1D inverse problems: regularization and the SVD

Bill Lionheart's quick tutorials on numerical methods for inverse problems

In this tutorial we explore some one-dimensional inverse problems , and look at the singular values and Tikhonov regularization. We use Per Christian Hanson's Regularization Tools (reg tools). Follow this link for the code and manual http://people.compute.dtu.dk/pcha/Regutools/index.html The manual itself is a nice tutorial for numerical treatment of 1D inverse problems

If you have the mlx file you should be able to see all the formatting while the m file juast has comments (but Matlab can convert it to Matlab live "notebook" format). In any case you can run each section at a time and see the results, and if you like make changes to the code.

```
addpath ../RegTools/  % use the subdirectory where you downloaded reg tools
```

The example deriv2 is esentially and inverse problem where the forward problem involves integrating twice, so the discrete inverse problem is differentiating twice numerically. Which we expect t be unstable. First set up the problem. In general the examples in reg tools generate a matrix A, a data vector b and a solution x. We can then play with solving Ax=b and see what happens when we add noise to b etc.

```
n=100 %Size of problem
```

```
n =
   100
```

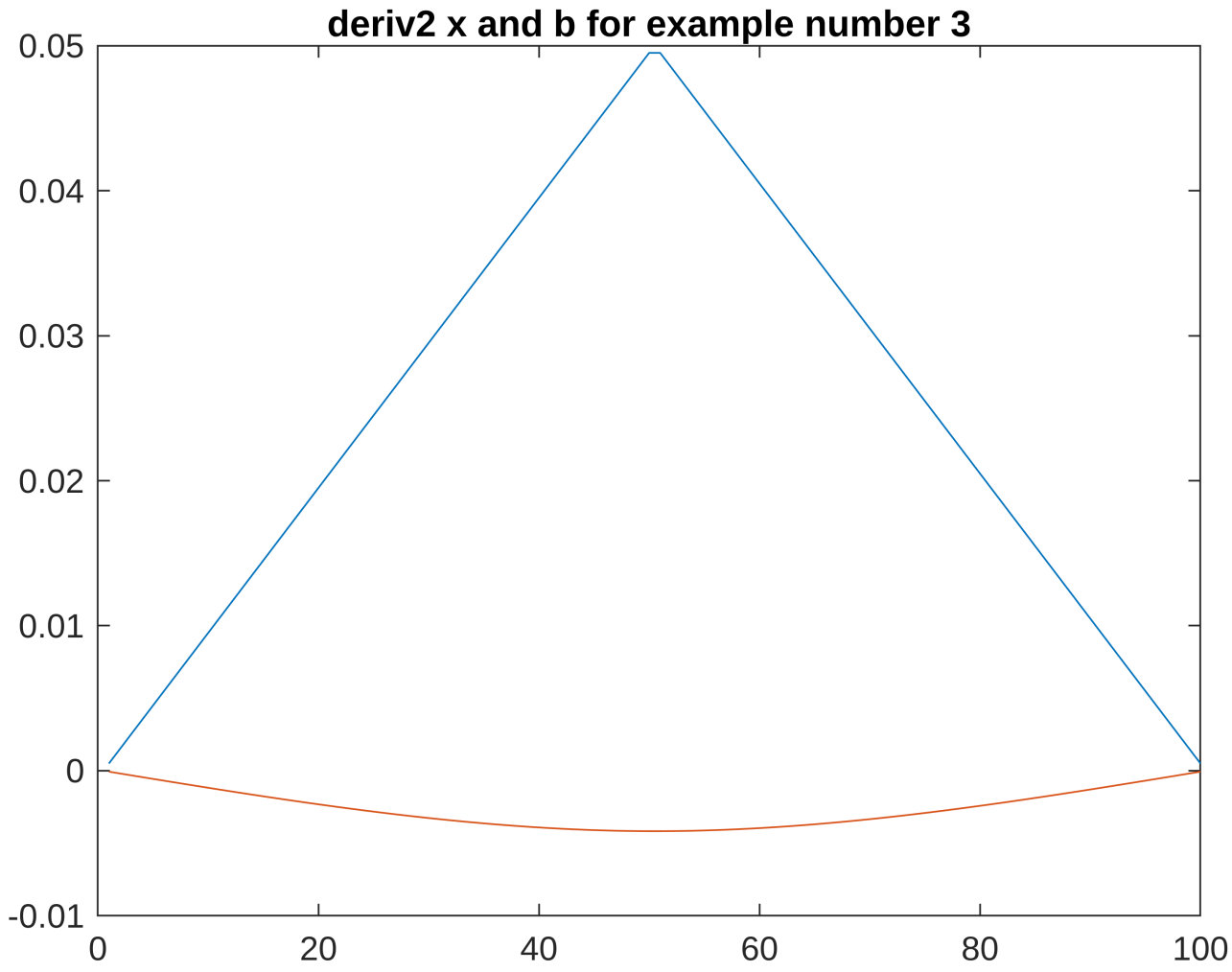
```
example=3;
[A,b,x]=deriv2(n,example); % The second derivative inverse problem
size(A)
```

```
ans = 1×2
   100   100
```

```
norm(A*x-b) % we see Ax is close to b
```

```
ans =
     2.405505837308437e-06
```

```
plot(x);hold on
plot(b)
hold off
title(['deriv2 x and b for example number ',num2str(example)])
```



