Magnetic inversion C++ framework.

Mikhail Tchernychev

January 20, 2006

Contents

1 Objective:

Develop flexible programming framework which would allow easy implementation of magnetic inversion for different types of objects (such as dipole, pipe line, power cable) and types of measured field (total magnetic field measurements or gradient measurements) as well as their possible combinations.

Any part of the framework should be easily replaceable without affecting much of the the already existing code. For instance there should be simple rules for implementing new object types (for instance, new type of magnetic body), measurement spatial forms and mathematical methods used for inversion. It should be easy to adapt any existing mathematical optimization routines to use them in magnetic inversion.

Framework should be carefully designed to serve its creators for the number of years to come.

2 Introduction

This document is indented as design draft for proposed framework. It explains main ideas behind the framework but does not cover all implemented or proposed features.

3 Interaction with optimization software

Flexible usage of the existing optimization software is primary goal of this magnetic inversion framework. Pure mathematical optimization routines used in this project are to be borrowed from third party libraries, not to be developed as part of the project. Framework just provides easy access to these routines and easy adaptation of new routines when they become available.

Here we need to consider some typical forms of mathematical linear and non-linear optimization software. This consideration is important because it shows how magnetic objects and data should represent themselves to be used in general routines.

For linear optimization task typical form is matrix equation:

$$\mathbf{A}\vec{x} = \vec{d} \tag{1}$$

Here $\mathbf{A}$ is $M \times N$ matrix which depends on objects used in the inversion, $\vec{x}$ is vector of unknown linear parameters M (for instance, magnetization components for the dipole), $\vec{d}$ is vector of measurements N. M is number of unknowns and N is number of measurements Based on the equation (1) it is clear that framework should be able to deliver all parts of that equation without goings into details. For instance, if set of profile lines and gridded data are used as field measurements framework should represent all measured points as one plain $\vec{x}$. Framework also should be capable to deliver $\mathbf{A}$ packed by rows or columns (latter is needed when FORTRAN optimization code is used)

Equation (1) imposes some requirements for object representation. Each object should know its place in the $\mathbf{A}$ matrix. For instance consider two magnetic dipoles each with three components of the total magnetization vector (J_x, J_y, J_z). Total of 6 parameters are to be estimated. Dipole #1 should know that it takes columns 0 to 2 in the matrix and dipole #2 should know that it takes columns 3 to 5 in the matrix. Certainly entire object collection is scanned to find "position in the matrix" for individual objects. In example above lets fix J_y for dipole #1; now collection has to be re-scanned and new "matrix positions" for dipole # are 0,1 and for dipole #2 are 2-4 (total number of unknowns is 5 now).

Non-linear optimization packages typically require user to provide routine which computes function to be minimized. In a simplest least squares sense this is just sum of squares of differences between observed and calculated field. The trick is to pass pointer to the whole inversion framework into optimization routine and then into route which computes value to be minimized. This could be done via global variable or by introducing additional `void *` pointer into optimization parameters list. This pointer later can be casted to something meaningful and used to access computational routines provided by the framework.

Some of the non-linear optimization routines can use Jacobian matrix which consist of partial derivatives taken from the field on unknown non-linear parameters. Note that $\mathbf{A}$ in equation above is part of Jacobian (if only linear parameters are estimated). It also required for the object to know its non-linear derivatives positions in the general Jacobian matrix. By design non-linear derivatives occupy first part of Jacobian matrix.

For example consider two dipoles as above. The following values for each dipole have to be estimated as result of non-linear optimization: position (x_0, y_0, z_0) and magnetization (J_x, J_y, J_z), total of 12 parameters. Dipole #1 knows that its non-linear derivatives $(\frac{\partial}{\partial x_0}, \frac{\partial}{\partial y_0}, \frac{\partial}{\partial z_0})$ are taking Jacobian columns 0-2, and its linear derivatives are taking columns 6-8, and dipole #2 knows that its non-linear derivatives occupy columns 3-5, and linear ones occupy columns 9-11.

This effectively separates Jacobian matrix into two parts based on linearity. Note that it is only design issue. However this separation might make sense if only non-linear parameters are involved in optimization, and linear parameters are re-estimated on each optimization iteration.

