UNITED STATES
DEPARTMENT OF THE INTERIOR
GEOLOGICAL SURVEY

THE GENERIC BUNDLE ADJUSTMENT

by
Keld S. Dueholm

Open-File Report 89-185

This report is preliminary and has not been reviewed for conformity with U.S. Geological Survey editorial standards. Any use of trade names is for descriptive purposes only and does not imply endorsement by the USGS. Although the described program has been used by the USGS, no warranty, expressed or implied, is made by the USGS as to the accuracy and functioning of the program and related program material nor shall the fact of distribution constitute any such warrenty, and no responsibility is assumed by the USGS in connection therewith.

The author was employed under contract with the USGS from September 1987 to August 1989 on leave from the Institute of Surveying and Photogrammetry, Technical University of Denmark, DK2800 Lyngby, Denmark.

Denver, Colorado
1989

BRIEF DESCRIPTION

The Generic Bundle Adjustment (GBA) is a set of Pascal procedures for least squares adjustment. The GBA includes: automatic trapping of blunders and reduction of weight on involved observations, automatic removal of singular columns, and a choice of defining individual unknowns as constants in a given adjustment. New types of unknowns and observation equations can be added to the adjustment in a standardized and relatively simple way.

CONTENTS

INTRODUCTION

The Generic Bundle Adjustment (GBA) is a set of Pascal procedures for least squares adjustment. The GBA includes: automatic trapping of blunders and reduction of weight on involved observations (The Danish Method, Krarup et al., 1980), automatic removal of singular columns (Poder, 1982), and a choice of defining individual unknowns as constants in a given adjustment. Unknowns that can be redefined as constants are called "elements" in the GBA. New types of elements and observation equations can be added to the adjustment in a standardized and relatively simple way, described in this documentation.

The GBA handles the administration of pointers in a generalized least squares adjustment, the forming, reduction and resolving of the normal equations, and the subsequent error calculations. The procedures are complete for bundle adjustment using points or geodetic observations (angles and distances) as ground control and with radial distortion, principal point, and camera constant as extra parameters. However, the input and output modules are primitive and meant for demonstration purposes only.

The GBA implementation is in part based on ideas and experiences derived from different bundle adjustment programs developed over the last 20 years at the Institute of Surveying and Photogrammetry, Technical University of Denmark (Jacobi, 1968; Alexandersen, 1974; Dueholm, 1983; Thomsen, 1987). The LINLIB 2.0 procedures (Weng, 1984; Weng, 1986) for forming, reduction and resolving sparse matrices are used to handle the normal equations.

The VAX PASCAL compiler version 3.5 running under VAX/VMS version 4.7 has been used for the developments. Experience in Pascal programming and a good understanding of least squares adjustment theory are required to use the procedures.

LIST OF PROCEDURES

The GBA modules consist of the global declarations and nine GBA core procedures.

Example observation equation procedures, modules for input of elements and observations, an example main program, and command files for compilation, linking, and printing are included in order to illustrate the use of the GBA procedures.

The following describe in brief all the distributed modules.

The global declarations:

MODULE BUNDGLO: Declares constants, types, variables, and external procedures.

The GBA procedures:

MODULE BUND01 (GBA_Initialize): Initializes global variables.

MODULE BUND02 (GBA_Elements): Defines the elements of the adjustment.

MODULE BUND03 (GBA_Observations): Defines the observations of the adjustment, and establish the link to the elements.

MODULE BUND04 (GBA_Pointers): Establishes the pointers between observations, elements, and normal equation columns.

MODULE BUND05 (GBA_Solve): Solves the adjustment by iteration.

MODULE BUND06 (GBA_Statistics): Calculates residuals, standard deviations, and redundancies after the adjustment.

MODULE BUND07 (GBA_Observation_Equation): Calls the observation equation procedures. This is the only GBA procedure that need to be changed when a new type of observation equation is added to the adjustment.

MODULE BUND08 (GBA_Write_Pass_Information): Allows the user to design and write the information he wants to see after each iteration, and to stop the adjustment if desired.

MODULE BUND09 (GBA_Write_String): Help procedure to write a string without tailing blanks.

The observation equation procedures:

MODULE OBSLO1 (Photo_Observation_Equation): Calculates normal equation coefficients and right hand side for sets of photo observations (x, y). The elements are described in the procedure.

MODULE OBSLO2 (Element_Observation_Equation): Calculates normal equation coefficients and right hand side for observation of an individual element (e.g. a coordinate observation).

MODULE OBSLO3 (Length_Observation_Equation): Calculates normal equation coefficients and right hand side for horizontal distance observations. The elements are described in the procedure.

MODULE OBSLO4 (TheoHZ_Observation_Equation): Calculates normal equation coefficients and right hand side for horizontal direction observations. The elements are described in the procedure.

The example input procedures:

MODULE READO1 (Read_String): Help procedure to enter a continuous string (no spaces) into a packed array of characters.

MODULE READELEO1 (Read_Camera_Elements): Reads and defines camera elements.

MODULE READELE02 (Read_Photo_Elements): Reads and defines photo elements.

MODULE READELE03 (Read_Point_Elements): Reads and defines point elements.

MODULE READOBS01 (Read_Photo_Observations): Reads and defines photo observations.

MODULE READOBS02 (Read_Point_Observations): Reads and defines point observations.

MODULE READOBS03 (Read_Length_Observations): Reads and defines horizontal distance observations.

MODULE READOBS04 (Read_TheoHZ_Observations): Reads and defines horizontal direction observations. For every

station a preliminary value for the teodolite circle zero angle is needed.

The example main program:

PROGRAM BUNDOO: A primitive program with the aim of demonstrating the use of the distributed procedures.

The command files:

BUND.COM: Command file for the compilation of all procedures.

BUND.LNK: Command file for linking.

BUND.PRT: Command file for the printing of the source code.

The data files:

TEST.*: Input data for test run of the program.

.OUT: Output from a calculation of TEST..

THE GLOBAL DECLARATIONS

It is crucial to be familiar with the global declarations. So, please study the following section carefully, with a listing of the global declarations (BUND.GLO) handy for reference. Program identifiers defined in the global's are written in quotation.

The constants.

'Max_Name_Length' limits the number of characters in names for elements and observations.

'Max_Number_Elements' limits the number of elements allowed. Because an element can be defined as unknown or constant in a given adjustment, 'Max_Number_Elements' must be the maximum number of unknowns and constants. Besides, also the right hand sides of the normal equations need to be allowed for in this constant.

'Max_Embedded_Elements' and 'Max_Embedded_Observations' limit the number of embedded elements and observations that

can be referred to by one observation equation. Each observation equation refers to a certain number of embedded elements and may handle several embedded observations. For instance, a photo observation as implemented in the procedure 'Photo_Observation_Equation' refers to 17 elements and handles 2 coordinate observations (MODULE OBSL01).

'Max_Number_Observations' limits the number of observation records allowed. The observations in a group of embedded observations counts as one only. For example, a photo observation that consist of 2 coordinate observations counts as one.

'Max_Normal_Equation_Size' is the maximum places reserved for the profiled storage of the normal equations.

The type declarations.

Elements are divided into blocks of 'Element_Block_Type' and placed in the normal equations according to the order of the block variable. This is done, because it is difficult to find an automatic column organizer, that handles a combination of photogrammetric and geodetic equations better than a controlled blocking due to physical relationships. It will make sense to organize the columns within each separate block by use of an automatic column organizer. Points are placed in the 'POINT_BLOCK'. However, if they are connected to other points via geodetic equations the program automatically transfers them to the 'SURVEY_BLOCK'. Geodetic elements, such as refraction parameters or instrument parameters are placed in the 'SURVEY_BLOCK'. Photo elements or exterior orientation elements for models are placed in the 'PHOTO_BLOCK', while camera elements are placed in the 'CAMERA_BLOCK'.

'Element_Kind_Type' is an enumerated type holding a unique identifier for each kind of element. This declaration has to be updated whenever a new kind of element is added to the adjustment.

'Element_Status_Type' describes weather an element is considered unknown (status: 'UNKNOWN') or constant (status: 'CONSTANT').

The elements are organized in an array 'Element_Array_Type' of records 'Element_Record_Type', one element each record. Elements, such as, for instance, camera parameters can be referred to as a group under one name. In such case, the elements in the group must be placed sequentially and in order in the element array. As input to the adjustment the

'Element_Name', 'Element_Kind', 'Element_Block', 'Element_Status', and 'Element_Value' must be assigned values. The 'Element_Value' is the preliminary value for an element with the status 'UNKNOWN' or a constant if the element status is 'CONSTANT'. The procedure 'GBA_Elements' handles these assignments. After the adjustment the best fit value (the adjusted value) is assigned to 'Element_Value', and the calculated standard deviation on the element is assigned to 'Element_Standard_Deviation'. The boolean 'Element_Used' will be TRUE if the element has been referred to by an observation. If not 'Element_Used' will be FALSE.

'Observation_Group_Type' holds an identifier for each observation equation. This declaration has to be updated whenever a new observation equation procedure is added to the adjustment.

'Observation_Kind_Type' holds a unique identifier for each different kind of observation. This declaration has to be updated whenever a new kind of observation is added to the adjustment. The purpose of this type is to identify and separate the single embedded observations in output, and to provide a list of already defined observation types for the application programmer.

The observations are maintained in an array 'Observation_Array_Type' of records 'Observation_Record_Type'. Each observation may consist of several embedded observations 'Number_Embedded_Observations' and refer to several embedded elements 'Number_Embedded_Elements'. All embedded observations and elements has to be handled by one call of an observation equation procedure. The array 'Element_Pointer' is assigned a pointer to the element array for each embedded element. As input to the adjustment the 'Observation_Name', 'Observation_Group', 'Number_Embedded_Observations', 'Observation_Kind', 'Observation', 'Apriori_Error', 'Number_Embedded_Elements', and 'Element_Pointer', must be assigned values. The weight of the observation is calculated as: 1 / ('Apriori_Error' * 'Apriori_Error'). The procedure 'GBA_Observations' handles these assignments. After the adjustment the calculated standard deviations on the observations are assigned to 'Aposteriori_Error'. Residuals and standard deviations on residuals are assigned to 'Residual' and 'Residaul_Error'. If the weight of the observation was reduced during the adjustment the applied weight factor is assigned to 'Weight_Factor', otherwise the 'Weight_Factor' is assigned a value of 1.

The variables.

'Element' is the variable name for the element array. 'End_Element_Array' is the number of entries in the element array.

'Obs' is the variable name for the observation array. 'End_Obs_Array' is the number of entries in the observation array.

'Max_Column_In_Block' is the highest normal equation column number for each element block.

'NEle' is the actual number of unknown elements in a given adjustment. 'NObs' is the total number of observations. Each embedded observation is counted as one.

'OUTERR' is the file variable for the file to which warnings and errors are written during execution of the program.

'Col' and 'Ele' interfaces the element array and the normal equation columns:

```
Column number= Col[element array number].
Element array number= Ele[column number].
```

'N' is the normal equation array, 'N_Size' is the actual size of the normal equation array in a given adjustment.

'Itt_no' is the actual number of performed iterations.

's0' is the standard deviation unit weight of the adjustment.

'Base' is a "LINLIB" integer array holding the virtual linear index of the imaginary element number zero of each column of the normal equation matrix.

'Sing' is a "LINLIB" integer array that return the number of lost digits multiplied by 100 for each column of the normal equation matrix and the convergence measure after each iteration. 'Sing[0]' holds the maximum number of lost digits in any column.

THE GBA PROCEDURES

GBA_Initialize :

Initializes global variables.

```
GBA_Elements
      (Name                    :  Name_String;
       Kind                    :  Element_Kind_Type;
       Block                   :  Element_Block_Type;
       Status                  :  Element_Status_Type;
       Value                   :  DOUBLE;
       VAR Warning, Error      :  BOOLEAN) :
```

Adds an element to the element array. 'GBA_Initialize' must be called before the first call of 'GBA_Elements'.

The element is defined by passing the name 'Name,' the element kind 'Kind', the block ('POINT_BLOCK', 'SURVEY_BLOCK', 'PHOTO_BLOCK', or 'CAMERA_BLOCK'), the status ('UNKNOWN' or 'CONSTANT'), and the preliminary value 'Value'.

Since the program automatically moves points with geodetic bindings from the 'POINT_BLOCK' to the 'SURVEY_BLOCK' elements in the two blocks cannot have the same name. Aside from that, elements in different blocks may be named alike.

A group of elements with the same name should be entered sequentially and in the order given by the 'Element_Kind_Type'.

If an element is already defined under the same name and block then the boolean 'Warning' is set to TRUE and the following warning is given:

```
   *WARNING: Element entered twice.  Last entered value
used.
             "Block" "Kind" Name:  ".
```

It is up to the programmer to check for the warning status and take appropriate action. The example program ignores this warning.

If the maximum number of elements defined by the constant 'Max_Number_Elements' is exceeded the boolean 'Error' is set to TRUE and the following error message is given:

```
   ***ERROR:  Maximum number of elements exceeded.
```

```
GBA_Observations
      (Obs_Name                 :  Name_String;
       Obs_Group                :  Observation_Group Type;
       Embedded_Obs             :  INTEGER;
       Obs_Kind                 :  ARRAY [f1..11: INTEGER]
                                     OF Observation_Kind Type
       Obs_Value                :  ARRAY [f2..12: INTEGER]
                                     OF DOUBLE;
       Apriori                  :  ARRAY [f3..13: INTEGER]
                                     OF DOUBLE;
       Number_Groups            :  INTEGER;
       Names                    :  ARRAY [f4..14: INTEGER]
                                     OF Name_String;
       Kinds                    :  ARRAY [f5..15: INTEGER]
                                     OF Element_Kind_Type;
       Blocks                   :  ARRAY [f6..16: INTEGER]
                                     OF Element_Block_Type;
       Embedded_Ele             :  INTEGER;
       VAR Warning, Error       :  BOOLEAN) :
```

Adds observations to the adjustment and establish the pointers to the element array. 'GBA_Initialize' must be called before the first call of 'GBA_Observations'. Besides all the embedded elements must have been previously defined by calls to GBA_Elements.

