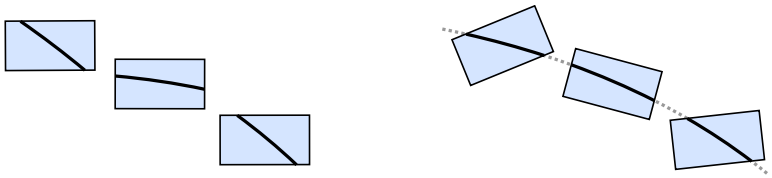
Track-Based Alignment Tutorial

Jim Pivarski

October 19, 2010

1 Introduction

Tracking detectors are designed to record the trajectories of charged particles with as much precision as possible, but if the location of a detector is not precisely known, then neither are the tracks it reports. The situation is even worse when a tracking system is composed of many subdetectors and each is imprecisely positioned relative to the others: smoothly-curving trajectories would appear to be erratic, and parameters of track-fits would be inaccurate. To "align" a tracking system is to correct this situation, not necessarily by physically moving the subdetectors into a specified configuration, but by precisely measuring the as-built positions and orientations of all subcomponents and supplying the information to track-fitting algorithms for better reconstruction.

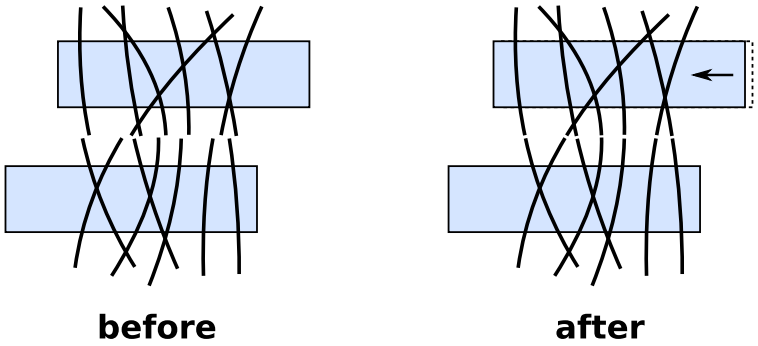


assumed configuration **real configuration**

There are many ways to measure the position of an object, but a particularly natural way to measure the position of a tracking detector is to use the tracks themselves. Tracking detectors are complicated objects with active (particle-sensing) components surrounded by support structures, often sealed to contain an ionizable gas. It can sometimes be difficult to relate measurements of the outside of the detector to the positions of the active components inside. The charged particles, however, are directly observed by the active elements, so if the trajectories of the particles could be known, then they would provide a position measurement of the important part of the detector.

This may sound like a circular argument: measure chamber positions with particle trajectories, and then measure particle trajectories with respect to the known chamber positions. The circularity is resolved by the fact that we know how particles propagate through space, at least in a statistical sense for an ensemble of tracks. For example, if all tracks passing through two subdetectors appear to deviate from extrapolated trajectories by the same offset somewhere between the subdetectors, then we can assume that the subdetectors are misaligned by the value of that offset. The difficult task is to understand the propagation of

particles in detail, so that the average of extrapolations from one subdetector to another is statistically precise and free of bias, to a given level of accuracy.



This note describes the central concepts of track-based alignment with toy examples and illustrations drawn from the alignment of the CMS muon system. It is a pedagogical text, rather than a how-to manual, to help the reader develop expertise in thinking about and solving alignment problems. The toy examples are open-ended to allow the reader to "tinker" with simplified alignment systems. Some familiarity with the Python scripting language is assumed, and the following software packages are required:

- Python `http://python.org/`;
- NumPy `http://numpy.scipy.org/` for linear algebra;
- PyMinuit `http://code.google.com/p/pyminuit/` for non-linear minimization;
- pyROOT `http://root.cern.ch/` for plotting.

Not all packages are required for all examples, and other tools that provide the same functionality could be substituted if desired.

The note covers three major topics: combining alignment measurements, measuring detector positions with tracks, and investigating track biases. The first is covered in section 2, which explains how relative measurements of subdetectors can be combined into a mutually consistent system, and how to propagate uncertainties through complex networks of inter-alignments. Section 3 shows how trends in tracking distributions can be used to measure the position and orientation of a detector. In section 4, common biases in track distributions are discussed with methods of diagnosing data from a tracking system. Finally, section 5 presents a configurable alignment package in CMSSW, so that alignment concepts can be applied in a real setting.

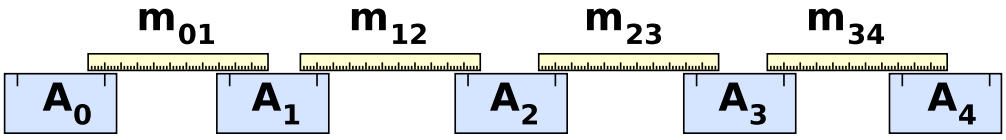
2 Propagating Alignment Measurements

Suppose that the positions of N_a alignable objects are related to one another by N_m measurements. How can these data be combined into a single configuration? Can such a configuration be internally consistent, given measurement uncertainties? How can measurement uncertainties be propagated to uncertainties in the positions of the aligned system? This

section covers the problem of deriving a global alignment configuration from relative measurements between alignable objects, a topic not restricted to track-based alignment. All alignment problems in this section are one-dimensional but with many objects to be aligned in that dimension; section 3 covers multi-dimensional problems, but usually with only one alignable object for simplicity.

