527E-13	28. DENSCHNB function
4	1417159	6039693	489	0.3150644529618E-12	29. DENSCHNC function
2	711418	2298258	65	0.3219646771413E-14	30. Griewank function
2	52622	184115	5	0.1293393700200E-14	31. Brent function
2	52304	201225	2	0.4104965908510E-13	32. Booth function
2	717578	2004520	24	0.2799993833942E-14	33. Matyas function
3	8195333	31914397	401	0.8289711921786E-07	34. Colville function (Andrei, U25)
2	1649974	5224920	233	-0.8110381996167E-04	35. Easom function
8	19783086	132373529	6021	0.6615259847323E-09	36. Beale function (Andrei, U16)
4	1559835	6876985	87	0.1285824459516E-08	37. Powell function (Andrei, U62)
2	4266	12315	0	-0.1913222954978E+01	38. McCormick function
2	237967	809851	6	0.4908320623628E-14	39. Himmelblau function (-11,-7)
2	63610	212134	2	0.1174796918677E-05	40. Leon function
2	1802097	6173224	53	0.1068218485126E-07	41. Price4 function
2	78275	246644	2	-0.3791237220468E-02	42. Zettl function
8	133777215	999990013	25833	0.1025689019328E+00	43. Sphere function
8	26546059	200996384	5589	0.3701062404756E-13	44. Elipsoid function
2	95	193	0	0.5924864255508E+01	45. Himmelblau (Problem 29/428)
3	1439	4699	0	0.7544044265105E-06	46. Himmelblau (Problem 30/428)
2	17452184	52356544	3420	0.1596984695221-312	47. Himmelblau (Problem 33/430)
2	18886	63700	1	-0.1526394417735E+00	48. Zirilli function
2	285	769	0	-0.7833233140632E+02	49. Styblinski function
2	755782	2507508	50	-0.2000000000000E+01	50. Trid function
2	110285272	441135983	2857	0.1892243360928E-12	51. Scaled Quadratic function
3	7510	29043	1	0.1143027552724E-03	52. Schittkowski 241, pp. 65
6	966	4340	0	0.2273917223761E-08	53. Schittkowski 271, pp. 95
2	1029	3374	0	0.7731990567225E+00	54. Schittkowski 308, pp. 131
5	34744	190100	3	0.2297883703351E-06	55. Brown's almost linear system
4	81407052	368163139	6071	0.1422943777975E-13	56. Kelley function. Andrei U72
4	831618	3641092	48	0.1050959372381E-11	57. A nonlinear system. Andrei U73
2	8908	24481	0	-0.1820000000000E+02	58. Zangwill function. Andrei U14
3	3969	14757	0	-0.3923048452658E+00	59. Circular function. Andrei U19
2	333330006	999990001	51593	0.3252892765785E-15	60. Polexp function. Andrei U21
2	18157983	63360019	1366	-0.5000000000000E+00	61. Dulce function. Andrei U20

4	95011309	435691620	52987	0.7500494352983E-17	62.	Cragg & Levy. Andrei U41, pp.49
5	732002	4006517	74	0.3752106106972E-14	63.	Broyden. Andrei U45, pp.50
8	5888828	44813788	1125	0.2027502384709E-13	64.	Broyden (n=10). Andrei U45, pp.50
10	5354896	45896404	2071	0.1657738109364E-13	65.	Broyden (n=20). Andrei U45, pp.50
5	32105835	172990716	5657	0.5600659081408E-15	66.	Full rank (n=5). Andrei U47, pp.51
8	48480940	345462434	24336	0.1455367377274E-12	67.	Full rank (n=10). Andrei U47, pp.51
4	10669233	51885942	1207	0.3021412301497E-13	68.	Full rank (n=20). Andrei U47, pp.51
5	26934031	142029901	33429	0.2760740072636E-14	69.	Trigonometric (n=5). Andrei U48, p
8	45798374	344886928	135910	0.1356472192863E-03	70.	Trigonometric (n=10). Andrei U48,
10	4250369	37671143	19528	0.2795056123570E-04	71.	Trigonometric (n=20). Andrei U48,
5	203955743	999990002	170652	0.1945484835526E-03	72.	Brown function. Andrei U75, pp.59
4	250	448	1	0.8582220220393E+05	73.	Brown & Dennis, Andrei U32, pp.46
2	7205	21445	0	-0.1127794026972E+01	74.	Hosaki function
5	38414	191802	2	0.2120535922759E-11	75.	Cosmin function
10	201534	1833324	58	0.1828116175371E+02	76.	BDQRTIC (CUTE)
10	3233930	28345714	652	0.9090909101913E-01	77.	DIXON3DQ (CUTE)
4	67202284	307509056	6519	0.2495604366214E+01	78.	ENGVAL1 (CUTE)
5	18287	107384	1	0.1522244423034E+01	79.	Extended Penalty Function
10	516284	4324701	157	0.2616410786262E-11	80.	Broyden pentadiagonal
2	114222	324189	13	0.9000000000000E+00	81.	Teo function
2	15239	44506	0	0.3603393492868E-11	82.	Coca function
2	220243	770819	6	-0.1250000000000E+01	83.	Nec function. Andrei, U30, pp.46
4	249997502	999990008	38652	0.6498415264253E-22	84.	QuadraticPowerExp. Andrei, U51,
8	157576603	999990001	32857	0.1061987603079E+00	85.	NONDQUAR (CUTE)
10	129703558	999990012	38047	0.2568702260532E+00	86.	ARWHEAD (CUTE)
4	1221660	5840002	98	0.1170242033020E-03	87.	CUBE (CUTE)
5	164045317	883130211	16416	0.9253097139544E-09	88.	NONSCOMP (CUTE)
10	1255946	11901044	483	0.2432712919620E-11	89.	DENSCHNF (CUTE)
12	45543080	371730317	15144	0.2014644810372E-11	90.	BIGGSB1 (CUTE)
10	133332459	999990011	49722	0.1683728521419E+02	91.	Borsec6
2	2424	7088	0	0.1273426289635E+03	92.	Three terms all quadratics (-10,-7
3	148	300	0	0.5510270894397E-03	93.	Mishra9
2	18567784	57590805	657	0.6454558509586E-14	94.	Wayburn1
2	50268833	142015420	1619	0.7079093388257E-15	95.	Wayburn2
10	136976950	999990005	37149	0.1005563480381E+04	96.	Dixon & Price
15	3688046	45451698	1895	0.2261032544726E-09	97.	Qing
2	1797	5885	0	-0.3873724182186E+04	98.	Quadratic 2 variables
2	253	826	0	-0.1640345171681E+02	99.	Rump
4	212439967	854777872	52900	0.3995732273681E+00	100.	Extended Cliff (CUTE)
10	642708	4988085	164	0.5455027073350E-12	101.	NONDIA (CUTE)
4	3424718	16773818	876	-0.3447779105873E+01	102.	EG2 (CUTE)
8	59110575	410824970	13094	0.1517431852535E-11	103.	LIARWHD (CUTE)
4	12770	57246	18	0.1212871287129E+02	104.	Full Hessian (m=50)
2	1875451	5018254	334	0.1833800208498E-15	105.	A nonlinear algebraic system (App.
4	104963	474395	7	-0.3739004995170E+02	106.	ENGVAL8 (CUTE)
10	79384557	699911844	312619	0.1000000000000E+01	107.	DIXMAANA (CUTE)
10	115678296	999990003	476546	0.1001251911045E+01	108.	DIXMAANB (CUTE)
5	519054	2611878	424	0.1000000000000E+01	109.	DIXMAANC (CUTE)
5	207132	1046318	18	0.3533704907427E-12	110.	DIAG-AUP1
10	685883	5148183	471	-0.8184677637534E+01	111.	EG3 (COS)
10	35829692	279603425	10377	0.4467510264809E-10	112.	VARDIM (CUTE)
4	7261027	38333938	595	-0.9999999949881E+00	113.	A narrow positive cone
10	281032	2414390	398	0.3841368041779E+01	114.	Ackley
10	7664639	69463156	3609	-0.1165685424949E+02	115.	Modified Wolfe
2	1609811	5320855	741	-0.4105766136014E-05	116.	Peak function
3	117079	492835	11	0.2492301853327E+02	117.	Function U18 (Andrei, pp. 43)
2	13116974	39350868	1855	0.3541982800515-316	118.	Function U23 (Andrei, pp. 44)
5	168469	738431	12	0.6048706368964E-13	119.	Sum Squares
10	17305874	131238244	6037	0.3070835264168E-10	120.	VARDIM MODIFIED (**8)

Total CPU time (centeseconds) = 1733441 (4.5 hours)

Figure 1 shows the Dolan and Moré's performance profiles of these algorithms

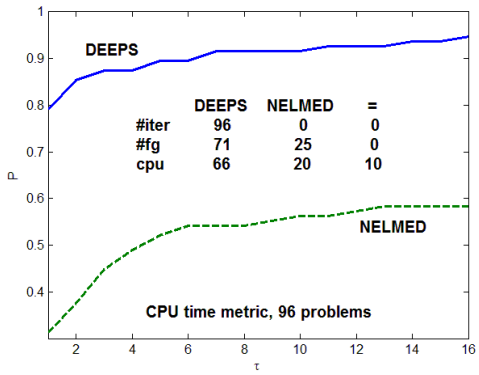


Fig. 1. Performance profiles of DEEPS and NELMED for solving 120 problems

DEEPS13

A two level random search method for unconstrained optimization.

This directory contains:

DEEPSPR.for (Fortran package)

FUNC120.TXT

NMPF.FOR (Fortran package of Nelder-Mead)

NELMIN.out
NELMIN.rez
NELMIN.tot
PERF2N.for (Fortran package for comparing algorithms)
R2020T13.DOC (The paper: A two level random search method for unconstrained optimization. 54 pages)
Figure 5 presents the performance profiles of these algorithms.

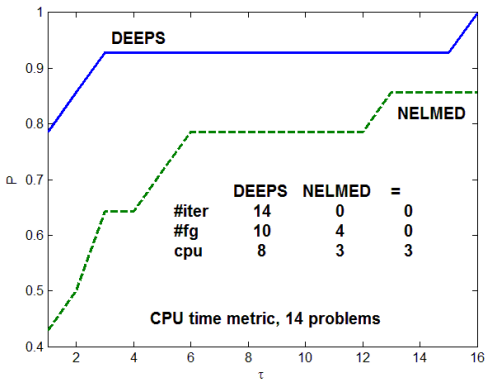


Fig. 5. Performance profiles of DEEPS and NELMED for solving 16 applications of unconstrained optimization

DEEPS14

Numerical experiments with DESCON for solving 14 applications of unconstrained optimization

This directory contains:
descon14.for
descon14.out
descon14.rez
func14.txt
R2020T14/doc (The paper: Numerical experiments with DESCON for solving 14 applications of unconstrained optimization. 10 pages)

Performances of DESCON

n	iter	fgcnt	time(c)	fx*	gnorm	Name of Application
2	1878	10001	0	-0.2644531414650E+03	0.8208740831576E+00	1. Weber Function (Andrei, U71)
4	48	143	0	0.3075056038514E-03	0.6886760990097E-08	2. Enzyme reaction (Andrei, U79) (A)
6	85	264	0	0.9665994663683E-15	0.4231231612938E-07	3. Solution of a chemical reactor (A)
8	1843	10006	2	0.5463981044793E-05	0.1781782618260E-02	4. Robot kinematics problem (A)
4	12	38	0	0.8312307692553E+01	0.8585722387648E-07	5. Solar Spectroscopy (A)
4	46	150	0	0.3185717881375E-01	0.6936429307668E-08	6. Estimation of parameters (A)
5	724	2246	0	0.1224151943762E-06	0.8166044868424E-07	7. Propan combustion in air (A)
2	14	154	0	0.1751192213346E+01	0.7760986494009E-07	8. Gear train with minimum inertia (A)
8	1916	10002	1	0.1120571259805E-01	0.1686188462847E-03	9. Human Heart Dipole. Andrei U84, pp.65
6	93	632	0	0.4539057615171E+01	0.4484463269794E-07	10. Neurophysiology (A)
10	51	142	0	0.6898812079492E-10	0.8219103504792E-07	11. Combustion application (A)
3	1839	10005	9	0.1726024568705E+03	0.1334938231384E+02	12. Thermistor (A)
4	1842	10004	0	0.2387780742094E-02	0.8995056760928E-04	13. Optimal design of a Gear Train (A)
9	739	2166	2	0.1454860731888E-13	0.1554041459749E-06	14. Circuit design (A)
TOTAL	11130	55953	14.00 centeseconds			

Date: --- Month: 6 Day: 3 Year: 2020

DEEPS15

Numerical experiments with CUBIC for solving 14 applications of unconstrained optimization

This directory contains:
cubic14.for
cubic14.out
cubic14.rez
func14.txt
R2020T15.doc (The paper: Numerical experiments with CUBIC for solving 14 applications of unconstrained optimization. 11 pages)

Performances of CUBIC

n	iter	fgcnt	time(c)	fx	gnorm	Name of Applications
2	1288	5001	1	-0.2643790043149E+03	0.6876989703321E+00	1. Weber Function (Andrei, U71)
4	39	116	0	0.3075056060090E-03	0.1140193575396E-06	2. Enzyme reaction (Andrei, U79) (A)
6	94	287	0	0.7468342084540E-15	0.4147895225729E-07	3. Solution of a chemical reactor (A)
8	333	5017	1	0.4829469121831E+00	0.4942442115598E+00	4. Robot kinematics problem (A)
4	10	31	0	0.8312307695614E+01	0.7104616347572E-06	5. Solar Spectroscopy (A)
4	30	96	0	0.3187570933023E-01	0.5862905229454E-06	6. Estimation of parameters (A)
5	555	1670	0	0.4799163454696E-05	0.1371052281519E-05	7. Propan combustion in air (A)
2	11	138	0	0.1751192330768E+01	0.9461910026439E-06	8. Gear train with minimum inertia (A)
8	940	5006	1	0.1199116073210E-01	0.8786818146807E-01	9. Human Heart Dipole. Andrei U84, pp.65
6	24	79	0	0.4539057615171E+01	0.3989872629134E-08	10. Neurophysiology (A)
10	50	139	0	0.4967405721874E-09	0.3630146049766E-06	11. Combustion application (A)
3	395	5003	6	0.1721497388951E+03	0.2361553857583E+01	12. Thermistor (A)
4	7	101	0	0.2322924674570E-04	0.7867065011382E-06	13. Optimal design of a Gear Train (A)
9	563	1639	3	0.8544333431670E-14	0.2890489586180E-06	14. Circuit design (A)
TOTAL	4339	24323	12.00 centeseconds			

Date: ---> Month: 6 Day: 3 Year: 2020

DEEPS16

Numerical experiments with CG-DESCENT for solving 14 applications of unconstrained optimization

This directory contains:
cgdescent14.for
descent14.out
descent14.rez
func14.txt
R2020T16.doc (The paper: Numerical experiments with CG-DESCENT for solving 14 applications of unconstrained optimization. 10 pages)

Performances of CG-DESCENT

n	iter	fgcnt	time(c)	fx	gnorm	Name of Applications
2	130	526	0	-0.2644531414650E+03	0.4360555021096E+00	1. Weber Function (Andrei, U71)
4	87	183	0	0.3075057506207E-03	0.9351232549738E-06	2. Enzyme reaction (Andrei, U79) (A)
6	242	531	0	0.1546034033470E-11	0.8328999653862E-06	3. Solution of a chemical reactor (A)
8	13	79	0	0.1045002080991E-04	0.2937120722379E-02	4. Robot kinematics problem (A)
4	34	73	0	0.6872367741557E+01	0.3753421634575E-06	5. Solar Spectroscopy (A)
4	638	1436	0	0.3194075831746E-01	0.9984546877919E-06	6. Estimation of parameters (A)
5	9001	18039	1	0.1327993904766E-03	0.3013599273355E-03	7. Propan combustion in air (A)
2	14	86	0	0.1745268282541E+01	0.6851854457169E-01	8. Gear train with minimum inertia (A)
8	2	57	0	0.1790818193032E+00	0.3756017838954E-01	9. Human Heart Dipole. Andrei U84, pp.65
6	39	100	0	0.4539057615171E+01	0.3103946255578E-07	10. Neurophysiology (A)
10	55	114	0	0.1279714516413E-09	0.1142557208812E-06	11. Combustion application (A)
3	32	462	1	0.1721680788246E+03	0.2931072079241E+03	12. Thermistor (A)
4	1	56	0	0.1743310601795E-01	0.5889113990961E-03	13. Optimal design of a Gear Train (A)
9	7485	15457	11	0.2419744215211E-10	0.8313476443084E-06	14. Circuit design (A)
TOTAL	17773	37199	13.00 centeseconds			

Date: --- Month: 6 Day: 4 Year: 2020

Line Search with Appproximate Wolfe conditions

DEEPS17

Comparison of modern conjugate gradient methods: DESCON, CUBIC, CG-DESCENT (4.1) for solving 14 small-scale applications of unconstrained optimization

This directory contains:
CGDESCENT
CUBIC
DESCON
R2020T17.doc (The paper: Comparison of modern conjugate gradient methods: DESCON, CUBIC, CG-DESCENT (4.1) for solving 14 small-scale applications of unconstrained optimization. 8pages)

DEEPS18

Numerical experiments with L-BFGS for solving 14 applications of unconstrained optimization

This directory contains:
func14.txt
lbfgs14.for
lbfgs14.out
lbfgs14.rez
R2020T18.doc (The paper: Numerical experiments with L-BFGS for solving 14 applications of unconstrained optimization. 10pages)
Table 5 contains the performances ofDESCON, CUBIC, CG-DESCENT

and L-BFGS for solving 14 applications of unconstarined optimization.

Table 5
Performances of DESCON, CUBIC, CG-DESCENT and L-BFGS

	iter	*fgcnt*	*time*
DESCON	11130	55953	15
CUBIC	4339	24323	12
CG-DESCENT(w)	17773	37199	14
CG-DESCENT(aw)	17773	37199	13
L-BFGS	22029	30723	6

DEEPS19
Properties of the DEEPS algorithm for solving unconstrained optimization problems
This directory contains:
deepsPR.for
func120.txt
R2020T19.doc (The paper: Properties of the DEEPS algorithm for solving unconstrained optimization problems. 6 pages)

DEEPS20
Performances of DEEPS for solving some difficult unconstrained optimization problems
This directory contains:
deepsPR.for
func120.txt
R2020T20.doc (The paper: Performances of DEEPS for solving some difficult unconstrained optimization problems. 7 pages)

April 20 – May 17, 2020

<><><><><><><><><><>

Conjugate Gradient Methods		
1.	**CGALL CGLOOP**	Package implementing 23 Conjugate Gradient Algorithms. The package implements 80 unconstrained test function examples. The following CG algorithms have been implemented: 1) Hestenes – Stiefel, 2) Fletcher – Reeves, 3) Polak-Ribiere-Polyak, 4) Polak-Ribiere-Polyak plus, 5) CD - Conjugate Descent (Fletcher), 6) Liu – Storey, 7) Dai – Yuan, 8) Dai - Liao, 9) Dai - Liao plus, 10) Andrei (SDC), (Please see the paper: **N. Andrei,** *A Dai-Yuan conjugate gradient algorithm with sufficient descent and conjugacy conditions for unconstrained optimization.* Applied Mathematics Letters, **21**, (2008), pp.165-171.) 11) hybrid Dai – Yuan,

		12) hybrid Dai - Yuan zero, 13) Gilbert – Nocedal, 14) Hu and Storey, 15) Touat-Ahmed and Storey, 16) Hybrid LS – CD, 17) Birgin - Martinez (scaled Perry), 18) Birgin - Martinez plus (scaled Perry), 19) scaled Polak-Ribiere-Polyak, 20) scaled Fletcher-Reeves, 21) New cg from PRP: beta=(ytg+ -yty stg+/gtg)/yts (Please, see (8.3.130) in the BOOK. Please, see (5.5.40) in the CG-BOOK.), 22) New cg from DY: beta = ytg+/yts-ytg+*stg/(yts**2) (Please, see (8.3.102) in the BOOK. Please, see (5.5.12) in the CG-BOOK.), 23) New cg from DY: beta=max(0,ytg/yts)*(1-stg/yts), 24) New cg: Please see the paper: W07P26.pdf Please see the books: 1) (**BOOK**) **N. Andrei,** *Critica Raţiunii Algoritmilor de Optimizare fără Restricţii*. Editura Academiei Române, Bucureşti, 2009. 2) (**CG-BOOK**) **N. Andrei**, *Metode Avansate de Ggradient Conjugat pentru Optimizare fără Restricţii*. Editura Academiei Oamenilor de Ştiinţă, Bucureşti, 2009. The Fortran program **CGLOOP.FOR** implements the above 20 conjugate gradient algorithms using the loop unrolling of depth 5. Subdirectory **APPLIC** contains 7 applications from MINPACK-2. Please see OPISAPL.DOC file. **February 8, 2007**
2.	**CG-ACCELERAT**	This package implements a number of 24 conjugate gradient algorithms accelerated by means of a procedure presented in: **N. Andrei**, *Acceleration of conjugate gradient algorithms for unconstrained optimization.* Applied Mathematics and Computation, vol. 213, Issue 2, 2009, pp. 361-369. DOI information: 10.1016/j.amc.2009.03.020 The package implements 80 unconstrained test function examples. The following conjugate gradient algorithms have been implemented: 1) Hestenes – Stiefel, 2) Fletcher – Reeves, 3) Polak-Ribiere-Polyak, 4) Polak-Ribiere-Polyak plus , 5) CD - Conjugate Descent (Fletcher), 6) Liu – Storey, 7) Dai – Yuan, 8) Dai – Liao, 9) Dai - Liao plus, 10) Andrei (ACGSD/2) (Please see the paper: Andrei, N., *A Dai-Yuan conjugate gradient algorithm with sufficient descent and conjugacy conditions for unconstrained optimization.* Applied Mathematics Letters, vol 21, 2008, pp. 165-171.