The observation (or group of embedded observations) is defined by passing the observation name 'Obs_Name' and group 'Obs_Group', the number of embedded observations 'Embedded_Obs', and for each embedded observation the kind of observation 'Obs_Kind', the observed values 'Obs_Value', and the Apriori errors 'Apriori'. The 'Obs_Name' is a user interface identifier only. Therefore, the program accepts two or more observations with the same name.

In order to establish the pointers to the element array the embedded elements need to be defined as input to the procedure. This is done by assigning the name, kind, and block of each embedded element to the arrays 'Names', 'Kinds', and 'Blocks'. In order to include all elements in a group, kind is set to 'WHOLE_GROUP'. 'Number_Groups' is the number of entries in Names, Kinds, and Blocks. Finally, the total number of embedded elements are assigned to 'Embedded_Ele'.

If an element or a whole group of elements cannot be found, or if 'Embedded_Ele' is different from the actual number of elements found, the boolean 'Warning' is set to TRUE and the following warning is given:

```
WARNING:  Element not found.  Observation ignored.
          "Blocks[1]" "Kinds[1]" Name:  "Names[1]"
           :          :     :   :       :
        >  "Blocks[n]" "Kinds[n]" Name:  "Names[n]"
     obs:  "Obs-Group" Name:  "Obs-Name"
```

The warning shows which elements are addressed in the call by the observation, the kind of observation, and the observation name. A ">" in front of the block name shows which specific element or element group was not found. In the example above it is element "Kinds[n]" in block "Blocks[n]" with name "Names[n]" that was missing. The observation is ignored, and it is up to the programmer to take appropriate action. The example program ignores this warning.

If the maximum number of observations defined by the constant 'Max_Number_Observations' or the maximum number of embedded observations defined by the constant 'Max_Embedded_Observations', or the maximum number of embedded elements defined by the constant 'Max_Embedded_Elements' are exceeded the boolean 'Error' is set to TRUE and one of the following error messages is given:

```
***ERROR:  Maximum number of observations exceeded.

***ERROR:  Maximum number of embedded observations
           exceeded.
     obs:  "Obs_Group" Name:  "Obs_Name"

***ERROR:  Maximum number of embedded elements exceeded.
     obs:  "Obs_Group" Name:  "Obs_Name"
```

```
GBA_Pointers
     (VAR Error     :  BOOLEAN) :
```

Establishes the pointers of the adjustment. All elements and observations must have been previously defined by calls to 'GBA_Elements' and 'GBA_Observations'. The following pointers are calculated by the procedure: 'NEle', 'NObs', 'Col', 'Ele', 'N_Size', 'Base', and 'Max_Column_In_Block'.

If the actual normal equation size, 'N_Size' is greater than 'Max_Normal_Equation_Size', or if the number of observations 'NObs' are less than the number of elements 'NEle', then the boolean 'Error' is set to TRUE and one of the following error message is given:

```
***ERROR:  Normal Equation  > Max_Normal_Equation_Size.

***ERROR:  Number_Observations < Number_Elements.
```

```
GBA_Solve
      (enp_stop              :  DOUBLE;
       max_itt               :  INTEGER;
       start_down_weight     :  INTEGER;
       max_vst               :  DOUBLE;
       max_lost_digits       :  INTEGER;
       VAR Error             :  BOOLEAN ) :
```

Solves the least squares adjustment by iteration. All elements and observations must have been previously defined by calls to 'GBA_Elements' and 'GBA_Observations' and the pointers set by 'GBA_Pointers'.

The exponent of numerical precision (enp) is used to verify the convergence of the iterations (Krarup, 1982). 'enp_stop' is the enp at which the user wants to stop the iterations. Normally, a value of 'enp_stop' between 3.0 and 5.0 assures satisfactory convergence.

In case the adjustments do not converge 'max_itt' sets a limit to the number of iterations that will be performed.

The procedure will perform an automatic search for gross errors according to the input parameters 'start_down_weight' and 'max_vst'. After each iteration the standard residuals (vst's) are compared to 'max_vst'. If a vst exceeds 'max_vst' the observation is given a reduced weight. 'max_vst' is applied when the enp is greater than zero. Normally values between 3.0 and 6.0 work well. In order to avoid unnecessary reduction of the weight on numerous observations during the first iterations a higher value for 'max_vst' is automatically calculated when the enp is less than zero. In this case 'max_vst' will be increased exponentially with decreasing enp. 'start_down_weight' is an integer variable defining the iteration number at which to initialize the search for blunders. If a number less than two is given the search starts at iteration number two.

'max_lost_digits' is the maximum number of digits that the user will accept lost during reduction of the diagonal elements of the normal equations. If more than 'max_lost_digits' are lost the column is considered singular and removed from the adjustment.

The following global variables are assigned values by the procedure: number of iterations before convergence or stop 'itt_no', the standard deviation unit weight 'sO', the adjusted element values 'Element.Element_Value', the weight factor applied to reduce the weight for observations

'Obs.Weight_Factor', and the array 'Sing' from the last iteration showing the number of lost digits multiplied by 100 for each column in the normal equation.

If an arithmetic error occurs during calculation of a observation equation, or if the Apriori error is zero for an observation the boolean 'Error' is set to TRUE and one of the following error message is given:

```
***ERROR:  Arithmetic Error during processing of:
     obs:  "Obs Group" Name:  "Obs Name"

***ERROR:  Apriori Error = zero for:
     obs:  "Obs Group" "Obs Kind" Name: "Obs Name"
```

```
GBA_Statistics
      (Residuals              :  BOOLEAN;
       Sd_elements            :  BOOLEAN;
       Aposteriori            :  BOOLEAN;
       VAR Warning, Error     :  BOOLEAN) :
```

Calculates standard deviations on elements and observations, residuals, and standard deviation on the residuals according to the specifications in the input parameters. The least squares adjustment must have been previously solved using the procedure GBA_Solve.

The input parameters specify which statistical parameters are calculated:

'Residuals'= TRUE: Residuals are calculated and assigned to the global variable 'Obs.Residual'.

'Sd_elements'= TRUE: Standard deviation on elements are calculated and assigned to the global variable 'Element.Element_Error'.

'Aposteriori'= TRUE: Standard deviations on observations and residuals are calculated and assigned to the global variables 'Obs.Aposteriori_Error' and 'Obs.Residual_Error'.

If an arithmetic error occurs during the calculation of statistics the boolean 'Warning' is set to TRUE and the following message is given:

```
*WARNING:  Arithmetic Error during processing of
           Statistics:
     obs:  "Obs Group" "Obs Kind" Name:  "Obs Name"
```

The procedure ignores this error and continues the calculation for the next observation.

The boolean 'Error' is set to TRUE if an arithmetic error occurs during calculation of an observation equation. In this case the calculation of statistics is stopped.

```
GBA_Observation_Equation
      (Obs I             :  Observation_Record_Type;
       C                 :  Real_Array_Embedded_Elements;
       Mask              :  Boolean_Array_Embedded_Elements;
       VAR Coefficients  :  Real_Array_Coefficients;
       VAR Error         :  BOOLEAN) :
```

Defines the calls to the user defined observation equation procedures. 'GBA_Observation_Equation' should not be called by the user and the input parameters can be ignored. However, the procedure has to be updated when a new observation equation procedure is added to the adjustment.

Add a new observation equation called 'New_Observation_Equation' with the observation group called 'New_Group' by updating the case statement in the following way:

```
   New_Group:  New_Observation_Equation
               (Observation, C, Mask, Coefficients, Error) :
```

The boolean 'Error' passes an error status from the user written observation equation procedures to the calling program 'GBA_Solve' or 'GBA_Statistics'.

```
PROCEDURE GBA_Write_Pass_Information
      (enp               :  DOUBLE;
       n_down            :  INTEGER;
       vstmax            :  DOUBLE;
       max_lost_digits   :  INTEGER;
       VAR Stop          :  BOOLEAN) :
```

This procedure is called by 'GBA_Solve' after each iteration. The values passed are the current values of local variables in 'GBA_Solve' of interest for the user. In here the user can design and write the action he desires to be taken after each iteration, based on the passed values and the current value of global variables.

'enp' is the current exponent of numerical precision, 'n_down' is the number of observations that have been given a reduced weight during the last iteration, 'vstmax' is the standard residual value used as test limit for the weight

reduction. See description of the procedure 'GBA_Solve' for explanation of 'max_lost_digits'.

'GBA_Write_Pass_Information' is pre-programmed to write the current values of the parameters including the global's 'sO' and 'itt_no' to 'OUTPUT', analyze the array 'Sing' for singularities and write the identification for unknowns that caused the singularity.

```
PROCEDURE GBA_Write_String
      (VAR Out_File : TEXT;
       String       : PACKED ARRAY [f..1: INTEGER] OF CHAR):
```

Help procedure for writing of names without tailing blanks. The procedure writes a packed array of character to the output media 'Out_File'. Any tailing blanks are ignored. Embedded blanks are allowed.

HOW TO ADD A NEW OBSERVATION EQUATION

The following explains in detail the steps to go through in order to add a new observation equation and a new kind of element to the adjustment.

(A) Write the observation equation procedure.
(B) Write a procedure to read the new kinds of elements.
(C) Write a procedure to read the new observations.
(D) Update the global's.
(E) Call the new observation equation procedure.
(F) Compile and link.

(A) Write the observation equation procedure.

Implement a procedure with the mathematical formulas for the new observation equation and the partial deviations. The procedure is placed in a module called OBSL*.PAS.

To exemplify the following, have 'Length_Observation_Equation' in MODULE OBSL03.PAS handy during study.

First define the embedded observations and elements in the arrays 'Observation' and 'Element'. Remember, that constants which you want to be able to redefine as unknowns should appear in the 'Element' array. The order chosen for the embedded observations and elements should be kept throughout the code. The order for groups of embedded elements with the same name, should be the same, as the order

in which the elements are listed in the 'Element_Kind_Type' in the global's.

In the example there is one embedded observation, namely the horizontal distance 'L', and four embedded elements, namely the horizontal coordinates for the two end points. Note, that an end point can be considered a given point by merely defining the coordinate as CONSTANT during the adjustment.

Assign the calculated coefficients and right hand sides to the columns of the matrix 'Coefficients'. One row is defined for each embedded observation. The coefficients are assigned to the columns corresponding to the order of the elements as defined above. The right hand side is assigned to the column with the number "number of embedded elements + 1". In the example a constant 'Konstant' is declared with the column number of the right hand side.

The array 'Mask' holds a boolean for each element. If 'Mask' is FALSE for an element, the element is a constant in the given adjustment and therefore, it is not necessary to calculate the coefficient for that particular element.

Include error traps for arithmetic errors and pass the error flag via the boolean 'Error'. In the example procedure an arithmetic error would result if the length was calculated to zero.

(B) Write a procedure to read the new kinds of elements.

If the observation involves new kinds of elements then implement a procedure to input and define the elements. The procedure should be placed in a module named READELE*.PAS.

To exemplify the following, have 'Read_Point_Elements' in MODULE READELE03.PAS handy during study.

In order to define elements the program needs to know the element kind, the block to which the element or element group belongs, the element status ('UNKNOWN' or 'CONSTANT'), and, of course, a preliminary value for the element. Read these values and assign them to the input identifiers of the procedure 'GBA_Elements'.

Note in the example, that 'GBA_Elements' are called three times, one for each coordinate (X, Y, Z). The three coordinates are entered as a group under one name. In order to refer for instance the Z coordinate one needs to specify both the 'Element_Name' and the 'Element_Kind' ('GROUND_Z'), as well as the block 'POINT_BLOCK' to which the element belongs.

(C) Write a procedure to read the new observations.

Implement a procedure that input and define the new observations. The procedure should be placed in a module named READOBS*.PAS.

To exemplify the following, have 'Read_Length_Observations' in MODULE READOBS03.PAS handy during study.

To define an observation the program needs to know the observed value, an Apriori error for the observation, the embedded elements, and the observation group that points to the observation equation procedure.

The example procedure shows how to enter a horizontal distance observation. The distance 'L', the Apriori error 'A1', and the names of the end points are entered. Then the procedure 'GBA_Observations' is called to define the observation and build the pointers to the element array.

Since we deal with a horizontal distance, there are four elements involved namely the plane coordinates of the end points as defined earlier in the observation equation procedure. Since point elements have been entered as groups of (X, Y, Z) coordinates under one name, we need to specify the kinds 'GROUND_X' and 'GROUND_Y'.

As a length observation introduces a geodetic binding between two points, the coordinate elements in the length observation need to be moved to the 'SURVEY_BLOCK'. Therefore 'SURVEY_BLOCK' is specified for the elements. 'GBA_Observation' now looks both under 'POINT_BLOCK' and 'SURVEY_BLOCK' in its search for the point names. If a point is found under 'POINT_BLOCK' it is automatically moved to the 'SURVEY_BLOCK'.

The first "4" in the call of 'GBA_Observations' is the number of entries in the 'Names', 'Kinds', and 'Blocks' arrays. The second "4" is the number of embedded elements as specified in the observation equation procedure. If all three elements for a point had been addressed by the observation, only the following two entries in the arrays : 'Names', 'Kinds', and 'Blocks' would have been needed.

```
Names[1]:=  Name1;          Names[2]:=  Name2;
Kinds[1]:=  WHOLE GROUP;    Kinds[2]:=  WHOLE GROUP;
Blocks[1]:= SURVEY BLOCK;   Blocks[2]:= SURVEY BLOCK;
```

In the example the name of the observation is the same as the name for the second end point. This is not a good

choice, but the program do not care which name you give the observation.

(D) Update the global's.

Declare the new procedures as EXTERNAL in the global module BUND.GLO.

Insert the identifiers for the new kind of elements in 'Element_Kind_Type'. In the example the identifiers are 'GROUND_X', 'GROUND_Y', and 'GROUND_Z'.