2.1 Motivating examples

We'll start with a simple case: $N_a = 5$ alignables A_i with $N_m = 4$ one-dimensional measurements $m_{i,i+1}$ of the displacement between neighbors.



We want to describe the positions of all A_i from the measurements. The first issue is that a global coordinate system has not been defined in which to express the solution, and even if an external coordinate system were to be provided, there would be no way to relate A_i to the origin of that system without an additional measurement. Providing both an external coordinate system and a measurement between it and the first or last alignable reduces the problem to what we had before, with its origin as a new A_i and the measurement as a new $m_{i,i+1}$. It is therefore equivalent to choose one of the A_i as the origin of the coordinate system. We could say that $A_0 = 0$ by definition.

Now we can solve the problem easily:

$$\begin{aligned} A_1 &= m_{01} \\ A_2 &= m_{01} + m_{12} \\ A_3 &= m_{01} + m_{12} + m_{23} \\ A_4 &= m_{01} + m_{12} + m_{23} + m_{34}. \end{aligned} \tag{1}$$

Where did this result come from? We intuitively solved the following system of equations:

$$\begin{aligned} A_0 &= 0 \\ m_{01} &= A_1 - A_0 \\ m_{12} &= A_2 - A_1 \\ m_{23} &= A_3 - A_2 \\ m_{34} &= A_4 - A_3. \end{aligned} \tag{2}$$

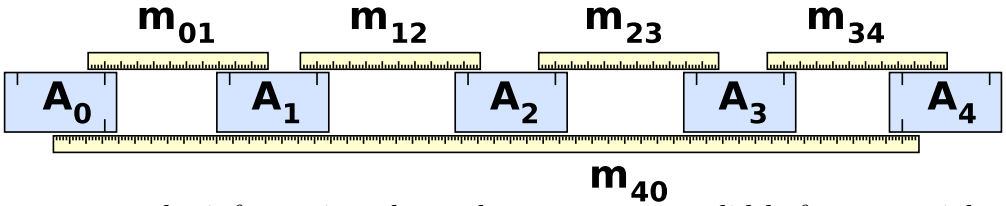
Now suppose that we want to know the uncertainty in A_4, given uncertainties in the measurements $\sigma_{i,i+1}$. Applying the standard error propagation formula to Eq. 1, the uncertainty in A_4 is

$$\sqrt{{\sigma_{01}}^2 + {\sigma_{12}}^2 + {\sigma_{23}}^2 + {\sigma_{34}}^2}. \tag{3}$$

Clearly, the uncertainty in A_4 must be larger than the uncertainty in any other alignable, but this is simply due to our choice of coordinate system. If we had chosen A_3 as the origin, then the uncertainty in A_4 would be only σ_{34}, and if we had chosen A_4 as the origin, then the uncertainty would be zero. Moreover, there are strong correlations between A_3 and A_4,

because if m_{23} fluctuates up by some amount, then A_3 and A_4 both fluctuate up together. Independent measurements do not result in independent alignment positions.

Let's add a little to our example by closing the loop: a new measurement m_{40} directly relates the position of A_4 to that of A_0.



If we propagate the information about the system as we did before, we might find that the new measurement is incompatible with the first four: $m_{40} \neq m_{01} + m_{12} + m_{23} + m_{34}$, possibly due to a statistical error, or possibly due to a mistake in the measurments. How do we solve the system now? $A_4 = m_{01} + m_{12} + m_{23} + m_{34}$ yields a different result than $A_4 = -m_{40}$, so which do we choose?

The new system is overconstrained, so in general we cannot find a solution that satisfies all measurements exactly. However, we can find a set of A_i that is the best fit to $m_{i,i+1}$ by minimizing an objective function like

$$\chi^2 = (A_0)^2 + (m_{01} - A_1 + A_0)^2 + (m_{12} - A_2 + A_1)^2 + (m_{23} - A_3 + A_2)^2 + (m_{34} - A_4 + A_3)^2 + (m_{40} - A_0 + A_4)^2. \quad (4)$$