		Please, see the book: *Advanced Conjugate Gradient Methods for Unconstrained Optimization.* Chapter 5, section 5, Remark 5.5.1. Academy of Romanian Scientists Publishing House, Bucharest, 2009.), 11) hybrid Dai – Yuan, 12) hybrid Dai - Yuan zero, 13) Gilbert – Nocedal, 14) Hu and Storey, 15) Touat-Ahmed and Storey, 16) Hybrid LS – CD, 17) Birgin - Martinez (scaled Perry), 18) Birgin - Martinez plus (scaled Perry), 19) scaled Polak-Ribiere-Polyak (Please see the paper: Andrei, N., ***Scaled memoryless BFGS preconditioned conjugate gradient algorithm for unconstrained optimization***. Optimization Methods and Software, vol.22, No.4, 2007, pp.561-571.), 20) scaled Fletcher-Reeves (Please see the paper: Andrei, N., ***Scaled memoryless BFGS preconditioned conjugate gradient algorithm for unconstrained optimization.*** Optimization Methods and Software, vol.22, No.4, 2007, pp.561-571.), 21) New cg from PRP Please, see the book: ***Advanced Conjugate Gradient Methods for Unconstrained Optimization.*** Chapter 5, section 5, Remark 5.5.2. Academy of Romanian Scientists Publishing House, Bucharest, 2009.), 22) New cg from DY (ACGSD) (Please see the paper: Andrei, N., ***Another nonlinear conjugate gradient algorithm for unconstrained optimization.*** Optimization Methods and Software, vol.24, No.1, 2009, pp.89-104.), 23) New CG from DY (ACGSDz) (Please see the paper: N. Andrei, ***Another nonlinear conjugate gradient algorithm for unconstrained optimization.*** Optimization Methods and Software, vol.24, No.1, February 2009, pp. 89-104.), 24) New cg from PRP and DYc Please see the paper: (N. Andrei, ***New Conjugate Gradient Algorithms for Unconstrained Optimization*** Encyclopedia of Optimization, Second Edition, 2009. C.A. Floudas and P.M. Pardalos (Eds.), Volume N, pp. 2560-2571, Springer.) The subdirectory APPLICATIONS contains 5 applications from MINPACK-II collection, as follows: APPL1.FOR - elastic-plastic torsion problem, APPL2.FOR - pressure distribution in a journal bearing problem, APPL3.FOR - optimal design with composite materials problem, APPL5.FOR - steady state combustion problem, APPL7.FOR - Minimal Surface Area Problem. All these applications have been solved using all 25 conjugate gradient algorithms. The results are enlisted in *.doc files.

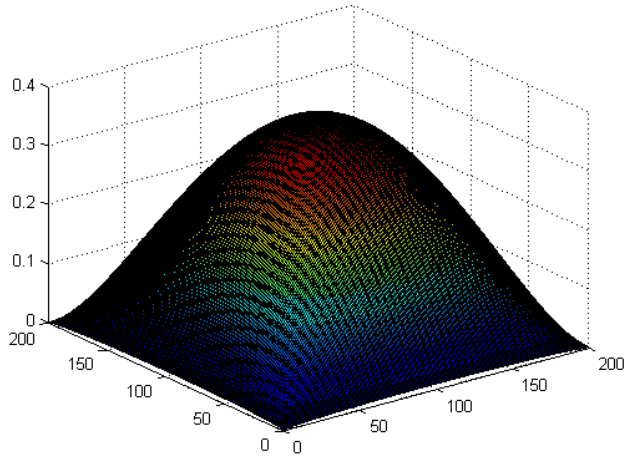


Fig. 1. Solution of the application A1 - Elastic-Plastic Torsion. $nx = 200, ny = 200$

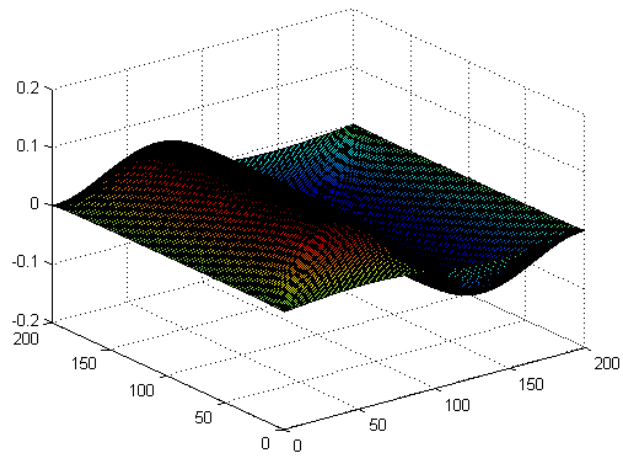


Fig. 2. Solution of the application A2 - Pressure Distribution in a Journal Bearing. $nx = 200, ny = 200$

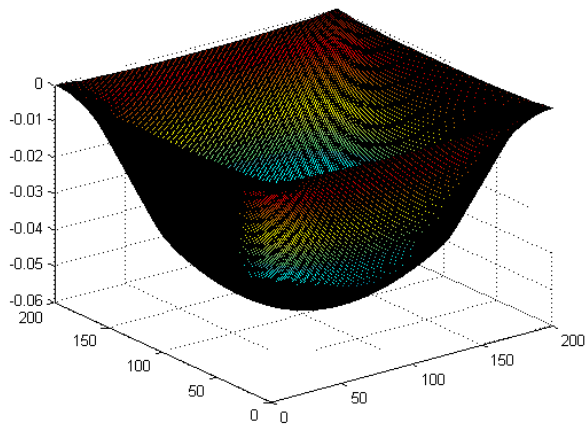


Fig. 3. Solution of the application A3 - Optimal Design with Composite Materials. $nx = 200, ny = 200$

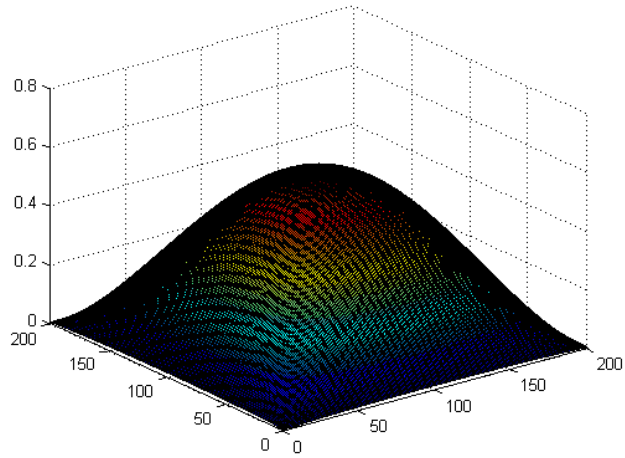


Fig. 4. Solution of the application A4 - Steady-State Combustion. $nx = 200, ny = 200$

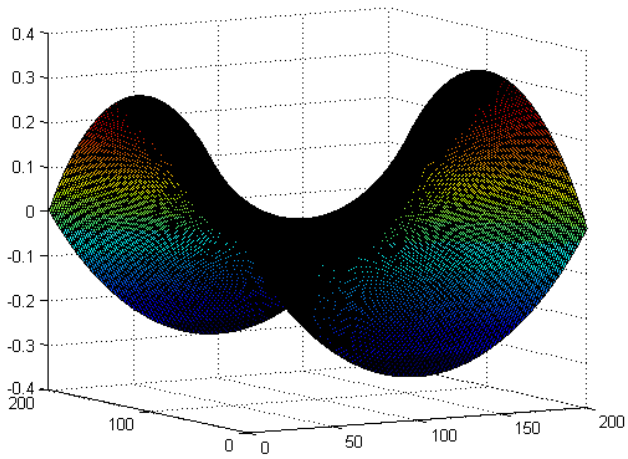


Fig. 5. Solution of the application A5 - Minimal Surfaces with Enneper boundary conditions. $nx = 200, ny = 200$

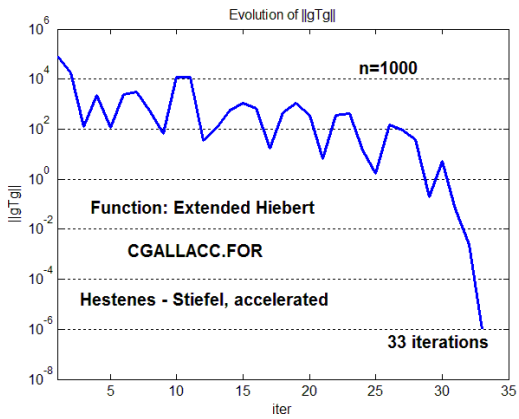


Fig. 6. Evolution of $\|g(k)\|$ given by HS algorithm for minimizing Extended Hiebert function.

		The norm of gradient $\|g(k)\|$ is: 0.8335591087430E+05 0.1770302860649E+05 0.1290104898224E+03 0.2202119533104E+04 0.1128337396401E+03 0.2510029662606E+04 0.3112527173225E+04 0.4760512089568E+03 0.6871298669484E+02 0.1197855311342E+05 0.1207581561453E+05 0.3458177452260E+02 0.1034945886632E+03 0.5609625857035E+03 0.1078967162139E+04 0.6620819548126E+03 0.1787862322948E+02 0.4539204111787E+03 0.1067589106626E+04 0.3678916480927E+03 0.6861727730327E+01 0.3594559195106E+03 0.4227395335853E+03 0.1523510891905E+02 0.1749839153294E+01 0.1539429121421E+03 0.9512788103020E+02 0.3761432997981E+02 0.1934700602985E+00 0.5261841571038E+01 0.6658363856712E-01 **0.2459542556525E-02** **0.9613587612598E-06** Observe that out 33 iterations only for the last two the norm of gradient is below 10^{-2} and 10^{-6} respectively. **March 30, 2009**
3.	**CCOMB** **NDOMB**	The package includes **two** hybrid conjugate gradient algorithms as a convex combination of PRP and DY. **CCOMB** is a Fortran package implementing a New Hybrid Conjugate Gradient Algorithm as a Convex Combination of PRP and DY conjugate gradient algorithms for unconstrained optimization in which the parameter theta is selected from the conjugacy condition. The search direction in CCOMB algorithm is as follows: $$d_{k+1} = -g_{k+1} + \beta_k^{CCOMB} s_k,$$ $$\beta_k^{CCOMB} = (1-\theta_k^{CCOMB})\beta_k^{PRP} + \theta_k^{CCOMB}\beta_k^{DY},$$ $$\theta_k^{CCOMB} = \frac{(y_k^T g_{k+1})(y_k^T s_k) - (y_k^T g_{k+1})\|g_k\|^2}{(y_k^T g_{k+1})(y_k^T s_k) - \|g_{k+1}\|^2\|g_k\|^2}.$$ **NDOMB** is a Fortran package implementing a New Hybrid Conjugate Gradient Algorithm as a Convex Combination of PRP and DY

		conjugate gradient algorithms for unconstrained optimization in which the parameter theta is selected from the Newton direction. The search direction in NDOMB algorithm is as follows: $d_{k+1} = -g_{k+1} + \beta_k^{NDOMB} s_k,$ $\beta_k^{NDOMB} = (1-\theta_k^{NDOMB})\beta_k^{PRP} + \theta_k^{NDOMB}\beta_k^{DY},$ $\theta_k^{NDOMB} = \dfrac{(y_k^T g_{k+1} - s_k^T g_{k+1})\lVert g_k \rVert^2 - (y_k^T g_{k+1})(y_k^T s_k)}{\lVert g_{k+1} \rVert^2 \lVert g_k \rVert^2 - (y_k^T g_{k+1})(y_k^T s_k)}.$ In both algorithms if $\theta_k \le 0,$ then set $\theta_k = 0,$ if $\theta_k \ge 1,$ set $\theta_k = 1.$ The CCOMB and NDOMB algorithms are detailed in the papers: **N. Andrei**, "*Hybrid conjugate gradient algorithm for unconstrained optimization*". Journal of Optimization Theory and Applications**,** vol.41, (2009), pp.249-264. **N. Andrei**, "*New hybrid conjugate gradient algorithms for unconstrained optimization*". C.A. Floudas and P.M. Pardalos, (Eds.) Encyclopedia of Optimization, second edition, 2009, Springer, pages: 2560-2571. **N. Andrei**, "*Performance profiles of conjugate gradient algorithms for unconstrained optimization*". C.A. Floudas and P.M. Pardalos, (Eds.) Encyclopedia of Optimization, second edition, 2009, Springer, pages: 2938-2953. **June 24, 2009**
4.	**CGSYS**	CGSYS is a package dedicated to compute the minimizer of a differentiable function with a large number of variables. The search direction of this algorithm is a linear combination of $-g_{k+1}$ and s_k, where the coefficients in this linear combination are computed in such a way that both the descent and the conjugacy conditions to be guaranteed at every iteration. The search direction is computed as: $d_{k+1} = -\theta_k g_{k+1} + \beta_k s_k,$ $\theta_k = \dfrac{-(y_k^T s_k)\lVert g_{k+1} \rVert^2 t + (s_k^T g_{k+1})^2 u}{\Delta_k},$ $\beta_k = \dfrac{-(y_k^T g_{k+1})\lVert g_{k+1} \rVert^2 t + (s_k^T g_{k+1})\lVert g_{k+1} \rVert^2 u}{\Delta_k},$ $\Delta_k = (y_k^T g_{k+1})(s_k^T g_{k+1}) - \lVert g_{k+1} \rVert^2 (y_k^T s_k).$ The parameters t and u are set $t = 7/8$ and $u = 0.01$. The algorithm is described in: **N. Andrei**, *An accelerated conjugate gradient algorithm with guaranteed descent and conjugacy conditions for unconstrained optimization*. Technical Report, March 6, 2009. (Please see the paper: cgsyspap.doc)

		The subdirectory APPLIC contains 7 applications from MINPACK-II collection. **October 24, 2008**
5.	**CGSECM**	Conjugate gradient algorithm based on the equality of the Newton direction with the conjugate gradient direction and modified secant condition. The algorithm depends on the scalar parameter δ. The search direction is as follows: $$d_{k+1} = -g_{k+1} + \beta_k s_k,$$ If $\delta = 0$, then: $$\beta_k = \max\left\{\frac{y_k^T g_{k+1}}{y_k^T s_k}, 0\right\} - \frac{s_k^T g_{k+1}}{y_k^T s_k},$$ If $\delta \neq 0$, then: $$\beta_k = \max\left\{\frac{y_k^T g_{k+1}}{y_k^T s_k + \delta\eta}, 0\right\} - \left(1 - \frac{\delta\eta}{\|s_k\|^2}\right)\frac{s_k^T g_{k+1}}{y_k^T s_k + \delta\eta},$$ $$\eta = 6(f_k - f_{k+1}) + 3(g_k + g_{k+1})^T s_k.$$ **February 12, 2008**
6.	**CGHSDY (HSDY9, HSDYNG, HSDYPLUS)**	A hybrid conjugate gradient algorithm with convex combination of HS and DY and Newton direction. There are three variants of hybrid conjugate gradient algorithms: **1) HSDY9 algorithm:** The search direction is computed as follows: $$d_{k+1} = -g_{k+1} + \beta_k s_k,$$ $$\theta_k = -\frac{s_k^T g_{k+1} + y_k^T g_{k+1}}{g_k^T g_{k+1}},$$ If $0 < \theta_k < 1$, then $\beta_k = -\frac{s_k^T g_{k+1}}{y_k^T s_k}$, If $\theta_k \geq 1$, then $\beta_k = \frac{\|g_{k+1}\|^2}{y_k^T s_k}$, (DY) If $\theta_k \leq 0$, then $\beta_k = \frac{y_k^T g_{k+1}}{y_k^T s_k}$. (HS) **2) HSDYNG algorithm** In this algorithm the parameter θ_k is computed in 6 different ways: a) Hybrid CG with Newton and secant equation: $$\theta_k = -\frac{s_k^T g_{k+1}}{g_k^T g_{k+1}},$$ b) Hybrid CG with Newton and spectral gradient:

$$\theta_k = \frac{(s_k^T g_{k+1})(y_k^T s_k^T)/\|s_k\|^2 - s_k^T g_{k+1} - y_k^T g_{k+1}}{g_k^T g_{k+1}},$$

c) Hybrid CG with Newton a modification of the above formula:

$$\theta_k = -\frac{s_k^T g_{k+1} + y_k^T g_{k+1}}{g_k^T g_{k+1}},$$

d) Hybrid CG with Newton and Zhang *et all* approximation of sHs:

$$\theta_k = -\frac{s_k^T g_{k+1} + (y_k^T g_{k+1}) * \eta/(y_k^T s_k)}{g_k^T g_{k+1}(1+\eta/(y_k^T s_k))},$$

$$\eta = 6(f_k - f_{k+1}) + 3(g_k + g_{k+1})^T s_k,$$

e) Hybrid CG with Newton and Zhang *et all* approximation of Hs and sHs:

$$\theta_k = \frac{\left(\frac{\delta\eta}{\|s_k\|^2} - 1\right)(s_k^T g_{k+1}) - \delta\eta \frac{y_k^T g_{k+1}}{y_k^T s_k}}{g_k^T g_{k+1}\left(1 + \frac{\delta\eta}{y_k^T s_k}\right)},$$

f) Hybrid CG with Newton and Zhang *et all* approximation of Hs and sHs:

$$\theta_k = -\frac{s_k^T g_{k+1}}{g_k^T g_{k+1}(1+\eta/y_k^T s_k)},$$

$$\eta = 6(f_k - f_{k+1}) + 3(g_k + g_{k+1})^T s_k,$$

With this value of θ_k the value of β_k is computed as:

$$\beta_k = (1-\theta_k)\beta_k^{HS} + \theta_k \beta_k^{DY},$$

If $\theta_k \geq 1$, then set $\beta_k = \beta_k^{DY}$,

If $\theta_k \leq 0$, then set $\beta_k = \beta_k^{HS}$.