4 Framework object and data classes

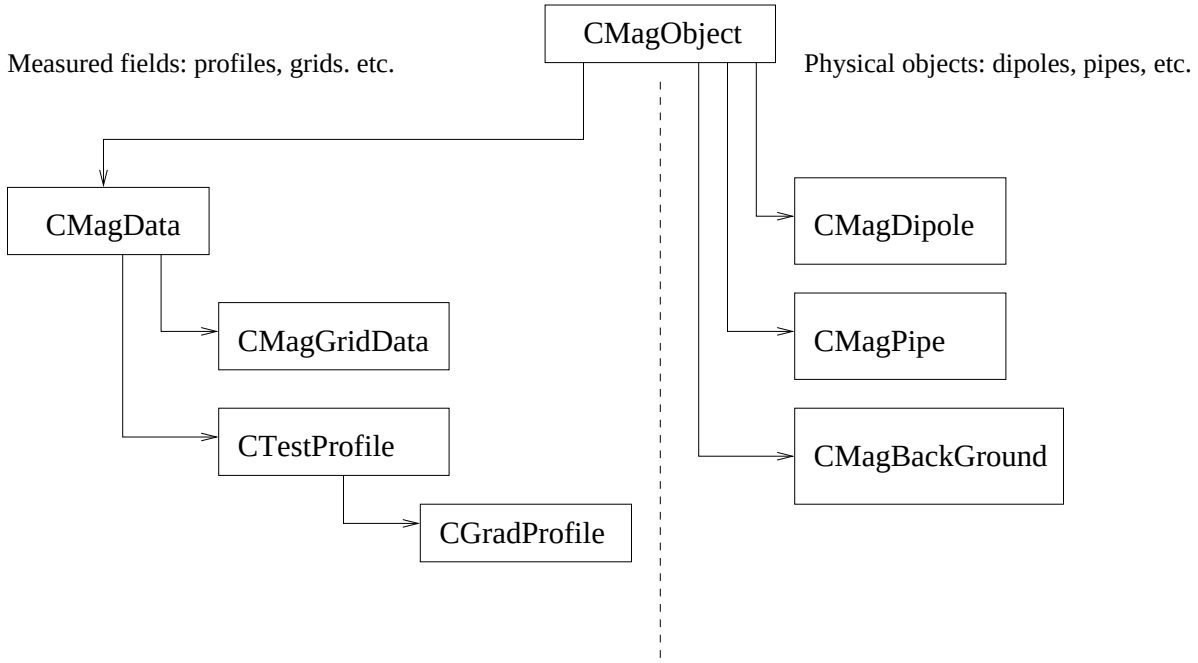

Figure 1: Framework object and data classes

Figure 1 shows main framework object and data classes. Note that all classes can be separated into part dealing with models (dipoles, pipes, etc) and measurements (profile, gridded data). However even data can have some unknown parameters to be estimated. For example multiple passes over the same magnetic objects could have errors in their positions (typical situation in marine environment). In this case not only objects parameters but also position corrections might be of interest. This is the reason why all classes are derived from CMagObject.

4.1 Base class: CMagObject

This class not a simple empty parent for all subclasses. It has some meaningful parameters and methods which are general for all objects.

- Holds separate spaces for linear (such as magnetization) and non-linear (position and geometry) parameters. Also reserves space for standard deviations of parameters(to be estimated along with parameters by linear or non-linear optimization) and lower and upper constrains (if optimization procedure uses them).
- Reports number of linear and non-linear parameters and separately reports number on unknowns in each category.
- Allows fixing some of the parameters thus excluding them from estimation procedure.
- Stores linear and nonlinear parameters positions in Jacobian matrix. These are set as result of scanning all object participating in the inversion.
- Stores field type object can compute. For instance, dipoles can be used to compute total field or gradients. Note that magnetic data sets as objects (for example marine profile where position correction are to be estimated) do not produce any field, but contribute into unknown parameters list.
- Reports if object is "real" (produces magnetic field) or not (such as data profile).
- Reports object computational abilities as a number. 0 means object can do nothing (even compute its magnetic field)), 1 means that object can compute field (and thus used in non-linear optimization with finite-difference approximation for Jacobian) and 2 means that object can compute non-linear derivatives (analytical Jacobian).

Each magnetic object has its position stored in designed place, as three first elements of non-linear parameters vector.

Magnetic object can have two states: induced or non-induced. In first case magnetization is aligned with Earth's magnetic field (for instance, for dipole only one linear parameter would be present in this case). In second case magnetization is considered as a vector (three linear parameters to be estimated in case of dipole source).