In the minimization, we are asking for $A_{i+1} - A_i$ to be as close as possible to $m_{i,i+1}$ for all measurements simultaneously, and for A_0 to be as close as possible to zero (compare with Eq. 2). The function can be minimized analytically:

$$\begin{aligned}
\frac{1}{2}\frac{\partial\chi^2}{\partial A_0} &= A_0 + (-1)(m_{40} - A_0 + A_4) + (1)(m_{01} - A_1 + A_0) &= 0 \\
\frac{1}{2}\frac{\partial\chi^2}{\partial A_1} &= (-1)(m_{01} - A_1 + A_0) + (1)(m_{12} - A_2 + A_1) &= 0 \\
\frac{1}{2}\frac{\partial\chi^2}{\partial A_2} &= (-1)(m_{12} - A_2 + A_1) + (1)(m_{23} - A_3 + A_2) &= 0 \\
\frac{1}{2}\frac{\partial\chi^2}{\partial A_3} &= (-1)(m_{23} - A_3 + A_2) + (1)(m_{34} - A_4 + A_3) &= 0 \\
\frac{1}{2}\frac{\partial\chi^2}{\partial A_4} &= (-1)(m_{34} - A_4 + A_3) + (1)(m_{40} - A_0 + A_4) &= 0.
\end{aligned} \quad (5)$$

Rearranging it and casting it in matrix form, the problem becomes

$$\begin{pmatrix} m_{40} - m_{01} \\ m_{01} - m_{12} \\ m_{12} - m_{23} \\ m_{23} - m_{34} \\ m_{34} - m_{40} \end{pmatrix} = \begin{pmatrix} 1+2 & -1 & 0 & 0 & -1 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ -1 & 0 & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \\ A_4 \end{pmatrix} \text{ or } \vec{v} = M \cdot \vec{A}, \quad (6)$$

which can be solved by matrix inversion: $\vec{A} = M^{-1} \cdot \vec{v}$.

If $m_{40} \neq m_{01} + m_{12} + m_{23} + m_{34}$, the matrix solution spreads non-zero values of $m_{i,+1} - A_{i+1} + A_i$ (the "fit residuals") equally among all alignables. This would be appropriate if all $\sigma_{i,i+1}$ were equal, but if not, we would like the pairs with the largest statistical uncertainties to absorb the largest part of the error, such that $(m_{i,+1} - A_{i+1} + A_i)/\sigma_{i,i+1}$ (the "fit pulls") are equal. For this case, we redefine the objective function to be

$$\chi^2 = (A_0)^2 + \sum_i \left(\frac{m_{i,i+1} - A_{i+1} + A_i}{\sigma_{i,i+1}} \right)^2 \tag{7}$$

with a convention that the indices are cyclic (A_{i+1} for $i = 4$ is A_0). The minimum of the new χ^2 is solved by

$$\begin{pmatrix} \frac{m_{40}}{{\sigma_{40}}^2} - \frac{m_{01}}{{\sigma_{01}}^2} \\ \frac{m_{01}}{{\sigma_{01}}^2} - \frac{m_{12}}{{\sigma_{12}}^2} \\ \frac{m_{12}}{{\sigma_{12}}^2} - \frac{m_{23}}{{\sigma_{23}}^2} \\ \frac{m_{23}}{{\sigma_{23}}^2} - \frac{m_{34}}{{\sigma_{34}}^2} \\ \frac{m_{34}}{{\sigma_{34}}^2} - \frac{m_{40}}{{\sigma_{40}}^2} \end{pmatrix} = \begin{pmatrix} 1 + \frac{1}{{\sigma_{40}}^2} + \frac{1}{{\sigma_{01}}^2} & \frac{-1}{{\sigma_{01}}^2} & 0 & 0 & \frac{-1}{{\sigma_{40}}^2} \\ \frac{-1}{{\sigma_{01}}^2} & \frac{1}{{\sigma_{01}}^2} + \frac{1}{{\sigma_{12}}^2} & \frac{-1}{{\sigma_{12}}^2} & 0 & 0 \\ 0 & \frac{-1}{{\sigma_{12}}^2} & \frac{1}{{\sigma_{12}}^2} + \frac{1}{{\sigma_{23}}^2} & \frac{-1}{{\sigma_{23}}^2} & 0 \\ 0 & 0 & \frac{-1}{{\sigma_{23}}^2} & \frac{1}{{\sigma_{23}}^2} + \frac{1}{{\sigma_{34}}^2} & \frac{-1}{{\sigma_{34}}^2} \\ \frac{-1}{{\sigma_{40}}^2} & 0 & 0 & \frac{-1}{{\sigma_{34}}^2} & \frac{1}{{\sigma_{34}}^2} + \frac{1}{{\sigma_{40}}^2} \end{pmatrix} \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \\ A_4 \end{pmatrix}. \tag{8}$$

Just as before, the solution is $\vec{A} = M^{-1} \cdot \vec{v}$, where M is the new matrix and $\vec{v}$ is the new left-hand side. We also have all the information we need to compute uncertainties in the alignment parameters: the function $\chi^2(A_0, A_1, A_2, A_3, A_4) - {\chi^2}_{\rm min}$ represents how much worse a given set of A_i is than the best-fit A_i in units of standard deviations. A set of A_i that raises χ^2 by 1 unit is one standard deviation from the best-fit. Since χ^2 is perfectly parabolic, the one-sigma deviation in A_i is

$${\sigma_{ij}}^2 = 2 \left(\frac{\partial^2 \chi^2}{\partial A_i \partial A_j} \right)^{-1} = 2 {M_{ij}}^{-1} \tag{9}$$

where $\left(\frac{\partial^2 \chi^2}{\partial A_i \partial A_j} \right)$ is a matrix of second derivatives of χ^2. This makes ${\sigma_{ij}}^2$ a matrix as well: it is the covariance matrix of all uncertainties and correlations in A_i. Thus, inverting M gives us both the solution and all of the uncertainties in the solution.