3) HSDYPLUS algorithm

In this algorithm the parameters θ_k and β_k are computed as follows:

$$\theta_k = \frac{\left(\frac{\delta\eta}{\|s_k\|^2} - 1\right)(s_k^T g_{k+1}) - \delta\eta \frac{y_k^T g_{k+1}}{y_k^T s_k}}{g_k^T g_{k+1}\left(1 + \frac{\delta\eta}{y_k^T s_k}\right)},$$

$$\eta = 6(f_k - f_{k+1}) + 3(g_k + g_{k+1})^T s_k,$$

$$\beta_k = \max\left\{\frac{y_k^T g_{k+1}}{y_k^T s_k + \delta\eta}, 0\right\} + \frac{(1-\delta\eta/\|s_k\|^2)(s_k^T g_{k+1})}{y_k^T s_k + \delta\eta}.$$

The theoretical developments of HYBRID algorithm are described into the papers:

N. Andrei, *Another hybrid conjugate gradient algorithm for*

		Unconstrained Optimization, Numerical Algorithms, vol.47, no.2, February 2008, pp.143-156. **N. Andrei**. *Accelerated hybrid conjugate gradient algorithm with modified secant condition for unconstrained optimization.* Numerical Algorithms, vol. 54 (2010), pp.23-46. **April 2, 2007**
7.	**CONGRAD (AML5382, PCONMIN, MCONMIN)**	Package for unconstrained minimization using the conjugate gradient algorithm of Shanno with Beale's restart procedure. Prof. Shanno sent me a copy of the package on October 17, 1983. I modified it in some respects, including the possibility to work on a train of numerical experiments. The algorithm is described in: Shanno, D.F., (1978) *Conjugate gradient methods with exact searches.* Mathematics of Operations Research, vol.3, no.3, August 1978, pp.244-256. The subdirectory MINPACK includes 7 applications from MINPACK-II collection. The package **aml5382.for** implements the conjugate gradient algorithm BFGS preconditioned, in variant given by Shanno, with a train of 80 unconstarined optimization test functions. This is a variant of the Shanno's package which I modified in some respects. The line search procedure is incorporated into the package. Another variant of this package is given by **PCONMIN**. The package **MCONGRAD**, wich includes the subroutine CONGRAD uses the numerical derivatives facilities. The subroutine NUMGRAD is designed for numerical derivatives computation. **November 26, 2001**
8.	**CONMINEX**	Another variant of the package for unconstrained minimization using the conjugate gradient algorithm by Shanno and Phua with Beale's restart procedure. Mainly, this package is the same as CONGRAD. **March: 27, 2007**
9.	**CONMIN**	Another variant of the package implementing the conjugate gradient by Shanno and Phua. Subroutine CONMIN is described in the papers: 1) Shanno, D.F., Conjugate gradient methods with inexact searches. Mathematics of Operations Research, vol. 3, No. 3, August 1978, pp. 244256. 2) Shanno, D.F., On the convergence of a new conjugate gradient algorithm. SIAM J. Numer. Anal., vol.15, No.6, December 1978, pp.1247-1257. 3) Shanno, D.F., Phua, K.H., Algorithm 500. Minimization of unconstrained multivariate functions. ACM TOMS, vol.2, No.1, March 1976, pp.87-94. 4) **Andrei, N.,** *Criticism of the unconstrained optimization*

		algorithms reasoning. Academy Publishing House, Bucharest 2009. ISBN 978-973-27-1669-4 (Chapter 8, pp.317-448.) Remark: Professor Shanno sent me the Fortran subrutine CONMIN in October 17, 1983. I modified it in some respects. **October 15, 2004**
10.	**DLDC**	DLDC is a subroutine dedicated to compute the minimizer of a differentiable function with a large number of variables. Mainly, this is a modification of the Dai-Liao conjugate gradient algorithm with guaranteed descent and conjugacy conditions. The search direction is computed as: $$d_{k+1} = -\theta_k g_{k+1} + \max\left\{\frac{y_k^T g_{k+1}}{y_k^T s_k}, 0\right\} s_k - t_k \frac{s_k^T g_{k+1}}{y_k^T s_k} s_k,$$ The parameters θ_k and t_k are computed as solution of the following linear algebraic system: $$\theta_k (y_k^T g_{k+1}) + t_k (s_k^T g_{k+1}) = a_k,$$ $$\theta_k (y_k^T s_k) \lVert g_{k+1} \rVert^2 + t_k (s_k^T g_{k+1})^2 = b_k,$$ where $$a_k = v(s_k^T g_{k+1}) + (y_k^T g_{k+1}),$$ $$b_k = w(s_k^T y_k) \lVert g_{k+1} \rVert^2 + (y_k^T g_{k+1})(s_k^T g_{k+1}).$$ The scalar parameters w and v are introduced in such a way that the algorithm to satisfies the sufficient descent condition and the conjugacy condition respectively. These parameters are assigned to the values: $w = (1 + \lVert s_k^T g_{k+1} \rVert),\ v = 0.1.$ The algorithm is described in: **N. Andrei,** (2009) *An accelerated modified Dai-Liao conjugate gradient algorithm with guaranteed descent and conjugacy conditions for unconstrained optimization*. Technical Report, July 16, 2009. (Please see the Technical Report: n41a09.doc) **N. Andrei,** *Another accelerated conjugate gradient algorithm with guaranteed descent and conjugacy conditions for large-scale unconstrained optimization.* Technical Report, January 29, 2010. (Please see the paper in DLDCNEW.DOC file) The directory MINPACK includes 5 applications from MINPACK-II collection. **May 8, 2009**
11.	**DLDN**	DLDN is a subroutine dedicated to compute the minimizer of a differentiable function with a large number of variables. Mainly, this is a variant of a modification of the Dai-Liao conjugate gradient algorithm with guaranteed descent and conjugacy conditions. The search direction is computed as:

		$$d_{k+1} = -\theta_k g_{k+1} + \max\left\{\frac{y_k^T g_{k+1}}{y_k^T s_k}, 0\right\} s_k - t_k \frac{s_k^T g_{k+1}}{y_k^T s_k} s_k,$$ The parameters θ_k and t_k are computed as solution of the following linear algebraic system: $$\theta_k (y_k^T g_{k+1}) + t_k (s_k^T g_{k+1}) = a_k,$$ $$\theta_k (y_k^T s_k) \|g_{k+1}\|^2 + t_k (s_k^T g_{k+1})^2 = b_k,$$ where $$a_k = v(s_k^T g_{k+1}) + (y_k^T g_{k+1}),$$ $$b_k = w(s_k^T y_k)\|g_{k+1}\|^2 + (y_k^T g_{k+1})(s_k^T g_{k+1}).$$ The scalar parameters w and v are introduced in such a way that the algorithm to satisfies the sufficient descent condition and the conjugacy condition respectively. These parameters are assigned to the values: $w = 7/8,\ v = 0.1.$ The algorithm is not too much sensitive to the values of these parameters. **January 29, 2010**
12.	**DESCON**	DESCON is a subroutine dedicated to compute the minimizer of a differentiable function with a large number of variables. The search direction is computed as: $$d_{k+1} = -\theta_k g_{k+1} + \beta_k s_k,$$ $$\beta_k = \frac{y_k^T g_{k+1} - t_k s_k^T g_{k+1}}{y_k^T s_k},$$ where θ_k and t_k are scalar parameters which are computed as: $$t_k = \frac{b_k (y_k^T g_{k+1}) - a_k (y_k^T s_k)\|g_{k+1}\|^2}{\Delta_k},$$ $$\theta_k = \frac{a_k - t_k (s_k^T g_{k+1})}{y_k^T g_{k+1}},$$ $$\bar{\Delta}_k = (y_k^T g_{k+1})(s_k^T g_{k+1}) - \|g_{k+1}\|^2 (y_k^T s_k),$$ $$\Delta_k = (s_k^T g_{k+1})\bar{\Delta}_k,$$ $$a_k = v(s_k^T g_{k+1}) + y_k^T g_{k+1},$$ $$b_k = w\|g_{k+1}\|^2 (y_k^T s_k) + (y_k^T g_{k+1})(s_k^T g_{k+1}).$$ $v > 0$ and $w > 0$ are *known* scalar parameters. The algorithm is described in: - **N. Andrei**, *An accelerated conjugate gradient algorithm with guaranteed descent and conjugacy conditions for large-scale unconstrained optimization.* ICI Technical Report, November 29, 2010. Please see the Technical Report: R5A11.DOC file. - **N. Andrei**, *Another conjugate gradient algorithm with guaranteed*

descent and conjugacy conditions for large-scale unconstrained optimization. Journal of Optimization Theory and Applications, vol. 159, Number 1, 2013, pp159-182.
Please, see the paper: JOTA-2013.pdf (paper published in JOTA)
- **N. Andrei**, *Nonlinear Conjugate Gradient Methods for Unconstrained Optimization*, vol. 158 Springer Optimization and Its Applications, Springer, 2020, (Chapter 7, pp.227-245)

A comprehensive numerical comparasions between DESCON and some other conjugate gradient algorithms are presented into the paper **N. Andrei**, *A numerical study on efficiency and robustness of some conjugate gradient algorithms for large-scale unconstrained optimization.* Technical Report, June 6, 2013.
(Please see the paper: ANpaper.doc.)

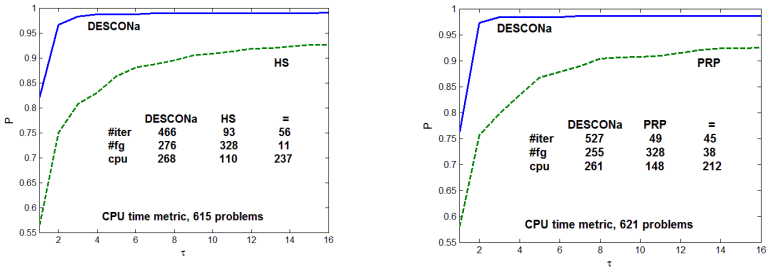


Fig. 1. Performance profile of DESCONa versus HS and versus PRP

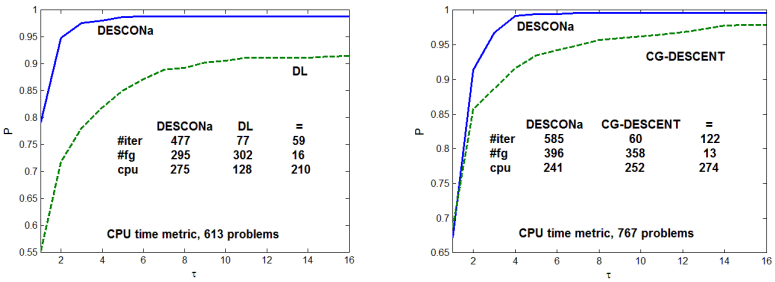


Fig. 2. Performance profile of DESCONa versus DL ($t=1$) and versus CG-DESCENT

The subdirectory MINPACK2 contains 5 applications from the MINPACK-2 Collection.

November 22 2010

13.	**HS**	This package implements the Hestenes and Stiefel (HS) conjugate gradient algorithm using loop unrollong of depth 5. The search direction is computed as: $$d_{k+1} = -g_{k+1} + \beta_k s_k,$$ $$\beta_k = \frac{y_k^T g_{k+1}}{y_k^T s_k}.$$ **January 10, 2013**
14.	**hDY**	The package implements the hybrid Day and Yuan conjugate gradient algorithm using loop unrolling of depth 5. The search direction is computed as:

		$$d_{k+1} = -g_{k+1} + \beta_k s_k,$$ $$\beta_k = \max\left\{ -c\frac{\|g_{k+1}\|^2}{y_k^T s_k}, \min\left\{ \alpha_k \frac{y_k^T g_{k+1}}{y_k^T s_k}, \frac{\|g_{k+1}\|^2}{y_k^T s_k} \right\}\right\},$$ where $c = 0.05263157$ and α_k is the step length computed by the Wolfe line search conditions. **March 29, 2013**
15.	**HYBRID, HYBRIDM, AHYBRIDM**	A hybrid conjugate gradient algorithm with Convex combination of HS and DY and Newton direction with secant condition. This subdirectory contains three packages: **HYBRID, HYBRIDM, AHYBRIDM.** In **HYBRID** it is assumed that the pair (s_k, y_k) satisfies the secant condition. The search direction is as follows: $$d_{k+1} = -g_{k+1} + \beta_k^C s_k.$$ A parameter θ_k is computed as: $$\theta_k = \frac{s_k^T g_{k+1}}{g_k^T g_{k+1}}.$$ If $0 < \theta_k < 1$, then $$\beta_k^C = (1-\theta_k)\frac{g_{k+1}^T y_k}{y_k^T s_k} + \theta_k \frac{\|g_{k+1}\|^2}{y_k^T s_k}.$$ If $\theta_k \geq 1$, then $\beta_k^C = \beta_k^{DY} = \frac{\|g_{k+1}\|^2}{y_k^T s_k}$. If $\theta_k \leq 0$, then $\beta_k^C = \beta_k^{HS} = \frac{g_{k+1}^T y_k}{y_k^T s_k}$. The theoretical developments of HYBRID algorithm are described into the paper: **N. Andrei**, (2008) *Another hybrid conjugate gradient algorithm for Unconstrained Optimization*, Numerical Algorithms, vol.47, no.2, February 2008, pp.143-156. **HYBRIDM** is an extension of the HYBRID package authored by N. Andrei. In HYBRIDM it is assumed that the pair (s_k, y_k) satisfies the modified secant condition given by Zhang, Deng and Chen into the paper: J.Z. Zhang, N.Y. Deng and L.H. Chen, "New quasi-Newton equation and related methods for unconstrained optimization", JOTA, 102 (1999), p. 147-167. **AHYBRIDM** is an acceleration of the HYBRIDM package. The directory APPLIC contains 7 applications from MINPACK-II

		collection. AHYBR1.FOR - Elastic-Plastic Torsion Problem, AHYBR2.FOR - Pressure Distribution in a Journal Bearing Problem, AHYBR3.FOR - Optimal Design with Composite Materials Problem, AHYBR4.FOR - Inhomogeneous Superconductors. Ginzburg-Landau (1-dimensional) problem, AHYBR5.FOR - Steady State Combustion Problem, AHYBR6.FOR - Jones Clusters (Molecular Conformation) Problem, AHYBR7.for - Minimal Surface Area Problem. **April 8, 2008**
16.	**ACGA**	A nonlinear conjugate gradient algorithm which is a modification of the Dai and Yuan [*A nonlinear conjugate gradient method with a strong global convergence property*, SIAM J. Optim. 10 (1999), pp. 177–182] conjugate gradient algorithm satisfying a parameterized sufficient descent condition with a parameter δ_k is proposed. The parameter δ_k is computed by means of the conjugacy condition, thus an algorithm which is a positive multiplicative modification of the Hestenes and Stiefel [*Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards Sec. B 48 (1952), pp. 409–436] algorithm is obtained. The algorithm can be viewed as an adaptive version of the Dai and Liao [*New conjugacy conditions and related nonlinear conjugate gradient methods*, Appl. Math. Optim. 43 (2001), pp. 87–101] conjugate gradient algorithm. The search direction is computed as: $$d_{k+1} = -g_{k+1} + \beta_k^A s_k,$$ $$\beta_k^A = \frac{1}{y_k^T s_k}\left(y_k - \frac{g_{k+1}^T y_k}{y_k^T s_k} s_k \right)^T g_{k+1}.$$ The algorithm is described in the paper: **N. Andrei**, *Another nonlinear conjugate gradient algorithm for unconstrained optimization.* Optimization Methods & Software, Vol. 24, No. 1, February 2009, 89–104. **July 31, 2008**
17.	**CGSECM**	Conjugate gradient algorithm based on the equality of the Newton direction with the conjugate gradient direction and modified secant condition. The search direction is computed as: $$d_{k+1} = -g_{k+1} + \beta_k s_k,$$ $$\beta_k = \max\left\{ \frac{y_k^T g_{k+1}}{y_k^T s_k + \delta\eta}, 0 \right\} - \left(1 - \frac{\delta\eta}{s_k^T s_k} \right) \frac{s_k^T g_{k+1}}{y_k^T s_k + \delta\eta},$$ $$\eta = 6(f_k - f_{k+1}) + 3(g_k + g_{k+1})^T s_k,$$ $$\delta = \begin{cases} 0, \\ 1 \end{cases} \text{ a parameter.}$$ **February 12, 2008**
18.	**HYBRID7**	Accelerated conjugate gradient algorithm based on the equality of the

Newton direction with the conjugate gradient direction and 7 BFGS approximations of the Hessian used in modified secant condition.

The algorithm is described in:
Andrei, N., *Another hybrid conjugate gradient algorithm for unconstrained optimization*. Numerical Algorithms, vol. 47, (2008), pp.143-156.
Andrei, N., *Accelerated hybrid conjugate gradient algorithm with modified secant condition for unconstrained optimization*. Numerical Algorithms (2010) vol.54, pp.23-46.
(Please see the Technical Report: n14a09.pdf)

Methods for BFGS updating:
1) Secant condition. The pair (s_k, y_k) satisfies the secant condition

$$B(k+1)s(k) = y(k)$$

Please see HYBRID algorithm described in:
Andrei, N., *Another hybrid conjugate gradient algorithm for unconstrained optimization*, Numerical Algorithms, vol. 47, (2008), pp.143-156.

2) The pair (s_k, y_k) satisfies the modified secant condition

$$B_{k+1}s_k = \bar{y}_k.$$

$$\bar{y}_k = y_k + \frac{\rho_k s_k}{\|s_k\|^2},$$

$$\rho_k = 2(f_k - f_{k+1}) + (g_k + g_{k+1})^T s_k.$$

3) The pair (s_k, y_k) satisfies the modified secant condition

$$B_{k+1}s_k = \tilde{y}_k.$$

$$\tilde{y}_k = y_k + c\|g_k\|s_k,$$

where c is a positive constant, suggested by Li and Fukushima [2001].

4) The pair (s_k, y_k) satisfies the modified secant condition

$$B_{k+1}s_k = y_k^*.$$

$$y_k^* = y_k + \max\{\rho_k, 0\}\frac{s_k}{\|s_k\|^2},$$

$$\rho_k = 2(f_k - f_{k+1}) + (g_k + g_{k+1})^T s_k.$$

suggested by Yuan and Wei [2010].

5) B_{k+1} is approximated by Yuan's formula [1991].

