

Using variational methods to establish the optimal precipitation ensemble forecast mean magnitude, areal extent, and ensemble-derived location using discrete field translation

Daniel Birkenheuer

Abstract

This memo describes a technique using fast integer arithmetic to match shapes from different sources that produce similar 2D output. The shapes, for example, could be precipitation forecasts from different computer forecast models. In the area of ensemble forecasting, this allows the computation of a mean forecast field from ensemble fields that are aligned using this algorithm (field alignment), followed by the translation of that mean field to the location described by the minimal translation from all ensemble members. This work was funded by director discretionary funding under ESRL's Global Systems Division.

Introduction

The popular method to establish an area-precipitation ensemble mean (simple field averaging) is deficient due to positional errors in model output (Alexander et al. 1998). This conventional method computes an ensemble mean by simply averaging the precipitation amounts at each location from the ensemble set of forecasts. This process results in: 1) a smearing of the precipitation areal coverage due to ensemble members placing the precipitation location in different positions within the domain; and 2) the reduction in precipitation amplitude(s), since even a slight miss-alignment of precipitation regions results in a precipitation magnitude reduction by the process of point numerical averaging – even if each forecast had exactly the same shape and amplitude(s) (just different placement). This is because simple averaging does not take into account the possibility that different model output might show similar fields but place the precipitation in slightly different locations – so-called position error.

In 2009, an unpublished study by Brad Beechler (ESRL/GSD) explored a new technique that was shown to have better performance by using observations (such as radar reflectivity) to aid in field alignment. The GSD-developed technique was considered an “observation dependent” approach and was limited to quickly updating observations such as satellite and radar, since observation latency had to be considered.

Other approaches, much more comprehensive, have been developed (Ravela et al. 2007) that take into account model error, translational shift, rotation, and warping of the fields to arrive at superior alignment. Once fields are better aligned by minimal shifting of the ensemble members, it is reasonable to assume the mean of the shifted fields will be a superior solution to unadjusted field average.

The proposed approach here is unsophisticated and meant to be observation independent, very efficient, and could even be applied to a 72h forecast ensemble set whereas the GSD method by Beechler could not do this because it required observations at the current time. The other objective was to derive a rudimentary field alignment method that was efficient and could run with minimal computation. The technique explored here can be applied to any set of horizontal spatial fields deemed representative of some variable; this can be considered a generic solution method applicable to any field. Precipitation is selected only as a “first test” parameter. The algorithm is developed first and the latter part of this memo describes ideas for application.

For this algorithm, the i,j element of each ensemble member is aligned with all others at all (i,j) discrete locations. The minimization that occurs considers all manner of orthogonal shifting from each translational set to compute a minimum in alignment without using observation data. This method is based on reduction in “miss match” of forecast amounts as all possible alignments are examined. The only place observations enter the picture would potentially be in validation. The resulting “shape” by this method can change by however the forecasts drive the result, there is no observation constraint. If precipitation areal shapes differ radically in a set of forecast ensemble members (larger or smaller), this method will cope with all of those types of changes, producing an ensemble average of the new “shape”. In contrast to observational dependent techniques, this is more robust, and since an integer-based alignment method makes this technique computationally efficient, the technique could quickly be rerun after field outliers were identified. This discussion will be explored at the end of the memo.

Variational minimization is used to optimally align the ensemble precipitation fields by determining the unique translation vector for each field, while maintaining the constraint of minimal translation. The minimized functional is the inverse of the summed product of all translated ensemble members. The derived set of translation vectors has dual use, first in the recovery of the true mean ensemble precipitation magnitude, and second in determining the ensemble mean field’s location.

J' , is defined as a maximum functional

$$J' = \sum_i \sum_j \vec{A}(a_{ij}) \vec{B}(b_{ij}) \vec{C}(c_{ij}) \dots \quad (1)$$

Where capital symbols A, B, C... are translation vectors for each ensemble member corresponding with lower case ensemble member fields – (lower case a,b,c...) and gridpoints are defined by i and j . The product of all translated ensembles is summed over all i,j . No shape considerations are accounted for, nor need to be in (1).

The minimization functional is the inverse of J'

$$J = 1 / J' \quad (2)$$

The minimization of J is the maximum of J' if J' is greater than zero. The translation vectors are determined such that J -prime is a maximum value that will occur when the set of ensembles is best aligned and produce the largest value in overall product precipitation value. When the best alignment of fields is obtained, J will have minimum value and render an optimal set of A,B,C... translation vectors. This is how the set of translational vectors set is obtained. Will J ever be singular? – Only if there is no precipitation anywhere in one ensemble member. That would be a pretest and would be avoided or a small non-zero value assigned to all zero points. This action would not affect J minimization.

The set of translation vectors are used in the averaging process to establish the shape and magnitude of the mean precipitation area. Specifically, the averaging is done after each ensemble member is translated to its optimal alignment location, as defined by the set of translation vectors derived from the above minimization scheme.

$$ensemble_mean_{ij} = [\vec{A}(a_{ij}) + \vec{B}(b_{ij}) + \vec{C}(c_{ij}) \dots] / member_total \quad (3)$$

The last step is to “place, or locate” the mean precipitation field [from (3) above - just computed] to the “ensemble location.” This is accomplished by averaging the set of translation vectors computed by minimization, then applying this one translation vector to the mean precipitation field established in the averaging step (3).

$$\vec{E}_{translation} = [\sum \vec{A} + \vec{B} + \vec{C}...] / member_total \quad (4)$$

The result translated ensemble mean is then defined as:

$$located_ensemble_mean_{ij} = \vec{E}_{translation}(ensemble_mean_{ij}) \quad (5)$$

At this point, the technique has been presented. The discussion will go beyond this stage however, describing some of the additional applications of the technique and explaining the technique step-by-step. For example, the translation vectors could be statistically assessed for their dispersive characteristics to ascertain information about the ensembles in general (whether there was good agreement, poor agreement, or contained statistically significant outliers). The minimization functional (1) and (2) could incorporate bias corrections depending on the model used for a specific ensemble member (for cases where the ensemble set comprises different models). The latter item mentioned is not a part of this technical memorandum, but illustrates the potential it serves for follow-on work if the new method for ensemble mean forecasts proves effective.

Computational Approach

Initially, it was envisioned that (2) could be minimized by using a package, such as the Powell (1962) method, that has been used successfully in the past for humidity solutions using a 1DVar approach. This is currently documented in the LAPS system (Birkenheuer 2006). It was discovered that the functional was not easily transformed into a method sought to iterate on discrete index values (integers), but worked best for a numerical solution of rational numbers. Therefore, a new approach was designed for discrete minimization, and it was devised particularly for this 2-D problem. Solving this problem in a discrete manner simplifies it greatly with the limitation that the integral solution is as close as can be obtained. However, considering that the data is only as good (in resolution) as the density of the gridded field, this is a reasonable result. One approach to obtaining better resolution in the result would be to use denser grid spacing and still rely on finite mathematics to achieve the result. One must also realize that even if a rational result were computed, it could be argued that the accuracy of that result’s translated position might only be as good as the parent grid resolution.

Sliding windows – a fast minimization solving algorithm using integer computation

A scheme was eventually invented in which the fields would be centered in a larger array that was exactly 2x larger the given field, referred to as the reference-field. Figure 1 shows such a configuration graphically.

