AUTOMATICALLY REGULARIZED NONNEGATIVE SOLUTONS FOR ILL-CONDITIONED LINEAR SYSTEMS

RONDALL E. JONES
Ktech Corporation, Albuquerque, NM 87185
rejones@sandia.gov
2006

Abstract. When performing Tikhonov or similar types of regularization of ill-conditioned linear systems, a free parameter λ must be determined. Common techniques such as use of L-curves are somewhat daunting for non-experts, and automated methods of choosing λ have not been widely applied. A given choice of λ implies a corresponding residual for the resulting regularized system. An appropriate residual is easily determined from the error level in the right hand side if that is known. This paper points out that a useful estimate for the error in the right hand side can be determined automatically by projecting the rows of the matrix onto the "usable" rows of an orthogonalized version of the system. Thus, the problem of picking λ is transformed into one of picking a "usable rank" at which to split the orthogonalized system. A widely applicable heuristic is presented which picks this usable rank. Realistic applications of such a general-purpose algorithm will require support for a variety of constraints. This paper presents straightforward methods to add a non-negativity constraint, equality constraints, and inequality constraints to the automatically regularized solution process. A software implementation is available from the author.

Key words. Linear algebraic systems, least squares, ill-posed problems, automatic regularization, constraints

1. Introduction. In this paper we present a method for producing an automatically regularized solution, with various constraints, for a wide class of ill-conditioned and/or rank-deficient linear systems, $Ax \approx b$, where A is $m \times n$ with m and n arbitrary.

Regularization is widely used in a variety of contexts, and especially in the field of inverse problems. Common regularization methods such as that of [Tikhonov] require choice of a free parameter, λ, which is typically a task for an expert in the field. Thus regularization is often difficult for non-experts or in automated contexts in which software would need to adapt the value of λ to the given data. A few automatic regularization schemes exist [Wahba], [Jones], [Hansen], [O'Leary] but automatic regularization schemes have not routinely been used for a variety of reasons, such as unreliability of early methods, and the lack of methods which guarantee a non-negative solution. Non-negativity is essential in many contexts, where negative values are not physically meaningful. Sometimes other constraints are needed.

The earliest proposed automatic regularization method was that of [Wahba], which never received wide use, perhaps because its reliability may be less than what users expect. The method of [Jones] was also an early approach. This method estimates the error in the right hand side automatically, by splitting the system of equations into usable and unusable rows based on a Picard-like condition. Then a projection scheme is used to estimate the right hand side error. With this error estimate in hand a variety of regularization methods can then be automated. This method was not widely published originally, and was also not as reliable as needed except for highly ill-conditioned problems. This method has been updated and improved for this paper. A more recent automatic method is Hansen's automated L-curve analysis, which is available primarily

as a Matlab plug-in [Regtools]. Hansen's method sets a higher standard for reliability than Wahba's method; a level apparently matched by the method presented here. Our method also provides an explicit threshold for a linear system to be considered ill-conditioned, which is a useful feature in practice. An automatic regularization method by O'Leary [O'Leary] was not evaluated for this paper.

None of the previously available automatic regularization methods provides realistic constraints on the solution, such as non-negativity, even though non-negativity methods are well known, at least for well-conditioned problems. For example, see [Hanson].

The goal of this paper and the software being offered is to advance the use of automated regularization. The method we present appears to be relatively reliable, is implemented in C++ in a form callable from other languages, includes a range of constraints, beginning with non-negativity, and is free to students and academic researchers.

In Section 2 below we present some of the necessary background information for our methods. In Section 3 we discuss the "usable rank" concept, which includes consideration of the right hand side in the rank. In Section 4 we present the second key point: how to accurately estimate the right hand side error, σ, given a chosen usable rank. In Section 5 we briefly discuss determination of the Tikhonov regularization parameter using the estimated σ. In Section 6 we present some background on implementing non-negativity, and suggest a useful method. In Section 7 we briefly present what is required to add general equality and inequality constraints. An example is shown in Section 8.