The correlations are typically large. For the case in which all $\sigma_{ij} = 1$,

$${\sigma_{ij}}^2 = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.5 & 0.9 & 0.8 & 0.7 & 0.6 \\ 0.5 & 0.8 & 1.1 & 0.9 & 0.7 \\ 0.5 & 0.7 & 0.9 & 1.1 & 0.8 \\ 0.5 & 0.6 & 0.7 & 0.8 & 0.9 \end{pmatrix}, \tag{10}$$

which is far from being diagonal. The term in the χ^2 (Eq. 7) holding A_0 at the origin of the coordinate system is being treated like a measurement with an uncertainty of 1, and that propagates into the final covariance matrix, though it has no physical meaning. We cannot simply remove it, because M would become non-invertable: the problem would have no solution because nothing glues the system as a whole to any coordinate frame. We can,

however, consider it a free parameter, and possibly make it very weak. Redefining the χ^2 again as

$$\chi^2 = \lambda(A_0)^2 + \sum_i \left(\frac{m_{i,i+1} - A_{i+1} + A_i}{\sigma_{i,i+1}} \right)^2 \tag{11}$$

and setting $\lambda = 10$ yields

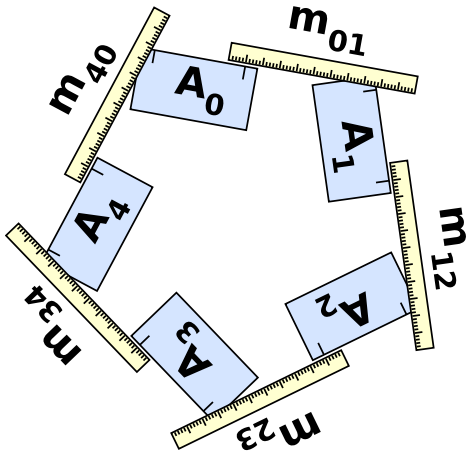
$$\sigma_{ij}{}^2 = \begin{pmatrix} 0.05 & 0.05 & 0.05 & 0.05 & 0.05 \\ 0.05 & 0.45 & 0.35 & 0.25 & 0.15 \\ 0.05 & 0.35 & 0.65 & 0.45 & 0.25 \\ 0.05 & 0.25 & 0.45 & 0.65 & 0.35 \\ 0.05 & 0.15 & 0.25 & 0.35 & 0.45 \end{pmatrix}. \tag{12}$$

Setting $\lambda = 100$ yields

$$\sigma_{ij}{}^2 = \begin{pmatrix} 0.005 & 0.005 & 0.005 & 0.005 & 0.005 \\ 0.005 & 0.405 & 0.305 & 0.205 & 0.105 \\ 0.005 & 0.305 & 0.605 & 0.405 & 0.205 \\ 0.005 & 0.205 & 0.405 & 0.605 & 0.305 \\ 0.005 & 0.105 & 0.205 & 0.305 & 0.405 \end{pmatrix}. \tag{13}$$

The first row and first column are determined exclusively by λ, and the rest of the matrix has an additive contribution from λ which can be made negligibly small. The $\lambda \to \infty$ limit makes the assertion that $A_0 = 0$ with perfect certainty (it is a definition after all), leaving only uncertainties and correlations among alignables 1–4. In each alignment solution, $A_0 = 0$ independent of λ because nothing is in conflict with this constraint.

The loop of measurements we have been discussing is equivalent to a ring of detectors, where the last is physically close to the first.



In this case, the measurements are not a linear distance but either an angle ϕ or an $r\phi$ displacement along an arc of constant radius r. This is a common configuration in particle physics, as tracking detectors are often built to encircle a beamline and collision point. Figure 1 shows an example from muon endcap alignment.

Given the symmetry of the problem, we may want to chose a more symmetric coordinate system constraint than $A_0 = 0$. Fixing the average of all alignment corrections, $\sum_i A_i = 0$