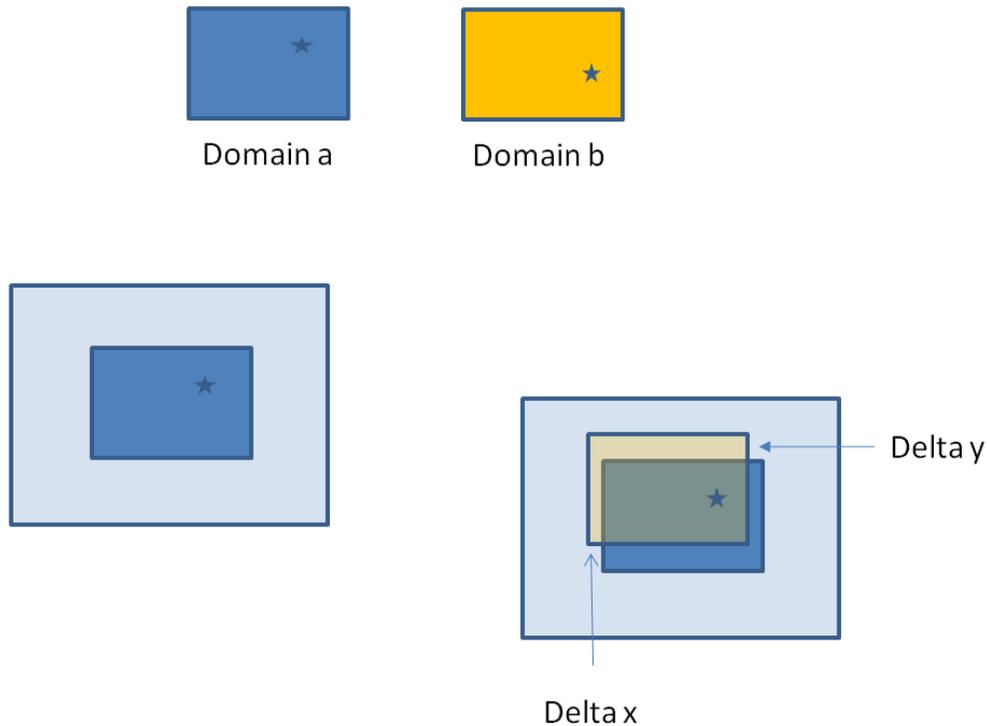


Fig. 1. An illustration of one field (domain a) placed into a larger field (light blue) referred to here as a reference-field, which would allow other fields to slide over it in roughly 50% of the possible i, j configurations. There is no reason to slide the window further since that would mean the precipitation field would be over $\frac{1}{2}$ a domain away from the centered reference-field, which would be an unlikely solution with current model precision. Here both domain a and b have a “feature” (star) and they are aligned as domain b is slid over the reference domain, revealing delta-x and delta-y that define a displacement vector that corresponds to differences in i and j values.

The algorithm is to take the information from performing what is done in Fig. 1 with all fields in the ensemble against the first domain (or domain “a”), and then working out the set of translation vectors used in (1), (3), and (4). This minimizes (2) and works out the translation vectors simultaneously. The best way to illustrate the algorithm is with the following algebra looking in only one dimension (x , or the i direction). Since this is orthogonal, the same algorithm can be used identically in the other dimension (y , or the j direction). This can be referred to as a rudimentary displacement solution. A minimization approach allows the ability to weight the terms differently to serve specific purposes.

The algorithm

Imagine n fields, and to make it easier, these are 1-dimensional features in the x (i direction). In an ideal case, each contains an identical feature, but at a different location in x , or with a different integral index location i . Assume that $i=0$ is a reference "first field" and other fields are related to this field to generate delta values comparable to delta- x in Fig. 1. The objective is to identify delta vectors for each field to move them the minimal amount so you can align them.

The process begins by not moving (or aligning the first location to itself), which is known to maximize the fit in (2).

$$\Delta x_1 = x_1 - x_1 = 0 \quad (6)$$

The delta x for the second field can be defined as

$$\Delta x_2 = x_2 - x_1 \quad (7)$$

Or generally stated,

$$\Delta x_n = x_n - x_1 \quad (8)$$

The next step is to determine the average delta, and computing this for an arbitrary number of cases. Note that this always includes the first term that is zero.

$$\overline{\Delta x} = \sum_{i=1}^n (x_1 - x_i) / n \quad (9)$$

The final step is to move the individual fields (now including the first field) to the minimized weighted location, thus simultaneously solving (2), minimizing the functional, and determining the vectors. This step breaks into two parts. The first part computes primed quantities, and these are different for the first field and all of the others rendering displacements. The second part uses the displacements to determine translation vectors. The first field (reference field) is moved toward all of the others, and the other fields toward the first field.

$$\Delta x'_1 = x_1 - \overline{\Delta x} \quad (10)$$

And following for the other fields

$$\Delta x'_n = -(x_n + \overline{\Delta x}), n > 1 \quad (11)$$

The DeltaPrime values are in fact the translation vector component in the x direction desired in (1) and (2).

The one-dimensional development can now be expanded to two dimensions. Performing the computations in (6) through (11) similarly in the y direction, when coupled with the x direction results, give us the 2-dimensional translation vectors.

$$(\Delta x'_1, \Delta y'_1) = \vec{A} \quad (12)$$

Likewise, to preserve the notation in (1)

$$(\Delta x'_2, \Delta y'_2) = \vec{B} \quad (13)$$

And so forth for vectors C, D, and E to the nth member. To translate the fields for computing the ensemble mean (3), the place to move field n in the x direction becomes,

$$x_{n,moved} = x_n + \Delta x_n' \quad (14)$$

There are a few details to mention before the description is complete. First, instead of working in x or y , computations are in i and j (integers). Second, when (9) is computed, it is generated from integers in the formulation here such that the fraction will become a rational number, and more than likely not be an integer. Since the grid translates by an integral amount, the mean delta (translation) term in (9) is assigned the nearest integer. This allows for discrete field translation.

Discrete field translation is accomplished as in Fig. 1, but for all fields as they are placed in the center of a 2x larger field and translated in i and j within that larger field. When this is done for all member fields, the 2x larger fields are averaged per (3). Then the center of the mean large 2x field, corresponding to the original domain, is used for the mean result. The result corresponds exactly to the initial domain, but now contains the mean ensemble values averaged and relocated to their optimal position.

The reference-field framework allowed the preservation of each field and subsequent realignment of each field. By beginning with field 1 (arbitrary - but maintained as the reference-field throughout the process) being centered into the reference-field, we provide a comparison framework. By manipulating field 2 to all i,j positions with respect to the reference-field, the functional (2) can be computed at each position and its minimum value recorded. Then the same reference-field is applied against the next field (field 3) and the minimization for that field is repeated, recording that aligned position and so forth, continuing in-kind with all n fields.

When all of the fields are best “aligned” to the reference-field, a set of translation i,j values satisfying the criteria defined by (10) and (11) are derived, and the minimal displacement to align all fields are computed as the vector average of all of the individual field displacements. The displacement vectors i.e., 12, 13 are applied as in (3), and this renders the ensemble mean and location.

Three-field minimization example

Figure 2 shows a three-field minimization example plotting the minimum functional value as an incremental alignment search occurs. Here, the decrease during the iterative search for the “best” alignment of fields 1 and 2 extends from iteration 1 through about 2500. The plot then shows the minimizing process repeated for fields 1 and 3. This comprises the second part of the plot from iteration 2500 to the end (about 5000). Both field comparisons reached a minimum value, but the second pair, 1 and 3, has a lower minimum result.