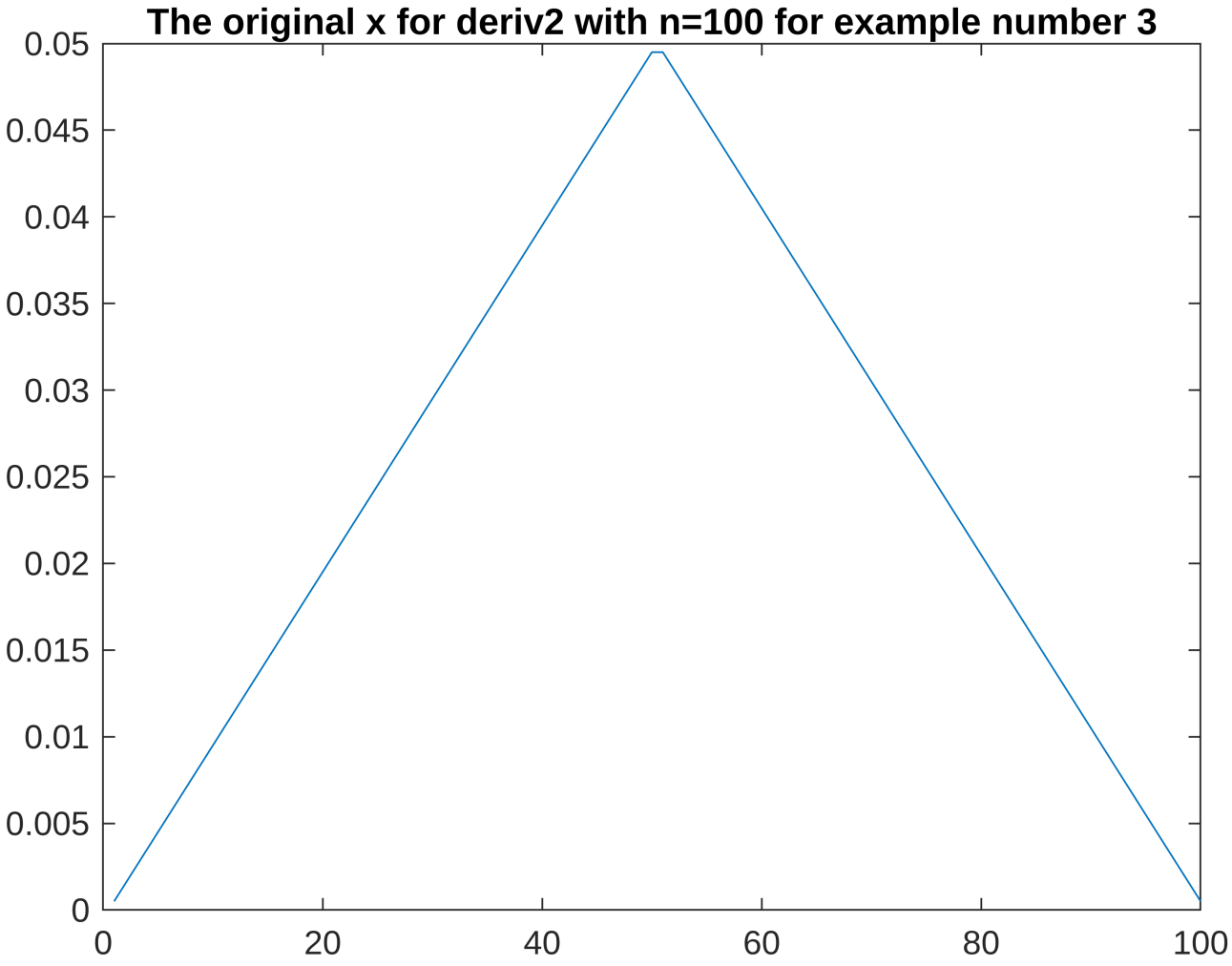
The matrix A is n by n, and the condition number measures who hard it is to invert

```
cond(A)
```