Based on computational abilities program can choose how to compute whole Jacobian matrix on non-linear optimization step. For instance if only dipoles (can compute Jacobian matrix analytically) are used in the inversion then framework uses analytical Jacobian. If dipoles and pipes are used (pipes can not compute analytical Jacobian) the whole Jacobian matrix is computed using finite-differencing.

4.2 Real magnetic objects: dipoles, pipes.

New magnetic object class must be derived from `CMagObject` and the following virtual methods are to be overwritten:

- `ComputeField` Computes field (or gradient) from the object at the observation point (passed as parameter). A must.
- `GetLinearDerivatives` Delivers derivatives from the object linear parameters at the observation point. This function can be omitted if object does not participate in the inversion.

- `GetNonLinearDerivatives` Delivers derivatives from object non-linear parameters at the observation point. This is optional. If function is not defined then non-linear inversion (optimization) uses finite difference approximation to compute Jacobian.

- `CanComputeField` Always returns 1 for real magnetic objects (such as dipoles, pipe lines) Returns 0 for data sets as objects.

4.3 Magnetic data: profiles, grids

All data set classes are derived from `CMagData` class. The primary reason is to make it possible to create data collection using only `CMagData` methods and thus hiding details of the particular data type. Each data type allows iteration through its data points one by one and `CDataCollection` class allows to do it seamlessly through different data sets. In allows combining data sets of different types into one inversion act.

Data class has member which indicates data type (primarily "total magnetic field" or gradient). In latter case each measurement point holds also gradient direction. Alternatively gradient direction can be set as common for all data points (for instance, assuming data represent vertical gradient).

To be used as data class should be derived from `CMagData` (in turn derived from `CMagObject`) and has the following virtual methods overwritten:

- `get_total_data_points()` returns number of data points in the data set. Data set has current position which ranges from 0 to `get_total_data_points() -1`.

- `get_data_origin())` return X, Y, Z position which is considered origin for the data set. It all computations are made based on positions relative to this origin point to avoid big X,Y numbers in the computations.

- `reset())` Each data class maintains internal counter of the current data point. Calling `reset()` functions sets this current data point to the beginning of the data set.

- `get_next_position()` returns current data point position and its magnetic field (or gradient) value. It also may return some additional parameters associated with data point position such as direction of the gradient. This function increases current position by one.

- `get_at_position()` Get position, field and its parameters at selected specified position. Does not change current position in the data set.

- `get_position()` returns coordinates (not field parameters) at specified position. Does not change current position.

- `GetBoundingBox()` returns spatial bounding box in X,Y,Z dimensions around all points included in the data set.

5 Framework collection classes

Objects and data can be joined into collections to allow inversion procedure use number of objects and data sets at the same time. For example if user wants to estimate position of the