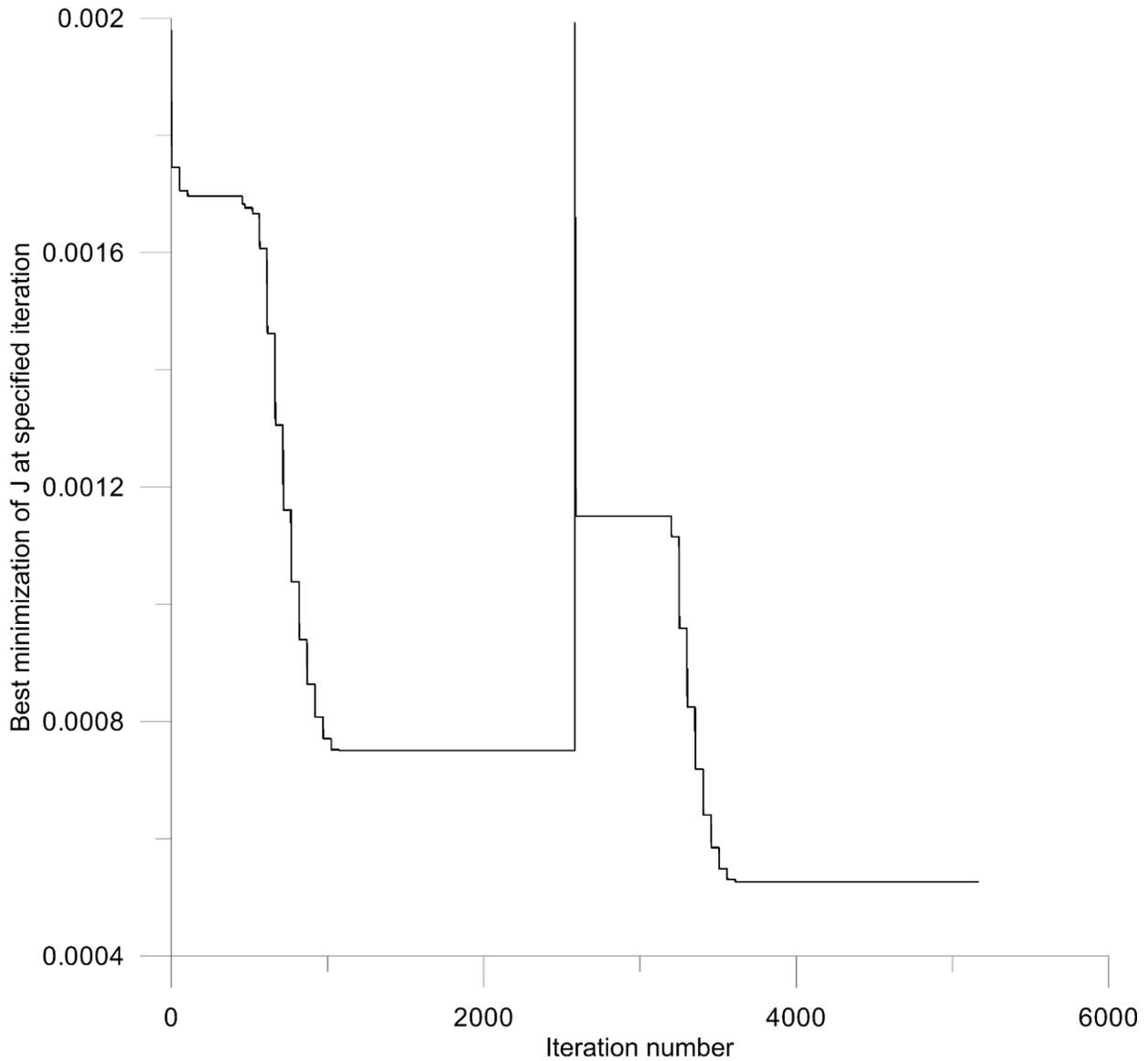


Fig. 2. A plot showing the minimization of the J functional as a function of iteration for two cases. It is showing the comparison between two sequential solutions, fields 1 and 2, and fields 1 and 3, during the processing. Here field 1 was treated as the reference-field. The plotted value is not the functional at that iteration, but rather the minimum value achieved up to that and including that iteration, thus showing minimization convergence over time. Each derived minima are independent. At iteration ~ 1000 , the minimum occurs for the first pair, and the displacement for that pair remains constant for subsequent iterations to the final one (~ 2500). At iteration ~ 2501 , the minimization for the second pair begins. At iteration ~ 3500 , the minimum for the second pair occurs, and remains constant until the end of the run. In this example, the second pair of fields (1,3) was a better match because it produced a minimum with a lower value. If field 1 and field n happened to be exactly the same, then $J(2)$ would be the smallest achievable number, and the shift of field n would be (0,0), placing it directly over the reference-field center, or aligned exactly with field 1.

As shown in Fig. 2, all possible displacements are examined, and when finished, the best alignment displacement of each field (2 to n) individually compared to field 1, the “reference-field,” is discovered.

It is apparent in Fig. 2 that a speedup of the algorithm is potentially possible. Since the precipitation fields will generally be somewhat similar to each other, the best match will occur near the “central” iteration. It may be safe to assume that once that iteration has occurred and the function has not changed for several hundred subsequent iterations, that the solution has likely been found.

Termination of the search at that point may be possible with CPU time savings. This assumption only holds if the fields are similar and the precipitation area remains localized in all fields.

Also apparent in Fig. 2 is the fact that the second pair was a better match with the given functional constraints defined by (2), since the minimum was less. This fact can be used with a large number of ensemble members to gauge the “set of best fits” when used with the displacement vector set. The minima and displacement vector “outliers” could be used as criteria to discard an ensemble member if it was deemed to be “too far away” from the solution cluster. Certainly not all problems lend themselves to this outlier treatment, but if one is desired, it is an objective way to determine and discard unwanted ensemble solutions. It should also be mentioned for such cases that it would not be that the smallest minimum would potentially represent a cluster (since all are compared to field 1), but rather the “most popular” minimum value. For example, if you had 6 fields and 4 paired minima were at a value near 0.2 and the 5th pair produced a minimum of 0.1, the 4 pairs at 0.2 would be deemed the “cluster” in regards to minimization, not because they were of the lowest value, but because they were of similar value.

The discrete advantage of this solution is in part due to the integral number of solutions. The problem is constrained to a finite number of known alignment configurations for the different fields, and a discrete (not rational) displacement is sought. Thus the problem is quite limited and tractable by today’s high-speed CPU computing machines. This is a brute force approach, with the best solution within easy grasp. If two truly equal minima are computed during this sliding window solution approach, it really does not matter which one to use. They are both equally valid. In this case, constraints could be placed on the solution that resolved the conflict, such as accepting the minimum translation result as superior.

Simple case – artificial data

Several artificial data cases were created and tested on the algorithm to assure its proper functioning. The first actually was a single point in the field assigned a value of 1.0 in a field of 0.0 values at all other locations. Three fields were set up, with the unity point at different locations in each. The algorithm ran, aligned each field, and then produced a relocated ensemble mean location and value. This was computed both by the algorithm and manually to assure the code worked.

Following this test, a more challenging test was run and presented below. In this case, a function was created both in x and y that resulted in maxima and minima in the 2D field with different magnitudes. The field was not allowed to go negative. Where the function had negative values, 0.0 values were imposed. This resulted in a pseudo-precipitation field with a distribution of differing max and min precipitation areas with dry regions between them. The function used is shown in Fig. 3.