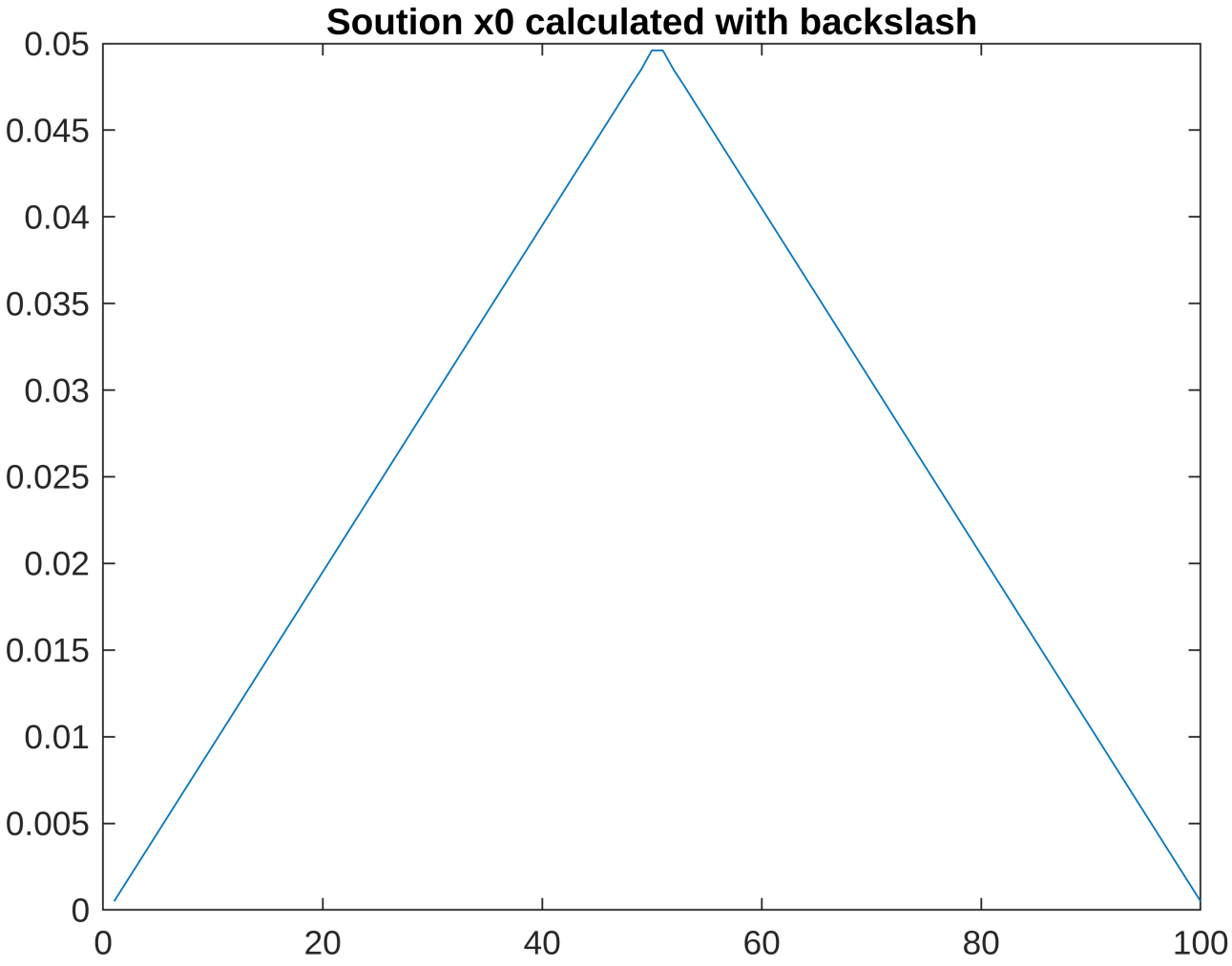
```
ans =
     1.215754208643192e+04
```

In tis case about 10^4 so *resonably* hard.

```
plot(x)
title(['The original x for deriv2 with n=',num2str(n),' for example number ',num2str(ex
```