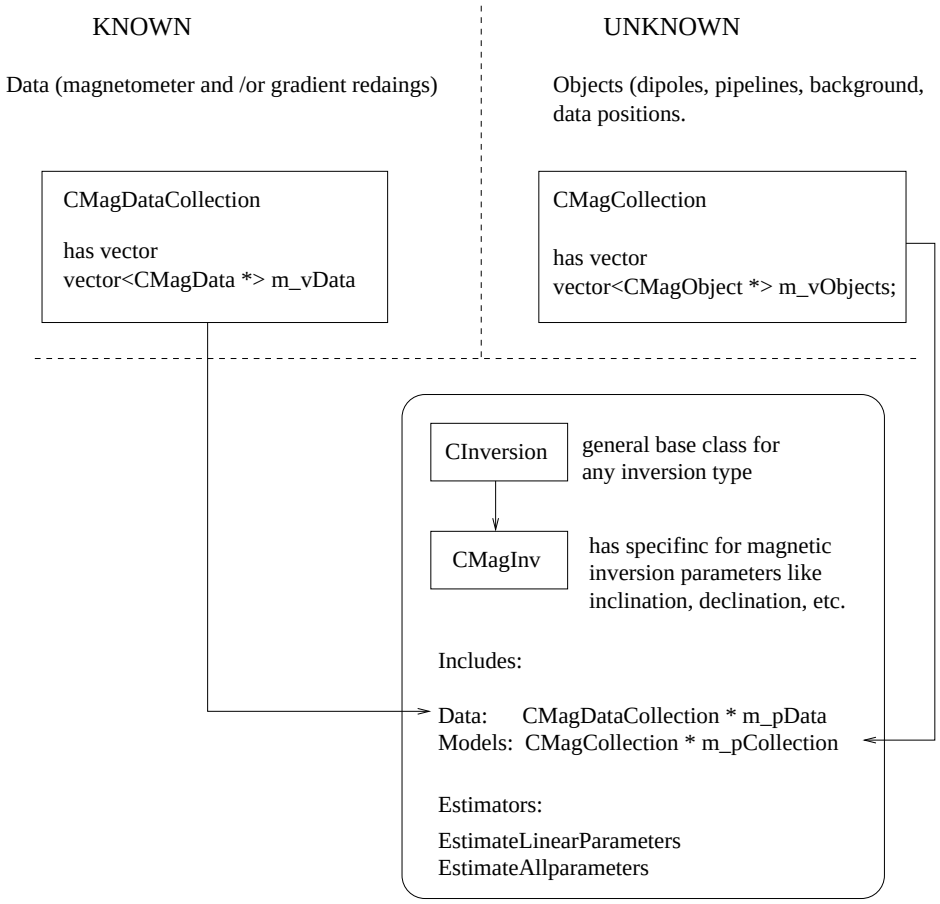

Figure 2: Framework collection and inversion classes

magnetic dipole and closely located pipe line he may create object collection with these two objects. If total magnetic field is used it is also good idea to estimate general background in the area (this reduces total field values to the local anomalies). Object collection would therefore include dipole, pipe and background.

Lets assume there are 5 profiles which cross area of interest. Therefore data collection should include these 5 lines. Lets also assume that we wanted to correct positions of 4 of these 5 profiles (we could not find corrections for all 5 profiles because task is undefined; we need to assume that at least one profile has "true" coordinates.). It means that 4 profiles are to be added to the object collection.

When collection is destroyed it deletes its "native" objects. For instance object collection deletes pipes, dipoles and other physical objects but not data objects included in the collection. When data collection is destroyed it deletes its data objects.

Collection classes realization is simple. Each class has vector of the collection items (data or physical objects). Because pointer to the base class is used (`CMagData *` and `CMagObject *`) collection can handle any items, allows easy addition of the new one types.

Primary goal of the data collection is to let user iterate trough data points as if they all would be stored in the single array. In certain cases (for instance, for linear inversion) it can lead to the data duplication.

Object collection is working the same way. Because it uses pointer to the base class

`CMagObject *` it can handle variety of objects, including not yet developed.

6 Inversion class

Main framework class is `CInversion` which is responsible for the inversion (optimization) itself. This class is pure virtual and is used only as a base for real inversion. The reason for this is to have ability to handle different inversion types (for instance, for the gravity or EM data) using the same mathematical routines as for magnetic inversion without altering them.

The real class which provides inversion for magnetic data is `CMagInversion`, based on `CInversion`. This class has the following features:

- Holds pointer to the collection of the data sets. Note that data sets are not destroyed when `CMagInversion` is deleted. This mission is designed to the data collection.
- Holds pointer to the collection of the magnetic objects. Note that object are not destroyed when inversion class is deleted.
- Has special methods which allow to carry out linear and non-linear inversion.

From the programmer's point of view inversion consist of the following steps:

1. Load data sets creating data set object for each of them. Create data set collection (`CMagDataCollection`). If data set already exists in the form of a class derive new class based on `CMagDataCollection` and adjust its methods.
2. Create magnetic objects (dipoles, pipes) and load its initial non-linear parameters. Most of the non-linear optimization routines need initial guess values to work.
3. Create `CMagInversion` class and include both collections into it.
4. Call `CMagInversion::EstimateLinearParameters())` method to estimate initial values for linear parameters. Note that this function takes as parameter estimation method. This allows using of new mathematical routines without altering the framework.
5. Call `CMagInversion::EstimateAllParameters())` to estimate all object parameters. Note that as in the linear case this function takes estimation method as a parameter. It allows to use future mathematical routines without altering the framework. Dump the solution and its standard variation.

6.1 Including new mathematical routines in the framework.

6.1.1 Linear case

Let's consider new linear inversion routine `LINEAR(A,X,Y ...)` Typically this would invert matrix $\mathbf{A}$ in some way and find solution $\vec{x}$. Note that `LINEAR()` can have number of additional parameters related to the procedure. Framework does not know about these parameters and is not going to use them.