2. Background Consider the system of equations

(2.1) $Ax \approx b$

where the shape and size of A are arbitrary. In the following work we assume that the user of this method has pre-scaled this system of equations as well as feasible to make the right-hand-side error estimates all have the same (unknown) standard deviation of σ. Failure to scale the problem well does not prevent use of our method, but it will effectively create an implicitly heavier weight for rows which have larger than average error and will skew the estimated value of σ away from the optimal value. And at some point large unevenness in the errors in the right hand side may degrade the quality of solution to an unusable point. We have not characterized this behavior well. The system can then be considered to have the form

(2.2) $Ax \approx b = b_0 + e$

where A is $m \times n$, with any shape for m and n, b_0 is the vector of unknown "true" values of the right hand side, and the e is a vector of random deviates from a distribution with mean zero and standard deviation σ.

We will solve such systems using a Singular Value Decomposition, or SVD. Available software for the SVD usually processes only square or overdetermined systems. Thus if $m>=n$, then

(2.3) $A = USV^T$

where U is $m \times n$ with orthonormal columns, S is $n \times n$ diagonal with diagonal elements s_i, and V is $n \times n$ with orthonormal rows and columns. (That is, $V^TV = VV^T = I.$) If A is

underdetermined, one simply applies the SVD to A^T, reversing the names for U and V and transposing the result. Then U is $m \times m$ with orthonormal rows and columns, S is $m \times m$ diagonal, and V is $m \times n$ with orthonormal rows. In both cases, then, setting $p = min(m,n)$, we have U as $m \times p$, S as $p \times p$, and V as $p \times n$, and the shape of A can be considered to be arbitrary.

Substituting the SVD for A into (2.1) and multiplying by the transpose of U we have

(2.4) $SV^Tx \approx U^Tb$

which we re-label as

(2.5) $S\hat{x} \approx \beta$

where x and $\hat{x}$ have the same norm, and b and β have the same norm. The solution to this is, formally,

(2.6) $\hat{x} = S^{-1}\beta$

except of course that S may contain zeros. The pseudo-inverse solution is then given by

(2.7) $\hat{x} = S^{+}\beta$

where S^+ is diagonal with diagonal elements equal to $1/s_i$ except when s_i is zero, when zero is substituted. The norm of x is given by

(2.8) $$\|x\|_2^2 = \sum_{i=1}^{r} (\beta_i / s_i)^2$$

where r is the number of non-zero diagonal elements of S. The magnitude of the quotients in (2.8) are important in the following development, so we define

(2.9) $t_i = |\beta_i / s_i|$

If the system (2.1) is simply rank deficient, without being otherwise ill-conditioned, (2.7) may provide a perfectly acceptable solution. This situation does occur, such as when there is a simple linear dependency between certain rows. But more generally the system is ill-conditioned after removing the zero singular values. Ill-conditioning causes the series of t_i values to behave like a diverging series: the successive values initially appear to be well behaved and even to be converging to zero. But at some point they reverse direction and grow in an unbounded manner, ruining the solution entirely. This behavior is contrary to the expected behavior, which is that the t_i should generally decline as a function of i. This requirement is sometimes called the discrete Picard condition. The first approach to addressing this behavior is sometimes to perform a truncated SVD solution. To do this one simply zeroes more of the t_i than those zeroed by the pseudo-inverse. The choice of how many t_i to zero is problematic, and is typically done by plotting the resulting solution for a range of possible "ranks", and choosing whichever one likes. This is obviously not a process which we can automate. A more sophisticated way to tame the ill-conditioning is to regularize the system by augmenting (2.1) to the form

(2.10) $$\begin{bmatrix} A \\ \lambda I \end{bmatrix} x \approx \begin{bmatrix} b \\ 0 \end{bmatrix}$$

This technique is associated with the name Tikhonov or the term "ridge regression" depending on the context. The larger the value of λ, the more the latter t_i are reduced.

The problem with this method is what value should be used for λ. This can be answered in a variety of ways, including plotting the solution for various values of λ, adjusting λ until a maximum tolerable residual is reached, or plotting an "L curve" which displays the norm of $x = x(\lambda)$ for a range of λ values, versus the corresponding residual (2.11).

(2.11) $\|Ax(\lambda) - b\|_2$

Using the L curve one hopes to find an optimum tradeoff point between decreasing norm of x versus increasing residual. One heuristic is to pick the point on the curve closest to the origin. If this choice is made in software, then an automatic regularization process results. An example implementation is in [Regtools]. This is an alternative to the method we develop in this paper, and could be used instead of our method by incorporating the constraint methods we present. We prefer to present a version of the method of [Jones 1985] due to the direct manner in which the regularization is based on the error present in the system. Our method directly estimates the unknown standard deviation σ.