Insert the identifier for the observation group in 'Observation_Group_Type'. In the example the identifier is 'LENGTH_GROUP'.

Insert the identifier for the observation kind in 'Observation_Kind_Type'. In the example the identifier is 'LENGTH'.

(E) Call the new observation equation procedure.

Add the call of the observation equation procedure to the CASE sentence in procedure 'GBA_Observation_Equation' in module BUND07.PAS. In the example the following CASE would be added:

```
LENGTH_GROUP:  Length_Observation_Equation
                 (Observation, C, Mask, Coefficients);
```

(F) Compile and link.

Add the new modules to the command file BUND.COM for compiling, BUND.LNK for linking, and BUND.PRT for printing.

ACKNOWLEDGEMENTS

Parts of the project were supported by the Danish National Research Councils and other parts were carried out under contract with the U.S. Geological Survey.

REFERENCES

Alexandersen, S.R., 1974. Aerotriangulation - Straaleudjaevning. Institute of Surveying and Photogrammetry, Technical University of Denmark.

Dueholm, K.S., 1983. CAMCAL: Camera Calibration Program. Institute of Surveying and Photogrammetry, Technical University of Denmark.

Jacobi, O., 1968. LFL100: Fotogrammetriprogram. Institute of Surveying and Photogrammetry, Technical University of Denmark.

Krarup, T., K. Kubik, J. Juhl, 1980. Goetterdammerung over Least Squares Adjustment. Proceedings, 14th Congress of ISP, Hamburg.

Krarup, T., 1982. Non-Linear Adjustment and Curvature. Daar heb ik veertis jaar over nasedacht... Deel 1, p. 146-159. Edited by M.J. Blotwijk et al. Delft.

Poder, K., 1982. Data Processing and Adjustment. Proceedings, FIG Study Group 5B. Survey Control Networks, p. 327-337. Aalbors, 1982.

Thomsen, B.V., 1987. NEWBUN: Principper & Systembeskrivelse. Report to the Danish National Research Councils. Institute of Surveying and Photogrammetry, Technical University of Denmark.

Weng, W.L., 1984. LINLIB. Institute of Surveying and Photogrammetry, Technical University of Denmark.

Weng, W.L., 1986. LINLIB 2.0. Institute of Surveying and Photogrammetry, Technical University of Denmark.