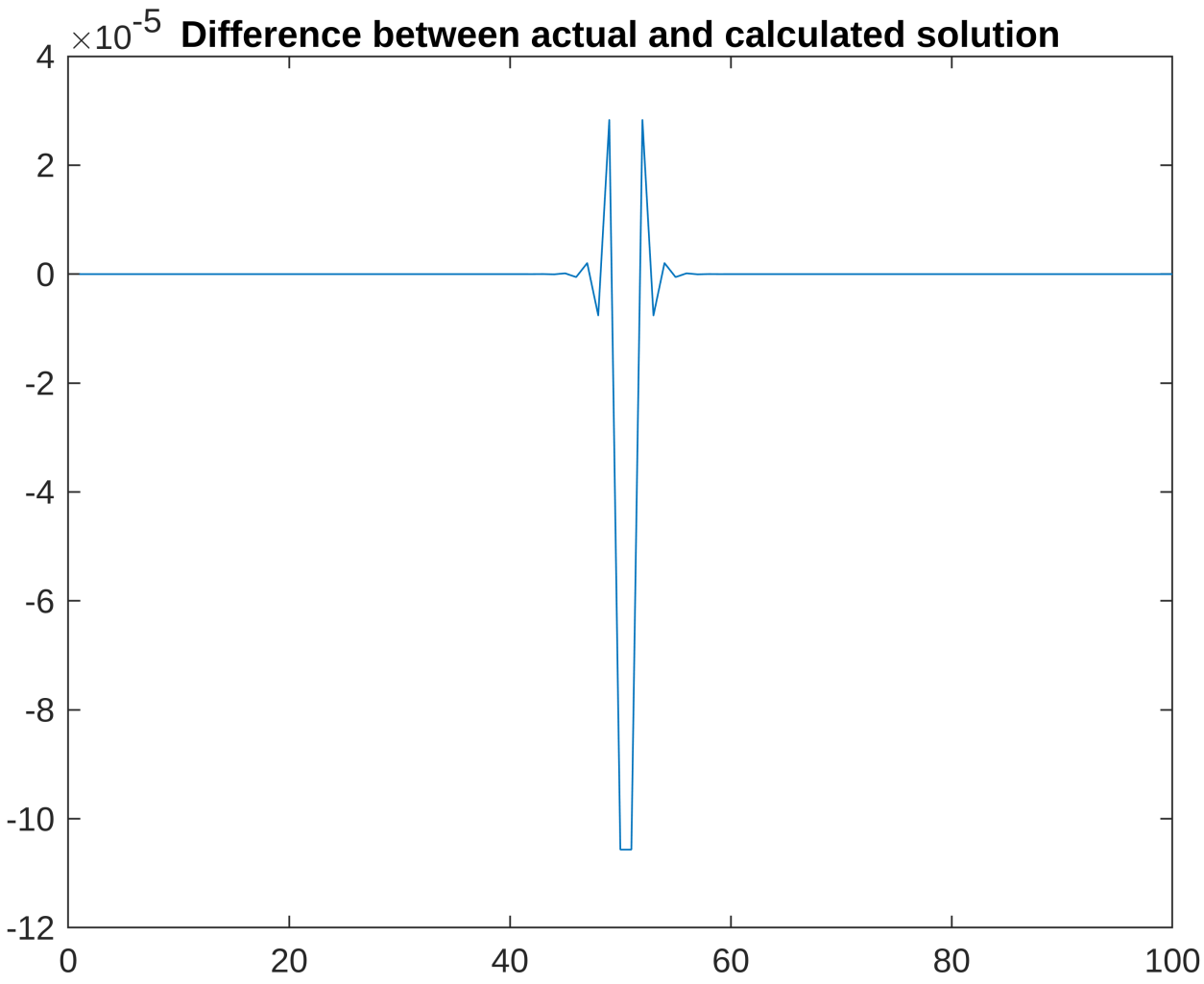

```
x0=A\b;
plot(x0)
title('Soution x0 calculated with backslash')
```




```
norm(A*x0-b)% error in solution
```

```
ans =
     9.974742848441219e-18
```

```
plot(x-x0)
title('Difference between actual and calculated solution')
```