When we have an estimate of the value of σ, then an appropriate value for λ can be computed using the "discrepancy" principle [Hansen]. That is, one adjusts λ until the average, or root-mean-square change to the b_i is equal to σ. That is, until

(2.12) $\|Ax(\lambda) - b\|_2^2 = n\sigma^2$

Where necessary, we will refer to the *i*-th column of a matrix *A* as A_i and the *i*-th row of a matrix *A* as A^i. We denote the dot product of two equal length one-dimensional structures *a* and *b* as <*a,b*> whether *a* and *b* are actually vectors, rows, columns, or combinations, as will be clear in context.

3. Determining the Usable Rank When the system (2.1) is well-conditioned the t_i coefficients in (2.9) should satisfy the so-called "discrete Picard condition" [Hansen], which says that the t_i should trend downward, or that the β_i should decrease (in magnitude) faster than the s_i. But as mentioned in the introduction, the typical behavior when the system is ill-conditioned is that the summation in (2.8) behaves much like a diverging series: the magnitudes of the successive t_i may initially drop off nicely as expected by the Picard condition, or they may fall initially then wander in a moderate range briefly. But the t_i eventually begin to grow in an unbounded manner. Were it not for the removal of zero singular values some t_i would be infinite. The graph below shows a semi-log plot of the t_i for an inverse heat problem with a matrix of size 15 × 51. The t_i appear to be trending downward slightly through index 6, then clearly reverse course and grow exponentially. The initial behavior may be a slight drop, as here, or a much stronger drop, or in same cases no noticeable drop. But the exponential growth after some point is typical of ill-conditioned systems, and especially so of inverse problems.

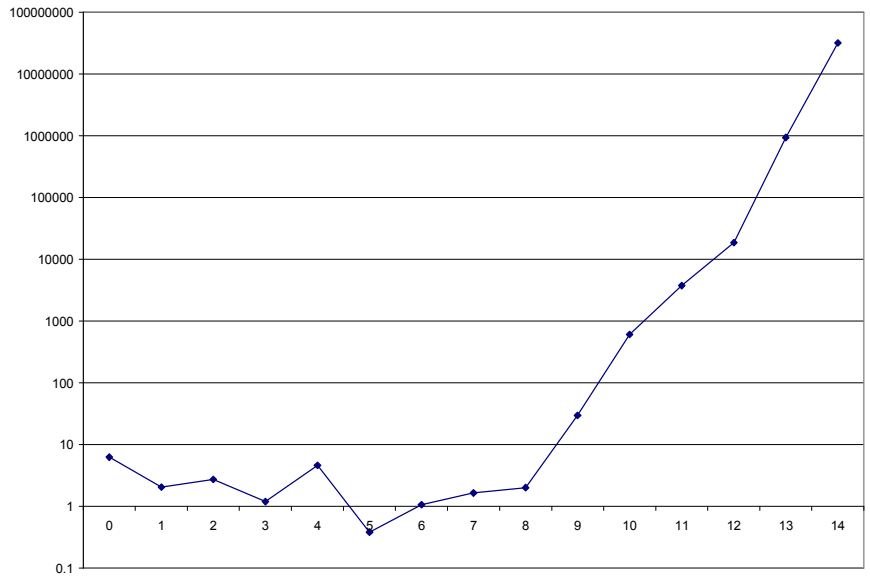

Figure 3.1. Typical behavior of the row contributions, t_i

What we want to do here, given a linear system, is to reliably find a row in (2.4) beyond which the discrete Picard condition is clearly violated. The portion of the system up to and including the final acceptable row will be called the "usable" portion, and to have a "usable rank" equal to the number of rows in the acceptable portion. It is not especially important that the usable rank be as small as possible… it is better to include some extra rows in the usable portion if there is doubt as to the best choice. But most of the rows in the range of the exponential growth of the t_i need to be excluded. We will assume that the usable part of the matrix A then represents for practical purposes a basis for the span of the non-null space of the A, and that the β_i beyond the usable rank represent for practical purposes images of the error in the right hand side. Indeed, a first estimate of σ can be obtained by simply computing the root-mean-square of the β_i for rows beyond the usable rank. But there are practical problems with that approach, such as the fact that the number of such error samples can be very small… possibly just one. A much better approach is available, as explained in the next section.