SOURCE CODE LISTING

```
[ENVIRONMENT ('BUND.PEN')] MODULE BUNDGLO (OUTERR) ;

CONST
  Pi = 3.1415926536 ;

  Max_Name_Length                 = 20 ;
  Max_Number_Elements             = 2000 ;
  Max_Embedded_Elements           = 30 ;
  Max_Embedded_Observations       = 4 ;
  Max_Number_Observations         = 4000 ;
  Max_Normal_Equation_Size        = 200000 ;


TYPE
  Name_String =
    PACKED ARRAY [1..Max_Name_Length] OF CHAR ;

  Boolean_Array_Embedded_Elements =
    ARRAY [1..Max_Embedded_Elements] OF BOOLEAN ;

  Integer_Array_Elements =
    ARRAY [1..Max_Number_Elements] Of INTEGER ;
  Integer_Array_Embedded_Elements =
    ARRAY [1..Max_Embedded_Elements] OF INTEGER ;

  Real_Array_2 =
    ARRAY [1..2] OF DOUBLE ;
  Real_Array_3_3 =
    ARRAY [1..3,1..3] OF DOUBLE ;

  Real_Array_Elements =
    ARRAY [1..Max_Number_Elements] Of DOUBLE ;
  Real_Array_Embedded_Elements =
    ARRAY [1..Max_Embedded_Elements] OF DOUBLE ;
  Real_Array_Normal_Equations =
    ARRAY [1..Max_Normal_Equation_Size] of DOUBLE ;
  Real_Array_Embedded_Observations =
    ARRAY [1..Max_Embedded_Observations] OF DOUBLE ;
  Real_Array_Coefficients =
    ARRAY [1..Max_Embedded_Observations,
           1..Max_Embedded_Elements] OF DOUBLE ;
```

```
{ ELEMENT TYPES }

Element_Block_Type    = ( POINT_BLOCK,
                          SURVEY_BLOCK,
                          PHOTO_BLOCK,
                          CAMERA_BLOCK
                        ) ;

Element_Kind_Type     = ( WHOLE_GROUP,

                          CAMERA_C,
                          CAMERA_DX,
                          CAMERA_DY,
                          CAMERA_A1,
                          CAMERA_A2,
                          CAMERA_A0,

                          PHOTO_X0,
                          PHOTO_Y0,
                          PHOTO_Z0,
                          PHOTO_OMEGA,
                          PHOTO_FI,
                          PHOTO_KAPPA,
                          PHOTO_DX,
                          PHOTO_DY,

                          GROUND_X,
                          GROUND_Y,
                          GROUND_Z,

                          THEO_R
                        ) ;

Element_Status_Type   = ( UNKNOWN,
                          CONSTANT
                        ) ;

Element_Record_Type =
  RECORD
  Element_Name              : Name_String ;
  Element_Kind              : Element_Kind_Type ;
  Element_Block             : Element_Block_Type ;
  Element_Status            : Element_Status_Type ;
  Element_Used              : BOOLEAN ;

  Element_Value,
  Element_Error             : DOUBLE ;
  END ;

Element_Array_Type =
  ARRAY [1..Max_Number_Elements] OF Element_Record_Type ;
```

```
  { OBSERVATION TYPES }

  Observation_Group_Type = ( PHOTO_GROUP,
                             LENGTH_GROUP,
                             THEOHZ_GROUP,
                             ELEMENT_GROUP
                           ) ;

  Observation_Kind_Type  = ( PHOTO_X,
                             PHOTO_Y,

                             POINT_X,
                             POINT_Y,
                             POINT_Z,

                             LENGTH,
                             THEO_HZ
                           ) ;

  Kind_Array_Embedded_Observations =
    ARRAY [1..Max_Embedded_Observations] OF Observation_Kind_Type ;

  Observation_Record_Type =
    RECORD
    Observation_Name             : Name_String ;
    Observation_Group            : Observation_Group_Type ;

    Number_Embedded_Elements     : INTEGER ;
    Element_Pointer              : Integer_Array_Embedded_Elements ;

    Number_Embedded_Observations: INTEGER ;
    Observation_Kind             : Kind_Array_Embedded_Observations ;
    Observation,
    Apriori_Error,
    Aposteriori_Error,
    Residual,
    Residual_Error,
    Weight_Factor                : Real_Array_Embedded_Observations ;
    END ;

  Observation_Array_Type =
    ARRAY [1..Max_Number_Observations] OF Observation_Record_Type ;


VAR

  { INPUT AND OUTPUT VARIABLES }

  { The element array }
```

```
  Element:                Element_Array_Type ;
  End_Element_Array:      INTEGER ;

  { The observation array }
  Obs:                    Observation_Array_Type ;
  End_Obs_Array:          INTEGER ;

  { heighest normal equation column number in the specified element block }
  Max_Column_In_Block:    ARRAY [Element_Block_Type] OF INTEGER ;

  { File for Output of Warnings and Errors }
  OUTERR: TEXT ;


  { ADJUSTMENT VARIABLES }

  { Actual number of elements and observations in the adjustment }
  NEle:                   INTEGER ;
  NObs:                   INTEGER ;

  { Normal equation pointers }
  Col, Ele:               Integer_Array_Elements ;

  { The normal equation array and size }
  N:                      Real_Array_Normal_Equations ;
  N_Size:                 INTEGER ;

  { pass information }
  itt_no:                 INTEGER ;
  s0:                     DOUBLE ;

  { Integer arrays form LINLIB used for profiled storage and solving }
  { of normal equation                                               }
  Base:                   ARRAY [0..Max_Number_Elements] OF INTEGER ;
  Sing:                   ARRAY [0..Max_Number_Elements] OF INTEGER ;


{ GBA CORE PROCEDURES }

[EXTERNAL] PROCEDURE GBA_Initialize
      ( VAR Error           : BOOLEAN ) ;
  EXTERNAL ; { bund01 }

[EXTERNAL] PROCEDURE GBA_Elements
          ( Name            : Name_String ;
            Kind            : Element_Kind_Type ;
            Block           : Element_Block_Type ;
            Status          : Element_Status_Type ;
            Value           : DOUBLE ;

        VAR Warning, Error : BOOLEAN ) ;
```

```
  EXTERNAL ; { bund02 }

[EXTERNAL] PROCEDURE GBA_Observations
          ( Obs_Name          : Name_String ;
            Obs_Group         : Observation_Group_Type ;
            Embedded_Obs      : INTEGER ;
            Obs_Kind          : ARRAY [f1..l1: INTEGER] OF Observation_Kind_Type;
            Obs_Value         : ARRAY [f2..l2: INTEGER] OF DOUBLE ;
            Apriori           : ARRAY [f3..l3: INTEGER] OF DOUBLE ;

            Number_Groups     : INTEGER ;
            Names             : ARRAY [f4..l4: INTEGER] OF Name_String ;
            Kinds             : ARRAY [f5..l5: INTEGER] OF Element_Kind_Type;
            Blocks            : ARRAY [f6..l6: INTEGER] OF Element_Block_Type ;
            Embedded_Ele      : INTEGER ;

        VAR Warning, Error : BOOLEAN ) ;
  EXTERNAL ; { bund03 }

[EXTERNAL] PROCEDURE GBA_Pointers
      ( VAR Error           : BOOLEAN );
  EXTERNAL ; { bund04 }

[EXTERNAL] PROCEDURE GBA_Solve
          ( enp_stop        : DOUBLE ;
            max_itt         : INTEGER ;

            start_down_weight: INTEGER ;
            max_vst         : DOUBLE ;

            max_lost_digits: INTEGER ;

        VAR Error           : BOOLEAN ) ;
  EXTERNAL ; { bund05 }

[EXTERNAL] PROCEDURE GBA_Statistics
          ( residuals       : BOOLEAN ;
            sd_elements     : BOOLEAN ;
            aposteriori     : BOOLEAN ;

        VAR Warning, Error : BOOLEAN ) ;
  EXTERNAL ; { bund06 }

[EXTERNAL] PROCEDURE GBA_Observation_Equation
          ( Obs_I           : Observation_Record_Type ;
            C               : Real_Array_Embedded_Elements ;
            Mask            : Boolean_Array_Embedded_Elements ;

        VAR Coefficients    : Real_Array_Coefficients ;

        VAR Error           : BOOLEAN ) ;
```

```
  EXTERNAL ; { bund07 }

[EXTERNAL] PROCEDURE GBA_Write_Pass_Information
          ( enp              : DOUBLE ;
            n_down           : INTEGER ;
            vstmax           : DOUBLE ;
            max_lost_digits  : INTEGER ;

        VAR Stop             : BOOLEAN ) ;
  EXTERNAL ; { bund08 }

[EXTERNAL] PROCEDURE GBA_Write_String
      ( VAR Out_File         : TEXT ;
            String           : PACKED ARRAY [f..l: INTEGER] OF CHAR ) ;
  EXTERNAL ; { bund09 }


{ OBSERVATION EQUATION PROCEDURES }

[EXTERNAL] PROCEDURE Photo_Observation_Equation
          ( Observation    : Real_Array_Embedded_Observations  ;
            Element        : Real_Array_Embedded_Elements  ;
            Mask           : Boolean_Array_Embedded_Elements  ;

        VAR Coefficient    : Real_Array_Coefficients ;

        VAR Error          : BOOLEAN ) ;
  EXTERNAL ; { obsl01 }

[EXTERNAL] PROCEDURE Element_Observation_Equation
          ( Observation    : Real_Array_Embedded_Observations  ;
            Element        : Real_Array_Embedded_Elements  ;
            Mask           : Boolean_Array_Embedded_Elements  ;

        VAR Coefficient    : Real_Array_Coefficients ;

        VAR Error          : BOOLEAN ) ;
  EXTERNAL ; { obsl02 }

[EXTERNAL] PROCEDURE Length_Observation_Equation
          ( Observation    : Real_Array_Embedded_Observations  ;
            Element        : Real_Array_Embedded_Elements  ;
            Mask           : Boolean_Array_Embedded_Elements  ;

        VAR Coefficient    : Real_Array_Coefficients ;

        VAR Error          : BOOLEAN ) ;
  EXTERNAL ; { obsl03 }

[EXTERNAL] PROCEDURE TheoHZ_Observation_Equation
          ( Observation    : Real_Array_Embedded_Observations  ;
```

```
            Element          : Real_Array_Embedded_Elements  ;
            Mask             : Boolean_Array_Embedded_Elements  ;

        VAR Coefficient      : Real_Array_Coefficients ;

        VAR Error            : BOOLEAN ) ;
  EXTERNAL ; { obsl04 }


{ INPUT PROCEDURES }

[EXTERNAL] PROCEDURE Read_String
           ( VAR In_File: TEXT ;
             VAR String: PACKED ARRAY [f..l: INTEGER] OF CHAR ) ;
  EXTERNAL ; { read01 }

[EXTERNAL] PROCEDURE Read_Camera_Elements
           ( VAR Error: BOOLEAN ) ;
  EXTERNAL ; { readele01 }

[EXTERNAL] PROCEDURE Read_Photo_Elements
           ( VAR Error: BOOLEAN ) ;
  EXTERNAL ; { readele02 }

[EXTERNAL] PROCEDURE Read_Point_Elements
           ( VAR Error: BOOLEAN ) ;
  EXTERNAL ; { readele03 }

[EXTERNAL] PROCEDURE Read_Photo_Observations
           ( VAR Error: BOOLEAN ) ;
  EXTERNAL ; { readobs01 }

[EXTERNAL] PROCEDURE Read_Point_Observations
           ( VAR Error: BOOLEAN ) ;
  EXTERNAL ; { readobs02 }

[EXTERNAL] PROCEDURE Read_Length_Observations
           ( VAR Error: BOOLEAN ) ;
  EXTERNAL ; { readobs03 }

[EXTERNAL] PROCEDURE Read_TheoHZ_Observations
           ( VAR Error: BOOLEAN ) ;
  EXTERNAL ; { readobs04 }


{ LINLIB PROCEDURES AND FUNCTIONS }

[external] procedure    chopro
(    order:  integer;
     rhs:    integer;
     limit:  integer;
```

```
 var N:      array[firstN..lastN: integer] of double;
 var Base:   array[firstB..lastB: integer] of integer;
 var Sing:   array[firstS..lastS: integer] of integer);
external;

[external] procedure    addObs
(    nn:     integer;
 var C:      array[firstC..lastC: integer] of double;
 var index:  array[firstI..lastI: integer] of integer;
 var N:      array[firstN..lastN: integer] of double;
 var Base:   array[firstB..lastB: integer] of integer);
external;

[external] function     quadic
(    nn:     integer;
 var C:      array[firstC..lastC: integer] of double;
 var index:  array[firstI..lastI: integer] of integer;
 var Q:      array[firstN..lastN: integer] of double;
 var Base:   array[firstB..lastB: integer] of integer) : double;
external;

[external] procedure    proinv
(    order:  integer;
 var N:      array[firstN..lastN: integer] of double;
 var Base:   array[firstB..lastB: integer] of integer;
 var Candi:  array[firstC..lastC: integer] of integer);
external;

END.
```

```
[INHERIT ('BUND.PEN')] MODULE BUND01 ;

[GLOBAL] PROCEDURE GBA_Initialize ( VAR Error: BOOLEAN ) ;

BEGIN { GBA_Initialize }

  End_Element_Array:= 0 ;

  End_Obs_Array:= 0 ;

  Error:= FALSE ;

END { GBA_Initialize } ;
END.
```

```
[INHERIT ('BUND.PEN')] MODULE BUND02 ;

{  }

[GLOBAL] PROCEDURE GBA_Elements
          ( Name            : Name_String ;
            Kind            : Element_Kind_Type ;
            Block           : Element_Block_Type ;
            Status          : Element_Status_Type ;
            Value           : DOUBLE ;
        VAR Warning, Error  : BOOLEAN ) ;

  VAR
    i, j     : INTEGER ;
    found    : BOOLEAN ;

  PROCEDURE Write_Warning ;
    VAR
      l: INTEGER ;

    BEGIN
    WRITELN (OUTERR) ;
    WRITELN (OUTERR,
      '*WARNING: Element entered twice. Last entered values used.');
    WRITE   (OUTERR,
      '        > ', Block, Kind, ',  Name: ') ;
    GBA_Write_String (OUTERR, Name); WRITELN (OUTERR) ;
    Warning:= TRUE ;