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + t_k \frac{y_k y_k^T}{y_k^T s_k}$$

where

		$$t_k = \frac{2}{s_k^T y_k}(f_k - f_{k+1} + s_k^T g_{k+1})$$ and t_k belongs to the interval $[0.01,100]$. 6) $s_k^T B_{k+1} s_k$ is from the interpolation condition by Yuan. $$s_k^T B_{k+1} s_k = 2(f_k - f_{k+1}) + s_k^T g_{k+1}$$ 7) $s_k^T B_{k+1} s_k$ is from the Hermite interpolation condition. If the function f is cubic along the line between x_{k-1} and x_k then by considering the Hermite interpolation we get: $$s_k^T B_{k+1} s_k = 4 s_k^T g_{k+1} + 2 s_k^T g_k - 6(f_{k+1} - f_k)$$ **October 6, 2010** **New version September 23, 2013**
19.	**PRP**	Polak-Ribière-Polyak conjugate gradient algorithm. The search direction is computed like: $$d_{k+1} = -g_{k+1} + \beta_k s_k,$$ $$\beta_k = \frac{y_k^T g_{k+1}}{g_k^T g_k}.$$ **January 15, 2013**
20.	**PRP-DC**	Three-term Conjugate Gradient Algorithms in three variants: 1) PRP Modified Method (Andrei) (PRPDC) 2) Zhang, Zhou and Li (ZZL) 3) Zhang, Xiao and Wei (ZXW) The search direction in version PRPDC: $$d_{k+1} = \frac{1}{\|g_k\|^2}\left[-(y_k^T s_k) g_{k+1} + (y_k^T g_{k+1}) s_k - (s_k^T g_{k+1}) y_k\right]$$ The search direction in version ZZL: $$d_{k+1} = -g_{k+1} + \frac{y_k^T g_{k+1}}{\alpha_k \|g_k\|^2} s_k - \frac{s_k^T g_{k+1}}{\alpha_k \|g_k\|^2} y_k.$$ The search direction in version ZXW: $$d_{k+1} = -g_{k+1} + \frac{1}{y_k^T s_k}\left[(y_k^T g_{k+1}) s_k - (s_k^T g_{k+1}) y_k\right]$$ The algorithm PRP modified (PRP-DC) is described in the paper: **N. Andrei**, *A modified Polak–Ribie`re–Polyak conjugate gradient algorithm for unconstrained optimization*. Optimization, Vol. 60, No. 12, December 2011, 1457–1471. (Please se the paper: optimiz11.pdf) **August 25, 2009**
21.	**ACGA**	Another Nonlinear Conjugate Gradient Algorithm for Unconstrained Optimization. The search direction in ACGA is computed as:

		$$d_{k+1} = -g_{k+1} + \beta_k^A s_k,$$ $$\beta_k^A = \frac{1}{y_k^T s_k}\left(y_k - \frac{g_{k+1}^T y_k}{y_k^T s_k} s_k \right)^T g_{k+1}.$$ Please see the paper: **Andrei, N.,** (2009), *Another Nonlinear Conjugate Gradient Algorithm for Unconstrained Optimization*, Optimization Methods and Software, vol.24, No.1, February 2009, pp. 89-104. **July 31, 2008**
22.	**ACGHES**	Accelerated conjugate gradient algorithm based on the equality of the Newton direction with the conjugate gradient direction and using the Hessian / vector product. The search direction in this algorithm is computed as: $$d_{k+1} = -g_{k+1} + \frac{s_k^T \nabla^2 f(x_{k+1}) g_{k+1} - s_k^T g_{k+1}}{s_k^T \nabla^2 f(x_{k+1}) s_k},$$ where the Hessian / vector product is computed using the finite difference: $$\nabla^2 f(x_{k+1}) s_k = \frac{\nabla f(x_{k+1} + \delta s_k) - \nabla f(x_{k+1})}{\delta},$$ $$\delta = \frac{2\sqrt{\varepsilon_m}(1 + \|x_{k+1}\|\sqrt{n})}{\|s_k\|},$$ ε_m is epsilon machine. The algorithm is described in: **Andrei., N.,** (2009) *Accelerated conjugate gradient algorithm with finite difference Hessian/vector product approximation for unconstrained optimization.* Journal of Computational and Applied Mathematics, vol. 230, 2009, pp. 570-582. Please see the paper jcam2009.pdf. The directory MINPACK contains 7 applications from MINPACK-II collection: ACGHES1.FOR - elastic-plastic torsion, ACGHES2.FOR – pressure distribution in a journal bearing problem, ACGBES3.FOR - optimal design with composite materials problem, ACGHES4.FOR - Inhomogeneous Superconductors. Ginzburg-Landau (1-dimensional) problem, ACGHES5.FOR - steady state combustion problem, ACGHES6.FOR - Jones Clusters (Molecular Conformation) Problem, ACGHES7.FOR - minimal surface area problem. **February 12, 2008**
23.	**ACGHESM**	Accelerated conjugate gradient algorithm based on the equality of the Newton direction with the conjugate gradient direction and using the Hessian / Vector product. The Hessian / vector product is computed by finite difference using 5

		different increments. The package is testing the numerical performances of this conjugate gradient algorithm subject to the values of increments for Hessian / vector product approximations. The search direction in this algorithm is computed as: $$d_{k+1} = -g_{k+1} + \frac{s_k^T \nabla^2 f(x_{k+1}) g_{k+1} - s_k^T g_{k+1}}{s_k^T \nabla^2 f(x_{k+1}) s_k},$$ where the Hessian / vector product is computed using the finite difference: $$\nabla^2 f(x_{k+1}) s_k = \frac{\nabla f(x_{k+1} + \delta s_k) - \nabla f(x_{k+1})}{\delta},$$ where ε_m is epsilon machine and δ is estimated by the following methods: 1) Schlick-Fogelson (TNPACK) (SF) $$\delta = \frac{2\sqrt{\varepsilon_m}(1 + \lVert x_{k+1} \rVert \sqrt{n})}{\lVert s_k \rVert},$$ 2) Schlick-Fogelson (variant) (SFV) $$\delta = \frac{2\sqrt{\varepsilon_m}(1 + \lVert x_{k+1} \rVert)}{\lVert s_k \rVert},$$ 3) Nash (Truncated-Newton) (NASH) $$\delta = \sqrt{\varepsilon_m (1 + \lVert x_{k+1} \rVert},$$ 4) Dembo-Steihaug (DS) $$\delta = \frac{\sqrt{\varepsilon_m}}{\lVert s_k \rVert},$$ 5) O'Leary (LEARY) $$\delta = \frac{2\sqrt{\varepsilon_m}(1 + \lVert x_{k+1} \rVert)}{\lVert s_k \rVert^2}.$$ The algorithm is described in: **N. Andrei,** *Accelerated conjugate gradient algorithm with finite difference Hessian/vector product approximation for unconstrained optimization*. Journal of Computational and Applied Mathematics, 230 (2009) 570-582. **February 23, 2010**
24.	**ACGSEC**	This algorithm uses a hybrid approach by considering a convex combination of Hestenes and Stiefel (HS) and Dai and Yuan (DY) conjugate gradient algorithms. **ACGSEC** is an accelerated conjugate gradient algorithm based on the equality of the Newton direction with the conjugate gradient direction and secant condition. The search direction is computed as: $$d_{k+1} = -g_{k+1} + \beta_k s_k,$$

		$$\beta_k = \max\left\{\frac{y_k^T g_{k+1}}{y_k^T s_k}, 0\right\} - \frac{s_k^T g_{k+1}}{y_k^T s_k}.$$ The algorithm is described in: **Andrei, N.,** (2010) *Accelerated hybrid conjugate gradient algorithm with modified secant condition for unconstrained optimization.* Numerical Algorithms (2010) vol.54, pp.23-46. (Please see the Technical Report n14a09.doc: „*Accelerated hybrid conjugate gradient algorithm with modified secant condition for unconstrained optimization*", February 23, 2009.) **February 22, 2008**
25.	**ACGMSEC**	This algorithm uses a hybrid approach by considering a convex combination of Hestenes and Stiefel (HS) and Dai and Yuan (DY) conjugate gradient algorithms. **ACGMSEC** is an accelerated conjugate gradient algorithm based on the equality of the Newton direction with the conjugate gradient direction and modified secant condition. The search direction is computed as: $$d_{k+1} = -g_{k+1} + \beta_k s_k,$$ $$\beta_k = \max\left\{\frac{y_k^T g_{k+1}}{y_k^T s_k + \delta\eta}, 0\right\} - \left(1 - \frac{\delta\eta}{\|s_k\|^2}\right)\frac{s_k^T g_{k+1}}{y_k^T s_k + \delta\eta},$$ $$\eta = 6(f_k - f_{k+1}) + 3(g_k + g_{k+1})^T s_k.$$ δ is a parameter. If $\delta = 0$ we get the **ACGSEC** algorithm. The algorithm is described in: **Andrei, N.,** (2010) *Accelerated hybrid conjugate gradient algorithm with modified secant condition for unconstrained optimization.* Numerical Algorithms (2010) vol.54, pp.23-46. (Please see the Technical Report n14a09.doc: „*Accelerated hybrid conjugate gradient algorithm with modified secant condition for unconstrained optimization*", February 23, 2009.) The directory MINPACK2 contains 7 applications from MINPACK-II collection. **February 11, 2008**
26.	**SCALCG** **ASCALCG**	Scaled Conjugate Gradient Algorithm BFGS Preconditioned with Powell restart. The package implements 80 unconstrained test function examples. The search direction in this algorithm is computed as: $$d_{k+1} = -\theta_{k+1} g_{k+1} + \theta_{k+1}\frac{s_k^T g_{k+1}}{y_k^T s_k} y_k - \left[\left(1 + \theta_{k+1}\frac{\|y_k\|^2}{y_k^T s_k}\right)\frac{s_k^T g_{k+1}}{y_k^T s_k} - \theta_{k+1}\frac{y_k^T g_{k+1}}{y_k^T s_k}\right] s_k.$$

At step r when $\left|g_r^T g_{r+1}\right| \geq 0.2\left\|g_{r+1}\right\|^2$ the algorithm is restarted using the above search direction. Otherwise, for any $k \geq r+1$ the search direction d_{k+1} is computed using a double upddate scheme as:

$$v = \theta_{r+1} g_{k+1} - \theta_{r+1} \frac{s_r^T g_{k+1}}{y_r^T s_r} y_r + \left[\left(1 + \theta_{r+1} \frac{\left\|y_r\right\|^2}{y_r^T s_r}\right) \frac{s_r^T g_{k+1}}{y_r^T s_r} - \theta_{r+1} \frac{y_r^T g_{k+1}}{y_r^T s_r}\right] s_r,$$

$$w = \theta_{r+1} y_r - \theta_{r+1} \frac{s_r^T y_k}{y_r^T s_r} y_r + \left[\left(1 + \theta_{r+1} \frac{\left\|y_r\right\|^2}{y_r^T s_r}\right) \frac{s_r^T y_k}{y_r^T s_r} - \theta_{r+1} \frac{y_k^T y_r}{y_r^T s_r}\right] s_r,$$

with wich the search direction is computed as follows:

$$d_{k+1} = -v + \frac{(g_{k+1}^T s_k) w + (g_{k+1}^T w) s_k}{y_k^T s_k} - \left(1 + \frac{y_k^T w}{y_k^T s_k}\right) \frac{g_{k+1}^T s_k}{y_k^T s_k} s_k.$$

The algorithm is presented in:
- **Andrei, N.,** (2007) *A scaled BFGS preconditioned conjugate gradient algorithm for unconstrained optimization.* Applied Mathematics Letters, 20 (2007), p.645-650.
- **Andrei, N.,** (2006) *Scaled memoryless BFGS preconditioned conjugate gradient algorithm for unconstrained optimization*. Optimization Methods and Software, vol.22, Number 4, August 2007, pp.561-571.
- **Andrei, N.,** (2007) *Scaled conjugate gradient algorithms for unconstrained optimization.* Computational Optimization and Applications, vol.38, Number 3, December 2007, pp.401-416.
- **Andrei, N.**, *Nonlinear Conjugate Gradient Methods for Unconstrained Optimization*, vol. 158 Springer Optimization and Its Applications, Springer, 2020, (Chapter 8, pp.261-277)

In the SCALCG subroutine there is the logical argument parameter **accelerat**. If **accelerat is false**, then SCALCG algorithm is used. Otherwise, if **accelerat is true**, then the ASCALCG algorithm is considered.

***

ASCALCG is Accelerated Scaled Conjugate Gradient Algorithm BFGS Preconditioned with Powell restart.
ASCALCG is an acceleration of the SCALCG algorithm.
ASCALCG is used when the logical parameter **accelerat** in SCALCG subroutine is set to **true**.

The algorithm is presented in:
Andrei, N., (2010) *Accelerated scaled memoryless BFGS preconditioned conjugate gradient algorithm for unconstrained optimization.* European Journal of Operational Research, Vol. 204, 2010, pp.410-420.
The package implements 80 unconstrained test function examples.

In SCALCG subdirectory there are three files with numerical comparisons, as follows: **comp-1.doc** contains comparison of SCALCG versus ASCALCG, **comp-2.doc** include comparison of ASCALCG versus DESCON, **comp-3.doc** gives comparisons of ASCALCG versus CG-DESCENT.

June 15, 2005
Implementation of the acceleration scheme, March 5, 2008

27. **THREECG**

A three-term conjugate gradient algorithm which satisfies both the descent condition and the conjugacy condition.
The direction is computed as:

$$d_{k+1} = -g_{k+1} - \delta_k s_k - \eta_k y_k,$$

$$\delta_k = \left(1 + \frac{\|y_k\|^2}{y_k^T s_k}\right)\frac{s_k^T g_{k+1}}{y_k^T s_k} - \frac{y_k^T g_{k+1}}{y_k^T s_k},$$

$$\eta_k = \frac{s_k^T g_{k+1}}{y_k^T s_k}.$$

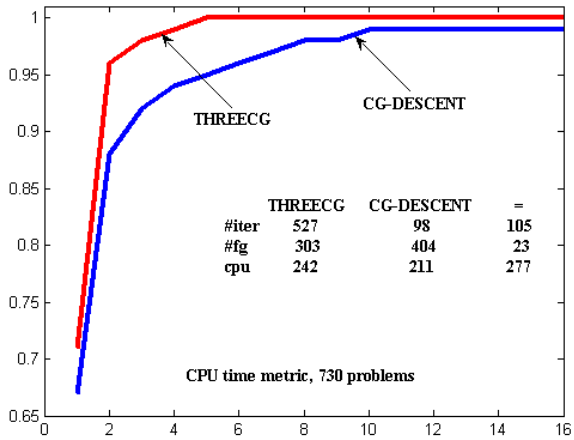


Fig. 1. THREECG versus CG_DESCENT.

Table 1. Performance of THREECG versus CG_DESCENT for solving 5 applications from MINPACK-2 collection. 1,000,000 variables. cpu seconds.

	THREECG			CG_DESCENT		
	#iter	#fg	cpu	#iter	#fg	cpu
A1	1111	2253	**306.04**	1145	2291	436.05
A2	2837	5702	**979.27**	3368	6737	1571.53
A3	4507	9104	**1904.79**	4841	9684	2904.12
A4	1413	2864	**1128.70**	1806	3613	2093.79

A5	1333	2689	**546.20**	1226	2453	713.89
TOTAL	11201	22612	**4865.00**	12386	24778	7719.38

The algorithm is described in:
N. Andrei, *A simple three-term conjugate gradient algorithm for unconstrained optimization.* Journal of Computational and Applied Mathematics, vol. 241, 2013, pp. 19-29.
(Please see the file: threecg-r2.doc)

September 28, 2012

28. **TTS**

An accelerated subspace minimization three-term conjugate gradient algorithm for unconstrained optimization.
This is a three-term conjugate gradient algorithm for which the search direction is computed as:

$$d_{k+1} = -g_{k+1} + a_k s_k + b_k y_k,$$

$$a_k = \frac{1}{\Delta_k}\left[\eta_k (y_k^T g_{k+1} - s_k^T g_{k+1}) - \|y_k\|^2 (\omega_k - y_k^T g_{k+1})\right],$$

$$b_k = \frac{1}{\Delta_k}\left[(y_k^T s_k)(\omega_k - y_k^T g_{k+1}) - \|y_k\|^2 (y_k^T g_{k+1} - s_k^T g_{k+1})\right],$$

$$\Delta_k \equiv (y_k^T y_k)^2,$$

$$\eta_k = 2\frac{(y_k^T y_k)^2}{y_k^T s_k},$$

$$\omega_k = g_{k+1}^T y_k + \frac{(g_{k+1}^T y_k)(y_k^T y_k)}{y_k^T s_k} - \frac{(g_{k+1}^T s_k)(s_k^T y_k)}{s_k^T s_k}.$$

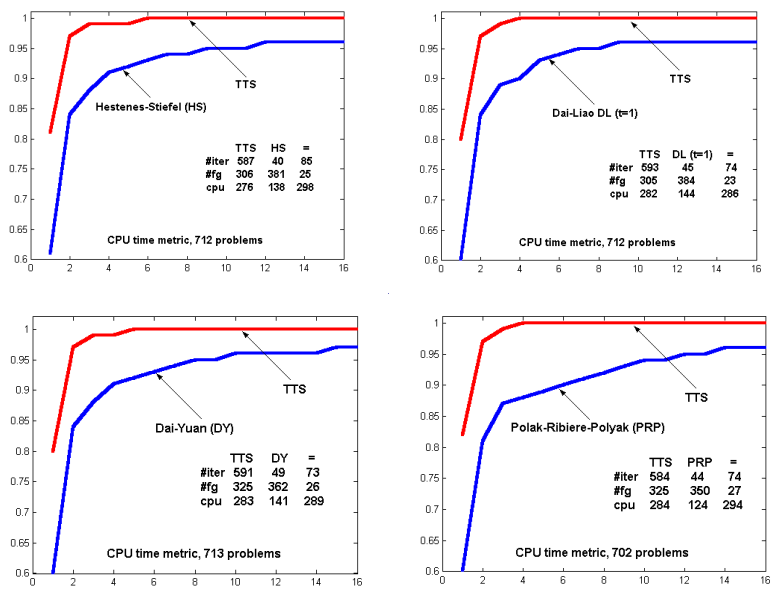
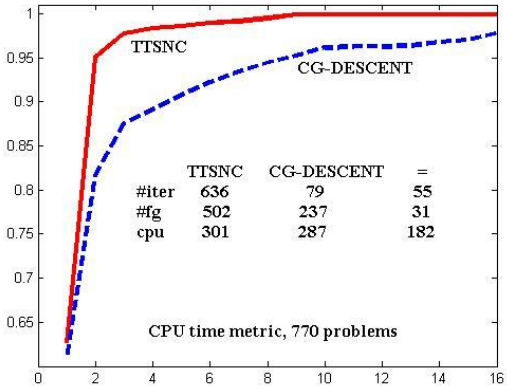


Fig. 1. TTS versus HS, DL (t=1), DY and PRP.

The algorithm is described in:
N. Andrei, *An accelerated subspace minimization three-term conjugate gradient algorithm for unconstrained optimization.* Numerical Algorithms, vol.65, (2014), pp.859-874.

		April, 5, 2013
29.	**TTSNC**	This is a variant of an accelerated subspace minimization three-term conjugate gradient algorithm for unconstrained optimization in which the three-term search direction is equal to the Newton direction. The search direction is computed as: $$d_{k+1} = -g_{k+1} + a_k s_k + b_k y_k ,$$ $$a_k = \frac{2(y_k^T g_{k+1})}{y_k^T s_k} - \frac{\omega_k - (y_k^T g_{k+1})}{y_k^T y_k},$$ $$b_k = \frac{1}{(y_k^T y_k)^2}\left[(y_k^T s_k)(\omega_k - y_k^T g_{k+1}) - (y_k^T g_{k+1})(y_k^T y_k)\right],$$ $$\omega_k = g_{k+1}^T y_k + \frac{(g_{k+1}^T y_k)(y_k^T y_k)}{y_k^T s_k} - \frac{(g_{k+1}^T s_k)(s_k^T y_k)}{s_k^T s_k}.$$ TTSNC is very close to TTS. The three-term conjugate gradient algorithm TTS is obtained as the minimization of the quadratic approximation model of function f in a subspace spanned by $-g_{k+1}, s_k$ and y_k. In this algorithm the searching direction is computed as above, where the parameters a_k and b_k are determined as: $$a_k = \frac{2(y_k^T g_{k+1} - s_k^T g_{k+1})}{y_k^T s_k} - \frac{\omega_k - y_k^T g_{k+1}}{y_k^T y_k},$$ $$b_k = \frac{(y_k^T s_k)(\omega_k - y_k^T g_{k+1})}{(y_k^T y_k)^2} - \frac{y_k^T g_{k+1} - s_k^T g_{k+1}}{y_k^T y_k}.$$ Observe the difference between these two formulae for a_k and b_k computation. It is the explicit presence of the term $s_k^T g_{k+1}$ into the formulae for a_k and b_k computation in TTS. The algorithm is presented in: **N. Andrei,** *Another three-term conjugate gradient algorithm for unconstrained optimization*. Technical Report, September 11, 2013 (Please see the file ttsnc.doc)



September 11, 2013

30. **THRCG2**

THRCG2 implements an accelerated conjugate gradient algorithm with three terms, that at each iteration both the descent and the conjugacy conditions are guaranteed.