For many problems it might actually be good enough to set the usable rank equal to the index of the smallest t_i. If the t_i decline more or less uniformly to that point then the discrete Picard condition is satisfied for that subset of the linear system, and not satisfied for the remainder, which is our goal. But that strategy is not nearly adequate for a general purpose algorithm which is expected to produce a good break point in a wide variety of contexts. Problems with that simple approach include the fact that sometimes a particular value of β_i may be nearly zero, but not indicate a meaningful minimum; that many problems exhibit highly oscillatory behavior within the usable portion, and there is no significant meaning to any of several near-minimum values of β_i; that there may be a well defined minimum point, but the rise after it may be so minor as to not warrant treating the problem as ill-conditioned; etc. Sometimes the Picard condition is tested

using some kind of smoothing formula centered on each index i. But formulas centered about each index necessarily include an odd number of values, and a sum of an odd number of terms will nearly replicate any problematic oscillation which is present. And, when the smoothing formula involves a geometric mean, as is typical, the smoothed result is subject to a false minimum when one term is near zero. All these difficulties can be resolved by minor adjustments: use an even number of terms in the smoothing average, and use an arithmetic mean rather than a geometric mean to avoid pathologies associated with insignificant near-zero term. A minor complication is that the smoothed terms no longer have an obvious one-to-one identification with the original terms. One obvious approach, then, is just to use the norm (or its square, which is equivalent here) of successive segments of the vector $\hat{x}$. That is,

$$(3.1) \qquad a_j = \left\| x(j:j+3) \right\|_2^2 = \sum_{i=j}^{j+3} (\beta_i / s_i)^2$$

We can find a usable rank more reliably by examing the sequence of these segment norms, a_j , with some care. We have found the following to work well in an extensive set of tests.

(3.2)
1. set *lo* = value of j at which a_j is minimal
2. set *hi* = value of first $j >$ lo, at which $a_j > min(15*a_{lo}\ ,\ 1.1\ *\ a_0)$
3. if no such value exists, then the problem does not require regularization,
   so set *ur=min(m,n)* and exit
4. set *hi* = value of first $j >$ lo, at which $a_j > min(3*a_{lo}\ ,\ 1.1\ *\ a_0)$
5. set *ur* = value of j within *hi* <= j <= *hi*+3 at which t_j is maximal
6. for j=*ur* down to j=*lo*+*1*…
   if $t_{j-1} < t_j$ then set *ur* = *j-1*; otherwise exit

In other words, we first seek the minimum of the (squared) segment norms. Then, we check to see if the segment norms ever grow significantly above that level. Note that we dealing with *squared* norms, and only ask for this measure to increase by a factor of 15 (which is arbitrary, but chosen through extensive tests). So not a great deal of growth is required, but this test avoids our applying regularization in parametric contexts in which some – or many -- of the systems exhibit no ill conditioning. If we decide to regularize, we look for the first point at which the squared segment norms rise by a factor of just 3. Note that we have an alternate limit of 1.1 times the first segment norm. This is to handle certain pathological cases in which the t_i never decrease significantly. We take the initial usable rank to be the peak of the t_i in that segment. Then, in the last step we allow the usable rank to walk down the exponential growth curve (to the left) as long as the t_i drop monotonically. Steps 5 and 6 are probably is no importance in most practical inverse problems. But they were found to be valuable for very small or very low rank problems.

The usable rank, as just defined, will typically be larger than the best choice of rank to be used in a truncated SVD solution. This is as planned, as we want to be sure to include all the usable data, and we are going to regularize the usable portion of the system, not just use a truncated SVD. If the usable rank is maximal (that is, equal to *min(m,n)*) then the system is not sufficiently ill-conditioned for our algorithm to apply. This can often be

corrected by added *more* equations (if available) to make the system more ill-conditioned. This is counter-intuitive, but highly effective, as the extra equations make the latter t_i rise higher, so the usable rank is easier to detect.