Function $\cos[(25-i)*0.1]+\cos[(32-i)*0.3]$

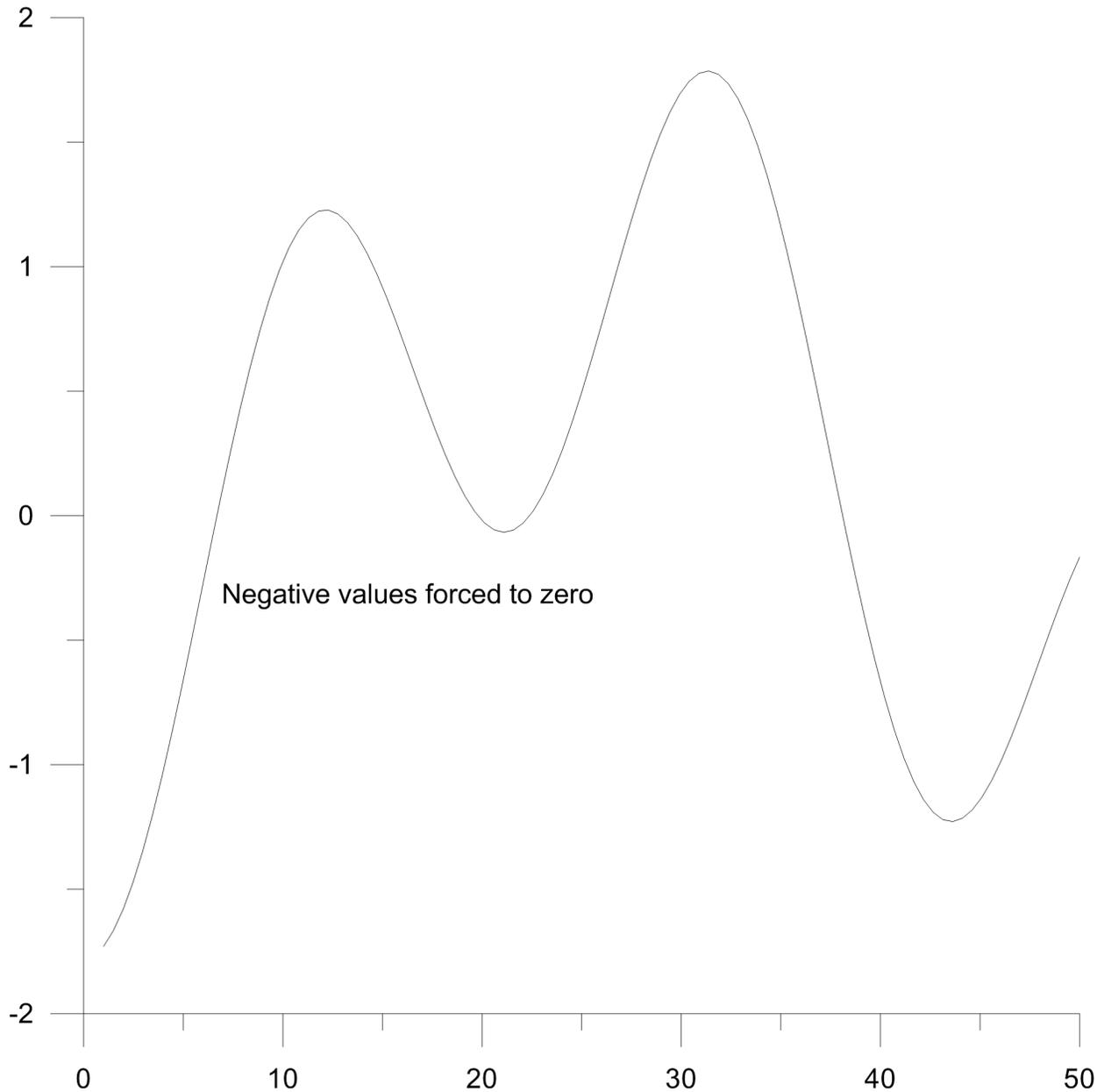


Fig. 3. The 1D continuous function used to simulate maximum, sub-maximum, and zero regions of precipitation. When used as a function in both the x and y directions (i and j), this yielded the 2D fields shown in subsequent figures. This function was designed for a 50x50 grid, so values are plotted ranging from 1 to 50. Values less than zero were assigned a zero value, simulating precipitation peaks that ranged from ~1.25 to a little under 2. One must keep in mind that the function was discretized to integral values at grid locations, thus true maxima amplitude and locations of both maxima and zero regions could differ slightly than plotted here.

The function in Fig. 3 could then be used in both x and y directions to render 2D fields of precipitation with varying amplitudes in the pseudo-precipitation areas. The arbitrary unit of inches was used for the

amplitude, so in this example just about 2 inches of precipitation was simulated by this function. Figure 3 is shown for i and it was also used in the j direction in the following plots.

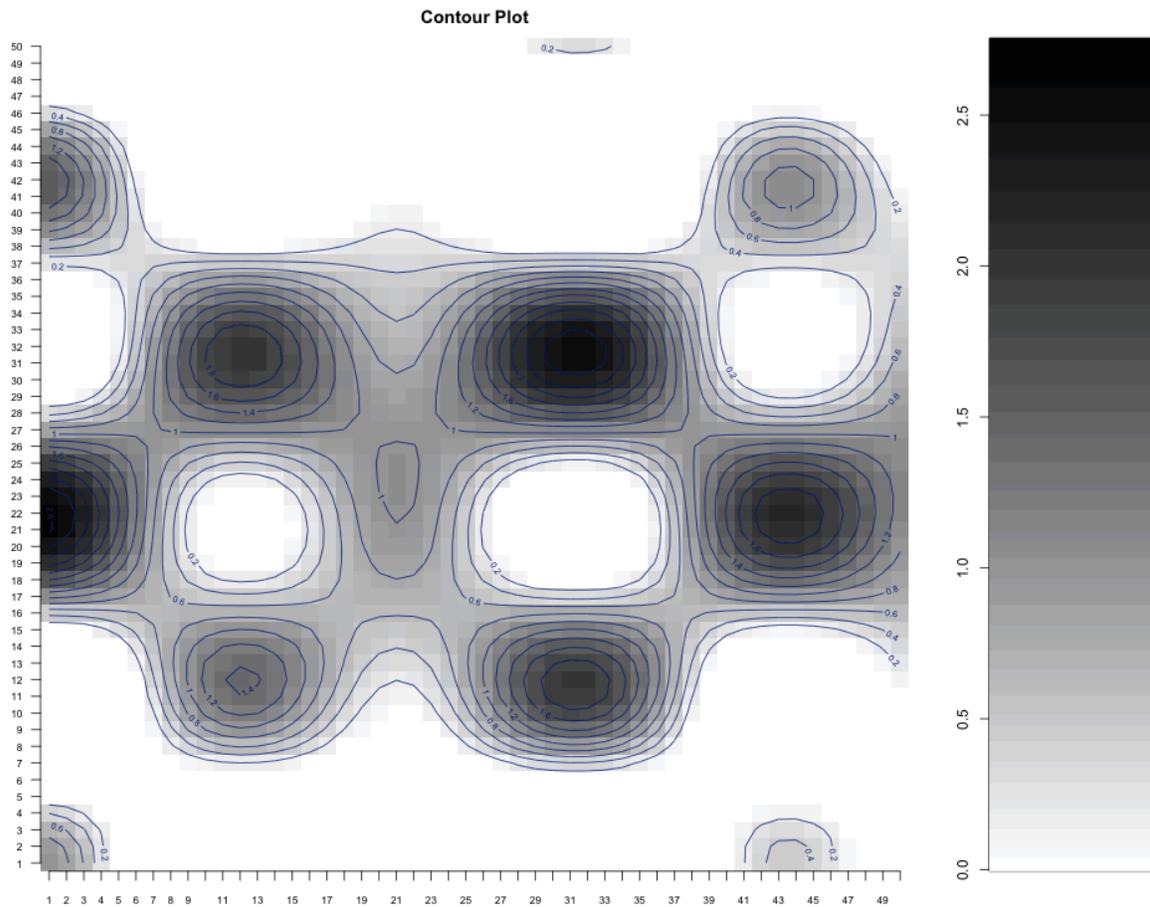


Fig. 4. Contour plot of the first of three pseudo-precipitation plots generated using the function shown in Fig. 3. The same function is used in the i, j directions with the following assignments. It is easy to see that there are several peaks and valleys and also extensive regions of no precipitation. The gray-scale is darker where there is greater precipitation and the gray-scale extends from zero (white) to 2.75 (black). The precipitation regions are also contoured in blue. In this plot, the equations for each location are based on Fig. 3, but are specifically based on: $Z = \text{Pos}(\cos[(25-i)*0.1] + \cos[(32-i)*0.3] * \cos[(25-j)*0.1] + \cos[(32-j)*0.3])$, where the "Pos" function returns only positive values with negative values forced to 0.0.

