EngSci 721

Inverse Problems and Learning From Data

Oliver Maclaren (oliver.maclaren@auckland.ac.nz)

1. *Basic concepts* [5 lectures + 1 Tutorial]

   Forward vs inverse problems. Well-posed vs ill-posed problems. Algebra and calculus of inverse problems (left and right inverses, generalised and pseudo inverses, resolution operators, matrix calculus). Representing higher dimensional problems (image data etc).

2. *Instability and regularisation in linear and nonlinear problems* [6 lectures + 1 Tutorial]

   Instability and related issues for generalised inverses. Introduction to regularisation and trade-offs. Tikhonov regularisation. Higher-order Tikhonov regularisation. Sparsity and regularisation using different norms. Truncated singular value decomposition. Iterative regularisation, including stochastic/mini-batch gradient descent.

3. *Further topics* [3 lectures + 1 Tutorial]

   Regularisation parameter choice, including statistical and machine learning views of regularisation. Confidence sets for linear and nonlinear models. Physics-informed machine learning and neural networks.

Module overview

Inverse Problems and Learning From Data (*Oliver Maclaren*)
[~14 lectures/3 tutorials]

Lecture 10: Iterative regularisation

Topics:

- Features of nonlinear and/or large linear inverse problems
  - Benefits of/need for iterative methods
- Brief overview of iterative minimisation for regularised problems
- Iteration plus early stopping as a regularisation method
  - Semi-convergence and singular limits

Eng Sci 721 : Lecture 10.

Iterative methods

For nonlinear &/or large, need:

- Iterative minimisation algorithms
- Iterative regularisation
  ↳ main focus.

Motivation I. Solving regularised equations.

- Nonlinear problems : $\min \|F(x)-y\|^2 + \lambda \|x\|^2$
  ↳ nonlinear optimisation → Iterative methods ✓

- Large linear systems → large sparse systems
  - factorisations like SVD, QR etc often denser than original matrix → require lots of memory :(
    (Though see eg CUR factorisations & matrix sketches)
  - methods that only require ability to compute 'actions' eg matrix times vector → Iterative methods ✓

Problem ? regularisation parameter search : λ in range

→ iteration takes more function evaluations than single solve

↳ doing iteration for each λ is expensive

Can we combine λ search & iteration ?

Motivation II As regularisation methods !

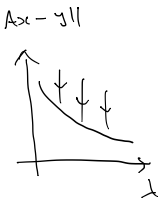
Key idea : early stopping rules for iterative opt.

Stopping rule plays role of regularisation
- too soon : bad fit / underfit / biased
- too late : overfit noise / too unstable

Advantage :

- rather than solve e.g.

for λ in range :
best $x(\lambda)$ via iterative



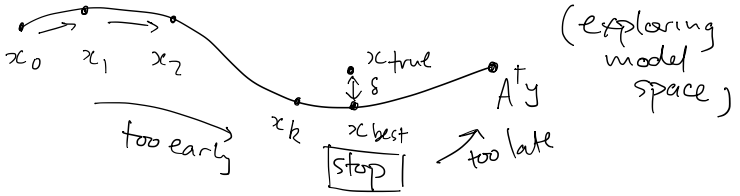
- generate :

sequence x_k, $k = 0, 1, \ldots$
where each is can be considered a regularised estimate &
$x_k \to x^+ = A^+ y$ as $k \to \infty$

Iterative regularisation

Want: - initial iterates close to regularised / 'simple' sol^{n}s
- better in middle stages, ie close to 'true' goal
- 'diverge' to exact but noise amplifying later

} controlled descent + stopping rule



(Note: continuation, ie use previous $x(\lambda_n)$ as guess for $x(\lambda_{n+1})$ is closely related!)

This idea is sometimes called 'semi-convergence'

Disadvantage:
- 'implicit' rather than explicit
- hard to analyse!
  ↳ But, really, true of nonlinear problems in general!

Regularisation schemes

	Tikhonov	TSVD	iterative
parameter:	λ	p (singular values)	k iteration number to stop.
parameter choice rule:	$\lambda \|\|Lx\|\|^2$ - L-curve / pareto - discrepancy principle - cross-validation etc	- pareto fit trade-off - Picard plot : etc	- Pareto fit stopping rule ('good enough') : etc

In general: start with 'simple'/biased model that doesn't fit so well, allow to become more complex/variable but not too complex.