4. Estimation of the Right Hand Side Error, σ. Now that we have a determined the usable rank, *ur* , we need to extract an estimate of the standard deviation of the right hand side, σ. We split (2.4) into "usable" and non-usable parts. The non-usable rows are the rows beyond row *ur* in the following system:

(4.1) $Cx = SV^T x \approx U^T b = U^T(b_0+e) = U^T b_0 + U^T e = \hat{b}_0 + \hat{e} = d$

Note that since U^T is orthonormal, the norm of $\hat{e}$ is the same as the norm of *e*. Were the system completely accurate, *e* would be a vector of zeroes, and the latter components of $\hat{b}_0$ would be small compared to the already small corresponding singular values, s_i, as the Picard condition requires. At rows below *ur* the right hand side values in (4.1) are dominated by the elements of $\hat{e}$, and so are almost purely samples from a random distribution with mean σ. Thus σ could be estimated as the root-mean-square of those right-hand-side values. That is,

(4.2) $$\sigma = \sqrt{\frac{1}{m-u}\sum_{i=u+1}^{i=m} d_i^2}$$

We note that this estimate of σ is biased very slightly to the high side due to the residual presence of the $\hat{b}_0$ term in (4.1). But the problem with this approach to estimating σ is that there may be a very small number – even just 1 – of samples in the summation in (4.2). That is an undesirable situation, and we can do better. We note that the "usable" rows of A in the orthogonalized system (4.1) constitute an approximate basis for the non-null space of the matrix *A*. The remaining rows can be taken as being in the null space of *A*. If we take each individual row of the original problem (2.1) and subtract from it its projection onto each of the usable rows of the orthogonalized system (and carry along the right hand side) we will be left with a (row) vector in *A*'s null space, and a right hand side value that should be zero were it not for error. That is, we compute

(4.3)
```
for i=1,m
        r_i = b_i
        for k=1 to u
                dot = <A^k,C^k>/<C^k,C^k>
                A^k = A^k – dot*C^k
                r_i = r_i – dot*b_k
```

Each resulting r_i is a value which would be zero except for presence of error, and each such value is a biased estimate of σ. It is biased primarily because the matrix *C* was computed from *A*, rather than being statistically independent of *A*. To compute an unbiased estimate of σ from the r_i we need to adjust the formula by the number of lost degrees of freedom, which results in the formula:

(4.4) $$\sigma = \sqrt{\frac{1}{m-u}\sum_{i=1}^{i=m} r_i^2}$$

which differs from (4.2) in that the sum always has a full length of *m*.

We should note that the development leading to formula (4.4) was for theoretical understanding. It is not necessary to do the actual computation in that fashion. The computation of *r* can be represented compactly as

(4.5) $r = b - A(VS_{ur}^{+}U^{T}b)$

where S_{ur}^{+} is the diagonal matrix of reciprocals of the first *ur* singular values, then zeroes. That is, *r* is the residual vector for the TSVD with *ur* retained singular values. Then σ is computed as in (4.4).

We have tested this approach on a range of problems, and find the resulting estimate of σ is often stunningly accurate. And it is almost always at least useful.

5. Determination of the Tikhonov Regularization Parameter λ. With a good estimate of σ on hand we are then free to proceed with any of a variety of regularization methods which are based on the discrepancy principle (2.12). We will now briefly discuss implementation of Tikhonov regularization using the discrepancy principle as it is probably the most widely used regularization, and because it can be done especially efficiently. [NR] includes a discussion of common variations of (2.10) using weighing matrices other than *I*.

We must adjust λ in (2.11) until the residual has the same norm as a vector with all entries equal to σ. Since the rows after the usable rank, *ur,* already played their role by determining σ, we can perform this regularization using only the usable rows. This reduction of the problem is, however, optional.

Note that one does not recompute the SVD of the left hand side matrix in (2.11) for each new λ. Rather, one proceeds as follows, substituting the SVD for *A*:

(5.1) $$\begin{bmatrix} USV^{T} \\ \lambda I \end{bmatrix} x \approx \begin{bmatrix} b \\ 0 \end{bmatrix}$$

Multiplying through by the transpose of the left hand side gives

(5.2) $(VSU^{T}USV^{T} + \lambda^{2}I)x = VSU^{T}b$

(5.3) $(VS^{2}V^{T} + \lambda^{2}VV^{T})x = VSU^{T}b$

(5.4) $V(S^{2} + \lambda^{2}I)V^{T}x = VSU^{T}b$

(5.5) $(S^{2} + \lambda^{2}I)V^{T}x = SU^{T}b$

(5.6) $\hat{x} = (S^{2} + \lambda^{2}I)^{-1}S\hat{b}$

So the addition of the conditioning equations, $\lambda Ix = 0$, replaces S^{+} in (2.7) with $(S^{2} + \lambda^{2}I)^{-1}S$, and there is no need to recomputed the SVD.