    END ;

  BEGIN { GBA_Elements }
  Warning:= FALSE ;

  {look for same element Name and block}
  found:= FALSE; i:= 0; j:= 0 ;
  IF End_Element_Array > 0 THEN
    REPEAT
    i:= i + 1 ;
    WITH Element[I] DO
      BEGIN
      IF (Element_Name= Name) AND (Element_Block= Block) THEN
        BEGIN
        j:= j + 1 ;
        IF Element_Kind= Kind THEN
          BEGIN
          found:= TRUE ;
          Write_Warning ;
          END ;
        END ;
      END ;
    UNTIL (i= End_Element_Array) OR (found)
```

```
  ELSE
    End_Element_Array:= 0 ;

  {create next element position}
  IF NOT found THEN IF End_Element_Array >= Max_Number_Elements THEN
    BEGIN
    WRITELN (OUTERR, '***ERROR: Maximum number of Elements exceeded') ;
    Error:= TRUE ;
    END
  ELSE
    BEGIN
    End_Element_Array:= End_Element_Array + 1 ;
    i:= End_Element_Array ;
    END ;

  {assign element values}
  IF NOT Error THEN WITH Element[i] DO
    BEGIN
    Element_Name:= Name ;
    Element_Kind:= Kind ;
    Element_Block:= Block ;
    Element_Status:= Status ;
    Element_Used:= FALSE ;

    Element_Value:= Value ;
    Element_Error:= 0.00  ;
    END ;

END { GBA_Elements } ;
END.
```

```
[INHERIT ('BUND.PEN')] MODULE BUND03 ;

[GLOBAL] PROCEDURE GBA_Observations
          ( Obs_Name        : Name_String ;
            Obs_Group       : Observation_Group_Type ;
            Embedded_Obs    : INTEGER ;
            Obs_Kind        : ARRAY [f1..l1: INTEGER] OF Observation_Kind_Type;
            Obs_Value       : ARRAY [f2..l2: INTEGER] OF DOUBLE ;
            Apriori         : ARRAY [f3..l3: INTEGER] OF DOUBLE ;

            Number_Groups   : INTEGER ;
            Names           : ARRAY [f4..l4: INTEGER] OF Name_String ;
            Kinds           : ARRAY [f5..l5: INTEGER] OF Element_Kind_Type;
            Blocks          : ARRAY [f6..l6: INTEGER] OF Element_Block_Type ;
            Embedded_Ele    : INTEGER ;

        VAR Warning, Error : BOOLEAN ) ;

  VAR
    pil         : ARRAY [1..Max_Embedded_Elements] OF INTEGER ;
    i, j, k     : INTEGER ;
    found       : BOOLEAN ;

  LABEL
    99 ;

  PROCEDURE Write_Warning ;
    VAR
      l: INTEGER ;
    BEGIN
    Warning:= TRUE ;
    WRITELN (OUTERR) ;
    WRITELN (OUTERR,
      '*WARNING: Element not found. Observation ignored.') ;

    FOR l:= 1 TO Number_Groups DO IF l<> i THEN
      BEGIN
      WRITE (OUTERR, '               ', Blocks[l], Kinds[l], ',  Name: ') ;
      GBA_Write_String (OUTERR, Names[l]); WRITELN (OUTERR) ;
      END
    ELSE
      BEGIN
      WRITE (OUTERR, '            >  ', Blocks[l], Kinds[l], ',  Name: ') ;
      GBA_Write_String (OUTERR, Names[l]) ; WRITELN (OUTERR) ;
      END ;
    WRITE (OUTERR, '          obs:', Obs_Group,  ',  Name: ' ) ;
    GBA_Write_String (OUTERR, Obs_Name); WRITELN (OUTERR) ;
    END ;

  BEGIN { GBA_Observations }
  IF End_Obs_Array >= Max_Number_Observations THEN
```

```
    BEGIN
    WRITELN (OUTERR,
      '***ERROR: Maximum number of observations exceeded') ;
    Error:= TRUE ;
    GOTO 99 ;
    END ;

  IF Embedded_Obs >= Max_Embedded_Observations THEN
    BEGIN
    WRITELN (OUTERR,
      '***ERROR: Maximum number of embedded observations exceeded') ;
    WRITE (OUTERR, '     Obs: ', Obs_Group, ',  Name: ') ;
    GBA_Write_String (OUTERR, Obs_Name); WRITELN (OUTERR) ;
    Error:= TRUE ;
    GOTO 99 ;
    END ;

  IF Embedded_Ele >= Max_Embedded_Elements THEN
    BEGIN
    WRITELN (OUTERR,
      '***ERROR: Maximum number of embedded elements exceeded') ;
    WRITE (OUTERR, '     Obs: ', Obs_Group, ',  Name: ') ;
    GBA_Write_String (OUTERR, Obs_Name); WRITELN (OUTERR);
    Error:= TRUE ;
    GOTO 99 ;
    END ;

  Warning:= FALSE ;
  End_Obs_Array:= End_Obs_Array + 1 ;

  FOR i:= 1 TO Number_Groups DO pil[i]:= 0 ; j:= 0 ;
  IF End_Element_Array > 0 THEN
    REPEAT
    found:= TRUE; j:= j + 1 ;
    FOR i:= 1 TO Number_Groups DO
      IF (Element[j].Element_Name= Names[i]) AND  (pil[i]= 0) THEN
        BEGIN
        IF (Element[j].Element_Block= Blocks[i])
        OR (((Blocks[i]= POINT_BLOCK) OR (Blocks[i]= SURVEY_BLOCK)) AND
            (Element[j].Element_Block IN [POINT_BLOCK, SURVEY_BLOCK]))
          THEN pil[i]:= j ;
        END ;
{       IF     (Element[j].Element_Block= Blocks[i])
        AND  (Element[j].Element_Name= Names[i])
        AND  (pil[i]= 0)
        THEN pil[i]:= j ;}
    FOR i:= 1 TO Number_Groups DO IF pil[i]= 0 THEN found:= FALSE ;
    UNTIL found OR (j= End_Element_Array) ;
  FOR i:= 1 TO Number_Groups DO IF pil[i]= 0 THEN Write_Warning ;

  IF Found THEN WITH Obs[End_Obs_Array] DO
```

```
    BEGIN
    j:= 0 ;
    FOR i:= 1 TO Number_Groups DO
      BEGIN
      found:= FALSE ;
      WHILE (Element[pil[i]].Element_Name= Names[i]) DO
        WITH Element[pil[i]] DO
        IF (Element_Block= Blocks[i])
        OR (((Blocks[i]= POINT_BLOCK) OR (Blocks[i]= SURVEY_BLOCK)) AND
            (Element_Block IN [POINT_BLOCK, SURVEY_BLOCK]))
          THEN BEGIN
               IF (Kinds[i]= WHOLE_GROUP) OR (Kinds[i]= Element_Kind)
                 THEN BEGIN
                      j:= j + 1; found:= TRUE ;
                      Element_Pointer[j]:= pil[i] ;
                      IF Blocks[i]= SURVEY_BLOCK
                        THEN Element_Block:= SURVEY_BLOCK ;
                      END ;
               pil[i]:= pil[i] + 1 ;
               END ;
      IF NOT found THEN Write_Warning ;
      END ;
      IF Embedded_Ele= j THEN
        BEGIN
        Observation_Name:= Obs_Name ;
        Observation_Group:= Obs_Group ;
        Number_Embedded_Elements     := Embedded_Ele ;
        Number_Embedded_Observations:= Embedded_Obs ;
        FOR I:= 1 TO Number_Embedded_Observations DO
          BEGIN
          Observation_Kind[i]:= Obs_Kind[i] ;
          Observation[i]:= Obs_Value[i] ;
          Apriori_Error[i]:= Apriori[i] ;
          Aposteriori_Error[i]:= Apriori[i] ;
          END ;
        END
      ELSE
        BEGIN
        i:= 0 ; Write_Warning ;
        END ;
    END ;

  IF Warning OR Error THEN End_Obs_Array:= End_Obs_Array - 1 ;

99: END { GBA_Observations } ;
END.
```

```
[INHERIT ('BUND.PEN')] MODULE BUND04 ;

[GLOBAL] PROCEDURE GBA_Pointers
      ( VAR Error           : BOOLEAN );

{ compute, Col, Ele, NEle, Base, NObs, N_Size }

  VAR
  B    : Element_Block_Type ;
  Index: Integer_Array_Embedded_Elements ;

  I, J, K, nn, ij, jIndex, row, column: INTEGER ;

  BEGIN { GBA_Pointers }

  FOR I:= 1 to End_Obs_Array DO WITH Obs[I] DO
    FOR J:= 1 TO Number_Embedded_Elements DO
      Element[Element_Pointer[J]].Element_Used:= TRUE ;

  K:= 0 ;
  FOR B:= POINT_BLOCK TO CAMERA_BLOCK DO
    BEGIN
    FOR J:= 1 TO End_Element_Array DO
      BEGIN
      IF (Element[J].Element_Block = B)
         AND (Element[J].Element_Status = UNKNOWN)
         AND (Element[J].Element_Used)                THEN
        BEGIN
        K:= K + 1 ;
        Col[J]:= K ;
        Ele[K]:= J ;
        END ;
      END ;
    Max_Column_In_Block[B]:= K ;
    END ;
  NEle:= K ;


  FOR I:= 1 TO NEle + 1 DO Base[I]:= I ;
  NObs:= 0 ;

  FOR I:= 1 to End_Obs_Array DO
    WITH Obs[I] DO
      BEGIN
       { get involved elements and constants }
      nn:= 0 ;
      FOR J:= 1 TO Number_Embedded_Elements DO
        BEGIN
        K:= Element_Pointer[J] ;
        WITH Element[K] DO
          IF Element_Status = UNKNOWN
```

```
        THEN BEGIN
             nn:= nn+1 ;
             index[nn]:= Col[K] { Normal Equation Column }
             END ;
    END ;

  {add right hand side}
  nn:= nn + 1; index[nn]:= NEle + 1 ;

  NObs:= NObs + Number_Embedded_Observations ;

  { add coefficients to normal equations }
    FOR J:= 1 TO nn DO
      BEGIN
      jIndex:= Index[J];

      FOR K:= 1 TO J DO
        BEGIN
        column:= jIndex;
        row:= index[K];
        IF row > column THEN
          BEGIN
          ij:= row;
          row:= column;
          column:= ij;
          END;

        IF Base[column] > row THEN Base[column]:= row ;
        END; {K:= 1 to J}
      END; {J:= 1 to nn}

  END ; {I:= 1 to End_Obs_Array}

Base[0]:= 0 ;
FOR I:= 1 TO NEle + 1 DO Base[I]:= Base[I-1] + I - Base[I] ;

N_Size:= Base[NEle + 1] + NEle + 1 ;

IF N_Size > Max_Normal_Equation_Size THEN
  BEGIN
  Error:= TRUE ;
  WRITELN (OUTERR) ;
  WRITELN (OUTERR,
    '***ERROR: Normal Equation > Max_Normal_Equation_Size');
  END ;

IF NObs < NEle THEN
  BEGIN
  Error:= TRUE ;
  WRITELN (OUTERR) ;
  WRITELN (OUTERR,
```

```
        '***ERROR: Number Observations < Number Elements');
      END ;

END { GBA_Pointers } ;
END.
```

```
[INHERIT ('BUND.PEN')] MODULE BUND05 (OUTPUT) ;

{  }

[GLOBAL] PROCEDURE GBA_Solve
          ( enp_stop        : DOUBLE ;
            max_itt         : INTEGER ;

            start_down_weight: INTEGER ;
            max_vst         : DOUBLE ;

            max_lost_digits: INTEGER ;

        VAR Error           : BOOLEAN ) ;

VAR
  I, J, n_down       : INTEGER ;
  vst, vstmax, enp   : DOUBLE ;
  Stop               : BOOLEAN ;

LABEL
  99 ;

PROCEDURE Normal_Equations ;

  VAR
    C                : Real_Array_Embedded_Elements ;
    Index            : Integer_Array_Embedded_Elements ;
    Mask             : Boolean_Array_Embedded_Elements ;
    Coefficients     : Real_Array_Coefficients ;

    nn, I, J, K, L : INTEGER ;
    P                : DOUBLE ;

  BEGIN { PROCEDURE Normal_Equations }

  n_down:= 0 ;

  FOR I:= 1 to End_Obs_Array DO
    WITH Obs[I] DO
      BEGIN

      { ------------------------------------------------------ }
      { test output to show elapsed time                       }
      {write (OUTPUT, chr(%O'33'), '[', 23: 1, ';', 1: 1, 'H' ) ;}
      {WRITELN ( OUTPUT, 'Normal Equations Forming       : ',
                      TRUNC(I*100.0/End_Obs_Array):3, '% done') ;}
      { ------------------------------------------------------ }

      { get involved elements and constants }
      nn:= 0 ;
```

```
FOR J:= 1 TO Number_Embedded_Elements DO
  BEGIN
  K:= Element_Pointer[J] ;
  WITH Element[K] DO
    BEGIN
    C[J]:= Element_Value ;
    IF Element_Status = UNKNOWN
      THEN BEGIN
           Mask[J]:= TRUE ;
           nn:= nn+1 ;
           index[nn]:= Col[K] { Normal Equation Column }
           END
      ELSE IF Element_Status = Constant THEN Mask[J]:= FALSE ;
    END ;
  END ;

{add right hand side}
nn:= nn + 1 ; index[nn]:= NEle + 1 ;
Mask[Number_Embedded_Elements+1]:= TRUE ;

{ calculate observation equations }
GBA_Observation_Equation ( Obs[I], C, Mask, Coefficients, Error ) ;
IF Error THEN GOTO 99 ;

{ add coefficients to normal equations }
FOR L:= 1 TO Number_Embedded_Observations DO
  BEGIN
  Residual[L]:= Coefficients[L,Number_Embedded_Elements+1] ;
    { look for gross errors }
    IF Apriori_Error[L] <= 0.00 THEN
      BEGIN
      Error:= TRUE ;
      WRITELN (OUTERR) ;
      WRITELN (OUTERR, '***ERROR: Apriori Error = zero for:') ;
      WRITE   (OUTERR, '     Obs: ', Observation_Group,
                        Observation_Kind[L], ',  Name: ') ;
      GBA_Write_String (OUTERR, Observation_Name);
      WRITELN (OUTERR) ;
      GOTO 99 ;
      END ;
    p:= 1.00 / Apriori_Error[L] ;
    vst:= abs (Residual[L]) / (s0*Apriori_Error[L]) ;
    IF (vst > vstmax) THEN
      BEGIN
      Weight_Factor[L]:= SQRT ( EXP (-vst/vstmax) ) ;
      P:= P * Weight_Factor[L] ;
      n_down:= n_down + 1 ;
      END
    ELSE Weight_Factor[L]:= 1.00 ;
  K:= 0 ;
  FOR J:= 1 TO Number_Embedded_Elements + 1 DO
```

```
          BEGIN
          IF Mask[J] THEN
            BEGIN
            K:= K + 1 ;
            C[K]:= Coefficients[L,J] * P ;
            END ;
          END ;
        addObs (nn, C, index, N, Base) ;
        END ;
      END ;

  END { Normal_Equations } ;


BEGIN { GBA_Solve }

  itt_no:= 0; s0:= 1; vstmax:= 1.0E38 ;
  IF max_vst <= 0.00 THEN max_vst:= vstmax ;
  IF start_down_weight < 2 THEN start_down_weight:= 2 ;

  REPEAT
    itt_no:= itt_no + 1 ;

    IF itt_no >= start_down_weight
      THEN if enp > 0.00
             THEN vstmax:= max_vst
             ELSE vstmax:= max_vst/EXP(enp) ;

    { setup normal equations }
    FOR I:= 1 to N_Size DO N[I]:= 0.00 ;
    Normal_Equations ;

    { solve normal equations }
    Chopro (NEle, 1, max_lost_digits, N, Base, Sing) ;

    { update elements }
    FOR I:= 1 to NEle DO WITH Element[Ele[I]] DO
      Element_Value:= Element_Value - N[Base[NEle + 1] + I] ;

    { calculate pass information }
    enp:= Sing[NEle + 1]/100.00 ;
    s0:= SQRT (N[N_Size]/(NObs - NEle)) ;