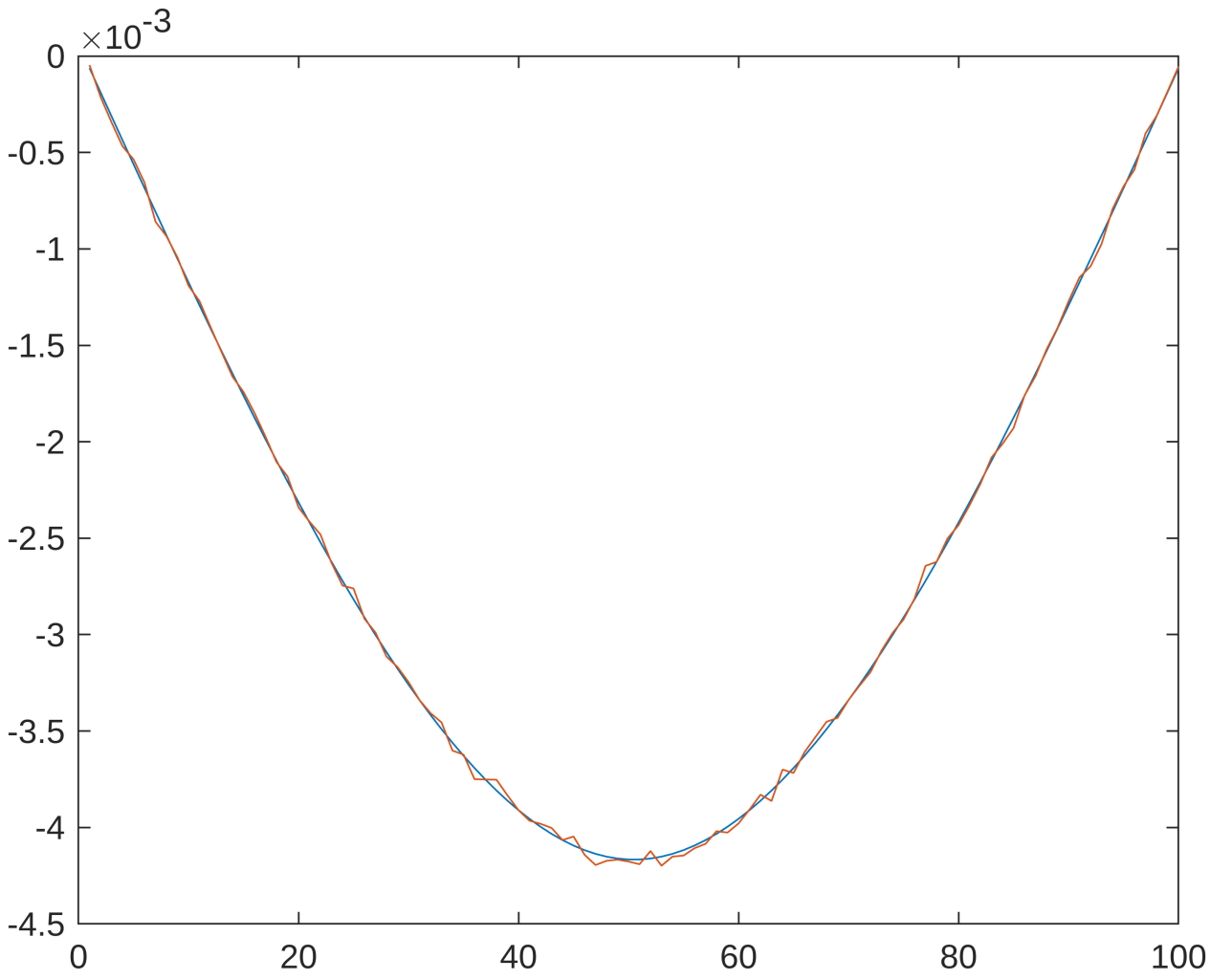


Suggestion: try different values of n and `example`

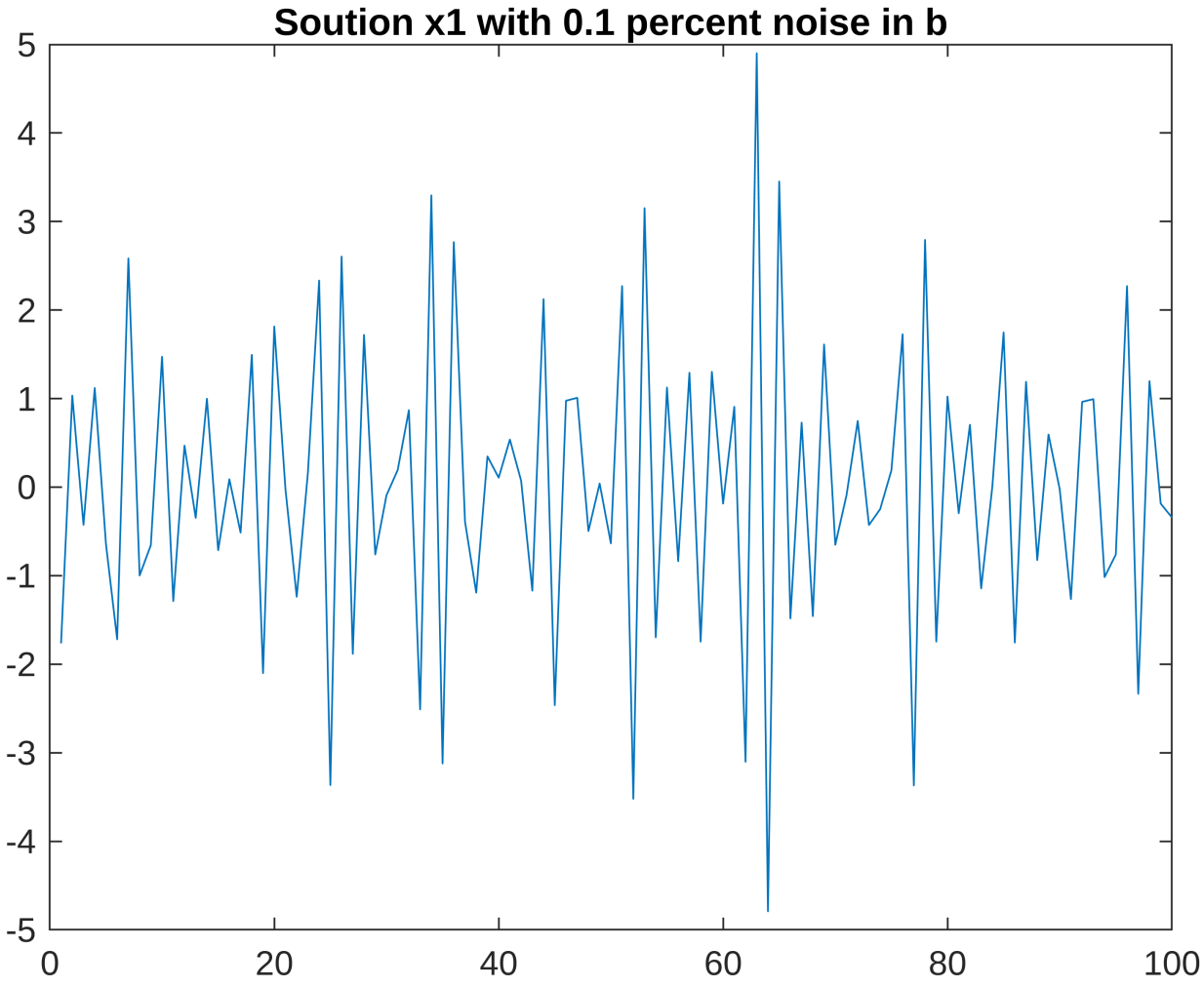
Clearly this is a fairly good solution but we did not deliberately add any noise to `b (ie we did an 'inverse crime').`