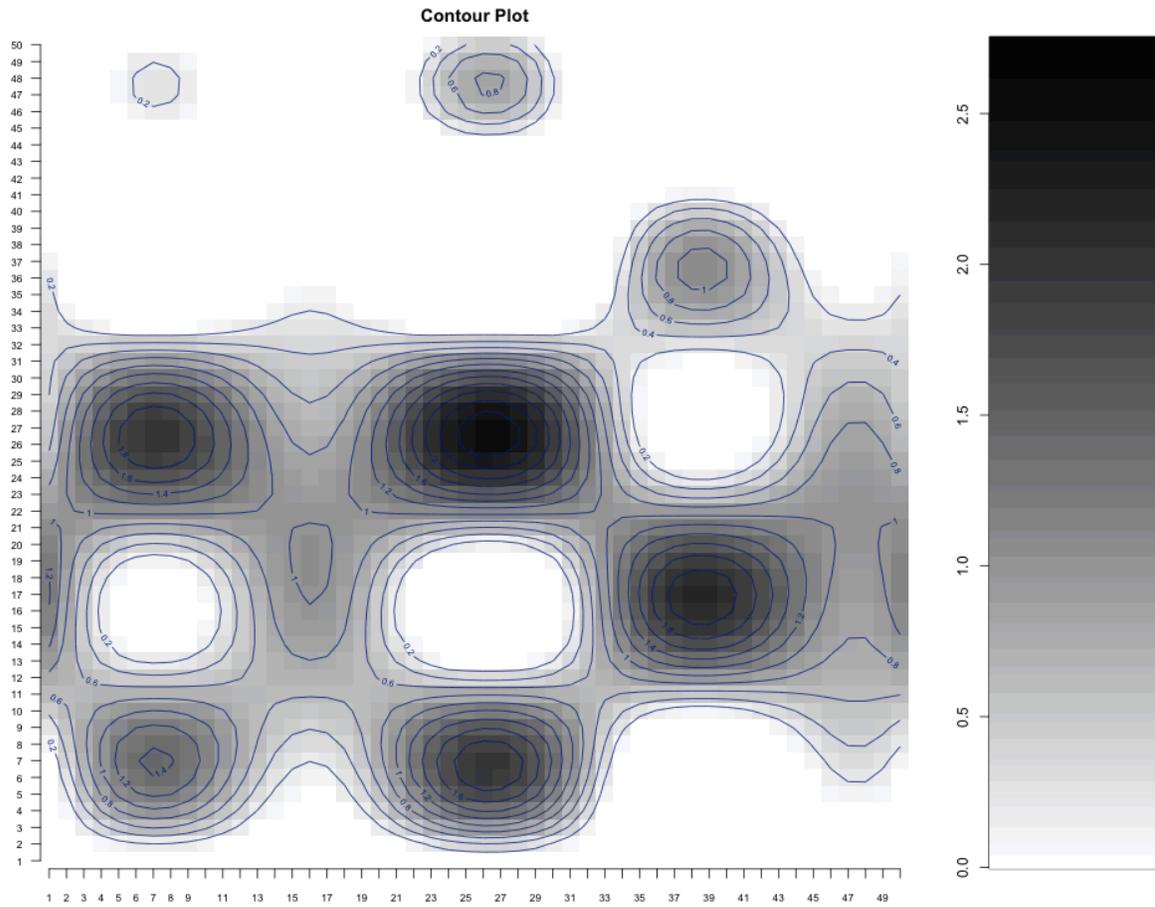


Fig. 5. As in Fig. 4, but with a different i, j displacement value so as to shift the plot as shown. Here:
 $Z = \text{Pos}(\cos[(20-i)*0.1] + \cos[(27-i)*0.3] * \cos[(20-j)*0.1] + \cos[(27-j)*0.3])$.

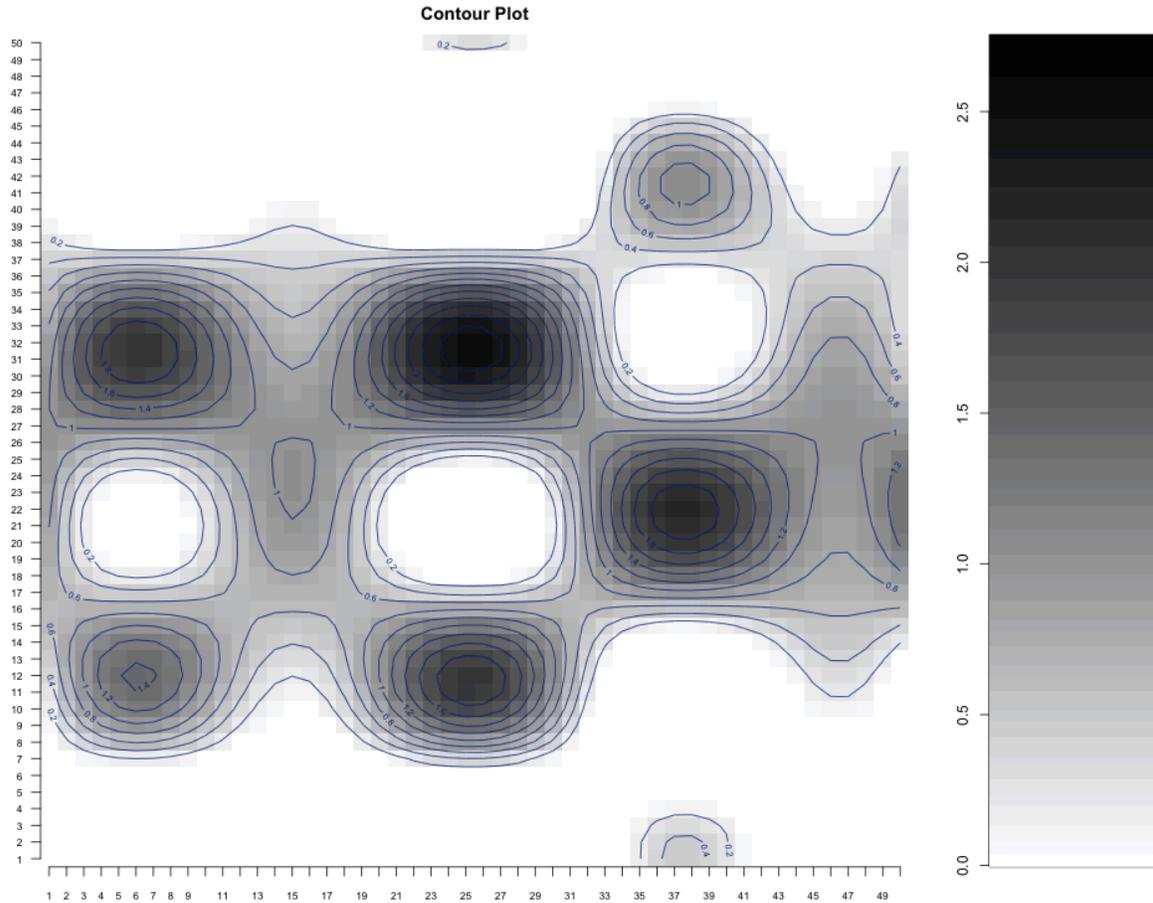


Fig. 6. As in Figs. 4 and 5 but of yet another offset in i , and j . Here: $Z = \text{Pos}(\cos[(19-i)*0.1]+\cos[(26-i)*0.3] * \cos[(25-j)*0.1]+\cos[(32-j)*0.3])$. This looks much like Fig. 5, but is shifted “up” and very slightly to the left (by one index).

Using the three examples shown as pseudo precipitation fields, the algorithm was applied to both to establish the average location of the resulting field. This was done by measuring the displacement vectors, moving field 2 to field 1 and then field 3 to 1, then determining the minimum average displacement for all three vectors and applying the respective translations to each field. The numerical average was then applied to the summed result.

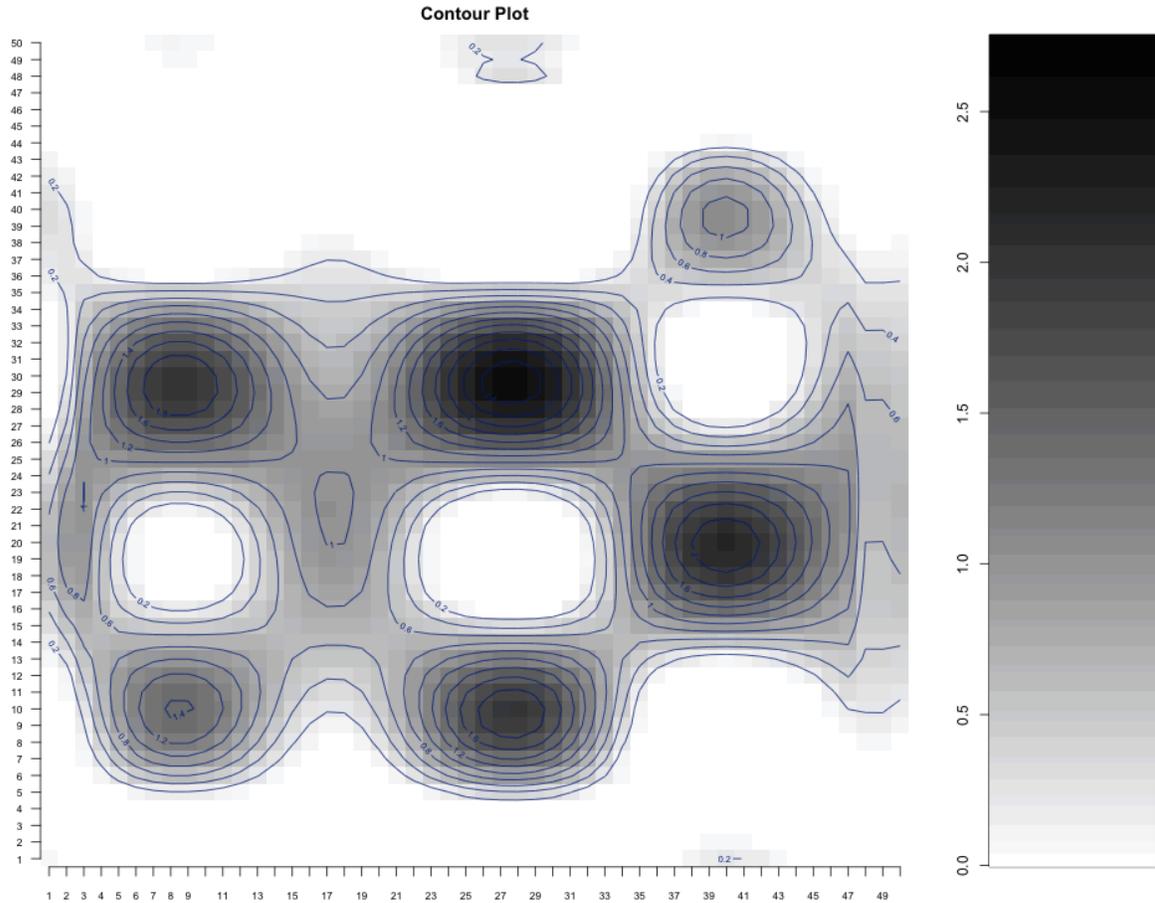


Fig. 7. Location and magnitude of the shifted and averaged result. Shown is the performance of the algorithm run on the three prior fields.