The implementation consist of the following steps:

- Create data structure which holds all parameters of the `LINEAR()` function. Make appropriate default constructor. For example sake let's name this structure `linear_params_struct`
- Write driver function with the following prototype:

```
typedef  int (*LINEAR_ESTIMATOR_FUNC)(int n, int m, double *a,
             double *y, double *x, void * param, double * synt,
             double * fit,  double * max_fit,
             double *st_err, char * message);
```

Let's name this function `linear_estimator()`. This function is called by the frame work. Its parameters have the following meaning:

1. n - number of data points (input).
2. m - number of variables (input).
3. a - matrix $\mathbf{A}$ in the equation (1). It can be packed by rows or columns (input).
4. y - $\vec{d}$ in the equation (1) (input)).
5. x - $\vec{x}$ to be estimated by `LINEAR()`. This is output parameter.
6. param - `void *` pointer to the newly created `linear_params_struct` structure. Use casting to access its data. `param` also can be NULL. In this case re-allocate entire `linear_params_struct` in the `linear_estimator()`. Some of the members in the structure could be arrays (like storage space for `LINEAR()`). Allocate appropriate space in `linear_estimator()` functions using m and n values.
7. synt - computed field after solution is completed (output parameter).
8. fit - RMS difference between computed and measured fields (output).
9. max_fit - maximum difference between observed and calculated fields (output).
10. st_err standard deviations of the estimates (output).
11. message - optional message from computational routines. Many of mathematical routines have termination indicator variable, typically integer. It is recommended to convert this variable into meaningful text and report it in the "message" parameter.

Inside `linear_estimator()` body do the following:

1. Cast `param` value or allocate new `linear_params_struct`.
2. Allocate all arrays needed by `LINEAR()` as part of the structure.
3. Call `LINEAR()` with appropriate parameters (use `linear_params_struct` members)
4. Compute fit and standard errors if it is not done by the `LINEAR()` itself.
5. Fill in output parameters and perform clean up.
6. Return "1" if estimation succeeded or "0" if it failed.

- Now when driver function is ready we can call

```
double fit;
linear_params_struct params;
int packing = 1;
inversion.EstimateLinearParameters(linear_estimator,
                                   &fit, packing, &params);
```

This will make actual estimation. Note that we could adjust any of the `LINEAR()` parameters by changing `params` members. Also note that we can use NULL in place of `params` (our driver function `linear_estimator()` should handle this).

6.1.2 Non-Linear case

Using new non-linear optimization routine in general follow the same steps as linear with some differences. Non-linear optimization task can not be expressed in simple matrix form therefore no need for framework to create $\mathbf{A}$ and $\vec{d}$ matrixes. Instead pointer to the whole framework `CInversion` is passed. It enables field computation with any parameters.

The general prototype for the driver is the following:

```
typedef  double * (*NONLINEAR_ESTIMATOR_FUNC)(void * func_param,
                                              void *maginv_param,
                                              int n, int m,
                                              double *x,
                                              double *x_up,
                                              double *x_low,
                                              double *fit,
                                              double *st_err,
                                              double *max_fit,
                                              char * message);
```

With the parameters:

1. func_param pointer to the parameter structure specific for particular non linear inversion routine. Use casting to access its members (input).
2. maginv_param pointer to the CInversion class. Use casting to access its methods (input).
3. n - number of data points (input).
4. m - number of variables (input).
5. x - initial value of the parameters on input and estimated parameters on output.
6. x_up upper constraints for parameters (input). May be disregarded by computational routine if it does not support constrained optimization.
7. x_low lower constraints for parameters (input). May be disregarded by computational routine if it does not support constrained optimization.

8. fit - RMS difference between computed and measured fields (output).

9. st_err standard deviations of the estimates (output).

10. max_fit - maximum difference between observed and calculated fields (output).

11. message - optional message from computational routines. Many of mathematical routines have termination indicator variable, typically integer. It is recommended to convert this variable into meaningful text and report it in the "message" parameter.

Some of the non-linear optimization routines require user to provide function to compute function value and possibly its derivatives. In this case simple modification of the original mathematical code is needed to pass `CInversion` pointer as `void *` to this function. Alternatively global variables can be used but this limits application.

Here is real example of non-linear optimization call for DNLS1 routine adopted from SLATEC fortran package:

```
DNLS1_STORAGE dstorage;
dstorage.iopt =  2;
char message[1024];
inversion.EstimateAllParameters(Estimator_dnls1, &dstorage, message);
```

7 Conclusions

Flexible inversion C++ framework is proposed. Simple rules for its extension are described. Each part of the framework can be easily developed without affecting other parts, such as data representation, magnetic models and numerical procedures.