`Now we add some gaussian pseudo random noise with standard deviation std*norm(b)`

```
std= 0.001;
bnoise = b + std*randn(size(b))*norm(b);
plot(b); hold on
plot(bnoise)
hold off
```



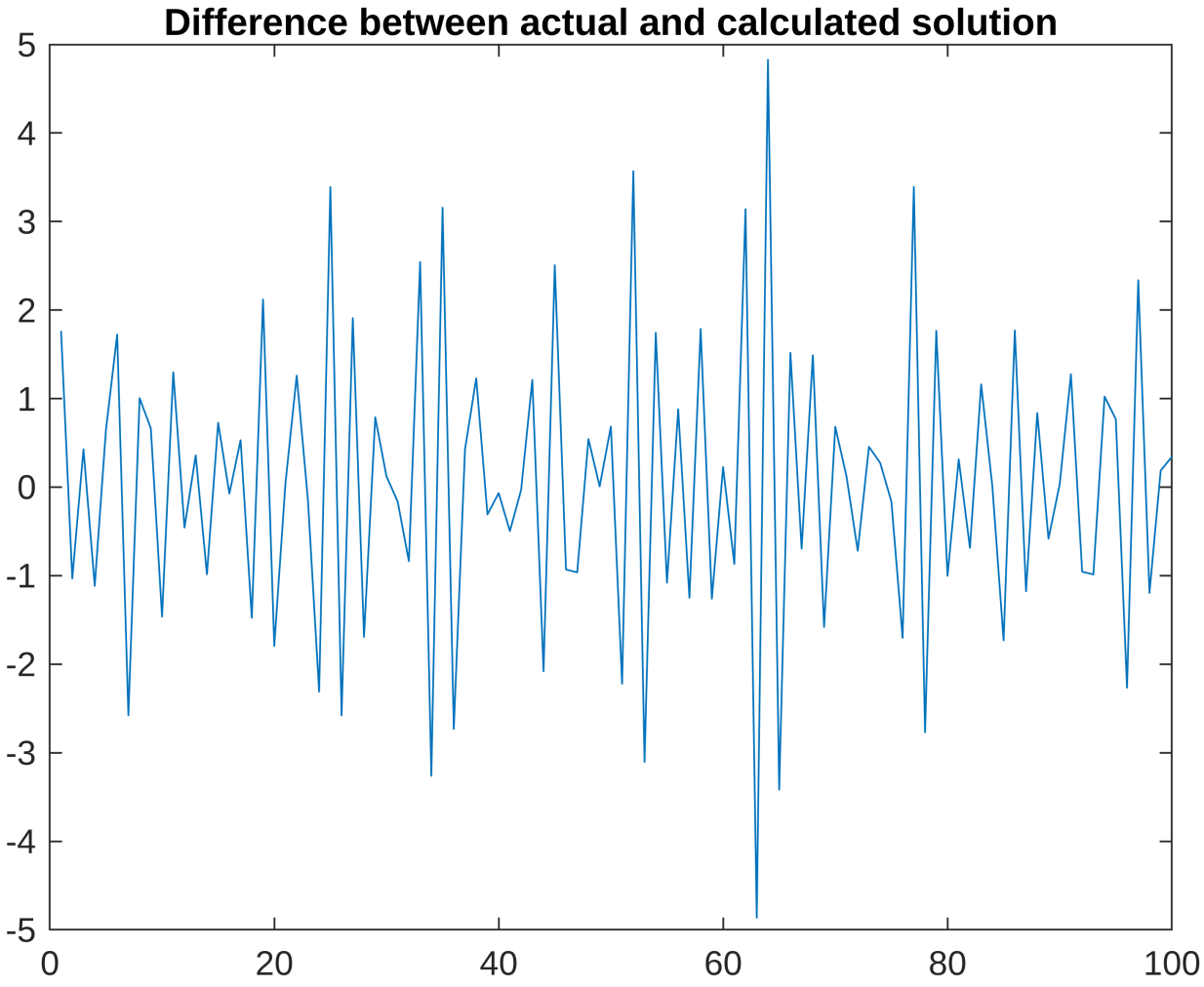

```
x1=A\bnoise;
plot(x1)
title(['Soution x1 with ',num2str(std*100),' percent noise in b'])
```




```
norm(A*x1-b)% error in solution
```

```
ans =
     2.678531627237602e-04
```

```
plot(x-x1)
title('Difference between actual and calculated solution')
```