Analysis ?

ML? Iterative methods as dynamical systems --

→ Singular limit : $A^\dagger y$ [nonhyperbolic?]

↳ goal 'in principle'

↳ unstable ! all models/data wrong

→ want to approach, but not too closely

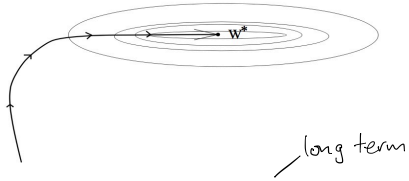
→ design a controlled approach to/flow on centre manifold ?

---

eg →

Even related to 'hot topics' like deep learning ! (eg (stochastic) gradient descent)

From LeCun et al "Efficient BackProp" in 'Neural Networks: Tricks of the trade' (book)



long term

Fig. 1.5. Convergence of the flow. During the final stage of learning the average flow is approximately one dimensional towards the minimum $\mathbf{w}^*$ and it is a good approximation of the minimum eigenvalue direction of the Hessian.

reduction

centre manifold

⇒ 'long term reduction to the centre manifold' !

CBMM Memo No. 073 **January 30, 2018**

Theory of Deep Learning III: the non-overfitting puzzle

by

T. Poggio[†], K. Kawaguchi [†⊤], Q. Liao[†], B. Miranda[†], L. Rosasco[†]
with
X. Boix[†], J. Hidary[††], H. Mhaskar[⋄],
[†]Center for Brains, Minds and Machines, MIT
[⊤]CSAIL, MIT
[††]Alphabet (Google) X
[⋄]Claremont Graduate University

explaining deep learning/non-overfitting using qualitative theory of ODEs, incl. centre manifolds.

Abstract: A main puzzle of deep networks revolves around the apparent absence of overfitting intended as robustness of the expected error against overparametrization, despite the large capacity demonstrated by zero training error on randomly labeled data.

In this note, we show that the dynamics associated to gradient descent minimization of nonlinear networks is topologically equivalent, near the asymptotically stable minima of the empirical error, to a gradient system in a quadratic potential with a degenerate (for square loss) or almost degenerate (for logistic or crossentropy loss) Hessian. The proposition depends on the qualitative theory of dynamical systems and is supported by numerical results. The result extends to deep nonlinear networks two key properties of gradient descent for linear networks, that have been recently recognized (*1*) to provide a form of implicit regularization:

qualitative theory of dynamical systems

One of the key ideas in stability theory is that the qualitative behavior of an orbit under perturbations can be analyzed using the linearization of the system near the orbit. Thus the first step is to linearize the system, which means considering the Jacobian of F or equivalently the Hessian of L at W^*, that is

$$\mathbf{H}_{ij} = \frac{\partial^2 L}{\partial w_i \partial w_j} \tag{2}$$

.

We obtain

$$\dot{W} = -HW, \tag{3}$$

where the matrix H, which has only real eigenvalues (since it is symmetric), defines in our case (by hypothesis we do not consider unstable critical points) two main subspaces:

- the stable subspace spanned by eigenvectors corresponding to negative eigenvalues
- the center subspace corresponding to zero eigenvalues.

centre manifold theory

The center manifold existence theorem (*16*) states then that if F has r derivatives (as in the case of deep polynomial networks) then at every equilibrium W^* there is a $\mathbf{C}^r$ stable manifold and a $\mathbf{C}^{r-1}$ center manifold which is sometimes called *slow manifold*. The center manifold emergence theorem says that there is a neighborhood of W^* such that all solutions from the neighborhood tend exponentially fast to a solution in the center manifold. In general, properties of the solutions in the center manifold depends on the nonlinear parts of F. We assume that the center manifold is not unstable in our case, reflecting empirical results in training networks. Of course, the dynamics associated with the center manifold may be non trivial, especially in the case of SGD. We simply assume for now that perturbations of it will not grow and trigger instability.

First: Iterative minimisation

- Quick review/recap.

  See [Nocedal & Wright 'Numerical Optimisation'
  Boyd & Vandenberghe 'convex optimisation']

  for more.

Descent methods for minimising a

function $f(x)$, $x \in \mathbb{R}^n$, $f(x) \in \mathbb{R}$

are all based on constructing a sequence

$$x_0, x_1, \dots x_k \dots$$

such that

$$f(x_{k+1}) < f(x_k)$$

unless x_k is optimal / a stopping condition is reached.