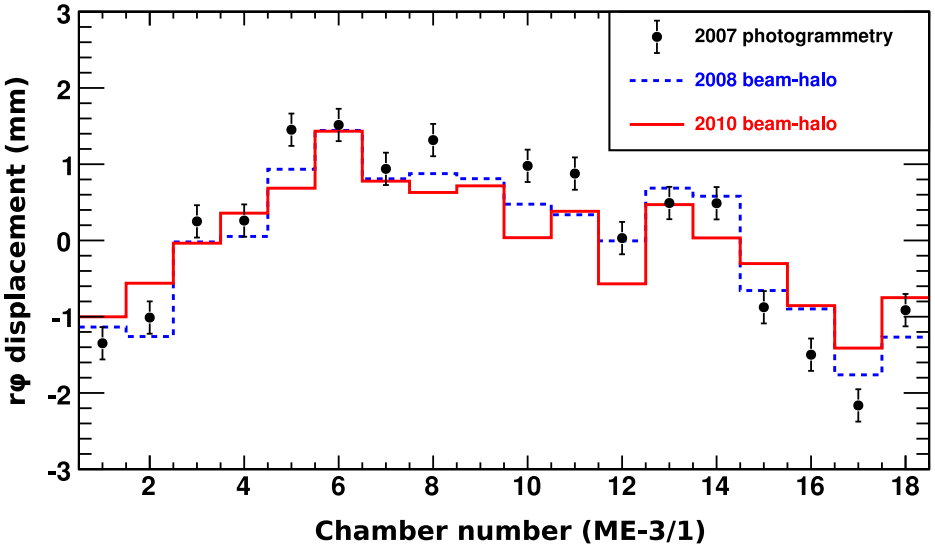
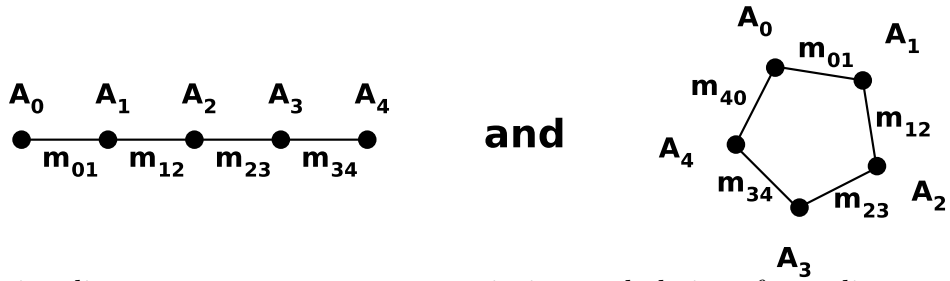


Figure 1: Beam-halo alignment measurements propagated around a circular ring as in Eq. 6 and photogrammetry (direct measurements relative to a single coordinate frame origin). The vertical axis is A_i expressed relative to a design configuration, and the horizontal axis is i.

(or any other subset), solves the same problem. The numerical results differ only in a shift of the overall coordinate system.

2.2 General Alignment Fit

In general, N_a alignables A_i are constrained by N_m measurements m_{ij}, where measurements can relate any two alignables. If we interpret the A_i as nodes and the m_{ij} as edges of a graph, we can draw a "constraint diagram" to represent the system. The two cases we have studied so far are



The constraint diagram, measurement uncertainties, and choice of coordinate system completely determine the matrix that needs to be inverted to solve a given alignment problem.

The objective function for a general one-dimensional alignment (centered on a subset of alignables $S \subseteq [0, N_a)$) is

$$\chi^2 = \sum_{i=0}^{N_a} \sum_{j=0}^{N_a} \left(q_{ij} \frac{m_{ij} - A_i + A_j}{\sigma_{ij}} \right)^2 + \lambda \left(\sum_{i \in S} A_i \right)^2 \tag{14}$$

where $q_{ij} = 1$ if a measurement exists and 0 if it does not exist. (We could have introduced fewer variables by saying that there is an edge between every two nodes and allowing some of them to have $1/\sigma_{ij} = 0$, but we don't usually think of unmeasured parameters as being "measured with infinite uncertainty.") The alignment solution is obtained when A_i are chosen such that χ^2 is minimized. The values of A_i may be "absolute" or "relative," where "absolute" means that each A_i is the distance between an alignable and a common origin, while "relative" means that an initial configuration is assumed (the "prior geometry") with m_{ij} being measurements that would be zero if the prior geometry were correct, and A_i are the changes needed to make the alignment correct. The latter is more useful for track-based alignment, because it is necessary to assume an initial alignment to fit tracks.

The last term in Eq. 14 is a Lagrange multiplier to enforce a coordinate system in which the average of A_i for $i \in S$ is zero. Any coordinate system could be chosen, and changes in A_i change the coordinate system— a fact which must be considered when comparing two alignment solutions.

Since Eq. 14 is quadratic in all A_i, its first derivatives are linear, so the minimum can be calculated by setting the first derivatives to zero and solving the system. The matrix that is used to solve the system of first derivatives is the matrix of second derivatives (the "Hessian"). With the following definitions,

$$v_k = \sum_{i=0}^{N_a} \frac{m_{ik}}{{\sigma_{ik}}^2} - \sum_{j=0}^{N_a} \frac{m_{kj}}{{\sigma_{kj}}^2} \tag{15}$$

$$M_{kl} = \frac{\partial^2 \chi^2}{\partial A_k \partial A_l} = \delta_{kl} \left(\sum_{i=0}^{N_a} \frac{q_{ik}}{{\sigma_{ik}}^2} + \sum_{j=0}^{N_a} \frac{q_{kj}}{{\sigma_{kj}}^2} \right) - \frac{q_{kl}}{{\sigma_{kl}}^2} - \frac{q_{lk}}{{\sigma_{lk}}^2} + \sum_{i \in S} \lambda, \tag{16}$$

the $\vec{A}$ (vector of A_i) that minimizes χ^2 is

$$\vec{v} = M \cdot \vec{A} \text{ or } \boxed{\vec{A} = M^{-1} \cdot \vec{v}.} \tag{17}$$