The search for the correct value of λ can be done in several ways. A sophisticated approach would be to use Newton's method to solve (2.12) as a general non-linear system. (Regtools does this.) The correct value of λ can also be found with a

straightforward bisection search, using a lower of zero. The upper limit can be set initially to a value such as s_1 and then increased geometrically until it is large enough to cause a residual of at least $n\ \sigma^2$ in (2.12). Bisection should continue until the residual is in a small interval about $n\ \sigma^2$. Note that a solution of x=0 is a feasible if

(5.8) $\|b\|_2^2 <= n\ \sigma^2$

This feasibility should be checked before starting the search, as a solution of zero can be immediately returned.

This method works well on the set of Fredholm integral equations problems we use for testing, plus pathological cases designed to stress the software. It also performs well on all test problems in Hansen's Regtools package [Regtools], using a problem size of 20 by 20, and on most reasonable problem sizes.

As a practical matter, a zero solution may be an undesirable or surprising result to return from an automatic algorithm. If so, one can put an upper limit how much the solution is "smoothed". For example, the upper limit for λ can be limited. If the upper end of the search range for λ is limited to $\lambda < 0.33\ s_1$ then reduction of the contribution to the solution from the first singular value will be limited to 10%.

It is perhaps worth noting that this entire automatic regularization process can be performed with slightly inferior results without the cost of the SVD. A row-wise Gram-Schmidt orthogonalization can be used instead, but several algorithmic adjustments are required, including designing an appropriate row pivoting scheme. And the resulting algorithm does not produce solutions as close to the true solution as the SVD method does. But in cases where execution time is critical it might be worth considering.

6. Non-negativity Constraints. There are a variety of ways to adjust a system of equations so that the resulting solution is nonnegative. The difficulty is to accomplish this constraint without unduly increasing the residual error or zeroing an unnecessarily large number of the x_j. One approach is to iteratively set to zero values of x_j chosen by some methodology. A specific x_j can be set to zero by simply setting the corresponding column of A to zero. This approach tends to produce a small increase in residual error over the unconstrained least squares result. The choice of which sequence of x_j to zero becomes the question. We are aware of no specific guidance in the literature for how to proceed with selecting which columns of A to zero when A is ill-conditioned, though methods such as [Hanson] seem appropriate. [Hanson] initially zeroes *all* columns, then one-by-one reintroduces columns back into the problem for which it can be determined that the corresponding x_j remain non-negative. Unfortunately, this introduction may cause other entries in x to go negative, so some backtracking is required.

It is of some concern that setting columns of A to zero essentially reduces the domain of the solution, which may not be desirable. An alternate approach might be to make adjustments to the right hand side. For example, instead of zeroing column j of A to make x_j zero, b could be made orthogonal to column j of A by subtracting away its projection onto column j. But such methods provide no direct or indirect control over the

residual error, and typically produce very large residuals versus zeroing appropriate columns of A.

We prefer a simple implementation of the non-negative constraint if possible, for a variety of reasons, including the extensibility of our algorithms to more complex constraints, as discussed below. Our experience with a number of variations of such column zeroing algorithms indicates that all reasonable approaches tend to zero nearly or exactly the same set of columns when applied to the inverse problems we use for testing. So we suggest a simple approach: simply zero the column of A corresponding to the most negative x_j, and iterate until the solution is non-negative. At each iteration the solution to the newly modified system must be computed. There seem to be three reasonable ways to proceed when solving each newly modified system:

1. start over completely, using the algorithms described above to compute a split, a value for σ, and a value for λ.
2. assume that the previous value for σ is still valid, and just compute a new value for λ.
3. simply reuse λ.

Our suggestion is to take the last option: simply reuse the initially determined value of λ for all subsequent solutions. One reason for this approach is that the there are theoretical questions as to the validity of our automated process when columns of A have been zeroed. And indeed we have experienced fairly frequent failures in recomputing the split and/or σ after many columns have been zeroed. Even starting with a fixed σ at each iteration has frequent difficulties. Due to these experiences, and the poorly understood theory behind applying our algorithms after columns have been zeroed, we recommend the third option: after each column is zeroed, re-solve regularizing with the initially determined value of λ. The result is a robust, powerful, yet seemingly near-optimal method, which definitely converges in a limited number of steps.