The search direction is computes as:

$$d_{k+1} = -g_{k+1} - \delta_k s_k - \eta_k y_k,$$

$$\delta_k = \left(1 + v\frac{\|y_k\|^2}{y_k^T s_k}\right)\frac{s_k^T g_{k+1}}{y_k^T s_k} + \frac{y_k^T g_{k+1}}{y_k^T s_k},$$

$$\eta_k = \frac{s_k^T g_{k+1}}{y_k^T s_k},$$

$$v = \max\left\{1 - \frac{y_k^T s_k}{\|y_k\|^2}, 0\right\}.$$

The algorithm is described in:

N. Andrei, *A variant of three-term conjugate gradient algorithm for unconstrained optimization.* Technical Report, August 9, 2013

(Please see the file thrcg2.doc)

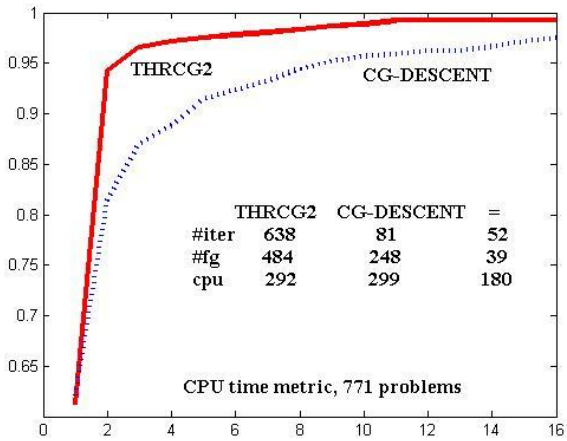
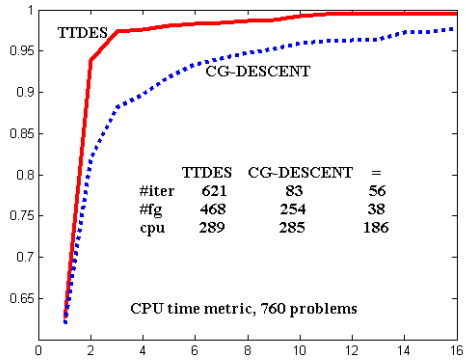


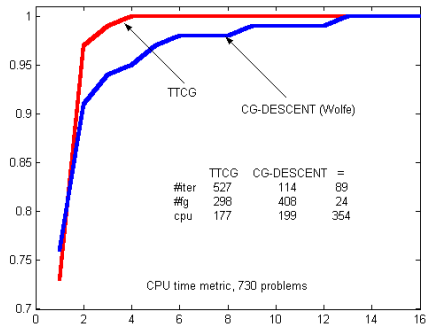
Fig. 1. THRCG2 versus CG-DESCENT

August 9, 2013

31.	**TTDES**	TTDES implements a three-term conjugate gradient algorithm obtained by minimizing the one-parameter quadratic model of the objective function in which the symmetrical approximation of the Hessian matrix satisfies the general quasi-Newton equation: $B_{k+1}s_k = \omega^{-1}y_k$, with $\omega \neq 0$. The search direction is computed as: $$d_{k+1} = -Q_{k+1}g_{k+1} = -g_{k+1} + \frac{y_k^T g_{k+1} - \omega s_k^T g_{k+1}}{y_k^T s_k} s_k - \frac{s_k^T g_{k+1}}{y_k^T s_k} y_k,$$ where $$\omega = \frac{2}{\|s_k\|^2}\sqrt{\|s_k\|^2\|y_k\|^2 - (y_k^T s_k)^2}.$$ This choice of the parameter ω makes the condition number of $$Q_{k+1} = I - \frac{s_k(y_k - \omega s_k)^T}{y_k^T s_k} + \frac{y_k s_k^T}{y_k^T s_k}$$ approach its minimum. The algorithm is described in: - **N. Andrei,** *A new three-term conjugate gradient algorithm for unconstrained optimization.* Numerical Algorithms, vol.68, (2015), pp.305-321. - **N. Andrei**, *Nonlinear Conjugate Gradient Methods for Unconstrained Optimization*, vol. 158 Springer Optimization and Its Applications, Springer, 2020, (Chapter 9, pp.334-345) (Please see the file: paper-ttdes.doc)  **October 24, 2013**
32.	**TTCG**	TTCG implements an accelerated conjugate gradient algorithm with three terms, that at each iteration both the descent and the conjugacy conditions are guaranteed. The search direction is computes as: $$d_{k+1} = -g_{k+1} - \delta_k s_k - \eta_k y_k,$$ $$\delta_k = \left(1 + 2\frac{\|y_k\|^2}{y_k^T s_k}\right)\frac{s_k^T g_{k+1}}{y_k^T s_k} + \frac{y_k^T g_{k+1}}{y_k^T s_k},$$

$$\eta_k = \frac{s_k^T g_{k+1}}{y_k^T s_k},$$

The algorithm is described in:
- **N. Andrei,** *On three-term conjugate gradient algorithms for unconstrained optimization.* Applied Mathematics and Computation, vol.219, 2013, pp.6316-6327.
(Please see the file AMC_17812.pdf)
- **N. Andrei**, *Nonlinear Conjugate Gradient Methods for Unconstrained Optimization*, vol. 158 Springer Optimization and Its Applications, Springer, 2020, (Chapter 9, pp.316-323)



March 14, 2012

33. **NADCG**

This program implements an adaptive conjugate gradient algorithm. The search direction is computed as the sum of the negative gradient and a vector determined by minimizing the quadratic approximation of the objective function at the current point. Using a special approximation of the inverse Hessian of the objective function, which depends by a positive parameter,a search direction is obtained which satisfies both the sufficient descent and the conjugacy conditions.
The search direction is computed as:

$$d_{k+1} = -g_{k+1} + \max\left\{\frac{y_k^T g_{k+1} - \omega_k s_k^T g_{k+1}}{y_k^T s_k}\right\} s_k - \frac{s_k^T g_{k+1}}{y_k^T s_k} y_k,$$

where

$$\omega_k = \begin{cases} 2\sqrt{\tau - 1}\dfrac{\|y_k\|}{\|s_k\|}, & a_k \geq \tau, \\ 2\sqrt{a_k - 1}\dfrac{\|y_k\|}{\|s_k\|} & a_k < \tau. \end{cases}$$

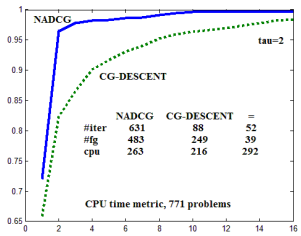
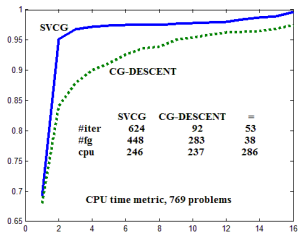
Here

$$a_k = \frac{\|y_k\|^2 \|s_k\|^2}{(y_k^T s_k)^2}.$$

The parameter $\tau > 1$ is the adaptive parameter. The algorithm is not very much sensitive the the values of τ.

The algorith is described in the paper:
N. Andrei, *A new adaptive conjugate gradient algorithm for large-*

		scale unconstrained optimization. Paper published into the book: „*Optimization and Applications in Control and Data Science*", edited by Boris Goldengorin. Springer Optimization and Its Applications, Vol.115. 2016, pp.1-16. This paper is written in honour of Prof. Boris T. Polyak celebrating his 80th anniversary. (Please see the file: nadcg.doc) **June 18, 2015**
34.	**ADCG**	An adaptive conjugate gradient algorithm for large-scale unconstrained optimization. The search direction is computed as $$d_{k+1} = -Q_{k+1} g_{k+1} = -g_{k+1} + \frac{(y_k - t_k s_k)^T g_{k+1}}{y_k^T s_k} s_k - \frac{s_k^T g_{k+1}}{y_k^T s_k} y_k,$$ where, $t_k \equiv \omega_k \lVert y_k \rVert^2 / y_k^T s_k$ is computed as: $$t_k = \begin{cases} 2\sqrt{\tau - 1}\dfrac{\lVert y_k \rVert}{\lVert s_k \rVert}, & \text{if } \dfrac{\lVert y_k \rVert^2 \lVert s_k \rVert^2}{(y_k^T s_k)^2} \geq \tau, \\ 0, & \text{otherwise,} \end{cases}$$ where $\tau > 1$ is a positive constant. The algorithm is described into the paper: **N. Andrei,** *An adaptive conjugate gradient algorithm for large-scale unconstrained optimization*, Journal of Computational and Applied Mathematics, 292 (2016), pp.83-91. **May 20, 2015**
35.	**EIGN-SING**	In this directory I placed the paper and the Fortran files: **N. Andrei**, *Eigenvalues versus singular values study in conjugate gradient algorithms for large-scale unconstrained optimization.* Technical Report, July 14, 2015. (See the file: paper10.doc) The Fortran packages SVCG.FOR and NADCG.FOR implements the singular value approach and eigenvalues approach in conjugate gradient algorithms, respectively. Directory MINPACK contains 5 applications from MINPACK-II collection. **1)** The NADCG algorithm implements the eigenvalues clustering in conjugate gradient algorithms. The search direction is computed as: $$d_{k+1} = -g_{k+1} + \max\left(\frac{y_k^T g_{k+1} - \omega_k s_k^T g_{k+1}}{y_k^T s_k}, 0\right) s_k - \frac{s_k^T g_{k+1}}{y_k^T s_k} y_k.$$ where $$\omega_k = \begin{cases} 2\sqrt{\tau - 1}\dfrac{\lVert y_k \rVert}{\lVert s_k \rVert}, & a_k \geq \tau, \\ 2\sqrt{a_k - 1}\dfrac{\lVert y_k \rVert}{\lVert s_k \rVert} & a_k < \tau. \end{cases}$$ Here $$a_k = \frac{\lVert y_k \rVert^2 \lVert s_k \rVert^2}{(y_k^T s_k)^2}.$$ The parameter $\tau > 1$ is the adaptive parameter. The algorithm is not

		very much sensitive the the values of τ. **2)** The SVCG algorithm implements the singular values approach (minimizing the comdition number) in conjugate gradient algorithms. The search direction is computed as: $$d_{k+1} = -g_{k+1} + \frac{y_k^T g_{k+1}}{y_k^T s_k} s_k - \frac{s_k^T g_{k+1}}{y_k^T s_k} y_k .$$ Observe that this is a modification of the Hestenes and Stiefel conjugate gradient algorithm. Some comparisons of these algorithms versus CG-DESCENT by Hager and Zhang, using 800 unconstrained optimization test problems, are as follows:   Please, see the paper: **N. Andrei**, *Eigenvalues versus singular values study in conjugate gradient algorithms for large-scale unconstrained optimization.* Optimization Methods and Software, vol. 32, no. 3, 2017, pp. 534-551. **July 14, 2015**
36.	**ACGSSV**	An adaptive class of nonlinear conjugate gradient algorithms is suggested. The search direction in these algorithms is given by symmetrization of the scaled Perry conjugate gradient direction [A. Perry, *A modified conjugate gradient algorithm.* Operations Research, 26 (1978) 1073-1078], which depends by a positive parameter. The value of this parameter is determined by minimizing the distance between the symmetrical scaled Perry conjugate gradient search direction matrix and the self-scaling memoryless BFGS update by Oren in the Frobenius norm. Two variants of the parameter in the search direction are presented as those given by: Oren and Luenberger [S.S. Oren, D. G. Luenberger, *Self-scaling variable metric (SSVM) algorithms. I. Criteria and sufficient conditions for scaling a class of algorithms*. Management Sci., 20 (1973/74) 845-862] and Oren and Spedicato [S.S. Oren, E. Spedicato, *Optimal conditioning of self-scaling variable metric algorithms*. Math. Program., 10 (1976) 70-90]. The corresponding algorithm, ACGSSV, is equipped with a vey well known acceleration scheme of conjugate gradient algorithms. The algorithm is described in the paper: **N. Andrei,** *Accelerated adaptive Perry conjugate gradient algorithms based on the self-scaling memoryless BFGS update.* Journal of Computational and Applied Mathematics, vol. 325, 2017, pp.149-164. (Please see the file JCAM-2017 (40).pdf) **January 16, 2017**
37.	**DLE**	A new value for the parameter in Dai and Liao conjugate gradient algorithm is presented. This is based on the clustering the eigenvalues of the matrix which determine the search direction of this algorithm.

This value of the parameter lead us to a variant of the Dai and Liao algorithm which is more efficient and more robust than the variants of the same algorithm based on the minimizing the condition number of the matrix associated to the search direction.
Babaie-Kafaki and Ghanbari [1] suggested two choices for the scaling parameter:

$$t_{k1}^* = \frac{y_k^T s_k}{\|s_k\|^2} + \frac{\|y_k\|}{\|s_k\|} \text{ (M1) and } t_{k2}^* = \frac{\|y_k\|}{\|s_k\|} \text{ (M2)}$$

In this paper I suggest the following value:

$$t_k^* = \frac{y_k^T s_k}{\|s_k\|^2}.$$

[1] Babaie-Kafaki, S., Ghanbari, R.: The Dai-Liao nonlinear conjugate gradient method with optimal parameter choices. European Journal of Operational Research 234, 625-630 (2014)

The algorithm is described in:
N. Andrei, (2018). *A Dai-Liao conjugate gradient algorithm with clustering the eigenvalues.* Numerical Algorithms, *77*(4), 1273-1282.
(Please see the file: Paper332R1.doc)

January 4, 2017

38. **CGLIN**

Solving linear algebraic systems with positive definite matrices using the linear conjugate gradient method.
Linear Conjugate Gradient Algorithm

1.	Select an initial point x_0 and $\varepsilon > 0$ sufficiently small
2.	Set $r_0 = Ax_0 - b,\ d_0 = -r_0$ and $k = 0$
3.	If $\|r_k\| \le \varepsilon,$ then stop. Otherwise continue with step 4
4.	Compute: $\alpha_k = \frac{r_k^T r_k}{d_k^T A d_k},\ x_{k+1} = x_k + \alpha_k d_k,\ r_{k+1} = r_k + \alpha_k A d_k,\ \beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k},$ $d_{k+1} = -r_{k+1} + \beta_k d_k$
5.	Set $k = k+1$ and continue with step 3 ♦

The linear algebraic system $Ax = b,$ where:

$$A = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}, \text{ and } b = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix},$$

is obtained from the finite difference numerical method to discretize the one-dimensional Poisson equation.
For $n = 1000,$ the linear conjugate gradient algorithm gives a solution in 500 iterations. Figure 1 shows the evolution of the error $\|b - Ax_k\|$ along the iterations for obtaining a solution with accuracy less than or

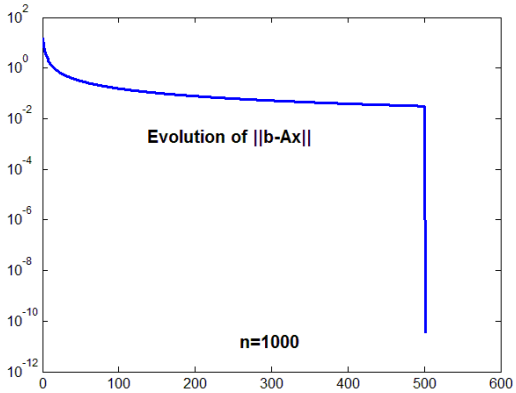
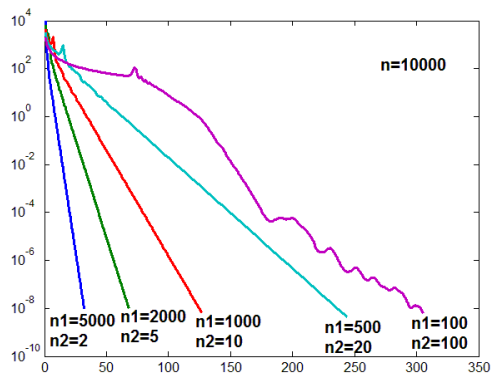
		equal to 10^{-8}.  **Fig. 1.** Evolution of the error $\|b - Ax_k\|$ See Chapter 2 of the book: N. Andrei, "*Nonlinear Conjugate Gradient Methods for Unconstrained Optimization*", Spriger, 2019. Example 2.1. **January 2, 2019**
39.	**CG4**	Program for solving linear algebraic systems $Ax = b$ obtained from the finite difference numerical method to discretize the two-dimensional Poisson equation. The matrix A has n_2 blocks B on the main diagonal, where each block $B \in \mathbb{R}^{n_1 \times n_1}$. Hence, $A \in \mathbb{R}^{n \times n}$, where $n = n_1 n_2$. Considering $n = 10{,}000$, the evolution of error $\|b - Ax_k\|$ computed by the linear conjugate gradient algorithm for five different values of n_1 and n_2 is presented in Figure 1. From Figure 1, for $n_1 = 5000$ and $n_2 = 2$, that is when there are only two blocks on the main diagonal of A, the linear conjugate gradient algorithm needs only 31 iterations. Therefore, the convergence is faster. On the other hand, when $n_2 = 100$, i.e. there are 100 blocks on the main diagonal of matrix A, then the algorithm needs 304 iterations. In other words, the smaller the number of blocks on the main diagonal of matrix A, the faster the convergence. 

Fig. 1. Evolution of the error $\|b - Ax_k\|$ of the linear conjugate gradient algorithm for different numbers (n_2) of blocks on the main diagonal of matrix A.

Please, see the Books:

1. **N. Andrei**, *Nonlinear Conjugate Gradient Methods for Unconstrained Optimization*. Springer, vol. 158 Springer Optimization and Its Applications, Springer, 2020.
2. **N. Andrei**, *Optimizare fără Restricţii – Metode de Direcţii Conjugate*, MATRIXROM, Bucureşti, 2000, pp. 78-79 şi 109-112.

(first version) **March 15, 1999**
(modified version) **January 3, 2019**

40. **ACGSYS (See: CGSYS)**

ACGSYS is the accelerated version of CGSYS. This is a subroutine dedicated to compute the minimizer of a differentiable function with a large number of variables.
The search direction of this algorithm is a linear combination of $-g_{k+1}$ and s_k, where the coefficients in this linear combination are computed in such a way that both the descent and the conjugacy conditions to be guaranteed at every iteration $k \geq 1$.

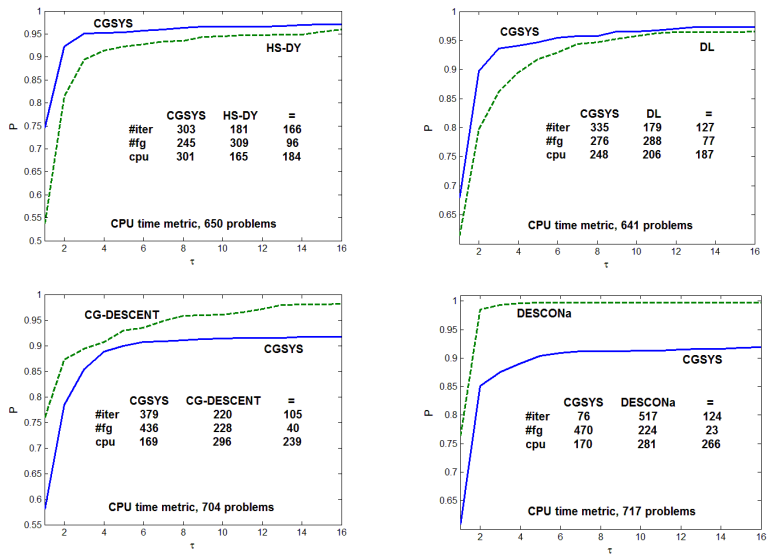


Fig. 1. Performance profiles of CGSYS versus HS-DY, DL ($t = 1$), CG-DESCENT and DESCONa

The accelerated version of CGSYS is described in the paper: **N. Andrei,** *An accelerated conjugate gradient algorithm with guaranteed descent and conjugacy conditions for unconstrained optimization*. (cgsyspap.doc). March 6, 2009.