To get a feeling of how this works, try executing the following Python program. It sets up a ring of alignment measurements and solves the system (a) by minimizing Eq. 14 with the general minimization package Minuit, and (b) by solving the linear system of Eq. 16–17.

```
# some definitions
lam = 1e8      # lambda (Lagrange multiplier in Eq. 14)
Na = 5         # number of alignables A_i
fixed = None   # if fixed is None, S = [0, Na) (fix the average of all)
               # if fixed is a number, S = [fixed] (fix one alignable)

# representation of the measurements m_ij
class Measurement:
    def __init__(self, i, j, value, error):
        self.i, self.j, self.value, self.error = i, j, value, error

measurements = [
    Measurement(0, 1,  0.12, 0.01),
```

```
    Measurement(1, 2,  0.00, 0.01),
    Measurement(2, 3, -0.15, 0.01),
    Measurement(3, 4, -0.15, 0.01),
    Measurement(4, 0,  0.18, 0.01),
    ]
Nm = len(measurements)    # number of measurements m_ij

# objective function (Eq. 14) with an arbitrary number of arguments
def chi2_arbitrary(*args):
    if fixed is None: s = lam * sum(args)**2
    else: s = lam * args[fixed]**2
    for m in measurements:
        s += (m.value - args[m.i] + args[m.j])**2 / m.error**2
    return s
# put it in a form that Minuit accepts (specific number of arguments)
chi2_arguments = ", ".join(["A%i" % i for i in range(Na)])
chi2 = eval("lambda %s: chi2_arbitrary(%s)" %
            (chi2_arguments, chi2_arguments))

# (a) minimize the objective function using Minuit
import minuit
minimizer = minuit.Minuit(chi2)
minimizer.migrad()
print "Minuit", [minimizer.values["A%i" % i] for i in range(Na)]

# (b) minimize the objective function using linear algebra
from numpy import matrix
from numpy.linalg.linalg import inv, dot

# Equation 15
def vk(k):
    s = 0.
    for m in measurements:
        d = 1.*m.value/m.error**2
        if m.i == k: s += d
        if m.j == k: s -= d
    return s
v = [vk(k) for k in range(Na)]

# Equation 16
def Mkl(k, l):
    if fixed is None: s = lam
    else:
        if k == 0 and l == 0: s = lam
        else: s = 0.
    for m in measurements:
        d = 1./m.error**2
```

```
        if k == l and (m.i == k or m.j == k):
            s += d
        if (m.i == k and m.j == l) or (m.j == k and m.i == l):
            s -= d
    return s
M = matrix([[Mkl(k, l) for k in range(Na)] for l in range(Na)])

# Equation 17
print "Linear algebra", dot(inv(M), v)
```

The output should be

```
Minuit [0.0060000000945981046, -0.11399999990522246, -0.11399999990562885,
                                0.036000000094937618, 0.18600000009519196]
Linear algebra [[ 0.006 -0.114 -0.114  0.036  0.186]]
```

The Minuit and algebraic results are nearly the same: Minuit searches $\chi^2(A_0, A_1, A_2, A_3, A_4)$ for a minimum and stops when it gets within a certain tolerance, while the matrix inversion finds the exact minimum in one step. Note that the Minuit minimization only accesses χ^2 while the algebraic solution only accesses v_k and M_{kl}, so comparing the two verifies that the derivatives were properly calculated.

To see uncertainties in the alignment (covariance matrices), implement the following function in your Python script:

```
def print_matrix(mat):
    if not isinstance(mat, dict):  # put all matrices in the same format
        m = {}
        for i in range(Na):
            for j in range(Na):
                m["A%d" % i, "A%d" % j] = mat[i, j]
    else:
        m = mat
    # pretty-print matrices which are in that format
    print "\n".join([" ".join(["%8.2g" % m["A%d" % i, "A%d" % j]
                               for j in range(Na)]) for i in range(Na)])
```

and print Minuit's covariance matrix and M^{-1}.

```
>>> print_matrix(minimizer.covariance)  # Minuit
   4e-05  5.7e-10   -2e-05   -2e-05  2.5e-10
 5.7e-10    4e-05  3.1e-10   -2e-05   -2e-05
  -2e-05  3.1e-10    4e-05    4e-10   -2e-05
  -2e-05   -2e-05    4e-10    4e-05  5.1e-10
 2.5e-10   -2e-05   -2e-05  5.1e-10    4e-05

>>> print_matrix(inv(M))  # Linear algebra
   4e-05    4e-10   -2e-05   -2e-05    4e-10
   4e-10    4e-05    4e-10   -2e-05   -2e-05
  -2e-05    4e-10    4e-05    4e-10   -2e-05
  -2e-05   -2e-05    4e-10    4e-05    4e-10
   4e-10   -2e-05   -2e-05    4e-10    4e-05
```