Fig. 7 maintains the dominant features of the displaced pattern. The location of the result does not align with any of the parent fields but is the mean location. The structure is fully maintained both in the amplitude and spacing of the features. Discrepancies do appear near the boundary of the domain where the most differences occur, due to the shifting inward of “no-data” or removal of data when it was shifted to a location outside of the domain. It would be possible to adjust the algorithm to ignore “zero” data that was translated inward from outside the domain.

To contrast this result with a simple average of the three fields, the following is a simple contoured average of the three.

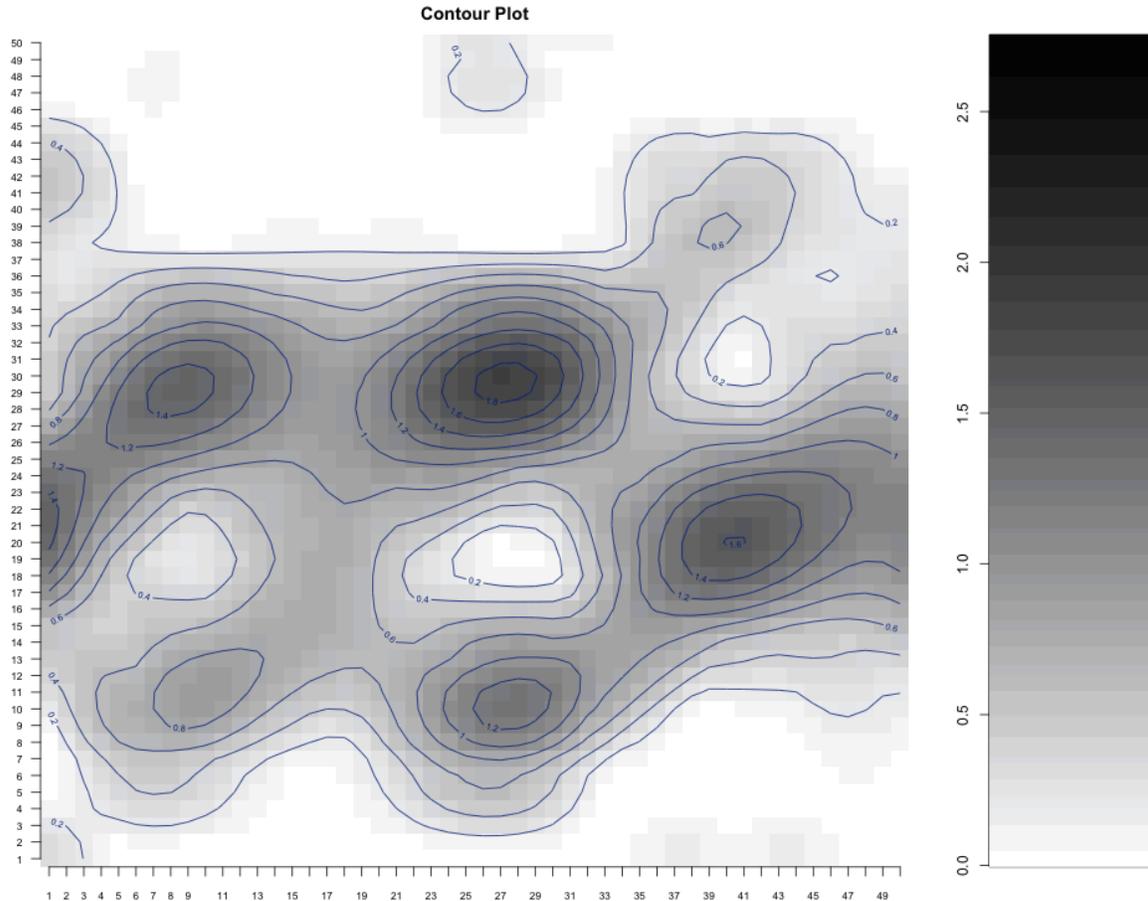


Fig. 8. Simple average of the three input fields shown in Figs. 3, 4, and 5 without regard for finding optimal alignment.

It is obvious when comparing Fig. 8 to all of the other fields (Figs. 4-7) that it is an inferior representation of any of the pseudo-precipitation fields as an average. It does tend to draw the peaks and valleys to the more optimal locations shown in Fig. 7, but that is because the alignment before averaging is inferior, the structure is “smeared,” and the amplitudes of the peaks are reduced. This is because by averaging the fields in which the peaks had not been aligned, some averaging takes into account regions of lower amplitude, thus giving a lower mean result.

The above pseudo-precipitation example illustrates that the alignment and averaging after alignment technique renders a superior ensemble average – at least in so far as an averaged displaced field. Figure 7 maintains the shapes and amplitudes of the input images, while shifting the majority of the field to an “average” location that is the location of the “minimal” shift of all fields.

As an ensemble post processing average, this appears to be the ideal solution. Recall that the full capabilities of the functional (1) can be modified in such a way as to render solutions that may favor other attributes that are deemed desirable in a solution, since this is a minimized functional. It is generally easy to modify such a functional to favor specific goals. For example, in this illustration, (1) was maximized to insure that the result aligned the peak amplitudes of the different fields. One could focus on major peaks by squaring each field (15).

$$J' = \sum_i \sum_j \vec{A}(a_{ij})^2 \vec{B}(b_{ij})^2 \vec{C}(c_{ij})^2 \dots \quad (15)$$

Thus, the solution would focus more on the extreme peaks and lessen the importance of smaller peaks. Also, data could be filtered before applying (1) such that small amounts of precipitation less than a given threshold would be forced to zero before the alignment. Then the true fields can later be restored for computing the ensemble average, but using the alignment criteria determined from the modified fields. This enables focusing the minimization to use the major precipitation maxima to dominate the alignment step.

The functional is only one way this can be modified. Another could be weighting specific microphysics packages over others in the field averaging step. Thus the displacement and mean location of the final field could be determined by all members equally, but the ensemble average would be weighted in favor of preferred microphysics schemes.

One could also perform a filter of the solutions as discussed following the explanation of Fig. 2, such that “outlier” members could be removed and a new ensemble average computed with the remaining members. This would focus more on the cluster of the resulting members. The member exclusion criteria could be devised in numerous ways, with different thresholds designed to satisfy specific probability levels as an example. Taking this one step further, cluster solutions could be identified. Each cluster could satisfy some pre-determined location-specific criterion. This could reveal different solution regimes that might be worth considering.

Weights could also be assigned by location, giving more influence to results that occurred in a specific locale. Such might be a way to tune the ensemble average to force adherence to orographic considerations that might be known.

The knowledge of agreement or how things translate to a result can be used in reverse to understand what is working and what is not working. This above tool can be used to verify ensembles. One can compute a new j functional as a match to the verifying observation field. Running the above algorithm on all ensemble members, where the observation set is taken at “field 1” or the reference field, will result in computation of j minimizations (Fig. 2) for each ensemble member paired with observation. In a verification setup case, the discovered j with the lowest value will be the ensemble member that **verifies** the best against the reference field. This knowledge could be useful in understanding which members do consistently well, and perhaps would be an efficient **verification tool**.

Summary

The major accomplishment of this activity was to approach the ensemble average as a solution to a minimized functional. Secondly, the established mechanism for solution was to switch from a method that solved for rational numbers to one that worked in a suitable, but far simpler, subspace of integers. The algorithm was written in FORTRAN 77 and demonstrated desired results in a test case.

Realizing that a rational solution would likely be rounded to the nearest integer to preserve the grid coordinate system of the solution and avoiding interpolation cost, the entire solution process could be

done much more simply and computationally economically using the brute force integer solution technique shown here. If integral gridded solutions prove too coarse, a better option is to use denser grids, the likely trajectory of future model and assimilation methods. Then the integer solution remains more favored, since there will be less and less need to even consider reanalysis and mapping back to a coarser grid.