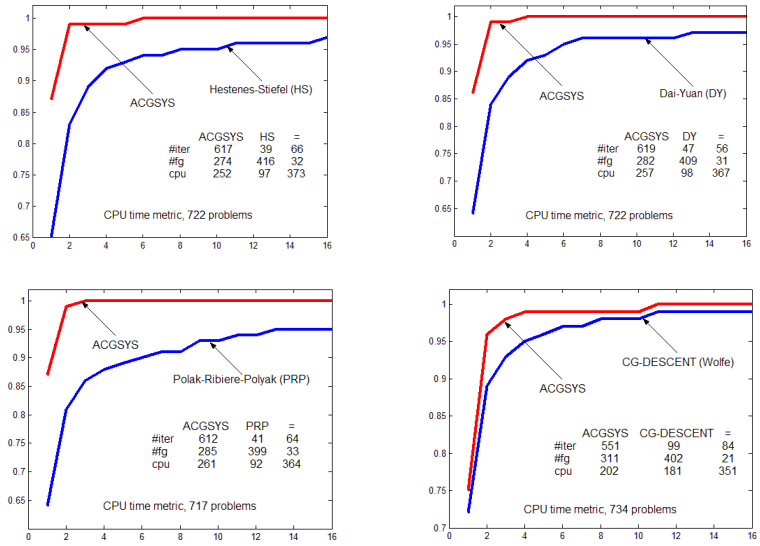


Fig. 2. ACGSYS versus HS, DY, PRP , CG-DESCENT

Also, please, see the book:
N. Andrei, *Nonlinear Conjugate Gradient Methods for Unconstrained Optimization*. Springer, vol. 158 Springer Optimization and Its Applications, Springer, 2020. (Chapter 11)

October 24, 2008

41. **CGSYSLBs**

Combination of the CGSYS algorithm with the limited memory L-BFGS algorithm by interlacing iterations of the CGSYS with iterations of the L-BFGS algorithms. In this algorithm, we called CGSYSLBs, the iterations of CGSYS are performed only if the stepsize is less or equal to a prespecified threshold. Otherwise, the iterations of L-BFGS ($m=5$) are performed. CGSYSLBsa is the accelerated version of CGSYSLBs.
Comparisons of CGSYSLBsa versus CGSYS and CG-DESCENT (version 1.4), Figure 1, illustrate that CGSYSLBsa is more robust than these algorithms.

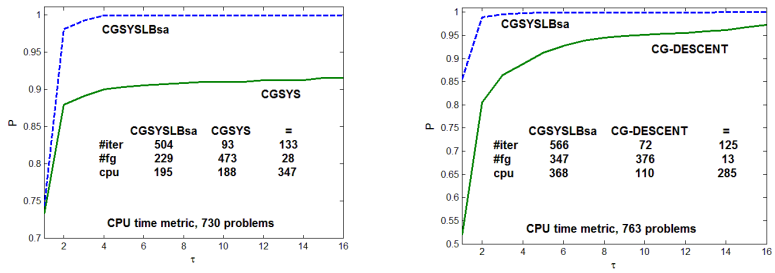


Fig. 1. CGSYSLBs versus CGSYS and versus CG-DESCENT

The program implements 81 unconstrained optimization test problems. The name of the minimizing functions is given in FUN80.TXT. The last problem is PALMER1C (ill-conditioned problem)
Comparisons of CGSYSLBsa versus DESCONa and versus DK+w are

given in Figure 2.

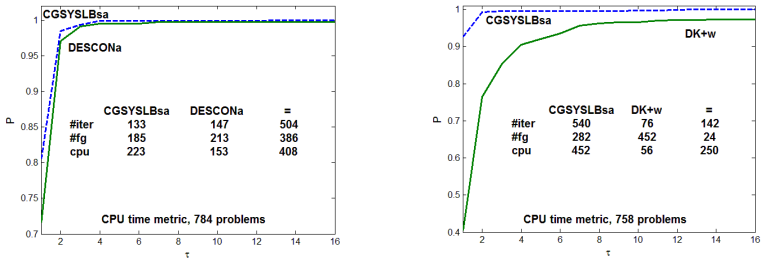


Fig. 2. Conmparisons of CGSYSLBsa versus DESCONa and versus DK+w.

Please, see the book:
- **N. Andrei**, *Nonlinear Conjugate Gradient Methods for Unconstrained Optimization*, vol. 158 Springer Optimization and Its Applications, Springer, 2020. (Chapter 11)

June 17, 2019

42. **CGSYSLBq**

Combination of the CGSYS algorithm with the limited memory L-BFGS algorithm by interlacing iterations of the CGSYS with iterations of the L-BFGS algorithms subject to the closeness of the minimizing function to a quadratic. Compute the parameter:

$$t_k = \left| \frac{2(f_k - f_{k+1} + g_{k+1}^T s_k)}{y_k^T s_k} - 1 \right|.$$

If t_k is close to zero, then φ_k is regarded as a quadratic function, otherwise not. In other words, if $t_k \le c$, where c is a small positive constant ($c = 10^{-8}$), we can conclude that φ_k is close to a quadratic function. If $t_k \ge c$, then the CGSYS iterations are performed, otherwise the L-BFGS ($m = 5$) iterations are considered. In this algorithm, we called CGSYSLBq, the iterations of CGSYS are performed only if $t_k \ge c$. Otherwise, the iterations of L-BFGS ($m = 5$) are performed. CGSYSLBqa is the accelerated version of CGSYSLBq.

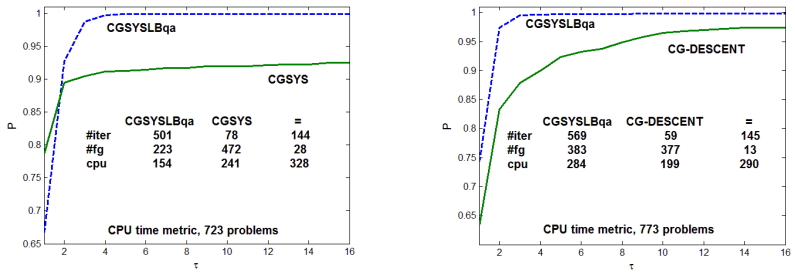


Fig. 1. CGSYSLBqa versus CGSYS and versus CG-DESCENT

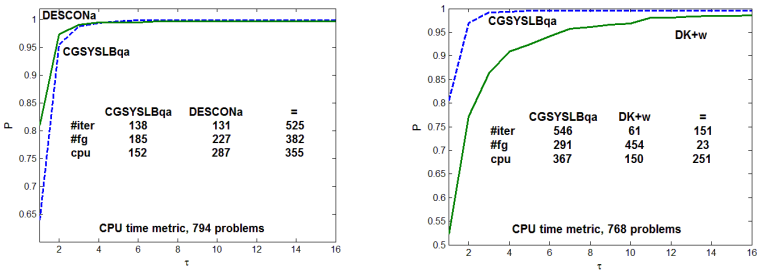
<table>
<tr>
<td></td>
<td></td>
<td>


Fig. 2. CGSYSLBqa versus DESCONa and versus DK+w

Please, see the book:
- N. Andrei, Nonlinear Conjugate Gradient Methods for Unconstrained Optimization, vol. 158 Springer Optimization and Its Applications, Springer, 2020, (Chapter 11)

June 17, 2019
</td>
</tr>
<tr>
<td>43.</td>
<td>CGSYSLBo</td>
<td>
Combination of the CGSYS algorithm with the limited memory L-BFGS algorithm by interlacing iterations of the CGSYS with iterations of the L-BFGS algorithms subject to the orthogonality of the current gradient to the previous search direction. In other words, in our algorithm we call CGSYSLBo the CGSYS and L-BFGS methods are combined as follows: if $g_{k+1}^T d_k \le c$, where c is a small positive constant ($c = 10^{-5}$), then the CGSYS iterations are performed, otherwise the L-BFGS ($m = 5$) iterations are considered.

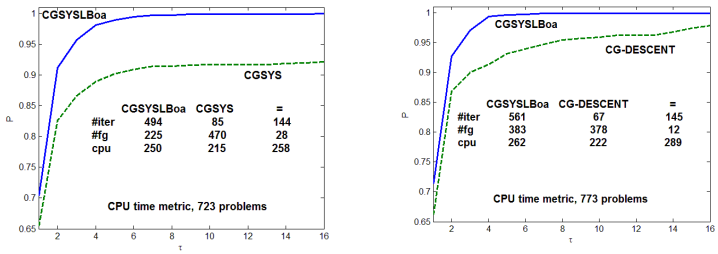


Fig. 1. Performance profiles of CGSYSLBoa versus CGSYS and versus CG-DESCENT

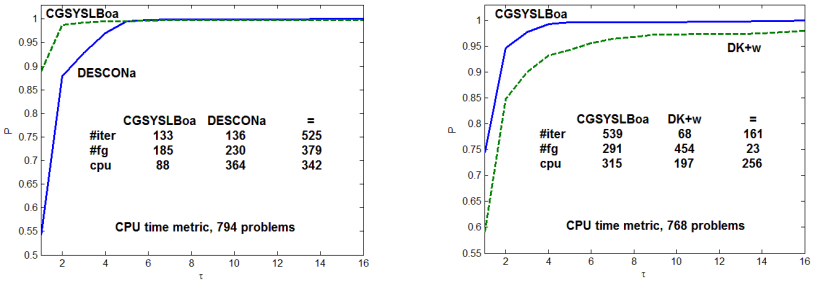


Fig. 2. Performance profiles of CGSYSLBoa versus DESCONa and versus DK+w

Please, see the book:
- N. Andrei, Nonlinear Conjugate Gradient Methods for Unconstrained Optimization, vol. 158 Springer Optimization and Its
</td>
</tr>
</table>

		Applications, Springer, 2020, (Chapter 11) **January 18, 2020**
44.	**CG3LS**	Fortran program for unconstrained optimization using 6 procedures for computation of the conjugate parameter β_k : Hager-Zhang, Dai-Kou, Hestenes-Stiefel, Polak-Ribière-Polyak, Dai-Yuan and minimizing the measure function φ of Byrd and Nocedal, under the 3 line search procedures: standard Wolfe, approximate Wolfe of Hager and Zhang and improved Wolfe of Dai and Kou. The program is that of Hager and Zhang (CG-DESCENT), where the formula for beta computation is modified as that given by Dai and Kou, or HS, PRP, DY, FI. Please see the Book: **N. Andrei,** *Nonlinear Conjugate Gradient Methods for Unconstrained Optimization*. Springer, vol. 158 Springer Optimization and Its Applications, Springer, 2020. (Chapter 8) **January 17, 2019**
45.	**CG3LSpre**	Fortran program for unconstrained optimization using 6 procedures for computation of the conjugate parameter β_k : PRECONDITIONED Hager-Zhang, Dai-Kou, Hestenes-Stiefel, Polak-Ribière-Polyak, Dai-Yuan and minimizing the measure function φ of Byrd and Nocedal, under the 3 line search procedures: standard Wolfe, approximate Wolfe of Hager and Zhang and improved Wolfe of Dai and Kou. Only the conjugate gradient parameter β_k of Hager and Zhang algorithm is preconditioned with a diagonal approximation of the Hessian. The program is that of Hager and Zhang (CG-DESCENT), where the formula for beta computation is modified as that given by Dai and Kou, or HS, PRP, DY, FI. Please see the Book: **N. Andrei,** *Nonlinear Conjugate Gradient Methods for Unconstrained Optimization*. Springer, vol. 158 Springer Optimization and Its Applications, Springer, 2020. (Chapter 1 for diagonal approximation to the Hessian and Chapter 8) **January 17, 2019**
46.	**CGALLpre**	30 conjugate gradient unconstrained optimization algorithms with standard Wolfe line search. The following conjugate gradient algorithms are implemented: betatype = (1) HS, (2) FR, (3) PRP, (4) PRP+, (5) CD, (6) LS, (7) DY, (8) DL(t=1), (9) DL+, (10) SDC, (11) hDY, (12) hDY0, (13) GN, (14) HuS, (15) TAS, (16) LS-CD, (17) Birgin-Martinez, (18) Birgin-Martinez+, (19) scaledPRP, (20) scaledFR, (21) new cg from PRP, (22) newDY, (23) variant of newDY, (24) another variant of newDY, (25)

		Hager-Zhang, (26) Hager-Zhang preconditioned, (27) Dai-Kou, (28) Dai-Kou preconditioned, (29) PRP preconditioned, (30) Hager-Zhang SSML-BFGS preconditioned The algorithms corresponding to betatype = 26, 28, 29 and 30 are preconditioned conjugate gradient algorithnms, where the preconditioner is computed as a diagonal approximation to the Hessian. Please see the Book: - **N. Andrei**, *Nonlinear Conjugate Gradient Methods for Unconstrained Optimization*, vol. 158 Springer Optimization and Its Applications, Springer, 2020. For betatype = 30 the Hager-Zhang algorithmmis preconditioned with the Self-Scaling Limited Memory C BFGS updation to the Hessian. The preconditioner is given by the self-scaling memoryless BFGS update of Perry and Shanno (8.104), and the scaling parameter tau is computed as in (8.111). Please see the Book: - **N. Andrei**, *Nonlinear Conjugate Gradient Methods for Unconstrained Optimization*, vol. 158 Springer Optimization and Its Applications, Springer, 2020. (Please, see Chapter 8 for self-scaling memoryless BFGS update of Perry and Shanno, and Chapter 10 for preconditioning.) *Criticism of preconditioning:* See the Book: - **N. Andrei**, *Nonlinear Conjugate Gradient Methods for Unconstrained Optimization*, vol. 158 Springer Optimization and Its Applications, Springer, 2020, (Chapter 10) The search direction can be computed as: 1) d(i) = -r(i) + beta * d(i), beta=0, i.e. d(i)=-r(i) In this case we have a quasi-Newton method in which the inverse approximation to the Hessian is computed as a self-scaled memoryless BFGS update. 2) d(i) = -r(i) + beta * d(i) In this case we have a preconditioned conjugate gradient method. Refering to Chapter 10. Finally, we emphasize that in 2) there must be a balance concerning the quality of the preconditioner (i.e. the closeness to the inverse Hessian), namely, if the definition of the preconditioner contains useful information about the inverse Hessian of the objective function, it is better to use the search direction d=-Pg , since the addition of the last term beta*d may prevent d=-Pg+beta*d from being an efficient descent direction, unless the line search is sufficiently accurate. Compare HZSS.rez where the search direction is computed as in 1) above, i.e d(i)=-r(i) versus HZSSc.rez where the search direction is computed as in 2), i.e. d(i) = -r(i) + beta * d(i). HZSS is better. (May 21, 2019)

Figure 1 shows the performance profiles of HZ+pa (accelerated version of HZ+p, where the acceleration is as in Remark 5.1) in which the search direction is computed as

$$d_{k+1} = -P_{k+1} g_{k+1} + \bar{\beta}_k^{HZ+} d_k,$$

where

$$P_{k+1} = \frac{1}{\tau_k}\left(I - \frac{s_k y_k^T + y_k s_k^T}{y_k^T s_k} \right) + \left(1 + \frac{1}{\tau_k} \frac{\|y_k\|^2}{y_k^T s_k} \right) \frac{s_k s_k^T}{y_k^T s_k},$$

with $\tau_k = \tau_k^{OL}$ and $\bar{\beta}_k^{HZ+}$ is computed as in (10.8) versus the performances of the accelerated self-scaling memoryless BFGS (SSML-BFGSa) update in which the search direction is computed as

$$d_{k+1} = -P_{k+1} g_{k+1},$$

where P_{k+1} is given above. See Chapter 10 of the Book.

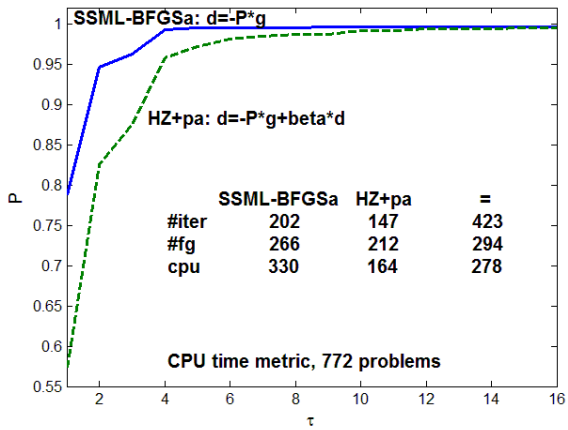


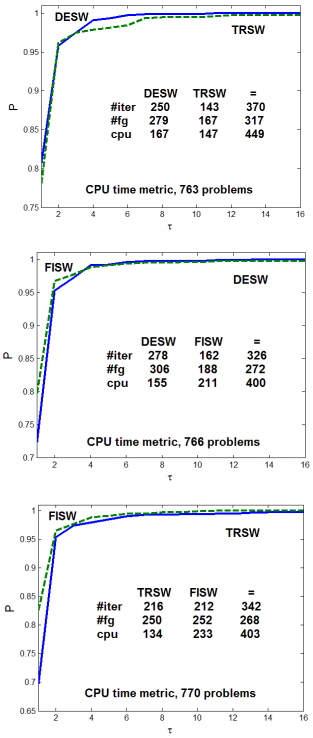
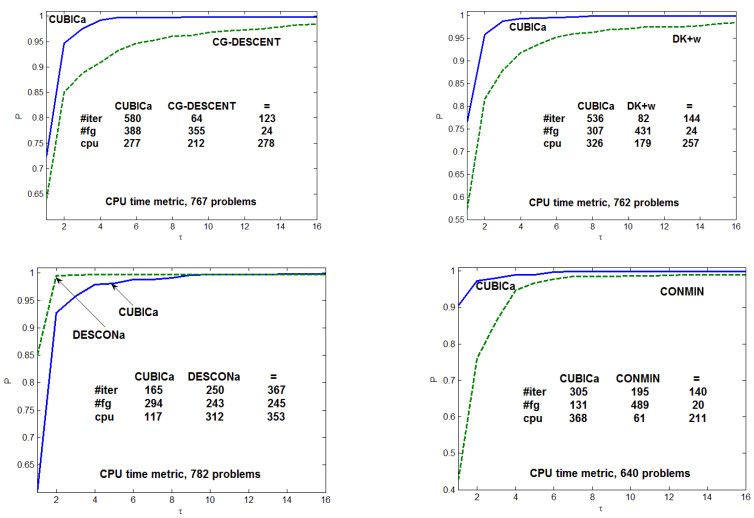
Fig. 1. Performance profiles of HZ+pa versus SSML-BFGSa

May 21, 2019

47. **CG3x8**

The program is a modification of the CG-DESCENT of Hager and Zhang (2005) to include different formulae for parameter beta computation under the three line search conditions: standard Wolfe, Approximate Wolfe and Improved Wolfe.
The program is that of Hager and Zhang (CG-DESCENT, version 1.4), where the formula for beta computation is modified as:
(1) Hager and Zhang (2005)
(2) Minim DETERMINANT (Andrei, Thechnical Report No.2/2019)
(3) Minim TRACE (Andrei, Thechnical Report No.2/2019)
(4) Minim Fi - measure function of Byrd and Nocedal
(5) Hestenes - Stiefel
(6) Dai - Yuan
(7) Polak-Ribiere-Polyak
(8) Minim of combination of DETERMINANT and TRACE

Please see: **N. Andrei**, *Conjugate Gradient Algorithms Closest to Self-Scaling Memoryless BFGS Method based on clustering the eigenvalues of the self-scaling memoryless BFGS iteration matrix or on minimizing the Byrd-Nocedal measure function with Different Wolfe Line Searches for*

		Unconstrained Optimization. Technical Report No.2/2019, Academy of Romanian Scientists, April 18, 2019. (TRR2-2019.doc)  **Fig. 1.** Performance profiles of DESW versus TRSW, of DESW versus FISW and of TRSW versus FISW **April 18, 2019**
48.	**CUBIC**	A variant of the conjugate gradient algorithm with subspace minimization based on the regularization model. The algorithm combines the minimization of a p-regularized model of the minimizing function with the subspace minimization.  **Fig. 1.** Performance profiles of CUBICa versus CG-DESCENT, DK+w, DESCONa and CONMIN