Below is incomplete list of possible extensions:

- Profile extension for multi sensor magnetometer array. Simplest way would be just compute each sensor position and store it as separate profile class using already existing code. However this would disintegrate measurements; for instance profile position correction can not be estimated. More flexible (and memory efficient) would be to store only positions of the middle of the array and orientation vectors at every position. Positions of the individual sensors can be computed on demand using these values. For instance for 8 sensor array "simple" storage would require $8 \times 4 = 32$ values for each reading; storing only central position and orientation vector would require $3 + 3 + 8 = 14$ values (3 for array position, 3 for array orientation vector and 8 for magnetometer readings). Most important that all 8 lines would appear as rigid entity.

- Multi-segment pipe. Real pipe lines consist of segments and produce complex magnetic signatures. It can be important to know pipe segment length for interpretation. Multi-segment pipe can appear as object with the following parameters:

    1. number of segments to the left and right from the center (this value framework can not estimate because of its discrete nature)

2. x_0, y_0, z_0 position of the center of the middle segment.
3. α, β direction angles for the pipe lines.
4. $J_{x_i}, J_{y_i}, J_{z_i}$ - magnetization components for each segment.

- Multi-segment pipe line where segments can have different orientation.

8 Dipole derivatives

$$R = \sqrt{z^2 + y^2 + x^2}$$

$$\frac{\mathrm{d2V_{xy}}}{\mathrm{dxdx_0}} = \frac{21xyz^2+21xy^3-84x^3y}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{24xy}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{xz}}}{\mathrm{dxdx_0}} = \frac{21xz^3+21xy^2z-84x^3z}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{24xz}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{YZ}}}{\mathrm{dxdx_0}} = \frac{15yz}{(z^2+y^2+x^2)^{\frac{7}{2}}} - \frac{105x^2yz}{(z^2+y^2+x^2)^{\frac{9}{2}}}$$

$$\frac{\mathrm{d2V_{xx}}}{\mathrm{dxdx_0}} = \frac{63x^2z^2+63x^2y^2-42x^4}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{-9z^2-9y^2+18x^2}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{xy}}}{\mathrm{dydx_0}} = \frac{21x^2z^2-84x^2y^2+21x^4}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{-3z^2+12y^2-9x^2}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{xz}}}{\mathrm{dydx_0}} = \frac{15yz}{(z^2+y^2+x^2)^{\frac{7}{2}}} - \frac{105x^2yz}{(z^2+y^2+x^2)^{\frac{9}{2}}}$$

$$\frac{\mathrm{d2V_{YZ}}}{\mathrm{dydx_0}} = \frac{21xz^3-84xy^2z+21x^3z}{(z^2+y^2+x^2)^{\frac{9}{2}}} - \frac{6xz}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{xx}}}{\mathrm{dydx_0}} = \frac{21xyz^2+21xy^3-84x^3y}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{24xy}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{yy}}}{\mathrm{dydx_0}} = \frac{63xyz^2-42xy^3+63x^3y}{(z^2+y^2+x^2)^{\frac{9}{2}}} - \frac{18xy}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{zz}}}{\mathrm{dydx_0}} = \frac{-84xyz^2+21xy^3+21x^3y}{(z^2+y^2+x^2)^{\frac{9}{2}}} - \frac{6xy}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{yy}}}{\mathrm{dxdx_0}} = \frac{21x^2z^2-84x^2y^2+21x^4}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{-3z^2+12y^2-9x^2}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{zz}}}{\mathrm{dxdx_0}} = \frac{-84x^2z^2+21x^2y^2+21x^4}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{12z^2-3y^2-9x^2}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{xy}}}{\mathrm{dzdx_0}} = \frac{15yz}{(z^2+y^2+x^2)^{\frac{7}{2}}} - \frac{105x^2yz}{(z^2+y^2+x^2)^{\frac{9}{2}}}$$