→ even non monotonic methods are usually descent methods for subsequences eg

$f(x_{k+m}) < f(x_k)$, $m \geq 1$ etc.

Descent method flavours

Two basic flavours:

1. **line search** given x_k
   - ↳ choose a **search direction** s_k (vector)
   - ↳ solve $\min_{\alpha > 0} f(x_k + \alpha s_k)$

     (scalar optimisation prob. in α)

2. **trust region** given x_k.
   - ↳ replace f by 'surrogate' model m_k that we 'trust' to represent f inside a given trust region
   - ↳ solve $\min_{s} m_k(x_k + s)$

     for all s inside trust region

     ↑ vectors

---

1. Choose direction, identify distance
2. Choose distance, identify direction.

Common search methods

- Nelder-Mead } derivative-free
- Gradient/steepest descent
- Newton
- Quasi-Newton
- Conjugate gradient

} derivative-based (at least first, often second)

etc

Trade-offs:

- computational expensive (derivative computation etc)
- exploration vs exploitation
  - ↘ look for better in new/wider neighbourhood
  - ↘ make use of what you have!
- robustness

etc

Tools:

Matlab:
- fminsearch (Nelder-Mead)
- optimisation toolbox

Python:
- scipy.optimize library
- sklearn
- etc.

Iterative regularisation

Recall: ____

Want:
- initial iterates close to regularised/'simple' soln's
- better in middle stages, close to 'true' goal
- 'diverge' to exact but noise amplifying later

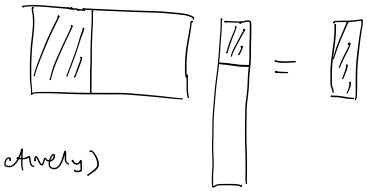
Linear example.

'start simple'

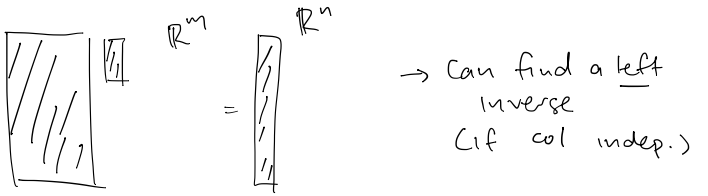
→ think overdetermined (effectively)

→ approach $A^\dagger$ from over → under det.

Full/initial:



So



→ Can find a left inverse (if col indep.)

Least squares & Normal equations...

But first!

Zeros & Fixed Points

'Theory of everything' (LeCun)

$$F(x) = 0 \quad \text{for some } F, x$$

(everything is a zero of some function)

Alternative:

$$T(x) = x \quad \text{for some } T, x.$$

(everything is a fixed point of some function)

Relationship

If eg

$$T(x) := \alpha F(x) + x, \quad \alpha \neq 0$$

Then

$$F(x) = 0 \iff T(x) = x$$

Fixed point iterations: (define goal as fixed point of some map.)

Define:

$$x_{k+1} = T(x_k)$$

If $x_k \to x^*$ as $k \to \infty$

Then get $x^* = T(x^*)$ & hence solves $F(x) = 0$

Contraction mappings

The above works if eg

$$\|Tx - Ty\| \leq a\|x - y\| \quad \text{for all } x, y$$

& $0 < a < 1$

More detail: functional analysis & dynamical systems

Difficult/interesting area for eg nonlinear T!

Normal equations

- $A^T A x = A^T y$
- $F(x) = A^T y - A^T A x = 0$
- $T(x) = \alpha F(x) + x = x$

Define:

$$x_{k+1} = T(x_k)$$
$$= x_k + \alpha F(x_k)$$
$$= x_k + \alpha A^T (y - A x_k)$$

→ 'Landweber iteration'

Note:

$$\nabla_x \|y - Ax\|^2 = -2 A^T (y - Ax)$$

⇒ gradient descent!

$$x_{k+1} = x_k - \alpha \nabla_x \phi(x_k)$$

for $\phi = \|y - Ax\|^2$

Landweber iteration

$$x_{k+1} = x_k + \alpha A^T (y - A x_k)$$

&

$$0 < \alpha < 2/s_1^2$$

← s_1: largest singular value

Stop when eg

- $\|A x_k - y\| \approx \epsilon$ = measure of noise level
- 'Flattens' off (monitor)

Note: only need matrix-vector products

→ no matrix factorisations

→ efficient for large sparse systems.