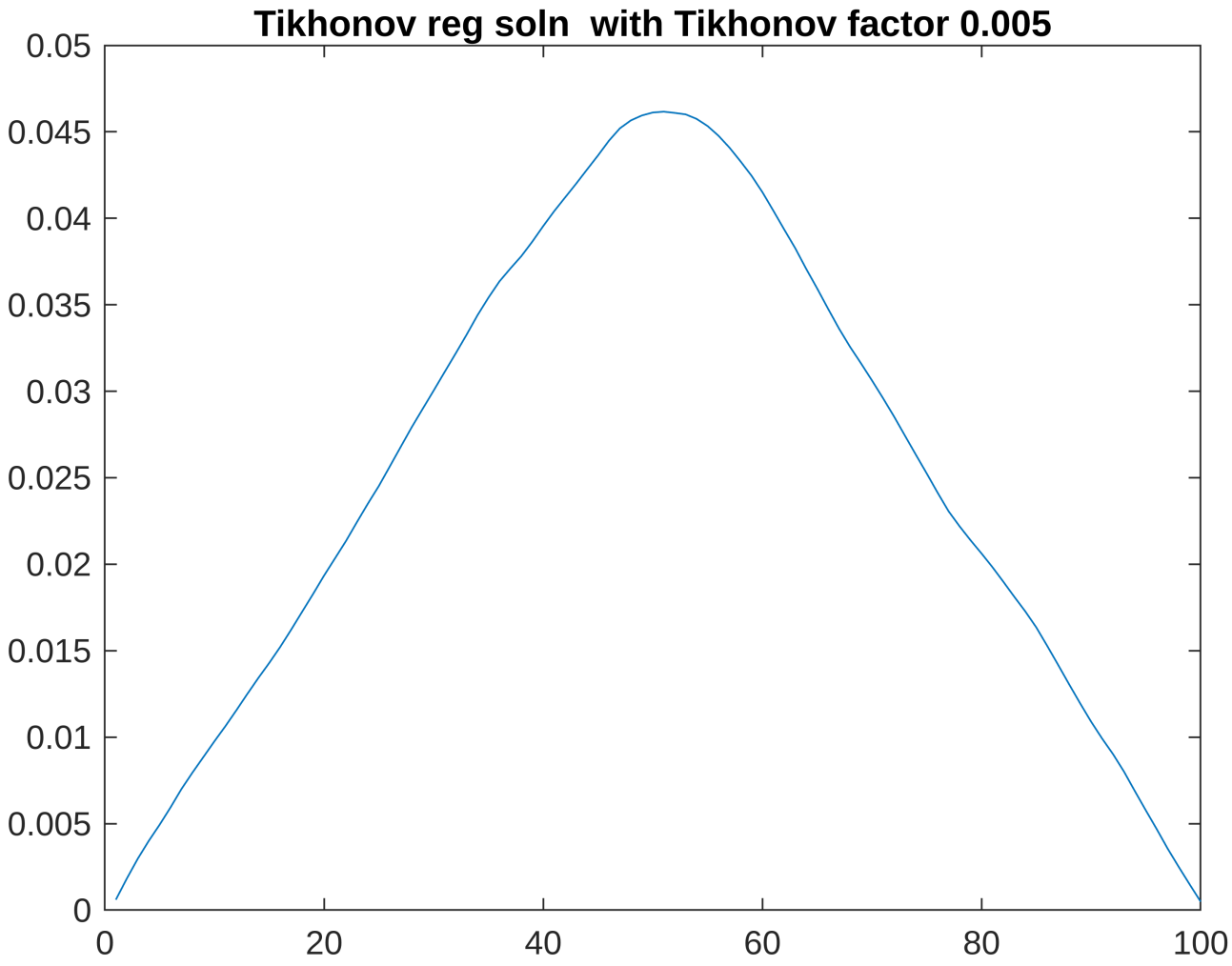
Suggestion: Try changing std to see what level of noise gives a recogisable solution.