7. Equality and General Inequality Constraints. Sometimes the user of such algorithms has extra constraints. For example, if the solution is known to start at zero, then there is an equality of constraint of $x_1 = 0$. Or, the user may know about the presence of otherwise un-modeled symmetry which might be expressed as $x_i = x_{n-i}$ for a range of i. (Of course, when feasible such constraints should be directly modeled.) Or, there may be information about bounds on some of the variables or other general inequality constraints. We denote such equality constraints as

(7.1) $Ex = f$, where E is of size $me \times n$.

and denote inequality constraints as

(7.2) $Gx >= h$, where G is of size $mg \times n$

Equality constraints can be handled in a standard fashion, by subtracting from each row of (2.1) its projection onto all rows of (7.1), or more precisely, onto all rows of a row-orthogonalized version of that system, $\hat{E}x = \hat{f}$. We use row-wise Gram-Schmidt orthogonalization [Golub] to do this orthogonalization. The reduced system is then re-solved by using the chosen value of λ, and the solution is added to the solution to (7.1), which is simply $\hat{E}^T \hat{f}$. Inequality constraints can be handled in a manner consistent with our chosen method of achieving non-negativity. That is, by iteratively selecting the row

of (7.2) which is most violated; moving it to the system of equality constraints; and re-solving. This is repeated until all remaining inequality constraints happen to be satisfied or all inequality constraints are so moved.

Note that we now have two methods to enforce non-negativity: the method of Section 6, or using (7.2) to enforce $x_i > 0$ for all i. Fortunately, these two methods are precisely equivalent.

The nature of equality constraints is typically such that the system (7.1) is underdetermined but otherwise well conditioned. However, since the inequality constraints are implemented by selecting some to become equality constraints, the resulting expanded set of equality constraints may be inconsistent, ill-conditioned, or singular. So, as we row-orthogonalize the expanded equality constraints we do row pivoting based on the largest remaining row norm, and discard rows whose norm is reduced dramatically as evidently redundant or conflicting.

8. Example. A companion paper in this conference's proceedings [Daun] illustrates the performance of our method on an important real problem. We present one example here for illustration. The problem was a heat-equation-like problem with a kernel of $e^{-(x-t)(x-t)}$. Discretization was chosen to result in a matrix A of size 11 by 51, and the intended exact solution was chosen with the tent shape shown in color (or grey) in (8.1). The right hand side was computed as $b=Ax$ to avoid any modeling error, and then Gaussian noise with $\sigma = 0.0003$ was added to the right hand side values. The automatically regularized non-negative solution proposed in this paper is shown in (8.1). This is an exceptionally good solution for this difficult problem. The peak is particularly close to the ideal peak, and the region of zero solution is nearly perfectly reproduced. The estimated value of σ was 0.00035. The estimated value of σ will typically be less accurate, as in this example, when m is small.