    GBA_Write_Pass_Information
      ( enp, n_down, vstmax, max_lost_digits, Stop ) ;

  UNTIL ((itt_no >= max_itt) OR (enp > enp_stop)) OR Stop ;

99: END ; { GBA_Solve }
END.
```

```
[INHERIT ('BUND.PEN')] MODULE BUND06 ;

[GLOBAL] PROCEDURE GBA_Statistics
          ( residuals      : BOOLEAN ;
            sd_elements    : BOOLEAN ;
            aposteriori    : BOOLEAN ;
        VAR Warning, Error : BOOLEAN ) ;

{  }

VAR
  C               : Real_Array_Embedded_Elements ;
  Index           : Integer_Array_Embedded_Elements ;
  Mask            : Boolean_Array_Embedded_Elements ;
  Coefficients    : Real_Array_Coefficients ;

  nn, I, J, K, L  : Integer ;
  Qua             : DOUBLE ;

  Candi           : ARRAY [0..Max_Number_Elements] OF INTEGER ;

LABEL
  99 ;

PROCEDURE Write_Warning ;
  BEGIN
  Warning:= TRUE ;
  WRITELN (OUTERR) ;
  WRITELN (OUTERR,
    '*WARNING: Arithmetric Error during processing of statistics:') ;
  WRITE   (OUTERR, '     Obs: ', Obs[I].Observation_Group,
                            Obs[I].Observation_Kind[L], ',  Name: ') ;
  GBA_Write_String (OUTERR, Obs[I].Observation_Name);
  WRITELN (OUTERR) ;
  END ;

BEGIN { GBA_Statistics }

  Warning:= FALSE ;

  { calculate inverse matrix }
  IF sd_elements OR aposteriori THEN
    proinv (NEle, N, Base, Candi) ;

  { calculate standard deviation on elements }
  IF sd_elements THEN
    FOR I:= 1 to NEle DO WITH Element[Ele[I]] DO
      IF N[Base[I] + I] >= 0.00 THEN
        Element_Error:= s0*SQRT (N[Base[I] + I])
      ELSE
        BEGIN
```

```
      Warning:= TRUE ;
      WRITELN (OUTERR) ;
      WRITELN (OUTERR,
        '*WARNING: Aritmetric Error during processing of statistics:') ;
      WRITE   (OUTERR,
        '     Ele: ', Element_Block, Element_Kind, ',  Name: ') ;
      GBA_Write_String (OUTERR, Element_Name);
      WRITELN (OUTERR) ;
      END ;

{ Calculate Residual, Residual_Error, and Aposteriori_Error }

IF residuals OR aposteriori THEN FOR I:= 1 to End_Obs_Array DO
WITH Obs[I] DO
  BEGIN
  { get involved elements and constants }
  nn:= 0 ;
  FOR J:= 1 TO Number_Embedded_Elements DO
    BEGIN
    K:= Element_Pointer[J] ;
    WITH Element[K] DO
      BEGIN
      C[J]:= Element_Value ;
      IF Element_Status = UNKNOWN
        THEN BEGIN
             Mask[J]:= TRUE ;
             nn:= nn+1 ;
             index[nn]:= Col[K] ;
             END
        ELSE IF Element_Status = Constant THEN Mask[J]:= FALSE ;
      END ;
    END ;

  {add right hand side}
  Mask[Number_Embedded_Elements+1]:= TRUE ;

  { calculate observation equations }
  GBA_Observation_Equation ( Obs[I], C, Mask, Coefficients, Error ) ;
  IF Error THEN GOTO 99 ;

  FOR L:= 1 TO Number_Embedded_Observations DO
    BEGIN
    Residual[L]:= Coefficients[L,Number_Embedded_Elements+1] ;
    IF aposteriori THEN
      BEGIN
      K:= 0 ;
      FOR J:= 1 TO Number_Embedded_Elements DO
        BEGIN
        IF Mask[J] THEN
          BEGIN
          K:= K + 1 ;
```

```
          C[K]:= Coefficients[L,J] ;
          END ;
        END ;
      { quadratic form calculations }
      Qua:= quadic (nn, C, index, N, Base) ;
      IF qua >= 0.00 THEN
        Aposteriori_Error[L]:= s0 * SQRT ( qua )
      ELSE
        BEGIN
        Write_Warning ;
        END ;
      IF Apriori_Error[L]*Apriori_Error[L] - qua >= 0.00 THEN
        Residual_Error[L]:=
        s0 * SQRT ( Apriori_Error[L]*Apriori_Error[L] - qua )
      ELSE
        BEGIN
        Write_Warning ;
        END ;
      END ;
    END ;
  END ;

99: END ; { GBA_Statistics }
END.
```

```
[INHERIT ('BUND.PEN')] MODULE BUND07 ;

[GLOBAL] PROCEDURE GBA_Observation_Equation
          ( Obs_I           : Observation_Record_Type ;
            C               : Real_Array_Embedded_Elements ;
            Mask            : Boolean_Array_Embedded_Elements ;
        VAR Coefficients    : Real_Array_Coefficients ;
        VAR Error           : BOOLEAN ) ;

BEGIN
  WITH Obs_I DO
    CASE Observation_Group OF
    PHOTO_GROUP:    Photo_Observation_Equation
                      (Observation, C, Mask, Coefficients, Error) ;
    LENGTH_GROUP:   Length_Observation_Equation
                      (Observation, C, Mask, Coefficients, Error) ;
    THEOHZ_GROUP:   TheoHZ_Observation_Equation
                      (Observation, C, Mask, Coefficients, Error) ;
    ELEMENT_GROUP:  Element_Observation_Equation
                      (Observation, C, Mask, Coefficients, Error) ;
    OTHERWISE {do nothing} ;
    END ;

  IF Error THEN
    BEGIN
    WRITELN (OUTERR) ;
    WRITELN (OUTERR,
      '***ERROR: Aritmetric Error during processing of:') ;
    WRITE   (OUTERR,
      '     Obs: ', Obs_I.Observation_Group, ',  Name: ') ;
    GBA_Write_String (OUTERR, Obs_I.Observation_Name );
    WRITELN (OUTERR) ;
    END ;

END ; { Observation_Equation }
END.
```

```
[INHERIT ('BUND.PEN')] MODULE BUND08 (OUTPUT) ;

{  }

[GLOBAL] PROCEDURE GBA_Write_Pass_Information
        ( enp              : DOUBLE ;
          n_down           : INTEGER ;
          vstmax           : DOUBLE ;
          max_lost_digits  : INTEGER ;

      VAR Stop             : BOOLEAN ) ;

VAR
  I, J:     Integer ;

BEGIN

  Stop:= FALSE ;

  WRITELN (OUTPUT) ;
  WRITELN (OUTPUT, 'Iteration Number ', itt_no:3) ;
  WRITELN (OUTPUT, '--------------------') ;
  WRITELN (OUTPUT,
    'ENP                                  : ', enp:4:1) ;
  WRITELN (OUTPUT,
    'Maximum Lost Decimal Digits          : ', TRUNC(Sing[0]/100.0):4) ;
  IF itt_no <> 1 THEN
  WRITELN (OUTPUT,
    'Number of Down Weighted Observations : ', n_down:4, ' > ',
                                                        vstmax:3:1) ;
  WRITELN (OUTPUT,
    'Standard Error Unit Weight           : ', s0:4:1) ;
  IF Sing[0]/100> max_lost_digits THEN
    BEGIN
    FOR I:= 1 TO NEle DO IF Sing[I]/100> max_lost_digits THEN
      BEGIN
      WRITELN (OUTPUT) ;
      WRITELN
        (OUTPUT, '*WARNING: Singularity! Lost Digits ',
                                              TRUNC(Sing[I]/100.0):2) ;
      WRITE   (OUTPUT, '           ', Element[Ele[I]].Element_Block,
                             Element[Ele[I]].Element_Kind, ',  Name: ') ;
      GBA_Write_String (OUTPUT, Element[Ele[I]].Element_Name );
      WRITELN (OUTPUT) ;
      END ;
    END ;

END { GBA_Write_Pass_Information } ;
END.
```

```
[INHERIT ('BUND.PEN')] MODULE BUND09 ;

[GLOBAL] PROCEDURE GBA_Write_String
          ( VAR Out_File: TEXT ;
                String: PACKED ARRAY [f..l: INTEGER] OF CHAR ) ;

{  }

VAR
  i, j: INTEGER ;

BEGIN

  j:= l; WHILE (j > f) AND (String[j]= ' ') DO j:= j - 1 ;

  FOR i:= f TO j DO WRITE (Out_File, String[i]) ;

END { GBA_Write_String } ;
END.
```

```
[INHERIT ('BUND.PEN')] MODULE OBSL01 ;

[GLOBAL] PROCEDURE Photo_Observation_Equation
          ( Observation     : Real_Array_Embedded_Observations  ;
            Element         : Real_Array_Embedded_Elements  ;
            Mask            : Boolean_Array_Embedded_Elements  ;
        VAR Coefficient     : Real_Array_Coefficients ;
        VAR Error           : BOOLEAN ) ;

{ Observation[1] = X Photo Observation }
{ Observation[2] = Y Photo Observation }

{ Element[1] = X Ground Coordinate }
{ Element[2] = Y Ground Coordinate }
{ Element[3] = Z Ground Coordinate }
{ Element[4] = X Projection Center }
{ Element[5] = Y Projection Center }
{ Element[6] = Z Projection Center }
{ Element[7] = Omega Photo }
{ Element[8] = Fi Photo }
{ Element[9] = Kappa Photo }
{ Element[10] = X Shift Photo Principal Point }
{ Element[11] = Y Shift Photo Principal Point }
{ Element[12] = Camera_Constant }
{ Element[13] = X Shift Camera Principal Point }
{ Element[14] = Y Shift Camera Principal Point }
{ Element[15] = Distortion a3 }
{ Element[16] = Distortion a5 }
{ Element[17] = Distortion a7 }

CONST
  Konstant = 18 ; {number elements + 1}

VAR
  XY_Photo:           Real_Array_2 ;
  Rotation_Matrix:    Real_Array_3_3 ;
  I, J:               INTEGER ;
  Delta:              DOUBLE ;

LABEL
  99 ;


PROCEDURE Calculate_Rotation_Matrix
  (     Omega, Fi, Kappa: Double ;
    VAR Rotation_Matrix:  Real_Array_3_3 ) ;

  VAR
    so, sf, sk, co, cf, ck: Double ;

  BEGIN
```

```
    so:= sin(Omega) ; sf:= sin(Fi) ; sk:= sin(Kappa) ;
    co:= cos(Omega) ; cf:= cos(Fi) ; ck:= cos(Kappa) ;

    Rotation_Matrix[1,1]:=  cf * ck ;
    Rotation_Matrix[1,2]:= -cf * sk ;
    Rotation_Matrix[1,3]:=  sf ;
    Rotation_Matrix[2,1]:=  so * sf * ck + co * sk ;
    Rotation_Matrix[2,2]:= -so * sf * sk + co * ck ;
    Rotation_Matrix[2,3]:= -so * cf ;
    Rotation_Matrix[3,1]:= -co * sf * ck + so * sk ;
    Rotation_Matrix[3,2]:=  co * sf * sk + so * ck ;
    Rotation_Matrix[3,3]:=  co * cf ;

  END { Calculate_Rotation_Matrix } ;


PROCEDURE Photo_Fundamental_Equation
  (     Element:           Real_Array_Embedded_Elements ;
        Rotation_Matrix:   Real_Array_3_3 ;
    VAR Photo_Coordinate: Real_Array_2 ) ;
  VAR
    I, J                  : INTEGER ;
    r2, r4, r6, e2, e4    : DOUBLE ;
    Delta, Radius, Nvner : DOUBLE ;

  BEGIN
    Nvner:= 0.00 ;
    FOR J:= 1 TO 3 DO
      Nvner:= Nvner + Rotation_Matrix[J,3]*(Element[J] - Element[3 + J]) ;

    IF Nvner <> 0.00 THEN
      BEGIN
      FOR I:= 1 TO 2 DO
        BEGIN
        Photo_Coordinate[I]:= 0.00 ;
        FOR J:= 1 TO 3 DO Photo_Coordinate[I]:=
          Photo_Coordinate[I] - Rotation_Matrix[J,I]
          * (Element[J] - Element[3 + J]) * Element[12] / Nvner ;
        END ;

      FOR I:= 1 TO 2 DO
        Photo_Coordinate[I]:= Photo_Coordinate[I] + Element[9 + I] ;

      r2:= ( Photo_Coordinate[1]*Photo_Coordinate[1]
           + Photo_Coordinate[2]*Photo_Coordinate[2] ) * 1000000.0 ;
      r4:= r2*r2 ;
      r6:= r2*r4 ;
      e2:= Element[17] ;
      e2:= e2 * e2; e4:= e2 * e2;
      Delta:= Element[15] * ( r2 - e2 ) + Element[16] * ( r4 - e4 ) ;
      {Delta:= (Element[15] * r2 + Element[16] * r4 + Element[17] * r6) ;}
```

```
    {Delta:= 0.00 ;}

    FOR I:= 1 TO 2 DO
      BEGIN
      Photo_Coordinate[I]:=
        Photo_Coordinate[I] * (1 + Delta) + Element[12+I] ;
      END ;
    END
  ELSE Error:= TRUE ;

END { Photo_Fundamental_Equation } ;


BEGIN { Photo_Observation_Equation }

  FOR I:= 1 TO 2 DO FOR J:=1 TO Konstant DO
    BEGIN
    Coefficient[I,J]:= 0.00 ;
    END ;

  Calculate_Rotation_Matrix
    (Element[7], Element[8], Element[9], Rotation_Matrix) ;

  { calculate right side of coefficient matrix }
  Photo_Fundamental_Equation
    ( Element, Rotation_Matrix, XY_Photo ) ;
  IF Error THEN GOTO 99 ;
  FOR I:= 1 TO 2 DO
    BEGIN
    Coefficient[I,Konstant]:= XY_Photo[I] - Observation[I] ;
    END ;


  { calculate coefficients by numerical differentiation }

  { X, Y, Z, Z0, Y0, Z0 }
  Delta:=  0.001 ;
  FOR J:= 1 TO 6 DO IF Mask[J] THEN

    BEGIN
    Element[J]:= Element[J] + Delta ;
    Photo_Fundamental_Equation
      ( Element, Rotation_Matrix, XY_Photo ) ;
    IF Error THEN GOTO 99 ;
    FOR I:= 1 TO 2 DO
      BEGIN
      Coefficient[I,J]:=
      (XY_Photo[I]-Observation[I]-Coefficient[I,Konstant])/Delta ;
      END ;
    Element[J]:= Element[J] - Delta ;
    END ;
```

```
{ photo principal point: dxp, dyp, and camera elements: C, dxc, dyc }
Delta:=  0.000001 ;
FOR J:= 10 TO 14 DO IF Mask[J] THEN
  BEGIN
  Element[J]:= Element[J] + Delta ;
  Photo_Fundamental_Equation
    ( Element, Rotation_Matrix, XY_Photo ) ;
  IF Error THEN GOTO 99 ;
  FOR I:= 1 TO 2 DO
    BEGIN
    Coefficient[I,J]:=
    (XY_Photo[I]-Observation[I]-Coefficient[I,Konstant])/Delta ;
    END ;
  Element[J]:= Element[J] - Delta ;
  END ;

{ distortion: a3, a5, a7 }
Delta:=  0.001 ;
FOR J:= 15 TO 17 DO
  BEGIN
  Delta:= Delta * 0.001 ;
  IF Mask[J] THEN
    BEGIN
    Element[J]:= Element[J] + Delta ;
    Photo_Fundamental_Equation
      ( Element, Rotation_Matrix, XY_Photo ) ;
    IF Error THEN GOTO 99 ;
    FOR I:= 1 TO 2 DO
      BEGIN
      Coefficient[I,J]:=
      (XY_Photo[I]-Observation[I]-Coefficient[I,Konstant])/Delta ;
      END ;
    Element[J]:= Element[J] - Delta ;
    END ;
  END ;

{ omega, fi, kappa }
Delta:=  0.000015 ;
FOR J:= 7 TO 9 DO IF Mask[J] THEN
  BEGIN
  Element[J]:= Element[J] + Delta ;
  Calculate_Rotation_Matrix
    (Element[7], Element[8], Element[9], Rotation_Matrix) ;
  Photo_Fundamental_Equation
    ( Element, Rotation_Matrix, XY_Photo ) ;
  IF Error THEN GOTO 99 ;
  FOR I:= 1 TO 2 DO
    BEGIN
    Coefficient[I,J]:=
    (XY_Photo[I]-Observation[I]-Coefficient[I,Konstant])/Delta ;
```

```
      END ;
    Element[J]:= Element[J] - Delta ;
    END ;