The author does not claim that the integer solution method shown here is the most optimal. It may only be one of several ways to solve the problem. However, by keeping the problem to one of shifting a grid by integral amounts, any discrete method can be numerically cheaper, while remaining very robust when compared to a rational solution.

References

G.D. Alexander, J.A. Weinman, J.L. Schols, 1998: The use of digital warping of microwave integrated water vapor imagery to improve forecasts of marine extratropical cyclones, *Mon. Weather Rev.* **126**, 1469–1495.

Birkenheuer, D., 2006: The Initial Formulation of a Technique to Employ Gradient Information in a Simple Variational Minimization Scheme, *OAR-GSD Tech Memo - 32*, 22pp.

Powell, M.J.D., 1962: An iterative method for finding stationary values of a function of several variables. *Computer J.*, **5**, 147-151.

Ravela S., K. Emanuel and D. McLaughlin, Data Assimilation by Field Alignment, *Physica D*, **230(1)**:127-145, 2007

Acknowledgements

Special thanks to Stanislav Stoytchev for his help in preparing most of the figures in this report.

Appendix 1: Software listing

Subroutines *minimize displacement* and *funct* are used to discover alignment and define the functional respectively. The software presented here is the code used to solve the minimization and produce the examples shown in Figs. 4-8. Source code can be downloaded from:

<http://laps.noaa.gov/birk/misc/driver3v3.f>

Program Driver3

```
c routine to oversee the execution of the post ensemble processing software
c in regard to DDF award for research on precipitation alignment.
c
c Author: Daniel Birkenheuer
c Date: 5/25/2010 Tuesday
c Second major rewrite: July 7 2010 revising the functional minimization
c approach from using tradition methods to one that I have created.
c
c July 21, 2010 arrived at version that converges a minimization
c solution of a field using index arithmetic, and can be upgrated to
c include other real number weak constraints to minimize J, a new
c discrete functional. Here to fore this new
c code will be version 2.0
c
c version 1.0 does not incorporate allocatable arrays
c this version is also testing the sync feature with backup between the imac
c and my home directory.
c
c Version 2.0 - solves array alignment by using index minimization
c
c
c 2.1 - solves the index translation (vector displacement) part of the
c problem
c July 29, 2010
c
c Aug 2, 2010 tested 3-point problem and resolved vectors
c also proved automatic expansion of problem
c vsn 2.1.1
c
c 2.2. - solves the average field problem Aug 3
c
c 2.3 - moves the average field back to regular coordinates.
c removing the uber array.
c COMPLETED AUG 5 2010 THIS NOW BECOMES VERSION 3.0
c
c version 3.0 - expand on experimentation of different shaped and types
c of fields
c
c
c version 3.1 - using a crafted cos function for rain amounts up to about
c 2 inches, same shape used in all three domains, but
c shifted like the box
c
c added also the average of all 3 fields without change of location.
c
c version 3.2 - outputs the Jmin to a text file for plotting
c
c implicit none
c
c integer nn,ii,jj,i,j
c parameter (nn=3, ii=50, jj=50) ! n must be greater than 1
c real a (ii,jj,nn) ! arrays of ensemble variables studied
c real v (2,nn) ! displacement vectors
c real ua (ii*2,jj*2,nn) ! large vector for individual displacement comps
c so call "uber" arrays.
```

```

real output (ii*2,jj*2) ! averaged output array
real a_out (ii,jj), a_out2(ii,jj) ! collapsed uber array "output"

c   call fill_array (a,ii,jj,nn)

call fill_array (a(1,1,1),ii,jj)
call fill_array_2 (a(1,1,2),ii,jj)
call fill_array_3 (a(1,1,3),ii,jj)

call image_array (a(1,1,1),ii,jj)

call print_array (a(1,1,1),ii,jj)
pause
call print_array (a(1,1,2),ii,jj)
pause
call print_array (a(1,1,3),ii,jj)
pause

c   write (6,*) 'a1'
c   pause
call image_array (a(1,1,2),ii,jj)
c   write (6,*) 'a2'
c   pause
call image_array (a(1,1,3),ii,jj)

c   write (6,*) ' got done with filling and imaging arrays'

call center_array (a,ua,ii,jj,nn)

call image_array (ua(1,1,1),ii*2,jj*2)
c   write (6,*) 'ua1'
c   pause

call image_array (ua(1,1,2),ii*2,jj*2)
c   write (6,*) 'ua2'
c   pause

write (6,*) 'done with center_array'

call minimize_displacement (v,ua,ii,jj,nn)

write (6,*) 'done with minimize displacement', v

c   compute the displaced array average using uber arrays

call average_result (v,ua,ii,jj,nn,output)

call dumb_average (a,ii,jj,nn,a_out2)

c   convert from uber to regular array

call uncenter_array (output,a_out,ii,jj)

call print_array (a_out,ii,jj)
pause

c   print the dumb average

```

```

call print_array (a_out2,ii,jj)
pause

c   write (6,*) 'output array', output

c   do i =1,ii*2
c     do j=1,jj*2
c       if (output(i,j) .ne. 0.0) then
c         write (6,*) 'non zero location'
c         write (6,*) i,j,output(i,j)
c       endif
c     enddo
c   enddo

c   write (6,*) 'test a_out'

c   do i = 1, ii
c     do j = 1, jj
c       if (a_out(i,j) .ne. 0.0) then
c         write (6,*) i,j,a_out(i,j)
c       endif
c     enddo
c   enddo

end

subroutine dumb_average(a,ii,jj,nn,a_out2)
implicit none
integer ii,jj,nn,i,j,n
real a(ii,jj,nn),a_out2(ii,jj)

do n = 1,nn
  do i = 1,ii
    do j = 1,jj
      a_out2(i,j) = a(i,j,n)+a_out2(i,j)
    enddo
  enddo
enddo

do i = 1,ii
  do j = 1,jj
    a_out2(i,j) = a_out2(i,j)/float(nn)
  enddo
enddo

return
end

```

```

subroutine uncenter_array (output,a_out,ii,jj)
implicit none
integer ii,jj,i,j,ioff,joff
real output(ii*2,jj*2),a_out(ii,jj)

c initialize a_out

do i = 1, ii
  do j = 1, jj
    a_out (i,j) = 0.0
  enddo
enddo

ioff = ii/4
joff = jj/4

c shift uber array into normal array consider ioff and joff

do i = 1,ii
  do j = 1,jj
    a_out(i,j) = output(i+ioff,j+joff)
  enddo
enddo

return
end

```

```

subroutine fill_array (a,ii,jj)

c fills the arrays with test data

implicit none

integer ii,jj,i,j
real a(ii,jj)

c initialized array a to zero

do i = 1,ii
  do j = 1, jj
    a(i,j) = 0.0 ! initialize
  enddo

```

```

        a(i,j) = 0.0
    enddo
enddo

do i = 1,ii
    do j = 1,jj
        a(i,j) = cos( (19-i)*0.1)+ cos( (26-i)*0.3)
        a(i,j) = cos( (25-j)*0.1)+ cos( (32-j)*0.3)*a(i,j)
        if (a(i,j).le.0.0) a(i,j) = 0.0
    enddo
enddo

return
end

```

```

subroutine center_array (a,ua,ii,jj,nn)
c   centers input array a into ua which is 2x as large
implicit none
integer ii,jj,nn,i,j,n,ioff, joff
real a(ii,jj,nn), ua(ii*2,jj*2,nn)
c   clean out ua to missing value or zero value
c   ua = 0.0 ! fortran 90 construct not working in test version
do n = 1, nn
    do j = 1,jj*2
        do i = 1,ii*2
            ua(i,j,n) = 0.0
        enddo
    enddo
enddo
c
c   place the a array in the center of the ua array
c   accomplished by putting the 1,1 point at ii/4 jj/4 (one
c   quarter of the way into the large array), this should about center
c   the array. it is not critical to put it exactly anywhere EXCEPT
c   that ALL of the "centered" arrays are done the same way so vectors
c   are computed fairly. That should be the case here.
c