Landweber iteration

Can show in linear case iterates have form

$$x_k = V F_k \Lambda^{-1} U^T y$$

V: model space basis; $U^T y$: coefficients; $F_k \Lambda^{-1}$: Filter factors × usual SVD

$$F_k = \begin{bmatrix} f_1^k & & \\ & \ddots & \\ & & f_n^k \end{bmatrix}$$

$$f_i^k = 1 - (1 - \alpha s_i^2)^k$$

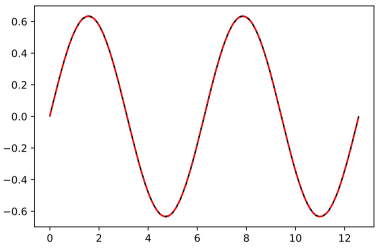
↑ using s_i for singular values

ie 'damps out singular values'
similar to Tikhonov / TSVD

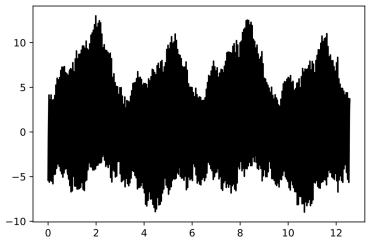
Iterative regularisation : Example.

Deconvolution again!

slightly noisy $Ax+\epsilon$



Naive $A^+ y$:



Landweber Iteration (Naive!)

```
niter = 1000
xs = np.zeros((niter,len(x)))
for i in range(0,niter-1):
    #update rule
    xs[i+1,:] = xs[i,:] + A.T@(y_noisy-A@xs[i,:])
```

Notes:

- just do 1000 & choose earlier stopping after (fast here so OK)
- largest s, s_1, is 1 here
  $\Rightarrow 0 < \alpha < 2$
  I used $\alpha = 1$ in iterations.

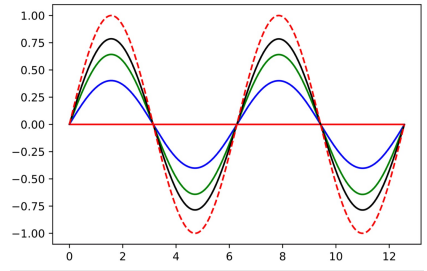
```
from scipy.sparse.linalg import svds

U, s, VT = svds(A, k=1)
s1 = s
print(s1)
```

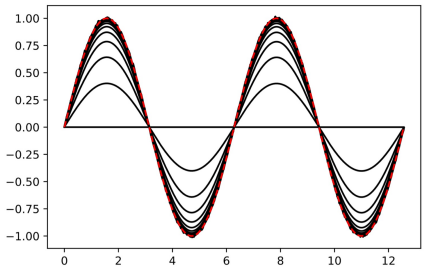
```
[ 1.]
```

Iterative regularisation : results

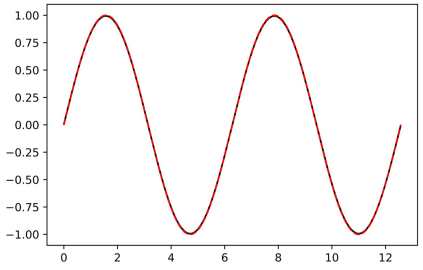
Solutions :



First few iterations



All iterations



10th iteration.

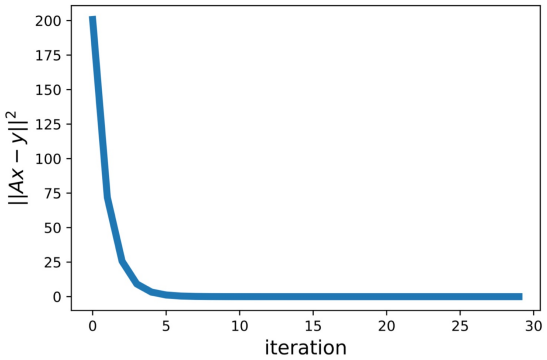
Pattern : underfit → good fit → overfit

ie 'semi-convergence'