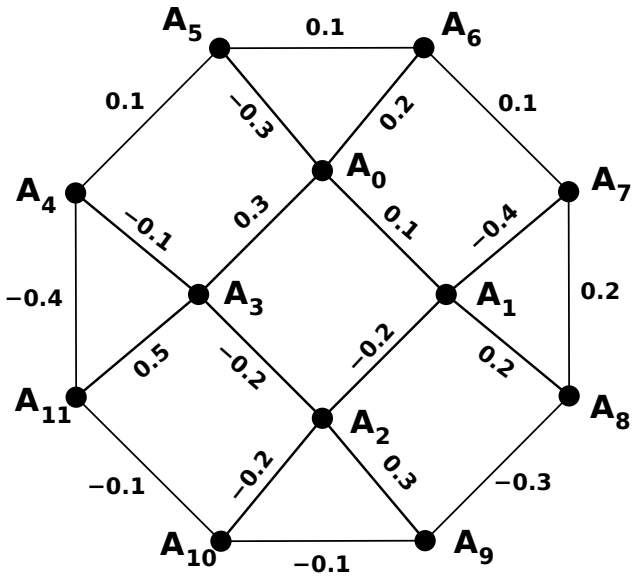


Figure 2: Constraint diagram used in several examples. Nodes are labeled alignables A_i and the values on the edges represent m_{ij}.

Again, Minuit provides an approximation to the matrix solution: only the smallest elements of the result differ.

As an exercise, try changing the collection of fixed alignables S, so that the minimization holds A_0 fixed, or A_1, etc. Are the results equivalent under global translation? Try changing the measurement constraints and their uncertainties: do the results make sense? Try to encode the constraint diagram in Fig. 2 as a list of measurements (measurements m_{ij} are the numbers marked on the edges; assume all σ_{ij} are 0.01).

Do you get the following result?

```
>>> print dot(inv(M), v)
[[ 0.01791667 -0.12458333  0.02791667  0.14541667  0.22625    -0.07625
  -0.17291667 -0.26041667 -0.18375     0.13375     0.05708333  0.20958333]]
```

Remember that this is still a one-dimensional example; the graph simply depicts which alignables are measured relative to which others.

2.3 Error Analysis

We have already seen how independent alignment measurements m_{ij} can be propagated to a final covariance matrix on A_i, but the covariance matrix doesn't give an intuitive sense of the uncertainties because it is far from being diagonal. This section shows how the uncertainties can be transformed to a more meaningful form.

We can recast the alignment solution (Eq. 17),

$$\vec{A} = M^{-1} \cdot \vec{v} \;\; \pm \;\; M^{-1}, \tag{18}$$

in a different basis in which the transformed A_i have linearly independent uncertainties. Suppose that B is a matrix such that $B^{-1}M^{-1}B$ is diagonal. By left-multiplying Eq. 18 by

B^{-1} and inserting an identity,

$$B^{-1}\vec{A} = \left(B^{-1}M^{-1}B\right) \cdot B^{-1}\vec{v}. \tag{19}$$

The vector $B^{-1}\vec{A}$ presents the same solution in different coordinates: linear combinations of the actual alignable positions, and by analogy with Eq. 18, the uncertainty in these coordinates is $B^{-1}M^{-1}B$, a diagonal covariance matrix. Therefore, the $B^{-1}\vec{A}$ coordinates are statistically independent of each other. Let's call these the "modes" of the alignment.

3 Measuring Detector Positions with Tracks

4 Realistic Tracking and Diagnostics of Track-Bias

5 CSCOverlapsAlignmentAlgorithm: a Complete Alignment Package

6 Conclusion

A Summarizing Datasets with Running Sums

It is often necessary to reduce a dataset to a few numbers by computing its mean or by fitting it to a straight line. There are many ways to do this, but the least intrusive method (no external packages) with the smallest memory usage is with running sums. Several quantities are initialized to zero, incremented in a loop over the data, and the desired quantities are calculated from the running sums at the end of the loop. These kinds of calculations are ubiquitous in alignment code, but easy to get slightly wrong, so it's good practice to check.

A.1 Mean and Weighted Mean

The mean is simply a weighted mean with all weights equal to 1. The uncertainty in the weighted mean (`weighted_mean_err`) does not reflect the width of the distribution, only the uncertainties given by the dataset (`xerr`).