		(CUBICa is the accelerated version of CUBIC.) See the Book: - **N. Andrei**, *Nonlinear Conjugate Gradient Methods for Unconstrained Optimization*, vol. 158 Springer Optimization and Its Applications, Springer, 2020, (Chapter 11, Section 11.4)
49.	**CGSECM**	Conjugate gradient algorithm based on the equality of the Newton direction with the conjugate gradient direction and modified secant condition. The search direction is computed as: $$d_{k+1} = -g_{k+1} + \beta_k s_k,$$ $$\beta_k = \max\left\{\frac{y_k^T g_{k+1}}{y_k^T s_k + \delta\eta}, 0\right\} - \left(1 - \frac{\delta\eta}{s_k^T s_k}\right)\frac{s_k^T g_{k+1}}{y_k^T s_k + \delta\eta},$$ $$\eta = 6(f_k - f_{k+1}) + 3(g_k + g_{k+1})^T s_k,$$ $$\delta = \begin{cases} 0, \\ 1 \end{cases} \text{ a parameter.}$$ **February 12, 2008**
50.	**DESCON14**	Performances of DESCON package for solving 14 applications of unconstrained optimization. The applications are as follows: 1. Weber Function (1) (Andrei, U71) 2. Enzyme reaction (Andrei, U79) (A) 3. Solution of a chemical reactor (A) 4. Robot kinematics problem (A) 5. Solar Spectroscopy (A) 6. Estimation of parameters (A) 7. Propan combustion in air (A) 8. Gear train with minimum inertia (A) 9. Human Heart Dipole. Andrei U84, pp.65 10. Neurophysiology (A) 11. Combustion application (A) 12. Thermistor (A) 13. Optimal design of a Gear Train (A) 14. Circuit design (A) Directory DESCON14: - DESCON14.FOR (Fortran package with all subroutines.) - FUNC14.TXT (Name of the applications) - R2020T14.DOC (Technical Report.) The performances of DESCON14 are presented in: **N. Andrei,** *Numerical experiments with DESCON for solving 14 applications of unconstrained optimization*. AOSR – Academy of Romanian Scientists, Bucharest, Romania, Technical Report No.14/2020, June 3, 2020. (Romanian Academy Library) (10 pages) **June 3, 2020**
51.	**CUBIC14**	Performances of CUBIC package for solving 14 applications of unconstrained optimization. The applications are as follows:

		1. Weber Function (1) (Andrei, U71) 2. Enzyme reaction (Andrei, U79) (A) 3. Solution of a chemical reactor (A) 4. Robot kinematics problem (A) 5. Solar Spectroscopy (A) 6. Estimation of parameters (A) 7. Propan combustion in air (A) 8. Gear train with minimum inertia (A) 9. Human Heart Dipole. Andrei U84, pp.65 10. Neurophysiology (A) 11. Combustion application (A) 12. Thermistor (A) 13. Optimal design of a Gear Train (A) 14. Circuit design (A) Directory CUBIC14: - CUBIC14.FOR (Fortran package with all subroutines.) - FUNC14.TXT (Name of the applications) - R2020T15.DOC (Technical Report.) The performances of CUBIC14 are presented in: N. Andrei, *Numerical experiments with CUBIC for solving 14 applications of unconstrained optimization.* AOSR – Academy of Romanian Scientists, Bucharest, Romania, Technical Report No.15/2020, June 3, 2020. (Romanian Academy Library) (11 pages) **June 3, 2020**
52.	**CG DESCENT14**	Performances of CG-DESCENT package for solving 14 applications of unconstrained optimization. The applications are as follows: 1. Weber Function (1) (Andrei, U71) 2. Enzyme reaction (Andrei, U79) (A) 3. Solution of a chemical reactor (A) 4. Robot kinematics problem (A) 5. Solar Spectroscopy (A) 6. Estimation of parameters (A) 7. Propan combustion in air (A) 8. Gear train with minimum inertia (A) 9. Human Heart Dipole. Andrei U84, pp.65 10. Neurophysiology (A) 11. Combustion application (A) 12. Thermistor (A) 13. Optimal design of a Gear Train (A) 14. Circuit design (A) Directory CGDESCENT14: - CGDESCENT14.FOR (Fortran package with all subroutines.) - FUNC14.TXT (Name of the applications) - R2020T16.DOC (Technical Report.) The performances of CGDESCENT14 are presented in: N. Andrei, *Numerical experiments with CG-DESCENT for solving 14 applications of unconstrained optimization.* AOSR – Academy of

Romanian Scientists, Bucharest, Romania, Technical Report No.16/2020, June 3, 2020. (Romanian Academy Library) (11 pages)

The Technical Report 17/2020: N. Andrei, *Comparison of modern conjugate gradient methods: DESCON, CUBIC, CG-DESCENT (4.1) for solving 14 small-scale applications of unconstrained optimization*, presents a comparison among the performances of DESCON14, CUBIC14 and CGDESCENT14 for solving 14 applications of unconstrained optimization.

Performances of DESCON14

n	iter	fgcnt	time(c)	fx*	gnorm	Name of Application
2	1878	10001	0	-0.2644531414650E+03	0.8208740831576E+00	1. Weber Function (Andrei, U71)
4	48	143	0	0.3075056038514E-03	0.6886760990097E-08	2. Enzyme reaction (Andrei, U79) (A)
6	85	264	0	0.9665994663683E-15	0.4231231612938E-07	3. Solution of a chemical reactor (A)
8	1843	10006	1	0.5463981044793E-05	0.1781782618260E-02	4. Robot kinematics problem (A)
4	12	38	0	0.8312307692553E+01	0.8585722387648E-07	5. Solar Spectroscopy (A)
4	46	150	0	0.3185717881375E-01	0.6936429307668E-08	6. Estimation of parameters (A)
5	724	2246	0	0.1224151943762E-06	0.8166044868424E-07	7. Propan combustion in air (A)
2	14	154	0	0.1751192213346E+01	0.7760986494009E-07	8. Gear train with minimum inertia (A)
8	1916	10002	1	0.1120571259805E-01	0.1686188462847E-03	9. Human Heart Dipole. Andrei U84, pp.65
6	93	632	1	0.4539057615171E+01	0.4484463269794E-07	10. Neurophysiology (A)
10	51	142	0	0.6898812079492E-10	0.8219103504792E-07	11. Combustion application (A)
3	1839	10005	10	0.1726024568705E+03	0.1334938231384E+02	12. Thermistor (A)
4	1842	10004	0	0.2387780742094E-02	0.8995056760928E-04	13. Optimal design of a Gear Train (A)
9	739	2166	2	0.1454860731888E-13	0.1554041459749E-06	14. Circuit design (A)
TOTAL	11130	55953	15.00 centeseconds			

Date: --- Month: 6 Day: 3 Year: 2020

Performances of CUBIC14

n	iter	fgcnt	time(c)	fx	gnorm	Name of Applications
2	1288	5001	1	-0.2643790043149E+03	0.6876989703321E+00	1. Weber Function (Andrei, U71)
4	39	116	0	0.3075056060090E-03	0.1140193575396E-06	2. Enzyme reaction (Andrei, U79) (A)
6	94	287	0	0.7468342084540E-15	0.4147895225729E-07	3. Solution of a chemical reactor (A)
8	333	5017	1	0.4829469121831E+00	0.4942442115598E+00	4. Robot kinematics problem (A)
4	10	31	0	0.8312307695614E+01	0.7104616347572E-06	5. Solar Spectroscopy (A)
4	30	96	0	0.3187570933023E-01	0.5862905229454E-06	6. Estimation of parameters (A)
5	555	1670	0	0.4799163454696E-05	0.1371052281519E-05	7. Propan combustion in air (A)
2	11	138	0	0.1751192330768E+01	0.9461910026439E-06	8. Gear train with minimum inertia (A)
8	940	5006	1	0.1199116073210E-01	0.8786818146807E-01	9. Human Heart Dipole. Andrei U84, pp.65
6	24	79	0	0.4539057615171E+01	0.3989872629134E-08	10. Neurophysiology (A)
10	50	139	0	0.4967405721874E-09	0.3630146049766E-06	11. Combustion application (A)
3	395	5003	6	0.1721497388951E+03	0.2361553857583E+01	12. Thermistor (A)
4	7	101	0	0.2322924674570E-04	0.7867065011382E-06	13. Optimal design of a Gear Train (A)
9	563	1639	3	0.8544333431670E-14	0.2890489586180E-06	14. Circuit design (A)
TOTAL	4339	24323	12.00 centeseconds			

Date: ---> Month: 6 Day: 3 Year: 2020

Performances of CG-DESCENT14(w)

n	iter	fgcnt	time(c)	fx	gnorm	Name of Applications
2	130	526	0	-0.2644531414650E+03	0.4360555021096E+00	1. Weber Function (Andrei, U71)
4	87	183	0	0.3075057506207E-03	0.9351232549738E-06	2. Enzyme reaction (Andrei, U79) (A)
6	242	531	0	0.1546034033470E-11	0.8328999653862E-06	3. Solution of a chemical reactor (A)
8	13	79	0	0.1045002080991E-04	0.2937120722379E-02	4. Robot kinematics problem (A)
4	34	73	0	0.6872367741557E+01	0.3753421634575E-06	5. Solar Spectroscopy (A)
4	638	1436	0	0.3194075831746E-01	0.9984546877919E-06	6. Estimation of parameters (A)
5	9001	18039	2	0.1327993904766E-03	0.3013599273355E-03	7. Propan combustion in air (A)
2	14	86	0	0.1745268282541E+01	0.6851854457169E-01	8. Gear train with minimum inertia (A)
8	2	57	0	0.1790818193032E+00	0.3756017838954E-01	9. Human Heart Dipole. Andrei U84, pp.65
6	39	100	0	0.4539057615171E+01	0.3103946255578E-07	10. Neurophysiology (A)
10	55	114	0	0.1279714516413E-09	0.1142557208812E-06	11. Combustion application (A)
3	32	462	0	0.1721680788246E+03	0.2931072079241E+03	12. Thermistor (A)
4	1	56	0	0.1743310601795E-01	0.5889113990961E-03	13. Optimal design of a Gear Train (A)
9	7485	15457	12	0.2419744215211E-10	0.8313476443084E-06	14. Circuit design (A)
TOTAL	17773	37199	14.00 centeseconds			

Date: --- Month: 6 Day: 4 Year: 2020

Line Search with Wolfe conditions

Performances of DESCON, CUBIC and CG-DESCENT

	iter	fgcnt	time
DESCON14	11130	55953	15
CUBIC14	4339	24323	12
CG-DESCENT14(w)	17773	37199	14
CG-DESCENT14(aw)	17773	37199	13

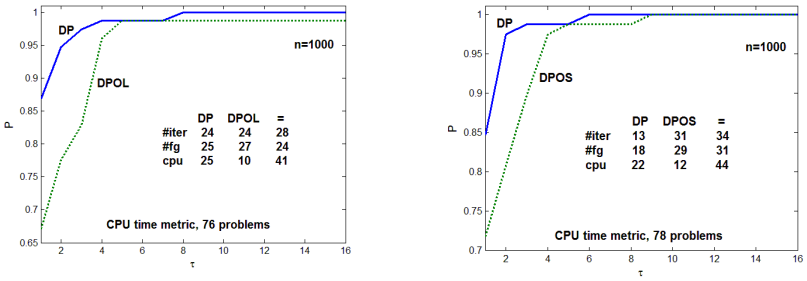
(w) – Wolfe line search
(aw) – approximate Wolfe line search

June 4, 2020

<><><><><><><><><><>

BFGS - MODIFIED

1.	**SBFGS**	A new adaptive scaled BFGS method for unconstrained optimization is presented. The third term in the standard BFGS update formula is scaled in order to reduce the large eigenvalues of the approximation to the Hessian of the minimizing function. Under the inexact Wolfe line search conditions, the global convergence of the adaptive scaled BFGS method is proved in very general conditions without assuming the convexity of the minimizing function. The algorithm is described in the paper: **N. Andrei,** *An adaptive scaled BFGS method for unconstrained optimization.* Numerical Algorithms, DOI: 10.1007/s11075-017-0321-1 (Please see the file: mbfgs-R2.doc) **March 18, 2017**
2.	**TPSBFGS**	A double parameter scaled BFGS method for unconstrained optimization is presented. In this method, the first two terms of the known BFGS update formula are scaled with a positive parameter while the third one is scaled with another positive parameter. These parameters are selected in such a way as to improve the eigenvalues structure of the BFGS update. The parameter scaling the first two terms of the BFGS update is determined by clustering the eigenvalues of the scaled BFGS matrix. On the other hand, the parameter scaling the third term is determined as a preconditioner to the Hessian of the minimizing function combined with the minimization of the conjugacy condition from conjugate gradient methods. The algorithm is described in the paper: **N. Andrei,** *A double parameter scaled BFGS method for unconstrained optimization*, Journal of Computational and Applied Mathematics, vol.332 (2018), pp.26-44 (Please see the file: JCAM-2018(43).pdf) **September 11, 2017**
3.	**DPSS**	A double parameter self-scaled memoryless BFGS method for unconstrained optimization is presented. In this method the first two terms of the self-scaled memoryless BFGS method are scaled with a positive parameter, while the third one is scaled with another positive parameter. The scaling parameters are selected in such a way to improve the eigenvalue structure of the BFGS update. The first parameter scaling the first two terms is determined to cluster the eigenvalues of the BFGS matrix. The second parameter scaling the third term is computed as a preconditioner to the Hessian of the minimizing function combined with minimization of the conjugacy condition from the conjugate gradient methods in order to shift the large eigenvalues of the self-scaled memoryless BFGS matrix to the left.

		 (a) (b) **Fig. 1.** Performance profiles of DP versus DPOL and versus DPOS. CPU time metric. $n = 1000$. The algorithm is described in the paper: **N. Andrei**, *A double parameter self-scaled memoryless BFGS method for unconstrained optimization.* Computational and Applied Mathematics, vol. , 2020. (Please see the file: COAMR1.doc) **November 12, 2017**
4.	**DSBFGS (ROMAN-POLYAK)**	A scaled BFGS method with two parameters for unconstrained optimization is presented. In this method the first two terms of the known BFGS update formula are scaled with a positive parameter and the third one is scaled with another positive parameter. The parameter scaling the first two terms of the BFGS update is determined by clustering the eigenvalues of the scaled BFGS matrix. On the other hand, the parameter scaling the third term is determined as a preconditioner to the Hessian of the minimizing function combined with the minimization of the conjugacy condition from conjugate gradient methods. This parameter is determined to reduce the large eigenvalues, thus obtaining a better distribution of them. The algorithm is described in the paper: **N. Andrei,** *A scaled BFGS method with two parameters for unconstrained optimization.* (Please see the file: Paper-Roman.doc) **May 5, 2017**
5.	**DNRTR**	A diagonal quasi-Newton updating algorithm. The elements of the diagonal matrix approximating the Hessian are determined by minimizing both the size of the change from the previous estimate and the trace of the update, subject to the weak secant equation. Figure 1 presents the performances of DNRTR versus steepest descent (SP) and versus Cauchy with Oren-Luenberger scaling in its complementary form (COL).

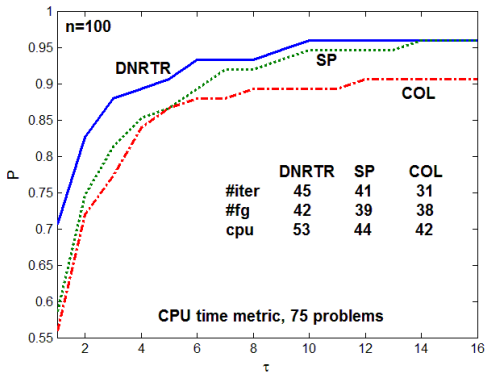
<table>
<tr>
<td></td>
<td></td>
<td>


Fig. 1 Performance profiles of DNRTR versus SP and versus COL. CPU time metric. $n = 100$.

Please see the paper:
N. Andrei, A diagonal quasi-Newton updating method for unconstrained optimization. Numerical Algorithms, vol.81(2), (2019), pp.575-590.

February 26, 2018
</td>
</tr>
<tr>
<td>6.</td>
<td>YONS</td>
<td>
A new diagonal quasi-Newton updating algorithm for unconstrained optimization is presented. The elements of the diagonal matrix approximating the Hessian, are determined as scaled forward finite differences directional derivatives of the components of the gradient. Under mild classical assumptions, the convergence of the algorithm is proved to be linear.
In this method the approximation Hessian B_{k+1} is a diagonal matrix computed as:
$$B_{k+1} = Y_k S_k^{-1},$$
where $Y_k = diag(y_k^1, \ldots, y_k^n)$ and $S_k = diag(s_k^1, \ldots, s_k^n)$, y_k^i, $i = 1, \ldots n$, being the components of the vector y_k and s_k^i, $i = 1, \ldots n$, being the components of vector s_k. Therefore, the diagonal elements of the matrix B_{k+1}, are computed as: $b_{k+1}^i = y_k^i / s_k^i$, $i = 1, \ldots, n$. In other words,
$$b_{k+1}^i = \frac{y_k^i}{s_k^i} = \frac{g_{k+1}^i - g_k^i}{x_{k+1}^i - x_k^i} = \frac{g^i(x_k + \alpha_k d_k) - g^i(x_k)}{\alpha_k d_k^i}, \qquad i = 1, \ldots n,$$
where g_k^i, is the $i-$th component of the gradient in x_k and d_k^i is the $i-$th component of the search direction.
Therefore, in this approach, the element b_{k+1}^i may be considered as an approximation of the second order derivative of function f, corresponding to the $i-$th diagonal element of the Hessian, computed in x_{k+1} by a scaled forward finite differences directional derivative scheme. Observe that $1/d_k^i$ is a scaling factor.
1) This directory contains the following Fortran files:
2) BFGS.FOR - Scaled BFGS method with Wolfe line search
</td>
</tr>
</table>

		3) **NTR.FOR -** $$\min \frac{1}{2}\left\|B_{k+1}-B_k\right\|^2+tr(B_{k+1})$$ subject to: $$s_k^T B_{k+1} s_k = s_k^T y_k.$$ 4) **PASDES.FOR** – Steepest descent metod. 5) **QNDIAG.FOR** – Diagonal Quasi-Newton updating. Only the diagonal elements from the BFGS updating are retained. The off diagonal elements of the BFGS update are neglected. $$b_{k+1}^i = b_k^i - \frac{(b_k^i)^2(s_k^i)^2}{\sum_{i=1}^n b_k^i (s_k^i)^2} + \frac{(y_k^i)^2}{y_k^T s_k}, \quad i=1,\ldots,n.$$ The algorithm was suggeted by Gilbert and Lemarechal; Gill and Murray. It is discussed by Zhu, Nazareth and Wolkowicz (SIAM 1999) 6) **WQND.FOR** - C This is obtained from the weak-quasi-Newton equation. The update proposed in this paper by Dennis and Wolkowicz: J.E. Dennis, H. Wolkowicz, Sizing and least-change secant methods, SIAM J. Numerical Analysis 30(5) (1993) 1291-1314. Only the diagonal elements from the BFGS updating are retained. The off diagonal elements of the BFGS update are neglected. $$b_{k+1}^i = b_k^i + \frac{s_k^T y_k - \sum_{i=1}^n b_k^i (s_k^i)^2}{\left(\sum_{i=1}^n b_k^i (s_k^i)^2\right)^2}(b_k^i)^2(s_k^i)^2, \quad i=1,\ldots,n,$$ 7) **YONS.FOR -** Te approximation Hessian B_{k+1} is a diagonal matrix computed as $B_{k+1} = Y_k S_k^{-1}$. (Please, see above.) The algorithm is described in the paper: **N. Andrei,** *A New Diagonal Quasi-Newton Updating Method With Scaled Forward Finite Differences Directional Derivative for Unconstrained Optimization*. Numerical Functional Analysis and Optimization, 2019, VOL. 40, NO. 13, 1467–1488. (Please see the files: NFAO2019.PDF and paperR1.doc) **February 26, 2018**

<><><><><><><><><><>

L-BFGS

1.	**LBFGS14**	Limited BFGS method for solving 14 applications of unconstrained optimization. The applications solved by LBFGS14 are: 1. Weber Function (1) (Andrei, U71) 2. Enzyme reaction (Andrei, U79) (A)

3. Solution of a chemical reactor (A)
4. Robot kinematics problem (A)
5. Solar Spectroscopy (A)
6. Estimation of parameters (A)
7. Propan combustion in air (A)
8. Gear train with minimum inertia (A)
9. Human Heart Dipole. Andrei U84, pp.65
10. Neurophysiology (A)
11. Combustion application (A)
12. Thermistor (A)
13. Optimal design of a Gear Train (A)
14. Circuit design (A)

The contents of the directory LBFGS14 is as follows:
LBFGS.FOR (Fortran package by Liu and Nocedal)
FUNC14.TXT
lbfgs14.out
lbfgs.rez
R2020T18.DOC (Technical Report with performances of LBFGS14.