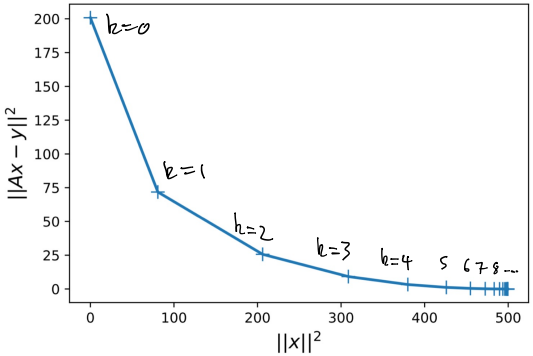
(not as obv here → see noise →)

Iterative regularisation : results

Convergence



Fit vs Iteration



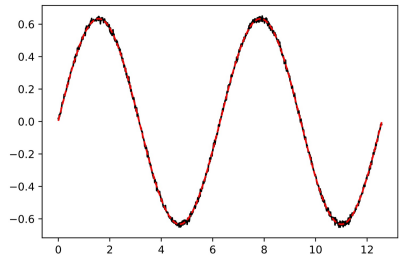
Lcurve

Note : 'slowing down' (slow manifold!)

→ smaller changes per it.

Iterative regularisation : Noisier

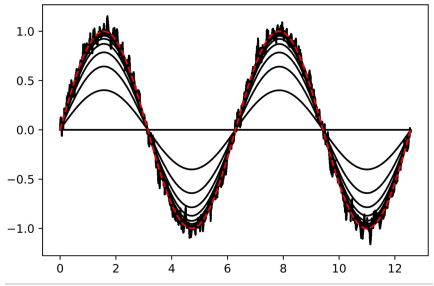
Noise ↑ x10



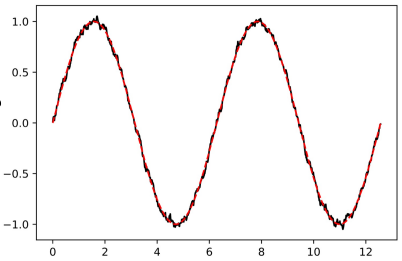
data.

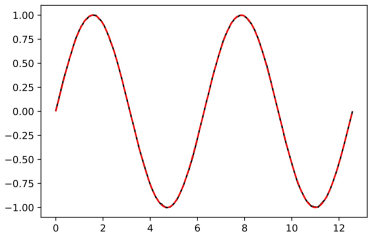
$Ax + \epsilon$

all inversions (1000)



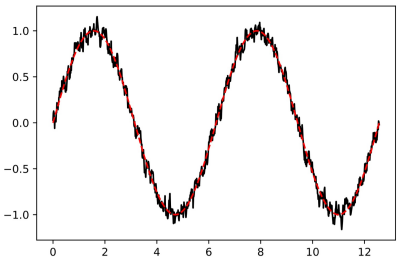
k = 300. Noise ampl.





k = 30 still good!

k = 1000. Noise ampl.



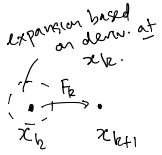
(overfitting noise)

Iterative regularisation : more sophisticated

→ Landweber iteration is basically (first-order) gradient descent

→ just as in optimisation, we can choose a search/update direction based on higher-order information

↳ more derivatives, 'larger' local neighbourhood



General $$x_{k+1} = x_k + F_k(x_k, y)$$

where F_k can take into account higher-derivatives* etc.

[* Think: Taylor series for $x(k+1) = T(x(k))$]

Trade-off: don't want it to optimise too well!

→ want to avoid $x^\dagger = A^\dagger y$ limit!

Iterative regularisation

Aster et al. (Chapter 6) Example 6.3 (see attached)

→ Deblurring 2D image (200 x 200 pixels)

→ with & without explicit regularisation

→ use conjugate gradient least squares

→ Blurring operator:
- 40,000 x 40,000
- very sparse
  - ↳ SVD produces denser matrices & requires a lot of storage
  - ↳ CGLS works well & only requires access to matrix-vector products

→ CGLS w/wout explicit regularisation
- → doesn't matter if only iterative!
- → much cheaper!

Note that this algorithm only requires one multiplication of $\mathbf{G}\mathbf{p}_k$ and one multiplication of $\mathbf{G}^T\mathbf{s}_{k+1}$ per iteration. We never explicitly compute $\mathbf{G}^T\mathbf{G}$, which might require considerable time, and which might have far more nonzero elements than $\mathbf{G}$ itself.