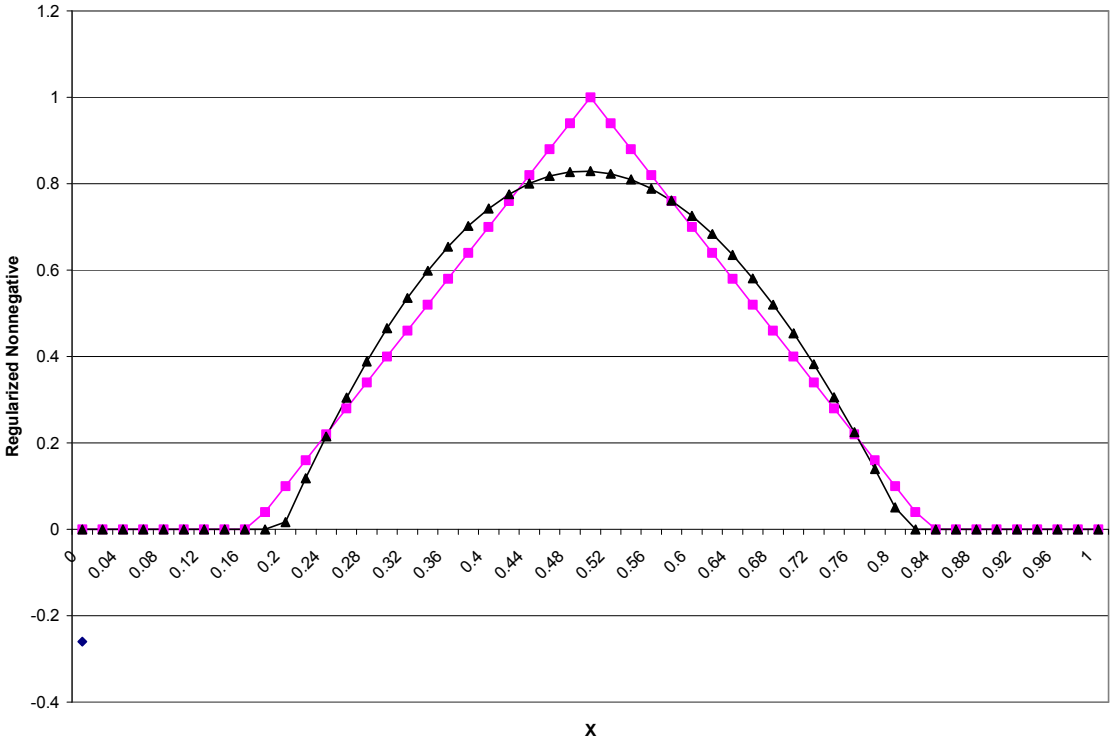

Figure 8.1

9. Summary. We have presented a method to estimate the right hand side error in a linear system when the "usable" rank is less than the number of rows. Having this estimate in hand then enables application of a variety of regularization methods which use the discrepancy principle. We focused on Tikhonov regularization due to its widespread use and efficient implementation. We presented a simple but effective approach to achieving non-negativity, and extended that method to general equality and inequality constraints. Implementations of the algorithm for C++, C, and Matlab are available from the author at www.rejonesconsulting.com or by email at rejones7@msn.com. These implementations are built on a public domain SVD algorithm [TNT].

10. Acknowledgements. The development of the original version of our automatic error estimation method was mentored by Cleve Moler. That work was supported in part by Sandia National Laboratories. The motivation to restart this work and incorporate constraints was inspired by interactions at the 2003 IPES. Kyle Daun has been a helpful critic and early adopter of this method. Per Christian Hansen provided a very helpful evaluation of an early implementation of the current methods, and his Regtools package provided useful test problems. My current employer, Ktech, provided access to Matlab and allowed me to develop the offered software on my time as my product. My wife, Mary Esther, has tolerated and even encouraged my obsession with developing this

method and this software at home when I should have been attending to many other things.

11. References

[Wahba] G. Golub, M. Heath, and G. Wahba, *Generalized cross-validation as a method for choosing a good ridge regression parameter*, Technometrics, 21 (1979) pp. 215-223

[Jones] Rondall E. Jones, *Solving Linear Algebraic Systems Arising in the Solution of Integral Equations of the First Kind*, dissertation, University of New Mexico, 1985

[Hansen] P. C. Hansen, *Rank-Deficient and Ill-Posed Problems: Numerical Aspects of Linear Inversion, SIAM, Philadelphia, 1997*

[O'Leary] Dianne P. O'Leary, *Near-Optimal Parameters for Tikhonov and Other Regularization Methods,* SIAM J. Sci. Comput., 23 (2001) pp. 1161-1171

[Regtools] P. C. Hansen, *Regularization Tools: A Matlab package for analysis and solution of discrete ill-posed problems,* Numer. Algorithms, 6 (1994); available online from http://www.mathworks.com/matlabcentral by searching for "regtools"

[Hanson] Richard J. Hanson and Karen H. Haskell, *Algorithm 587: Two Algorithms for the Linearly Constrained Least Squares Problem,* ACM Transactions on Mathematical Software (TOMS), Volume 8 Issue 3, ACM Press, September 1982

[Golub] – Gene H. Golub and Charles F. Van Loan, *Matrix Computations,* John Hopkins Studies in Mathematical Sciences, 1996

[Daun] Kyle Daun and Kevin Thomson, *"Abel Inversion for Flame Analysis",* in Proceedings of 14th Inverse Problems in Engineering conference, 2006

[NR] William H. Press and others, *Numerical Recipes in C++,* Chapter 18, Cambridge University Press, 2002

[Matlab] The MathWorks, *Matlab & Simulink*, release 14, http://www.mathworks.com

[TNT] National Institute of Standards and Technology, *Template Numerical Toolkit*, available online at http://math.nist.gov/tnt/download.html