$$\frac{\mathrm{d2V_{xy}}}{\mathrm{dzdx_0}} = \frac{-84x^2z^2+21x^2y^2+21x^4}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{12z^2-3y^2-9x^2}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{YZ}}}{\mathrm{dzdx_0}} = \frac{-84xyz^2+21xy^3+21x^3y}{(z^2+y^2+x^2)^{\frac{9}{2}}} - \frac{6xy}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{xx}}}{\mathrm{dzdx_0}} = \frac{21xz^3+21xy^2z-84x^3z}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{24xz}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{yy}}}{\mathrm{dzdx_0}} = \frac{21xz^3-84xy^2z+21x^3z}{(z^2+y^2+x^2)^{\frac{9}{2}}} - \frac{6xz}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{zz}}}{\mathrm{dzdx_0}} = \frac{-42xz^3+63xy^2z+63x^3z}{(z^2+y^2+x^2)^{\frac{9}{2}}} - \frac{18xz}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{xy}}}{\mathrm{dxdy_0}} = \frac{21y^2z^2+21y^4-84x^2y^2}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{-3z^2-9y^2+12x^2}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{xz}}}{\mathrm{dxdy_0}} = \frac{21yz^3+21y^3z-84x^2yz}{(z^2+y^2+x^2)^{\frac{9}{2}}} - \frac{6yz}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{YZ}}}{\mathrm{dxdy_0}} = \frac{15xz}{(z^2+y^2+x^2)^{\frac{7}{2}}} - \frac{105xy^2z}{(z^2+y^2+x^2)^{\frac{9}{2}}}$$

$$\frac{\mathrm{d2V_{xx}}}{\mathrm{dxdy_0}} = \frac{63xyz^2+63xy^3-42x^3y}{(z^2+y^2+x^2)^{\frac{9}{2}}} - \frac{18xy}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{xy}}}{\mathrm{dydy_0}} = \frac{21xyz^2-84xy^3+21x^3y}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{24xy}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{xz}}}{\mathrm{dydy_0}} = \frac{15xz}{(z^2+y^2+x^2)^{\frac{7}{2}}} - \frac{105xy^2z}{(z^2+y^2+x^2)^{\frac{9}{2}}}$$

$$\frac{\mathrm{d2V_{YZ}}}{\mathrm{dydy_0}} = \frac{21yz^3-84y^3z+21x^2yz}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{24yz}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{xx}}}{\mathrm{dydy_0}} = \frac{21y^2z^2+21y^4-84x^2y^2}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{-3z^2-9y^2+12x^2}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{yy}}}{\mathrm{dydy_0}} = \frac{63y^2z^2-42y^4+63x^2y^2}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{-9z^2+18y^2-9x^2}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{zz}}}{\mathrm{dydy_0}} = \frac{-84y^2z^2+21y^4+21x^2y^2}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{12z^2-9y^2-3x^2}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{yy}}}{\mathrm{dxdy_0}} = \frac{21xyz^2-84xy^3+21x^3y}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{24xy}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$