The CGLS algorithm has an important property that makes it particularly useful for ill-posed problems. It can be shown that, for exact arithmetic, $\|\mathbf{m}_k\|_2$ increases monotonically and $\|\mathbf{G}\mathbf{m}_k - \mathbf{d}\|_2$ decreases monotonically with increasing iterations [61, 65]. We can use the discrepancy principle together with this property to obtain a regularized solution. Simply stop the CGLS algorithm as soon as $\|\mathbf{G}\mathbf{m}_k - \mathbf{d}\|_2 < \delta$. In practice, this algorithm typically gives good solutions after a very small number of iterations.

An alternative way to use CGLS is to solve the Tikhonov regularization problem (4.4) by applying CGLS to

$$\min \left\| \begin{bmatrix} \mathbf{G} \\ \alpha\mathbf{L} \end{bmatrix} \mathbf{m} - \begin{bmatrix} \mathbf{d} \\ \mathbf{0} \end{bmatrix} \right\|_2^2 . \tag{6.76}$$

For nonzero values of the regularization parameter α, this least squares problem should be reasonably well-conditioned. By solving this problem for several values of α, we can compute an L-curve. The disadvantage of this approach is that the number of CGLS iterations for each value of α may be large, and we need to solve the problem for several values of α. Thus the computational effort is far greater.

Example 6.3

A commonly used mathematical model of image blurring involves the two-dimensional convolution of the true image $I_{\text{true}}(x, y)$ with a **point spread function**, $\Psi(u, v)$ [14]:

$$I_{\text{blurred}}(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} I_{\text{true}}(x - u, y - v)\Psi(u, v)\, du\, dv. \tag{6.77}$$

Here the point spread function shows how a point in the true image is altered in the blurred image. A point spread function that is commonly used to represent the blurring that occurs because an image is out of focus is the **Gaussian point spread function**

$$\Psi(u, v) = e^{\frac{u^2+v^2}{2\sigma^2}}. \tag{6.78}$$

Here the parameter σ controls the relative width of the point spread function. In practice, the blurred image and point spread function are discretized into pixels. In theory, Ψ is nonzero for all values of u and v. However, it becomes small quickly as u and v increase. If we set small values of Ψ to 0, then the $\mathbf{G}$ matrix in the discretized problem will be sparse.

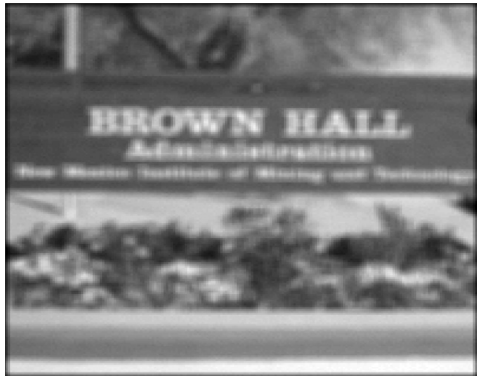


Figure 6.8 Blurred image.

Figure 6.8 shows an image that has been blurred and also has a small amount of added noise. This image is of size 200 pixels by 200 pixels, so the $\mathbf{G}$ matrix for the blurring operator is of size 40,000 by 40,000. Fortunately, the blurring matrix $\mathbf{G}$ is quite sparse, with less than 0.1% nonzero elements. The sparse matrix requires about 12 MB of storage. A dense matrix of this size would require about 13 GB of storage. Using the SVD approach to Tikhonov regularization would require far more storage than many current computers have. However, CGLS works quite well on this problem.

Figure 6.9 shows the L-curve for the solution of this problem by CGLS without explicit regularization and by CGLS with explicit regularization. The line with circles shows the solutions obtained by CGLS without explicit regularization. For the first 30 or so iterations of CGLS without explicit regularization, $\|\mathbf{G}\mathbf{m} - \mathbf{d}\|_2$ decreases quickly. After that point, the improvement in misfit slows down, while $\|\mathbf{m}\|_2$ increases rapidly.