99: END { Photo_Observation_Equation } ;
END.
```

```
[INHERIT ('BUND.PEN')] MODULE OBSL02 ;

[GLOBAL] PROCEDURE Element_Observation_Equation
          ( Observation     : Real_Array_Embedded_Observations  ;
            Element         : Real_Array_Embedded_Elements  ;
            Mask            : Boolean_Array_Embedded_Elements  ;
        VAR Coefficient     : Real_Array_Coefficients ;
        VAR Error           : BOOLEAN ) ;

{ Observation[1] = Coordinate Observation }
{ Element[1] =     Ground Coordinate      }

CONST
  Konstant = 2 ; {number elements + 1}

BEGIN

  { calculate right side of coefficient matrix }
  Coefficient [1,Konstant]:= Element[1] - Observation[1] ;

  { calculate coefficients by differentiation }
  Coefficient [1,1]:= 1 ;

END { Element_Observation_Equation } ;
END.
```

```
[INHERIT ('BUND.PEN')] MODULE OBSL03 ;

[GLOBAL] PROCEDURE Length_Observation_Equation
          ( Observation    : Real_Array_Embedded_Observations  ;
            Element        : Real_Array_Embedded_Elements  ;
            Mask           : Boolean_Array_Embedded_Elements  ;
        VAR Coefficient    : Real_Array_Coefficients ;
        VAR Error          : BOOLEAN ) ;

{ Observation[1] = Horizontal Distance }

{ Element[1] = X1 Ground Coordinate }
{ Element[2] = Y1 Ground Coordinate }
{ Element[3] = X2 Ground Coordinate }
{ Element[4] = Y2 Ground Coordinate }

CONST
  Konstant = 5 ; {number elements + 1}

VAR
  I, J   : INTEGER ;
  l      : DOUBLE ;

BEGIN

  FOR I:= 1 TO 1 DO FOR J:=1 TO Konstant DO
    BEGIN
    Coefficient[I,J]:= 0.00 ;
    END ;

  l:= SQRT ( (Element[3]-Element[1])*(Element[3]-Element[1])
           + (Element[4]-Element[2])*(Element[4]-Element[2]) ) ;

  IF l > 0.00 THEN
    BEGIN
    { calculate right hand side of error equations }
    Coefficient[1,Konstant]:= l - Observation[I] ;

    { calculate coefficients of error equations }
    IF Mask[1] THEN Coefficient [1,1]:= - (Element[3]-Element[1]) / l ;
    IF Mask[2] THEN Coefficient [1,2]:= - (Element[4]-Element[2]) / l ;
    IF Mask[3] THEN Coefficient [1,3]:= - Coefficient [1,1] ;
    IF Mask[4] THEN Coefficient [1,4]:= - Coefficient [1,2] ;
    END
  ELSE Error:= TRUE ;

END { Length_Observation_Equation } ;
END.
```

```
[INHERIT ('BUND.PEN')] MODULE OBSL04 ;

[GLOBAL] PROCEDURE TheoHZ_Observation_Equation
          ( Observation     : Real_Array_Embedded_Observations  ;
            Element         : Real_Array_Embedded_Elements  ;
            Mask            : Boolean_Array_Embedded_Elements  ;
        VAR Coefficient     : Real_Array_Coefficients ;
        VAR Error           : BOOLEAN ) ;

{ Observation[1] = Horizontal Direction from Circle Zero in radian }

{ Element[1] = X Station Ground Coordinate }
{ Element[2] = Y Station Ground Coordinate }
{ Element[3] = X Aim Ground Coordinate }
{ Element[4] = Y Aim Ground Coordinate }
{ Element[5] = Circle Zero Angle }

CONST
  Konstant = 6 ; {number elements + 1}

VAR
  I, J   : INTEGER ;
  LL     : DOUBLE ;

  FUNCTION  alfa(x,y: double): double;
    VAR a, offset: DOUBLE; direct, octet: INTEGER;

    BEGIN
    IF      x >= 0 THEN octet:= 4 else octet:= 0;
    IF      y >= 0 THEN octet:= octet + 2;
    IF abs(x) <= abs(y) THEN BEGIN a:= x; x:= y; y:= a; direct:= -1 END
    ELSE direct:= 1;
    IF direct >  0 then octet:= octet + 1;
    CASE octet of
    7:   offset:= 0;
    6,2: offset:=   Pi/2;
    3,1: offset:=   Pi;
    0,4: offset:= 3*Pi/2;
    5:   offset:= 2*Pi;
    end;
    if x=0 THEN alfa:= 0 ELSE alfa:= offset + direct*arctan(y/x);
    end; { function alfa }


BEGIN { TheoHZ_Observation_Equation }

  FOR I:= 1 TO 1 DO FOR J:=1 TO Konstant DO
    BEGIN
    Coefficient[I,J]:= 0.00 ;
    END ;
```

```
  ll:=  (Element[3]-Element[1])*(Element[3]-Element[1])
      + (Element[4]-Element[2])*(Element[4]-Element[2]) ;

  IF ll > 0.00 THEN
    BEGIN
    { calculate right side of coefficient matrix }
    Coefficient[1,Konstant]:=
      alfa ((Element[3]-Element[1]), (Element[4]-Element[2]))
    - Element[5] - Observation[I] ;

    { calculate coefficients by differentiation }
    IF Mask[1] THEN Coefficient [1,1]:= + (Element[4]-Element[2]) / ll ;
    IF Mask[2] THEN Coefficient [1,2]:= - (Element[3]-Element[1]) / ll ;
    IF Mask[3] THEN Coefficient [1,3]:= - Coefficient [1,1] ;
    IF Mask[4] THEN Coefficient [1,4]:= - Coefficient [1,2] ;
    IF Mask[5] THEN Coefficient [1,5]:= - 1 ;
    END
  ELSE Error:= TRUE ;

END { TheoHZ_Observation_Equation } ;
END.
```

```
[INHERIT ('BUND.PEN')] MODULE READ01 ;

[GLOBAL] PROCEDURE Read_String
          ( VAR In_File: TEXT ;
            VAR String: PACKED ARRAY [f..l: INTEGER] OF CHAR ) ;

{  }

VAR
  i       : INTEGER ;
  ch      : CHAR ;
  found   : BOOLEAN ;

BEGIN
  FOR i:= f TO l DO String[i]:= ' ' ;
  found:= FALSE ;
  i:= 0 ;

  WHILE NOT EOF (In_File) AND NOT EOLN (In_File)
  AND NOT (found AND (ch= ' ')) AND (i< l) DO
    BEGIN
    READ (In_File, ch) ;
    IF NOT found AND (ch <> ' ')
      THEN found:= TRUE ;
    IF found AND (ch <> ' ') THEN
      BEGIN
      i:= i + 1 ;
      String[i]:= ch ;
      END ;
    END ;

END { Read_String } ;
END.
```

```
[INHERIT ('BUND.PEN')] MODULE READELE01 ;

[GLOBAL] PROCEDURE Read_Camera_Elements
          ( VAR Error: BOOLEAN ) ;

VAR
  Name        : Name_String ;
  Block       : Element_Block_Type ;
  Status      : Element_Status_Type ;

  Value       : DOUBLE ;
  Warning     : BOOLEAN ;
  I           : INTEGER ;
  J           : Element_Kind_Type ;

  In_File     : TEXT ;

BEGIN

  OPEN (FILE_VARIABLE:= In_File,
        FILE_NAME     := 'TEST.CAM',
        HISTORY       := OLD,
        ERROR         := CONTINUE ) ;
  RESET (In_File, ERROR:= CONTINUE ) ;

  Block:= CAMERA_BLOCK ;
  WHILE NOT EOF (In_File) AND NOT Error DO
    BEGIN
    Read_String (In_File, Name); READLN (In_File) ;
    FOR j:= CAMERA_C to CAMERA_A0 DO
      BEGIN
      READLN (In_File, Value, i) ;
      IF i= 0 THEN Status:= Constant ELSE Status:= UNKNOWN ;
      GBA_Elements ( Name, J, Block, Status, Value, Warning, Error ) ;
      END ;
    END ;

  CLOSE ( In_File ) ;

END { Read_Camera_Elements } ;
END.
```

```
[INHERIT ('BUND.PEN')] MODULE READELE02 ;

[GLOBAL] PROCEDURE Read_Photo_Elements
          ( VAR Error: BOOLEAN ) ;

VAR
  Name        : Name_String ;
  Block       : Element_Block_Type ;
  Status      : Element_Status_Type ;

  Warning     : BOOLEAN ;
  I           : INTEGER ;
  J           : Element_Kind_Type ;
  Value       : DOUBLE ;

  In_File     : TEXT ;

BEGIN

  OPEN (FILE_VARIABLE:= In_File,
        FILE_NAME     := 'TEST.PHOTO',
        HISTORY       := OLD,
        ERROR         := CONTINUE ) ;
  RESET (In_File, ERROR:= CONTINUE ) ;

  Block:= PHOTO_BLOCK ;
  WHILE NOT EOF (In_File) AND NOT Error DO
    BEGIN
    Read_String (In_File, Name); READLN (In_File) ;
    FOR j:= PHOTO_X0 to PHOTO_DY DO
      BEGIN
      READLN (In_File, Value, i) ;
      IF i= 0 THEN Status:= Constant ELSE Status:= UNKNOWN ;
      IF (J= PHOTO_X0) OR (J= PHOTO_Y0) OR (J= PHOTO_Z0)
        THEN Value:= Value {*0.3048} ;
      IF (J= PHOTO_OMEGA) OR (J= PHOTO_FI) OR (J= PHOTO_KAPPA)
        THEN Value:= Value *Pi/200.00 ;
      GBA_Elements ( Name, J, Block, Status, Value, Warning, Error ) ;
      END
    END ;

  CLOSE ( In_File ) ;

END { Read_Photo_Elements } ;
END.
```

```
[INHERIT ('BUND.PEN')] MODULE READELE03 ;

[GLOBAL] PROCEDURE Read_Point_Elements
          ( VAR Error: BOOLEAN ) ;

VAR
  Name        : Name_String ;
  Block       : Element_Block_Type ;
  Status      : Element_Status_Type ;

  Warning     : BOOLEAN ;
  I, J        : INTEGER ;
  X, Y, Z     : DOUBLE ;
  Value       : DOUBLE ;

  In_File     : TEXT ;

BEGIN

  OPEN (FILE_VARIABLE:= In_File,
        FILE_NAME     := 'TEST.ELE',
        HISTORY       := OLD,
        ERROR         := CONTINUE ) ;
  RESET (In_File, ERROR:= CONTINUE ) ;

  Block:= POINT_BLOCK ;
  WHILE NOT EOF (In_File) AND NOT Error DO
    BEGIN
    Read_String (In_File, Name); READLN (In_File, X, Y, Z, I) ;
    IF i= 0 THEN Status:= Constant ELSE Status:= UNKNOWN ;
    GBA_Elements
      ( Name, GROUND_X, Block, Status, X {*0.3048}, Warning, Error ) ;
    GBA_Elements
      ( Name, GROUND_Y, Block, Status, Y {*0.3048}, Warning, Error ) ;
    GBA_Elements
      ( Name, GROUND_Z, Block, Status, Z {*0.3048}, Warning, Error ) ;
    END ;

  CLOSE ( In_File ) ;

END { Read_Point_Elements } ;
END.
```

```
[INHERIT ('BUND.PEN')] MODULE READOBS01 ;

[GLOBAL] PROCEDURE Read_Photo_Observations
          ( VAR Error: BOOLEAN ) ;

VAR
  Names           : ARRAY [1..3] OF Name_String ;
  Kinds           : ARRAY [1..3] OF Element_Kind_Type ;
  Blocks          : ARRAY [1..3] OF Element_Block_Type ;

  Obs_Kind        : ARRAY [1..2] OF Observation_Kind_Type ;
  Obs_Value       : ARRAY [1..2] OF DOUBLE ;
  Apriori         : ARRAY [1..2] OF DOUBLE ;

  Name            : Name_String ;
  Camera_Name     : Name_String ;
  Photo_Name      : Name_String ;

  Warning         : BOOLEAN ;
  X, Y, AX, AY    : DOUBLE ;
  In_File         : TEXT ;

BEGIN
  Camera_Name:= '                    ' ;
  Photo_Name:=  '                    ' ;

  OPEN (FILE_VARIABLE:= In_File,
        FILE_NAME    := 'TEST.OBS',
        HISTORY      := OLD,
        ERROR        := CONTINUE ) ;
  RESET (In_File, ERROR:= CONTINUE ) ;

  IF NOT EOF (In_File) THEN
    BEGIN
    Read_String (In_File, Camera_Name); READLN (In_File) ;
    END ;

  WHILE NOT EOF (In_File) AND NOT Error DO
    BEGIN
    Read_String (In_File, Photo_Name); READLN (In_File) ;
      REPEAT
      Name:= '                    ' ; Read_String (In_File, Name);
      IF Name <> '                    ' THEN
        BEGIN
        READLN (In_File, X, Y, AX, AY) ;
        Names[1]:= Name ;
        Names[2]:= Photo_Name ;
        Names[3]:= Camera_Name ;
        Kinds[1]:= WHOLE_GROUP ;
        Kinds[2]:= WHOLE_GROUP ;
        Kinds[3]:= WHOLE_GROUP ;
```

```
      Blocks[1]:= POINT_BLOCK ;
      Blocks[2]:= PHOTO_BLOCK ;
      Blocks[3]:= CAMERA_BLOCK ;
      Obs_Kind[1]:= PHOTO_X ;
      Obs_Kind[2]:= PHOTO_Y ;
      Obs_Value[1]:=  X/1000000.00 ;
      Obs_Value[2]:=  Y/1000000.00 ;
      Apriori[1]:=   AX/1000000.00 ;
      Apriori[2]:=   AY/1000000.00 ;
      GBA_Observations
        ( Name, PHOTO_GROUP, 2, Obs_Kind, Obs_Value, Apriori,
          3, Names, Kinds, Blocks, 17, Warning, Error ) ;
      END
    ELSE
      BEGIN
      READLN (In_File) ;
      END ;
    UNTIL (Name='                    ') OR EOF (In_File) ;
  END ;

  CLOSE ( In_File ) ;

END { Read_Photo_Observations } ;
END.
```

```
[INHERIT ('BUND.PEN')] MODULE READOBS02 ;

[GLOBAL] PROCEDURE Read_Point_Observations
          ( VAR Error: BOOLEAN ) ;

VAR
  Name            : Name_String ;

  Names           : ARRAY [1..1] OF Name_String ;
  Kinds           : ARRAY [1..1] OF Element_Kind_Type ;
  Blocks          : ARRAY [1..1] OF Element_Block_Type ;

  Obs_Kind        : ARRAY [1..1] OF Observation_Kind_Type ;
  Obs_Value       : ARRAY [1..1] OF DOUBLE ;
  Apriori         : ARRAY [1..1] OF DOUBLE ;

  Warning         : BOOLEAN ;
  X, Y, Z         : DOUBLE ;
  AX, AY, AZ      : DOUBLE ;

  In_File         : TEXT ;

BEGIN
  OPEN (FILE_VARIABLE:= In_File,
        FILE_NAME    := 'TEST.PAS',
        HISTORY      := OLD,
        ERROR        := CONTINUE ) ;
  RESET (In_File, ERROR:= CONTINUE ) ;

  WHILE NOT EOF (In_File) AND NOT Error DO
    BEGIN
    Read_String (In_File, Name); READLN (In_File, X, Y, Z, AX, AY, AZ) ;
    Names[1]    := Name ;
    Blocks[1]   := POINT_BLOCK ;

    Kinds[1]    :=  GROUND_X ;
    Obs_Kind[1] :=  POINT_X ;
    Obs_Value[1]:=  X {*0.3048} ;
    Apriori[1]  := AX {*0.3048} ;
    GBA_Observations
      ( Name, ELEMENT_GROUP, 1, Obs_Kind, Obs_Value, Apriori,
        1, Names, Kinds, Blocks, 1, Warning, Error ) ;

    Kinds[1]:=  GROUND_Y ;
    Obs_Kind[1] :=  POINT_Y ;
    Obs_Value[1]:=  Y {*0.3048} ;
    Apriori[1]  := AY {*0.3048} ;
    GBA_Observations
      ( Name, ELEMENT_GROUP, 1, Obs_Kind, Obs_Value, Apriori,
        1, Names, Kinds, Blocks, 1, Warning, Error ) ;
```

```
    Kinds[1]:=  GROUND_Z ;
    Obs_Kind[1] :=  POINT_Z ;
    Obs_Value[1]:=  Z {*0.3048} ;
    Apriori[1]  := AZ {*0.3048} ;
    GBA_Observations
      ( Name, ELEMENT_GROUP, 1, Obs_Kind, Obs_Value, Apriori,
        1, Names, Kinds, Blocks, 1, Warning, Error ) ;
    END ;

  CLOSE ( In_File ) ;

END { Read_Point_Observations } ;
END.
```

```
[INHERIT ('BUND.PEN')] MODULE READOBS03 ;

[GLOBAL] PROCEDURE Read_Length_Observations