Mean

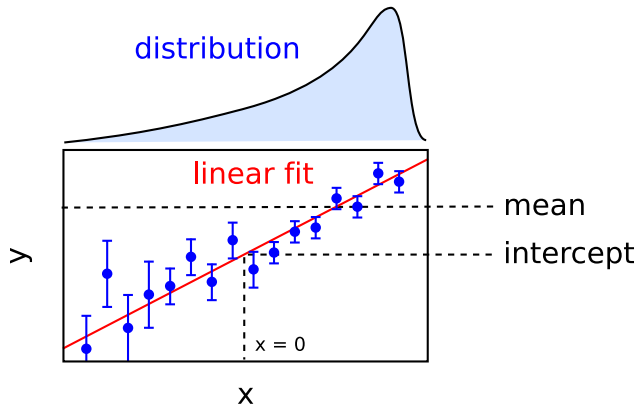
```
sum_1 = 0.
sum_y = 0.
for y in data:
    sum_1 += 1.
    sum_y += y
if sum_1 != 0.:
    mean = sum_y / sum_1
else: raise Exception
```

Weighted mean

```
from math import sqrt
sum_1 = 0.
sum_y = 0.
for y, yerr in data:
    weight = 1./yerr**2
    sum_1 += weight
    sum_y += weight * y
if sum_1 != 0.:
    weighted_mean = sum_y / sum_1
    weighted_mean_err = sqrt(1./sum_1)
else: raise Exception
```

A.2 Linear fitting

A linear fit is an extension of the weighted mean to account for a trend in the data as a function of an independent coordinate x. Sometimes knowing the slope is useful in itself, but sometimes a linear fit is needed only to remove the effect of non-uniformity in the x distribution. For example, suppose that a dataset is non-uniformly distributed in x and also has a linear trend in y vs. x. The fitted intercept quantifies the typical y value at x=0, but the mean of y does not.




```
from math import sqrt
sum_1 = 0.
sum_x = 0.
sum_xx = 0.
sum_y = 0.
sum_xy = 0.
for x, y, yerr in data:
    weight = 1./yerr**2
    sum_1 += weight
    sum_x += weight * x
```

```
    sum_xx += weight * x**2
    sum_y += weight * y
    sum_xy += weight * x * y
delta = (sum_1 * sum_xx) - (sum_x * sum_x)
if delta != 0.:
    intercept = ((sum_xx * sum_y) - (sum_x * sum_xy)) / delta
    intercept_err = sqrt(sum_xx / delta)

    slope = ((sum_1 * sum_xy) - (sum_x * sum_y)) / delta
    slope_err = sqrt(sum_1 / delta)

else: raise Exception
```

A.3 Root-mean-square and Standard Deviation

The root-mean-square (RMS) characterizes both the width of a distribution and its deviation from zero, while the standard deviation (stdev) only characterizes the width of the distribution. Note that $\texttt{rms}^2 = \texttt{stdev}^2 + \texttt{mean}^2$.

Root-mean-square (RMS)

```
from math import sqrt
sum_1 = 0.
sum_yy = 0.
for y in data:
    sum_1 += 1.
    sum_yy += y**2
if sum_1 != 0.:
    rms = sqrt(sum_yy / sum_1)
else: raise Exception
```

Standard deviation (stdev)

```
from math import sqrt
sum_1 = 0.
sum_y = 0.
sum_yy = 0.
for y in data:
    sum_1 += 1.
    sum_y += y
    sum_yy += y**2
if sum_1 != 0. and (sum_yy / sum_1) > (sum_y
    stdev = sqrt((sum_yy / sum_1) -
                 (sum_y / sum_1)**2)
else: raise Exception
```

The stdev may be used instead of a dataset's given uncertainties (`yerr`) to quantify the uncertainty in the mean. This implicitly assumes that the width of the distribution is due to statistical uncertainty, rather than an unnoticed trend. The uncertainty in the mean is $\sqrt{\texttt{stdev}/\texttt{sum\_1}}$ and the uncertainty in the RMS and stdev are $\sqrt{\texttt{stdev}/(2 \cdot \texttt{sum\_1})}$.

A.4 Covariance and Correlation

The covariance of a distribution is an extension of the stdev in two (or more) dimensions, quantifying the off-diagonal elements in a covariance matrix

$$\sigma = \begin{pmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} \end{pmatrix} \tag{20}$$

where $\sqrt{\sigma_{xx}}$ and $\sqrt{\sigma_{yy}}$ are the stdev of the x and y distributions, respectively, and σ_{xy} is the covariance. Correlation is the same thing, normalized such that diagonal elements are 1 (quantifies only the shape of the error ellipse).

Covariance

```
def mean(data):
    return sum(data)/len(data)
xmean = mean(xdata)
ymean = mean(ydata)

sum_xy = 0.
for x, y in zip(xdata, ydata):
    sum_xy += (x - xmean) *
              (y - ymean)
covariance = sum_xy/len(data)
```

Correlation

```
from math import sqrt
def mean(data):
    return sum(data)/len(data)
xmean = mean(xdata)
ymean = mean(ydata)

sum_xx = 0.
sum_yy = 0.
sum_xy = 0.
for x, y in zip(xdata, ydata):
    sum_xx += (x - xmean)**2
    sum_yy += (y - ymean)**2
    sum_xy += (x - xmean) *
              (y - ymean)
if sum_xx * sum_yy != 0.:
    correlation = sum_xy /
            sqrt(sum_xx * sum_yy)
else: raise Exception
```