Figure 6.10 shows the CGLS solution without explicit regularization after 30 iterations. The blurring has been greatly improved. Note that 30 iterations is far less than the size of the matrix ($n = 40{,}000$). Unfortunately, further CGLS iterations do not significantly improve the image. In fact, noise builds up rapidly, both because of the accumulation of round-off errors and because the algorithm is converging slowly towards an unregularized least squares solution. Figure 6.11 shows the CGLS solution after 100 iterations. In this image the noise has been greatly amplified, with little or no improvement in the clarity of the image.

We also computed CGLS solutions with explicit Tikhonov regularization for 22 values of α. For each value of α, 200 iterations of CGLS were performed. The resulting

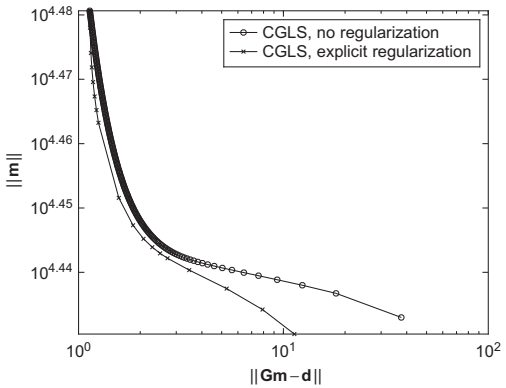


Figure 6.9 L-curves for CGLS deblurring.

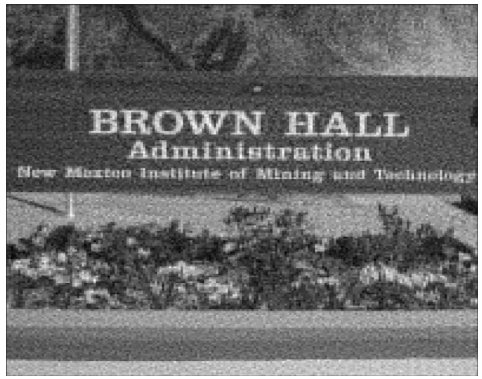


Figure 6.10 CGLS solution after 30 iterations, no explicit regularization.

L-curve is shown in Figure 6.9 with "x" markers for each regularized solution that was obtained. This L-curve is slightly better than the L-curve from the CGLS solution without explicit regularization in that the values of $\|\mathbf{m}\|_2$ and $\|\mathbf{Gm}-\mathbf{d}\|_2$ are smaller. However, it required 40 times as much computational effort. The corner solution for $\alpha = 7.0 \times 10^{-4}$ is shown in Figure 6.12. This solution is similar to the one shown in Figure 6.10.

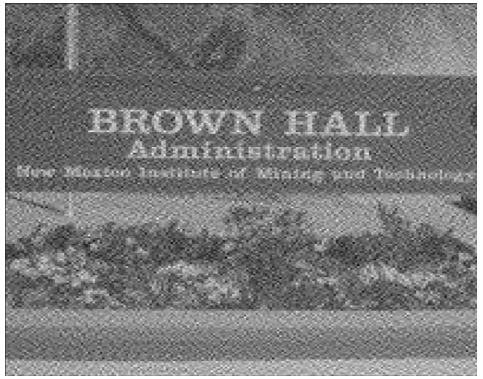


Figure 6.11 CGLS solution after 100 iterations, no explicit regularization.

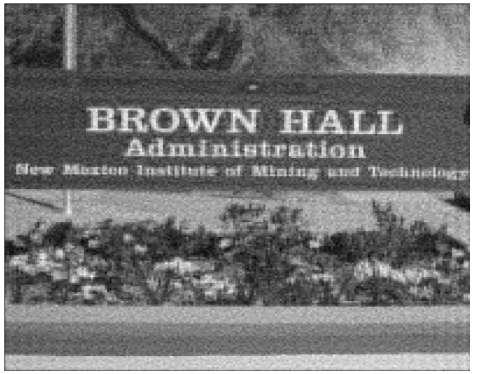


Figure 6.12 CGLS solution, explicit regularization, $\alpha = 7.0 \times 10^{-4}$.

We have focused on the CGLS method in this section because it is particularly easy to derive and because of the implicit regularization effect of the CGLS method. However, many other iterative methods have been developed for large-scale least squares problems [5, 135]. An analysis of the implicit regularizing effects of these methods can be found in Hanke [61]. The LSQR method of Paige and Saunders has been very widely used in many geophysical applications [123, 124]. MATLAB has an efficient and robust implementation of LSQR in its **lsqr** routine.