```

```

c      compute offsets, the exact offset is not important, just that they are all the same
c      the puts the array into the approximate center of the "uber-array"

      ioff = ii/4
      joff = jj/4

c      write (6,*) ioff,joff, 'ioff, joff'

      do n = 1,nn
        do j = 1,jj
          do i = 1,ii

              ua(i+ioff,j+joff,n) = a (i,j,n)

          enddo
        enddo
      enddo

c      simply tests for nan anywhere in the array

      do n = 1,nn
        do i = 1, ii
          do j = 1, jj
            if (ua(i,j,n) .ne. ua(i,j,n) ) then
              write (6,*) 'nan in center array routine', i,j,n
            endif
          enddo
        enddo
      enddo

      return
      end

```

```

subroutine minimize_displacement (v,ua,ii,jj,nn)

```

```

c      routine to actually do the displacement vector computation v

      implicit none

      integer ii,jj,nn,i,j,n
      real v(2,nn), ua (ii*2,jj*2,nn), vv(nn)!temp vector

      real ua_build (ii*2,jj*2) ! array to allow to grow (multiply alignment)

```

```

do i = 1, ii*2
  do j = 1, jj*2
    ua_build(i,j) = ua(i,j,1) ! initial build array
  enddo
enddo

v(1,1) = 0.0
v(2,1) = 0.0 ! initial array is a location zero

do n = 2, nn ! perform the minimization for nn fields (start with 2)
  call funct (v(1,n), ua_build, ua(1,1,n), ii,jj)
enddo ! minimization should have occurred

do n = 1, nn
  write(6,*) ' displacement vectors i,j,n', v(1,n), v(2,n), n
enddo

c   compute actual displacement vector in x

do n = 1,nn
  vv(n) = v(1,n)
enddo
call compute_vector (vv,nn)
do n = 1,n
  v(1,n) = vv(n)
enddo

c   compute actual displacement vector in y

do n = 1,nn
  vv(n) = v(2,n)
enddo
call compute_vector (vv,nn)
do n = 1,nn
  v(2,n) = vv(n)
enddo

write (6,*) 'actual vector displacement v '
do n = 1, nn
  write(6,*) ' displacement vectors i,j,n', v(1,n), v(2,n), n
enddo

c   compute total displacement for each field and report
do n = 1,nn
  write (6,*) 'total displacement for field ', n
  write (6,*) sqrt (v(1,n)**2+v(2,n)**2), n
enddo

c   apply normalized displacement vectors to compute the average
c   ensemble field

c   report normalized displacement vectors

return
end

```

```

subroutine funct (v2, ua1, ua2, ii, jj)

c   started working on this section June 7, 2010, Monday
c   Author: Daniel Birkenheuer
c
c   subroutine 'funct' is the minimization routine for 2 fields.
c

implicit none

integer ii,jj,n

data n /1/

real v2(2) ! displacement vector
real ua1 (ii*2,jj*2), ua2(ii*2,jj*2) ! large vector for
c                                     individual displacement comps
c                                     so call "uber" arrays.

integer i,j
integer is, js ! ishift and jshift indexes

real j_func, j_func_inv (ii*2,jj*2)
real j_func_min, v_min(2), j_func_inv_save(ii*2,jj*2)

c   initialize

j_func_min = 1000000000000.
v_min(1) = 0.0
v_min(2) = 0.0

do i = 1, ii*2
  do j = 1, jj*2
    j_func_inv (i,j) = 0.0
    j_func_inv_save(i,j) = 0.0
    if(ua1(i,j) .ne. ua1(i,j)) then
      write (6,*) 'ua1 bad ', i,j
      stop
    endif
    if (ua2(i,j) .ne. ua2(i,j)) then
      write (6,*) 'ua2 bad ',i,j
      stop
    endif
  enddo
enddo

c   compute explicit ishift and jshift from the v2 array minimization

do is = -ii/2, ii/2
  do js = -jj/2, jj/2

```

```

c      apply the ishift and jshifts to the array ua and product the result

      do j = 1,jj*2
        do i = 1,ii*2
          if ( ((i+is.le.ii*2) .and. (i+is.ge.1))
c           .and.
c           ((j+js.le.jj*2) .and. (j+js.ge.1)))
c           then

            j_func_inv(i,j) = ua1(i,j) * ua2(i+is,j+js)

            if (j_func_inv(i,j) .ne. 0.0) then
c              write (6,*) j_func_inv(i,j), ua1(i,j), ua2(i+is
c              ,j+js)
              continue
            endif

            if(j_func_inv(i,j) .ne. j_func_inv(i,j)) then ! nan
              write (6,*) i,j,'nan', ua1(i,j), ua2(i+is,j+js)
              pause
            endif

          else
            continue
c          write (6,*) is, js, 'bounds 2'
          endif

        enddo ! j
      enddo ! i

c      now j_func_inv is the product of all nn values at i,j, now sum and
c      inverse the problem
c

      j_func = 0.0

      do i = 1,ii*2
        do j = 1,jj*2
          j_func = j_func + j_func_inv(i,j)
        enddo
      enddo

c      write (6,*) j_func , 'first before inverse'

      if (j_func .eq. 0.0) then

        j_func = 400. ! some large number
        continue ! avoid divide by zero
      else

        j_func = 1./j_func

        if (j_func .ne. j_func) then

          write (6,*) j_func, 'jfunc nan', is,js
          stop
        endif
      endif

```

```

        endif

c     make minimization decision
        if (j_func_min .lt. 500) then
            write (44,*) n,j_func, j_func_min
            n=n+1
        endif

        if( j_func .lt. j_func_min ) then
            j_func_min = j_func
            v_min(1) = float (is)
            v_min(2) = float (js)
            do i = 1,ii*2
                do j = 1, jj*2
                    j_func_inv_save(i,j) = j_func_inv(i,j)
                enddo
            enddo

            write (6,*) 'new min found' , j_func,is,js

        endif

        enddo ! js
    enddo ! is

c     minimization state should be known for vector 2
c     return values

    do i = 1, ii*2
        do j = 1, jj*2
            ual(i,j) = j_func_inv_save(i,j)
        enddo
    enddo

    v2(1) = v_min(1)
    v2(2) = v_min(2)

    write (6,*) 'should have an answer', v_min
    pause

    return
end

```

```

subroutine image_array (a,ii,jj)

```

```

c     temporary subruoutine to image the array input this can be devised
c     to better diagnose algorithm function and to also (eventually) provide
c     output that will be of publication quality.

```

```

implicit none

```

```

integer ii,jj, i,j
real a(ii,jj)

do j = 1, jj

  write (6,*) (a(i,j), i = 1, ii)
  do i = 1,ii
    if(a(i,j) .ne. a(i,j) ) then ! nan
      write (6,*) 'nan found image array test' , i,j
    endif
  enddo

enddo

return
end

```

```

subroutine print_array(a,ii,jj)
implicit none
integer ii,jj,i,j
real a (ii,jj)

open (45)
do j = 1, jj
  write (45,23) (a(i,j), i=1,ii)
  format (50(1x,f5.3))
enddo

close (45)

return
end

```

23

c subroutine `compute_vector` (`v`, `nn`) ! take in the offset vectors and returns the true displacement vector

c note that here `v` is input and output variable.

```

implicit none

integer nn
real v(nn)

real ri ! relative movement in i
real sum

integer n

```

```

sum = 0.0
ri = 0.0

do n = 1,nn
  sum = v(n) + sum
enddo
c  sum = -sum ! change sign of sum

ri = sum / float (nn) ! this gives the relative movement that must
c  be applied to all v values to convert them from offsets to vector changes

do n = 1,nn
  if(n .eq. 1) then
    v(n) = v(n)+ri
  else
    v(n) = -(v(n)-ri)
  endif
enddo
return

end

```

```

subroutine average_result (v,ua,ii,jj,nn,output)
implicit none
integer ii,jj,nn
real v(2,nn),ua(ii*2,jj*2,nn),output(ii*2,jj*2)

integer i,j,n

write (6,*) 'entering average_result module'

c  initialize output array

do i = 1,ii*2
  do j = 1,jj*2
    output(i,j) = 0.0
  enddo
enddo

c  sum output array before averaging process relocating as we go

do n =1,nn
  do i=1,ii*2
    do j=1,jj*2
      output(i,j) = output(i,j) +
c      ua(i-nint(v(1,n)),j-nint(v(2,n)),n)
    enddo
  enddo
enddo

```

```
c    divide array sums by number of elements for average value
```

```
    do i=1,ii*2
      do j=1,jj*2
        output(i,j) = output(i,j)/float(nn)
      enddo
    enddo
```

```
c    move and average uber arrays
```

```
    return
  end
```