Performances of L-BFGS

n	iter	fgcnt	time(c)	fx	gnorm	Name of Applications
2	55	218	0	-0.2644531414650E+03	0.5851360369660E+01	1. Weber Function (Andrei, U71)
4	41	48	0	0.3075056038494E-03	0.8903176713965E-08	2. Enzyme reaction (Andrei, U79) (A)
6	15316	22822	4	0.5038172519506E+00	0.1643286688653E-04	3. Solution of a chemical reactor (A)
8	8	30	1	0.6957468202726E-05	0.3204751579182E-02	4. Robot kinematics problem (A)
4	23	29	0	0.8312307693160E+01	0.7194096296787E-07	5. Solar Spectroscopy (A)
4	46	53	0	0.3185717487911E-01	0.2214569790005E-07	6. Estimation of parameters (A)
5	282	351	0	0.5884317208105E-18	0.6963808917363E-07	7. Propan combustion in air (A)
2	10	33	1	0.1746908655419E+01	0.4035725583973E-01	8. Gear train with minimum inertia (A)
8	2	23	0	0.1789025922492E+00	0.7325794325147E-01	9. Human Heart Dipole. Andrei U84,
6	23	38	0	0.4539057615171E+01	0.2024779517274E-07	10. Neurophysiology (A)
10	155	179	1	0.3338493557009E-11	0.1532274549876E-06	11. Combustion application (A)
3	22	51	0	0.1721671200203E+03	0.1496223984167E+02	12. Thermistor (A)
4	9	10	0	0.1971065573361E-05	0.6630614440024E-07	13. Optimal design of a Gear Train (A)
9	6037	6838	6	0.2350848678996E-12	0.1601637091248E-06	14. Circuit design (A)
TOTAL	22029	30723	6.00	centeseconds		

Number of stored pairs (sk,yk) in L-BFGS: M = 5
Date: --- Month: 6 Day: 6 Year: 2020

June 6, 2020

<><><><><><><><><><>

Other Programs		
1.	**GRADSYS**	Gradient Flow Algorithm for Unconstrained Optimization. The algorithm is: $$x_{k+1} = x_k + \delta x_k,$$ where δx_k is computed as the solution of the following linear algebraic systems of equations: $$\left[I + h_k \theta \nabla^2 f(x_k)\right]\delta x_k = -h_k \nabla f(x_k).$$ $\theta \in [0,1]$. If $\theta = 1$ and $h_k \to 1$, then the algorithm is quadratically convergent.

		The algorithm is described in: **N. Andrei,** *Gradient flow algorithm for unconstrained optimization*. Technical Report, March 23, 2004. (Please see the file diff.pdf) **March 23, 2004**
2.	**NEWGRAD**	Relaxed Gradient Descent and a New Gradient Descent Methods for Unconstrained Optimization. Mainly, the algorithm is the steepest gradient where the step-length is modified by a multiplicative parameter: $$x_{k+1} = x_k - \theta_k \alpha_k \nabla f(x_k),$$ where $\theta_k \in [0,1]$ and α_k is the step-length computed by backtracking. The algorithm and its theory is described in: **N. Andrei,** *Theory versus empiricism in analysis of optimization algorithms*. Technical Press, Bucharest, 2004. ISBN: 973-31-2233-5. **N. Andrei,** *Criticism of the unconstrained optimization algorithms reasoning*. Romanian Academy Publishing House, Bucharest, 2009. ISBN: 978-973-27-1669-4. Please see the papers: **N. Andrei,** Numerical Experiments with Gradient Descent with Backtracking for Unconstrained Optimization. March 2, 2005. (File: CGAD.DOC, 10 pages) **N. Andrei,** Numerical Experiments with Relaxed Gradient Descent with Backtracking for Unconstrained Optimization. March 5, 2005. (File: RELAXED.DOC, 12 pages) **March 17, 2005**

<><><><><><><><><><><>

	Constrained optimization	
1.	**KKT**	Karush-Kuhn-Tucker methods for solving inequality constraints optimization problems of the following form: $$\min f(x),$$ subject to: $$g(x) \le 0,$$ where $f : R^n \to R,$ and $g : R^n \to R^m$. It is supposed that all functions of the problem are continuousli differentiable. Directorul KKT conţine un număr de 8 probleme de optimizare cu restricţii inegalităţi: NEWTON1.FOR, NEWTON2.FOR,... NEWTON8.FOR, SIMEQ.FOR, LS.FOR.

		Problemele sunt descrise şi rezolvate în lucrarea: **N. Andrei,** *Metode bazate pe condiţiile Karush-Kuhn-Tucker, Manuscript*, 1995, cu CD. (În biblioteca mea.) **March 3, 1995**
2.	**PREDCOR**	Interior-Point Predictor-Corrector algorithm for linear constrained optimization. Directorul conţine 15 exemple de probleme de optimizare cu restricţii linare utilizând metoda de punct interior, într-o implementare naivă. Sistemele de ecuaţii algebrice linare asociate metodei sun rezolvate cu subruruinele: DLINEQ.FOR (LU decomposition) şi DRESLV.FOR (Substitutions). Exemplul LCPC10.FOR rezolvă aplicaţia: Chemical Equilibrium Problem. **December 24, 1996**
3.	**SPG**	SIMPLE BOUNDED OPTIMIZATION by Birgin, Martinez and Rydan $\min\{f(x), l \le x \le u\}$ where $f(x)$ is a continuously differentiable and its gradient is available. l and u are simple margins on the variables. It is assumed that $l \le u$. - First version: February 02, 2001 by E.G.Birgin, J.M.Martinez and M.Raydan. - Final revision: April 30, 2001 by E.G.Birgin, J.M.Martinez and M.Raydan. - Modified final version: May 12, 2008 by Neculai Andrei to include the safeguarded cubic interpolation. The algorithm and its performances are presented in: **N. Andrei,** *Criticism of the Constrained Optimization Algorithms Reasoning*, Editura Academiei Române, Bucureşti, 2015. ISBN: 978-973-27-2527-6 (pp. 169-177) The following applications are considered: APPL1.FOR - Elastic-Plastic Torsion problem APPL2.FOR - Pressure Distribution in a Journal Bearing APPL3.FOR - Optimal Design with Composite Materials APPL4.FOR - Ginzburg-Landau (1-dimensional) problem APPL5.FOR - Steady State Combustion The program MSPG.FOR implements the SPG subroutine for solving a train of 730 problems with simple bounds. The line search subroutine is modified by Neculai Andrei to include the safeguarded cubic interpolation. The following examples are presented: 1) SPGEX1.FOR is for minimizing the Freudenstein & Roth function with n=1000, ..., 10000. 2) SPGEX2.FOR is for minimizing the Extended Penalty function with n=1000, ..., 10000. 3) SPGEX3.FOR is for minimizing the Broyden Tridiagonal function with n=1000, ..., 10000.

		May 12, 2008
4.	**SPENBAR**	Package for large-scale nonlinear, equality and inequality constrained optimization. The optimization problem solved by SPENBAR is as follows: $$\min F(x),$$ subject to $$c_i(x) \geq 0, \quad i = 1,\ldots,m,$$ $$e_k(x) = 0, \quad k = 1,\ldots,me,$$ $$l_j \leq x_j \leq u_j, \quad j = 1,\ldots,n,$$ where all the functions are continuously differential. The program implements a modified penalty-barrier method. The unconstrained optimization problems is solved by means of truncated Newton method implemented in subroutine LMQN written by Stephen Nash. This directory contains 4 sub-directories (DOC, examples, PROB, REZMOD) and 4 Fortran files (HS108.FOR, IP.FOR, IP1.FOR and SPENBAR.FOR). The algorithm is described in a number of papers and Technical Reports as: **N. Andrei,** (1996) *Computational Experience with a Modified Penalty-Barrier Method for Large-Scale Nonlinear Constrained Optimization. (FORTRAN subroutines)* ICI Working Paper No. AMOL-96-1, February 6, 1996. **N. Andrei,** (1996) *Computational Experience with SPENBAR a Sparse Variant of a Modified Penalty-Barrier Method for Large-Scale Nonlinear, Equality and Inequality Constrained Optimization.* ICI Technical Paper No. AMOL-96-4, March 11, 1996, pp.1-69. **N. Andrei,** (2006) *Numerical Examples with SPENBAR for Large-Scale Nonlinear, Equality and Inequality Constrained Optimization with Zero Columns in Jacobian Matrices.* ICI Technical Paper No. AMOL-96-5, March 29, 1996. **N. Andrei,** (2001) *Numerical Examples with SPENBAR - Modified penalty barrier method for large-scale nonlinear programming problems. Part I.* ICI Technical Report, ICI-TR-01/2001, Bucharest, February 2001. Technical Report placed in Library of Romanian Academy. **N. Andrei,** (2001) *Computational experience with SPENBAR. A sparse modified penalty-barrier method for large-scale nonlinear, equality and inequality, constrained optimization.* Technical Report No.4/2001, February 19, 2001. (*Manuscript. În biblioteca mea.*) **N. Andrei,** (2015) *Criticism of the Constrained Optimization Algorithms Reasoning,* Editura Academiei Române, Bucureşti, 2015. ISBN: 978-973-27-2527-6 (pp. 517-537)

		N. Andrei, (2017) *Continuous Nonlinear Optimization for Engineering Applications in GAMS Technology*. Springer Optimization and Its Applications, Volume 121, Springer Science+Business Media New York 2017, ISBN: 978-3-319-58356-3, e-book ISBN: 978-3-319-58356-3, ISSN: 1931-6828, DOI: 10.1007/978-3-319-58356-3, Springer New York Heidelberg Dordrecht London, 508 + XXIV pages. OPIS.TXT contains the list of problems from SPENBAR collection. **February 19, 2001**
5.	**TOLMINV**	Package of subroutines that calculate the the least value of a differentiable function of several variables subject to linear constraints on the values of the variables written by M.J.D. Powell. TOLMIN, written by Powell, works with two-dimensional arrays and solves the problems of the following types: $\min F(x),$ subject to: $a_j^T x = b_j, \quad j = 1,\ldots,MEQ,$ $a_j^T x \le b_j, \quad j = MEQ+1,\ldots,m,$ $l_i \le x_i \le u_i, \quad i = 1,\ldots,n.$ All the subroutines of the program are modified by N. Andrei to work with vectors, without considering the spsrsity the the matrix corresponding to linear constraints. This is TOLMINV package. This Directory contains three sub-directories: TOLMIN14, TOLMINMA, TOLMINVE. Subdirectory TOLMIN14 includes three programs for solving constrained optimization problems as follows: MAIN01.FOR is the main program for solving the nonlinear optimization problem presented in Example 14.1 in the book: **N. Andrei,** *Critica Ratiunii Algoritmilot de Optimizare cu Restrictii*, Editura Academiei, 2015, pp. 629. THE COMPUTED SOLUTION POINT IS 1 0.8750081257267E-07 2 0.4629495896370E-04 3 0.9999514660798E+00 4 0.5001292705362E+00 5 0.9999908024081E+00 6 0.4999956364708E+01 7 0.3000004697421E+01 8 0.1000000000000E+01 X(I) X(I)-XL(I) XU(I)-X(I) 1 8.7500813E-08 8.7500813E-08 1.9999999E+00 2 4.6294959E-05 5.0000463E+00 9.9995371E-01 3 9.9995147E-01 9.9995147E-01 1.0000485E+00 4 5.0012927E-01 1.5001293E+00 1.4998707E+00 5 9.9999080E-01 9.9999080E-01 3.0000092E+00 6 4.9999564E+00 5.9999564E+00 5.0000436E+00 7 3.0000047E+00 3.0000047E+00 2.9999953E+00 8 1.0000000E+00 2.0000000E+00 0.0000000E+00 FINAL CONSTRAINT RESIDUALS = 0.0000E+00 0.0000E+00 1.7764E-15 0.0000E+00 Function value in optimal point= 0.2999999978913E+01 Execution Time: 0: 0: 0: 0

MAIN02.FOR is the main program for solving the nonlinear optimization problem presented in Example 5.3 in the book:
N. Andrei, *Critica Ratiunii Algoritmilot de Optimizare cu Restrictii*, Editura Academiei, 2015, pp. 266.

```
   THE COMPUTED SOLUTION POINT IS
      1     0.4002737835268E+00
      2     0.1305663058260E+00
      3     0.0000000000000E+00
      4     0.0000000000000E+00
      5     0.9033793559852E+00
      6     0.4168581403599E+00
      7     0.0000000000000E+00
      8     0.1509334100456E+01
      9     0.1522805326970E+01
     10     0.5379458016316E+00
     11     0.1013056630583E+01
     12     0.5527662953395E+00
     13     0.0000000000000E+00
     14     0.0000000000000E+00
     15     0.6010093996177E+00

            X(I)           X(I)-XL(I)     XU(I)-X(I)
      1    4.0027378E-01  4.0027378E-01  1.5997262E+00
      2    1.3056631E-01  1.3056631E-01  1.8694337E+00
      3    0.0000000E+00  0.0000000E+00  2.0000000E+00
      4    0.0000000E+00  0.0000000E+00  2.0000000E+00
      5    9.0337936E-01  9.0337936E-01  1.0966206E+00
      6    4.1685814E-01  4.1685814E-01  1.5831419E+00
      7    0.0000000E+00  0.0000000E+00  2.0000000E+00
      8    1.5093341E+00  1.5093341E+00  4.9066590E-01
      9    1.5228053E+00  1.5228053E+00  4.7719467E-01
     10    5.3794580E-01  5.3794580E-01  1.4620542E+00
     11    1.0130566E+00  1.0130566E+00  9.8694337E-01
     12    5.5276630E-01  5.5276630E-01  1.4472337E+00
     13    0.0000000E+00  0.0000000E+00  2.0000000E+00
     14    0.0000000E+00  0.0000000E+00  2.0000000E+00
     15    6.0100940E-01  6.0100940E-01  1.3989906E+00

    FINAL CONSTRAINT RESIDUALS =
-1.3878E-16  0.0000E+00  4.4409E-16  3.3307E-16  5.5511E-16  6.6613E-16
 1.1102E-15

    Function value in optimal point= 0.2192129651805E+02

    Execution Time:  0: 0: 0: 0
```

MAIN03.FOR is the main program for solving the nonlinear optimization problem presented in Example 14.3 in the book:
N. Andrei, *Critica Ratiunii Algoritmilot de Optimizare cu Restrictii*, Editura Academiei, 2015, pp. 266.

```
    THE COMPUTED SOLUTION POINT IS
      1     0.2812500000000E+01
      2     0.0000000000000E+00
      3     0.7187500000000E+01
      4     0.3750000000000E+01
      5     0.0000000000000E+00
      6     0.0000000000000E+00
      7     0.0000000000000E+00
      8     0.3125000000000E+01
      9     0.0000000000000E+00
     10     0.0000000000000E+00
     11     0.0000000000000E+00
     12     0.5718750000000E+02
     13     0.2562500000000E+02

            X(I)           X(I)-XL(I)     XU(I)-X(I)
      1    2.8125000E+00  2.8125000E+00  9.7187500E+01
      2    0.0000000E+00  0.0000000E+00  1.0000000E+02
      3    7.1875000E+00  7.1875000E+00  9.2812500E+01
      4    3.7500000E+00  3.7500000E+00  9.6250000E+01
      5    0.0000000E+00  0.0000000E+00  1.0000000E+02
      6    0.0000000E+00  0.0000000E+00  1.0000000E+02
      7    0.0000000E+00  0.0000000E+00  1.0000000E+02
      8    3.1250000E+00  3.1250000E+00  9.6875000E+01
      9    0.0000000E+00  0.0000000E+00  1.0000000E+02
     10    0.0000000E+00  0.0000000E+00  1.0000000E+02
```

		11 0.0000000E+00 0.0000000E+00 1.0000000E+02 12 5.7187500E+01 5.7187500E+01 4.2812500E+01 13 2.5625000E+01 2.5625000E+01 7.4375000E+01 FINAL CONSTRAINT RESIDUALS = -1.7764E-15 0.0000E+00 3.5527E-15 5.3291E-15 1.4211E-14 0.0000E+00 Function value in optimal point= 0.1568830990135E+07 Execution Time: 0: 0: 0: 0 **November 22, 1995**
6.	**PSO-CO**	**Particle Swarm Optimization** (PSO). In this directory I included a number of Fortran packages for constrained optimization using the particle swarm optimization method. For solving the problem $\min\{f(x), c_i(x) \le 0, i = 1,\ldots,m\}$ the algorithm for PSO considers the following strategy. Using the PSO algorithm for unconstrained optimization minimize the penalty function: $F(x) = f(x) + h(t)H(x),$ where: $H(x) = \sum_{i=1}^{m} \theta(q_i(x))(q_i(x))^{\gamma(q_i(x))},$ $q_i(x) = \max\{0, c_i(x)\}, \quad i = 1,\ldots,m,$ $\theta(q_i(x)) = \begin{cases} 10, & \text{dacă} \quad q_i(x) < 0.001, \\ 20, & \text{dacă} \quad 0.001 \le q_i(x) < 0.1, \\ 100, & \text{dacă} \quad 0.1 \le q_i(x) < 1, \\ 300, & \text{dacă} \quad q_i(x) \ge 1, \end{cases}$ $\gamma(q_i(x)) = \begin{cases} 1, & \text{dacă} \quad q_i(x) < 1, \\ 2, & \text{dacă} \quad q_i(x) \ge 1, \end{cases}$ $h(t) = t\sqrt{t},$ Here, t is the number of iteration. The applications solved by this method are as follows: ALKI-PSO - Optimization of an alkylation process, Variant 1, CAM-PSO - Shape optimization of a cam, DES-PSO - Distribution of electrons on a sphere, HANG-PSO - Hanging chain, MSP3-PSO - 3-stage membrane separation, MSP5-PSO - A 5-stage membrane separation process, PPSE-PSO - Static Power Scheduling, PREC-PSO - Optimal Reactor Design, TRAFO-PSO - Transformer design. BRAKE-PSO - Design of a disc brake, EX1-PSO – Example 1, EX2-PSO – Example 2, LATHE.PSO - Multi-spindle automatic lathe SPRING.PSO - Minimizing the weight of a tension/compression spring WESSEL.PSO - Pressure vessel

		Please, see the book: „*Critica Raţiunii Algoritmilor de Optimizare cu Restricţii*”, Bucureşti, Editura Academiei Române, 2015, Capitolul 19. See also: the paper **Anale-PSO.doc** and the technical report **PSO.doc** (October 9, 2014). Please, see the directory PSO-CO in CONSTRAINED-OPTIM. **May 21, 2014**
7.	**CAON**	**A collection of nonlinear optimization applications in GAMS language**. Se prezintă 25 de modele de optimizare neliniară, exprimate în limbajul GAMS. See: **N. Andrei,** *CAON: O colecţie de aplicaţii de optimizare neliniară în limbajul GAMS.* Technical Report No.1/2011, January 31, 2011. (105 pages with CD). Please, see the directory CAON in CONSTRAINED_OPTIM. Please, see the Technical Report: **r1a11.doc**. The mathematical models in GAMS are placed in directory CD-GAMS. **January 31, 2011**