          ( VAR Error: BOOLEAN ) ;

VAR
  Name1, Name2    : Name_String ;

  Names           : ARRAY [1..4] OF Name_String ;
  Kinds           : ARRAY [1..4] OF Element_Kind_Type ;
  Blocks          : ARRAY [1..4] OF Element_Block_Type ;

  Obs_Kind        : ARRAY [1..1] OF Observation_Kind_Type ;
  Obs_Value       : ARRAY [1..1] OF DOUBLE ;
  Apriori         : ARRAY [1..1] OF DOUBLE ;

  Warning         : BOOLEAN ;
  l, al           : DOUBLE ;

  In_File         : TEXT ;

BEGIN
  OPEN (FILE_VARIABLE:= In_File,
        FILE_NAME    := 'TEST.HZL',
        HISTORY      := OLD,
        ERROR        := CONTINUE ) ;
  RESET (In_File, ERROR:= CONTINUE ) ;

  WHILE NOT EOF (In_File) AND NOT Error DO
    BEGIN
    Read_String (In_File, Name1) ;
    Read_String (In_File, Name2) ;
    READLN (In_File, l, al) ;
    Names[1]     := Name1 ;
    Names[2]     := Name1 ;
    Names[3]     := Name2 ;
    Names[4]     := Name2 ;
    Kinds[1]   :=  GROUND_X ;
    Kinds[2]   :=  GROUND_Y ;
    Kinds[3]   :=  GROUND_X ;
    Kinds[4]   :=  GROUND_Y ;
    Blocks[1]    := SURVEY_BLOCK ;
    Blocks[2]    := SURVEY_BLOCK ;
    Blocks[3]    := SURVEY_BLOCK ;
    Blocks[4]    := SURVEY_BLOCK ;

    Obs_Kind[1] := LENGTH ;
    Obs_Value[1]:=  l ;
    Apriori[1]  := al ;
    GBA_Observations
      ( Name2, LENGTH_GROUP, 1, Obs_Kind, Obs_Value, Apriori,
```

```
        4, Names, Kinds, Blocks, 4, Warning, Error ) ;

    END ;

  CLOSE ( In_File ) ;

END { Read_Length_Observations } ;
END.
```

```
[INHERIT ('BUND.PEN')] MODULE READOBS04 ;

[GLOBAL] PROCEDURE Read_TheoHZ_Observations
          ( VAR Error: BOOLEAN ) ;

VAR
  Name1, Name2    : Name_String ;
  Kreds           : Name_String ;

  Block           : Element_Block_Type ;
  Status          : Element_Status_Type ;

  Names           : ARRAY [1..5] OF Name_String ;
  Kinds           : ARRAY [1..5] OF Element_Kind_Type ;
  Blocks          : ARRAY [1..5] OF Element_Block_Type ;

  Obs_Kind        : ARRAY [1..1] OF Observation_Kind_Type ;
  Obs_Value       : ARRAY [1..1] OF DOUBLE ;
  Apriori         : ARRAY [1..1] OF DOUBLE ;

  Warning         : BOOLEAN ;
  d, ad, r        : DOUBLE ;


  In_File         : TEXT ;

BEGIN
  OPEN (FILE_VARIABLE:= In_File,
        FILE_NAME    := 'TEST.HZR',
        HISTORY      := OLD,
        ERROR        := CONTINUE ) ;
  RESET (In_File, ERROR:= CONTINUE ) ;

  IF NOT EOF (In_File) THEN
    BEGIN
    Read_String (In_File, Kreds); READLN (In_File, r) ;
    Block:= SURVEY_BLOCK ;
    Status:= UNKNOWN ;
    GBA_Elements ( Kreds, THEO_R, Block, Status, r*pi/200.0, Warning, Error ) ;

    END ;

  WHILE NOT EOF (In_File) AND NOT Error DO
    BEGIN
    Read_String (In_File, Name1);
    Read_String (In_File, Name2);
    READLN (In_File, d, ad) ;
    Names[1]     := Name1 ;
    Names[2]     := Name1 ;
    Names[3]     := Name2 ;
    Names[4]     := Name2 ;
```

```
    Names[5]     := Kreds ;
    Kinds[1]   :=  GROUND_X ;
    Kinds[2]   :=  GROUND_Y ;
    Kinds[3]   :=  GROUND_X ;
    Kinds[4]   :=  GROUND_Y ;
    Kinds[5]   :=  THEO_R ;
    Blocks[1]    := SURVEY_BLOCK ;
    Blocks[2]    := SURVEY_BLOCK ;
    Blocks[3]    := SURVEY_BLOCK ;
    Blocks[4]    := SURVEY_BLOCK ;
    Blocks[5]    := SURVEY_BLOCK ;

    Obs_Kind[1] := THEO_HZ ;
    Obs_Value[1]:=  d*pi/200.0 ;
    Apriori[1]  := ad*pi/200.0 ;
    GBA_Observations
      ( Name2, THEOHZ_GROUP, 1, Obs_Kind, Obs_Value, Apriori,
        5, Names, Kinds, Blocks, 5, Warning, Error ) ;

    END ;

  CLOSE ( In_File ) ;

END { Read_TheoHZ_Observations } ;
END.
```

```
[INHERIT ('BUND.PEN')] PROGRAM BUND00 (INPUT, OUTPUT, OUT_FILE) ;

VAR
  I, J, K                                  : INTEGER ;
  max_itt,
  start_down_weight,
  max_lost_digits                          : INTEGER ;

  max_vst, enp_stop                        : DOUBLE ;

  Warning, Error                           : BOOLEAN ;
  residuals, sd_elements, aposteriori      : BOOLEAN ;

  Name                                     : Name_String ;

  Out_File                                 : TEXT ;


LABEL
  999 ;

BEGIN
  { open file for output of warnings and errors }
  OPEN (FILE_VARIABLE:= OUTERR,
        FILE_NAME    := 'ERROR.OUT',
        HISTORY      := NEW,
        ERROR        := CONTINUE ) ;
  REWRITE (OUTERR, ERROR:= CONTINUE ) ;


  { initialize the GBA variables }
  GBA_Initialize ( Error ); IF Error THEN GOTO 999 ;

  { read elements }
  WRITELN (OUTPUT) ;
  WRITELN (OUTPUT, 'Reading elements') ;
  WRITELN (OUTPUT, '----------------') ;
  Read_Camera_Elements ( Error ) ;     IF Error THEN GOTO 999 ;
  Read_Photo_Elements ( Error ) ;      IF Error THEN GOTO 999 ;
  Read_Point_Elements ( Error ) ;      IF Error THEN GOTO 999 ;

  { read observations }
  WRITELN (OUTPUT) ;
  WRITELN (OUTPUT, 'Reading observations') ;
  WRITELN (OUTPUT, '--------------------') ;
  Read_Photo_Observations ( Error ) ;  IF Error THEN GOTO 999 ;
  Read_Point_Observations ( Error ) ;  IF Error THEN GOTO 999 ;
  Read_Length_Observations ( Error ) ; IF Error THEN GOTO 999 ;
  Read_TheoHZ_Observations ( Error ) ; IF Error THEN GOTO 999 ;

  { write the entered elements }
```

```
{
OPEN (FILE_VARIABLE:= Out_File,
      FILE_NAME    := 'ELE.OUT',
      HISTORY      := NEW,
      ERROR        := CONTINUE ) ;
REWRITE (Out_File, ERROR:= CONTINUE ) ;

FOR I:= 1 TO End_Element_Array DO
  WITH Element[I] DO
    BEGIN
    WRITE (Out_File, Element_Block, Element_Kind, Element_Status,
           ',  Name: ') ;
    GBA_Write_String (Out_File, Element_Name);
    WRITELN (Out_File, Element_Value:10:6 ) ;
    END ;
CLOSE (Out_File) ;
}

{ write the entered observations }
{
OPEN (FILE_VARIABLE:= Out_File,
      FILE_NAME    := 'OBS.OUT',
      HISTORY      := NEW,
      ERROR        := CONTINUE ) ;
REWRITE (Out_File, ERROR:= CONTINUE ) ;

FOR I:= 1 TO End_Obs_Array DO
  WITH Obs[I] DO
    BEGIN
    WRITE (Out_File, Observation_Group, ',  Name: ') ;
    GBA_Write_String (Out_File, Observation_Name) ;
    WRITELN (Out_File) ;
    WRITE (Out_File, Number_Embedded_Elements:3) ;
    FOR J:= 1 TO Number_Embedded_Elements DO
      WRITE (Out_File, Element_Pointer[J]:4) ;
    WRITELN (Out_File) ;
    WRITELN (Out_File, Number_Embedded_Observations:2) ;
    FOR J:= 1 TO Number_Embedded_Observations DO
      WRITELN (Out_File, Observation_Kind[J], '   ',
               Observation[J]:10:6, Apriori_Error[J]:10:6) ;
    WRITELN (Out_File) ;
    END ;
CLOSE (Out_File) ;
}

{ calculate the adjustment pointers }
WRITELN (OUTPUT) ;
WRITELN (OUTPUT, 'Calculating pointers') ;
WRITELN (OUTPUT, '--------------------') ;
GBA_Pointers ( Error ) ;              IF Error THEN GOTO 999 ;
```

```
{ write the calculated pointers }
{
OPEN (FILE_VARIABLE:= Out_File,
      FILE_NAME    := 'PNT.OUT',
      HISTORY      := NEW,
      ERROR        := CONTINUE ) ;
REWRITE (Out_File, ERROR:= CONTINUE ) ;

WRITELN (Out_File, NEle:5, NObs:5, N_Size:10) ;
FOR I:= 1 TO NEle DO
  WRITELN (Out_File, Ele[I]:5, Base[I]:10,
                                Element[Ele[I]].Element_Kind ) ;
FOR I:= 1 TO End_Element_Array DO
  WRITELN (Out_File, Element[I].Element_Kind, '   ', Col[I]:5 ) ;
CLOSE (Out_File) ;
}

{ demo input parameters to procedures Iterate and Statistics }
{
WRITELN (OUTPUT) ;
WRITELN (OUTPUT, 'STOP CRITERIA:') ;
WRITE (OUTPUT, 'Enter enp criteria for stop: ' ) ;
READLN (INPUT, enp_stop) ;
WRITE (OUTPUT, 'Enter maximum number of iterations: ' ) ;
READLN (INPUT, max_itt) ;

WRITELN (OUTPUT) ; WRITELN (OUTPUT) ;
WRITELN (OUTPUT, 'AUTOMATIC SEARCH OF GROSS ERRORS') ;
WRITE (OUTPUT, 'Enter iteration number for start down weight: ' ) ;
READLN (INPUT, start_down_weight) ;
WRITE (OUTPUT, 'Enter maximum allowed standard residual: ' );
READLN (INPUT, max_vst) ;

WRITELN (OUTPUT) ; WRITELN (OUTPUT) ;
WRITELN (OUTPUT, 'SINGULAR COLUMNS') ;
WRITE (OUTPUT, 'Enter maximum allowed lost digits: ' ) ;
READLN (INPUT, max_lost_digits) ;

WRITELN (OUTPUT) ; WRITELN (OUTPUT) ;
WRITELN (OUTPUT, 'DESIRED STATISTICS. ANSWER TRUE OR FALSE:') ;
WRITE (OUTPUT, 'Residuals on observations?: ' ) ;
READLN (INPUT, residuals) ; WRITELN (OUTPUT, residuals) ;
WRITE (OUTPUT, 'Standard deviations on elements?: ' ) ;
READLN (INPUT, sd_elements) ; WRITELN (OUTPUT, sd_elements) ;
WRITE (OUTPUT, 'Standard deviations on observations and residuals?: ' ) ;
READLN (INPUT, aposteriori) ;  WRITELN (OUTPUT, aposteriori) ;
}
enp_stop:= 3.0 ; max_itt:= 25 ;
start_down_weight:= 2 ; max_vst:= 3.0 ;
max_lost_digits:= 12 ;
residuals:= TRUE ; sd_elements:= TRUE ; aposteriori:= TRUE ;
```

```
{ solve the adjustment by iteration }
WRITELN (OUTPUT) ;
WRITELN (OUTPUT, 'Start iterations') ;
WRITELN (OUTPUT, '-----------------') ;
GBA_Solve ( enp_stop, max_itt, start_down_weight,
            max_vst, max_lost_digits, Error );
                                  IF Error THEN GOTO 999 ;

{ calculate residuals and standard deviations }
{ ignore any error, we want as much output as possible }
WRITELN (OUTPUT) ;
WRITELN (OUTPUT, 'Calculating statistics') ;
WRITELN (OUTPUT, '----------------------') ;
GBA_Statistics
  ( residuals, sd_elements, aposteriori, Warning, Error ) ;

{ search and display the down weighted points }
WRITELN (OUTPUT) ;
WRITELN (OUTPUT, '  RESIDUAL        SR       SAP   WF' ) ;
FOR I:= 1 to End_Obs_Array DO WITH Obs[I] DO
FOR J:= 1 TO Number_Embedded_Observations DO
  IF Weight_Factor[J] < 1.00 THEN
    BEGIN
    WRITE( OUTPUT, Residual[J]:10:6,
                   Residual_Error[J]:10:6,
                   Aposteriori_Error[J]:10:6,
                   Weight_Factor[J]:5:2,
                   Observation_Kind[J],
                   ',  Name: ') ;
    GBA_Write_String (OUTPUT, Observation_Name) ; WRITELN (OUTPUT);
    END ;

{ write the results of the adjustment }
OPEN (FILE_VARIABLE:= Out_File,
      FILE_NAME    := 'RESULT.OUT',
      HISTORY      := NEW,
      ERROR        := CONTINUE ) ;
REWRITE ( Out_File, ERROR:= CONTINUE ) ;

WRITELN (Out_File, 'Number of Iterations          :', itt_no:6  ) ;
WRITELN (Out_File) ;
WRITELN (Out_File, 'Standard Deviation Unit Weight:', s0:6:2  ) ;
WRITELN (Out_File) ;
WRITELN (Out_File, 'Observations with Reduced Weight') ;
WRITELN (Out_File, '  RESIDUAL        SR       SAP   WF' ) ;
FOR I:= 1 to End_Obs_Array DO WITH Obs[I] DO
FOR J:= 1 TO Number_Embedded_Observations DO
  IF Weight_Factor[J] < 1.00 THEN
    BEGIN
    WRITE( Out_File, Residual[J]:10:6,
```

```
                    Residual_Error[J]:10:6,
                    Aposteriori_Error[J]:10:6,
                    Weight_Factor[J]:5:2,
                    Observation_Kind[J],
                    ',  Name: ') ;
    GBA_Write_String (Out_File, Observation_Name) ; WRITELN (Out_File);
    END ;

IF Sing[0]/100> max_lost_digits THEN
  BEGIN
  WRITELN (Out_File) ;
  WRITELN (Out_File, 'Singular columns') ;
  FOR I:= 1 TO NEle DO IF Sing[I]/100> max_lost_digits THEN
    BEGIN
    WRITE   (Out_File, Element[Ele[I]].Element_Block,
                       Element[Ele[I]].Element_Kind, ',  Name: ') ;
    GBA_Write_String (Out_File, Element[Ele[I]].Element_Name );
    WRITELN (Out_File) ;
    END ;
  END ;

WRITELN (Out_File) ;
WRITELN (Out_File, 'Best Fit Element Values and Standard Deviation' ) ;

Name:= '                    ' ;
FOR I:= 1 to NEle DO WITH Element[Ele[I]] DO
  BEGIN
  IF Element_Name <> Name THEN
    BEGIN
    Name:= Element_Name ;
    WRITELN (Out_File) ;
    GBA_Write_String (Out_File, Element_Name) ; WRITELN (Out_File);
    END ;
  WRITELN (Out_File, Element_Kind, '   ',
           Element_Value:10:6, Element_Error:10:6 ) ;
  END ;

WRITELN (Out_File) ;
WRITELN (Out_File,
  'Residuals with Standard Deviation, Aposteriori Error, Weight Factor' ) ;
Name:= '                    ' ;
FOR I:= 1 to End_Obs_Array DO WITH Obs[I] DO
FOR J:= 1 TO Number_Embedded_Observations DO
  BEGIN
  IF Observation_Name <> Name THEN
    BEGIN
    Name:= Observation_Name ;
    WRITELN (Out_File) ;
    WRITE (Out_File, Observation_Group, ',  Name: ') ;
    GBA_Write_String (Out_File, Observation_Name) ;
    IF Observation_Group= PHOTO_GROUP THEN
```

```
      BEGIN
      WRITE (Out_File, '  Photo: ') ;
      GBA_Write_String
        (Out_File, Element[Element_Pointer[4]].Element_Name) ;
      WRITELN (Out_File) ;
      END
    ELSE
      WRITELN (Out_File) ;
    END ;
  WRITELN (Out_File, Observation_Kind[J], '   ',
           Residual[J]:10:6, Residual_Error[J]:10:6,
           Aposteriori_Error[J]:10:6, Weight_Factor[J]:5:2 ) ;
  END ;
CLOSE (Out_File) ;

999:
END.
```