$$\frac{\mathrm{d2V_{zz}}}{\mathrm{dxdy_0}} = \frac{-84xyz^2+21xy^3+21x^3y}{(z^2+y^2+x^2)^{\frac{9}{2}}} - \frac{6xy}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$
$$\frac{\mathrm{d2V_{xy}}}{\mathrm{dzdy_0}} = \frac{15xz}{(z^2+y^2+x^2)^{\frac{7}{2}}} - \frac{105xy^2z}{(z^2+y^2+x^2)^{\frac{9}{2}}}$$
$$\frac{\mathrm{d2V_{xy}}}{\mathrm{dzdy_0}} = \frac{-84xyz^2+21xy^3+21x^3y}{(z^2+y^2+x^2)^{\frac{9}{2}}} - \frac{6xy}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$
$$\frac{\mathrm{d2V_{YZ}}}{\mathrm{dzdy_0}} = \frac{-84y^2z^2+21y^4+21x^2y^2}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{12z^2-9y^2-3x^2}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$
$$\frac{\mathrm{d2V_{xx}}}{\mathrm{dzdy_0}} = \frac{21yz^3+21y^3z-84x^2yz}{(z^2+y^2+x^2)^{\frac{9}{2}}} - \frac{6yz}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$
$$\frac{\mathrm{d2V_{yy}}}{\mathrm{dzdy_0}} = \frac{21yz^3-84y^3z+21x^2yz}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{24yz}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$
$$\frac{\mathrm{d2V_{zz}}}{\mathrm{dzdy_0}} = \frac{-42yz^3+63y^3z+63x^2yz}{(z^2+y^2+x^2)^{\frac{9}{2}}} - \frac{18yz}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$
$$\frac{\mathrm{d2V_{xy}}}{\mathrm{dxdz_0}} = \mathrm{COMBINE}\,(\%)$$
$$\frac{\mathrm{d2V_{xy}}}{\mathrm{dxdz_0}} = \frac{21yz^3+21y^3z-84x^2yz}{(z^2+y^2+x^2)^{\frac{9}{2}}} - \frac{6yz}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$
$$\frac{\mathrm{d2V_{xz}}}{\mathrm{dxdz_0}} = \frac{21z^4+21y^2z^2-84x^2z^2}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{-9z^2-3y^2+12x^2}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$
$$\frac{\mathrm{d2V_{YZ}}}{\mathrm{dxdz_0}} = \frac{15xy}{(z^2+y^2+x^2)^{\frac{7}{2}}} - \frac{105xyz^2}{(z^2+y^2+x^2)^{\frac{9}{2}}}$$
$$\frac{\mathrm{d2V_{xx}}}{\mathrm{dxdz_0}} = \frac{63xz^3+63xy^2z-42x^3z}{(z^2+y^2+x^2)^{\frac{9}{2}}} - \frac{18xz}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$
$$\frac{\mathrm{d2V_{xy}}}{\mathrm{dydz_0}} = \frac{21xz^3-84xy^2z+21x^3z}{(z^2+y^2+x^2)^{\frac{9}{2}}} - \frac{6xz}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$
$$\frac{\mathrm{d2V_{xz}}}{\mathrm{dydz_0}} = \frac{15xy}{(z^2+y^2+x^2)^{\frac{7}{2}}} - \frac{105xyz^2}{(z^2+y^2+x^2)^{\frac{9}{2}}}$$
$$\frac{\mathrm{d2V_{YZ}}}{\mathrm{dydz_0}} = \frac{21z^4-84y^2z^2+21x^2z^2}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{-9z^2+12y^2-3x^2}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$
$$\frac{\mathrm{d2V_{xx}}}{\mathrm{dydz_0}} = \frac{21yz^3+21y^3z-84x^2yz}{(z^2+y^2+x^2)^{\frac{9}{2}}} - \frac{6yz}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$
$$\frac{\mathrm{d2V_{yy}}}{\mathrm{dydz_0}} = \frac{63yz^3-42y^3z+63x^2yz}{(z^2+y^2+x^2)^{\frac{9}{2}}} - \frac{18yz}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$
$$\frac{\mathrm{d2V_{zz}}}{\mathrm{dydz_0}} = \frac{-84yz^3+21y^3z+21x^2yz}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{24yz}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$
$$\frac{\mathrm{d2V_{yy}}}{\mathrm{dxdz_0}} = \frac{21xz^3-84xy^2z+21x^3z}{(z^2+y^2+x^2)^{\frac{9}{2}}} - \frac{6xz}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$
$$\frac{\mathrm{d2V_{zz}}}{\mathrm{dxdz_0}} = \frac{-84xz^3+21xy^2z+21x^3z}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{24xz}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$
$$\frac{\mathrm{d2V_{xy}}}{\mathrm{dzdz_0}} = \frac{15xy}{(z^2+y^2+x^2)^{\frac{7}{2}}} - \frac{105xyz^2}{(z^2+y^2+x^2)^{\frac{9}{2}}}$$
$$\frac{\mathrm{d2V_{xy}}}{\mathrm{dzdz_0}} = \frac{-84xz^3+21xy^2z+21x^3z}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{24xz}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$
$$\frac{\mathrm{d2V_{YZ}}}{\mathrm{dzdz_0}} = \frac{-84yz^3+21y^3z+21x^2yz}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{24yz}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$
$$\frac{\mathrm{d2V_{xx}}}{\mathrm{dzdz_0}} = \frac{21z^4+21y^2z^2-84x^2z^2}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{-9z^2-3y^2+12x^2}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$
$$\frac{\mathrm{d2V_{yy}}}{\mathrm{dzdz_0}} = \frac{21z^4-84y^2z^2+21x^2z^2}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{-9z^2+12y^2-3x^2}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$
$$\frac{\mathrm{d2V_{zz}}}{\mathrm{dzdz_0}} = \frac{-42z^4+63y^2z^2+63x^2z^2}{(z^2+y^2+x^2)^{\frac{9}{2}}} + \frac{18z^2-9y^2-9x^2}{(z^2+y^2+x^2)^{\frac{7}{2}}}$$