Now lets try one of the simplest tricks for solving an inverse problem: Tikonov regularization.

```
TikFac=5e-3
```

```
TikFac =
   0.005000000000000
```

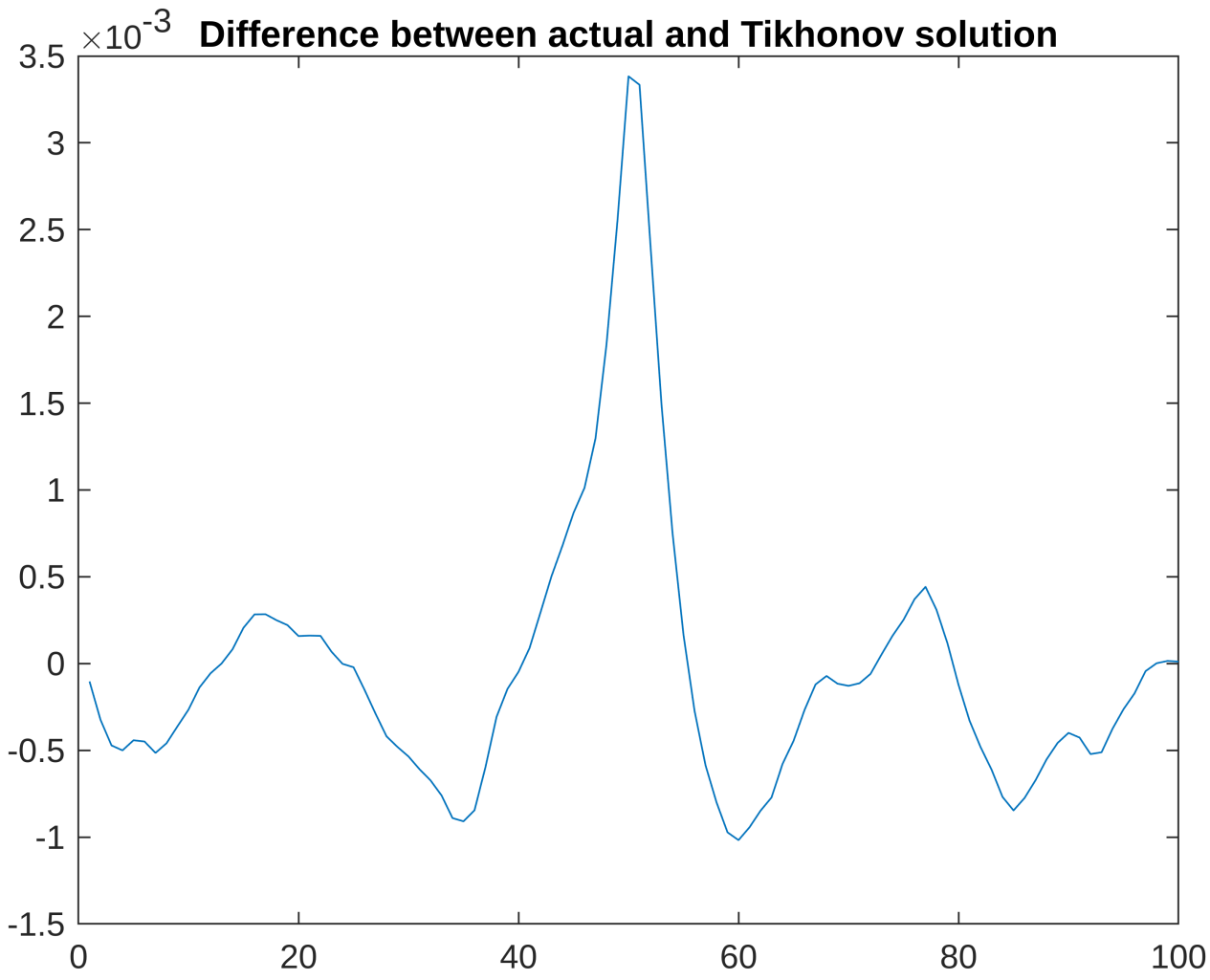
```
xtik=(A'*A+ TikFac^2 *eye(n))\(A'*bnoise);
plot(xtik)
title(['Tikhonov reg soln  with Tikhonov factor ',num2str(TikFac)])
```




```
norm(A*xtik-b)% error in solution
```

```
ans =
     8.065419571696940e-05
```

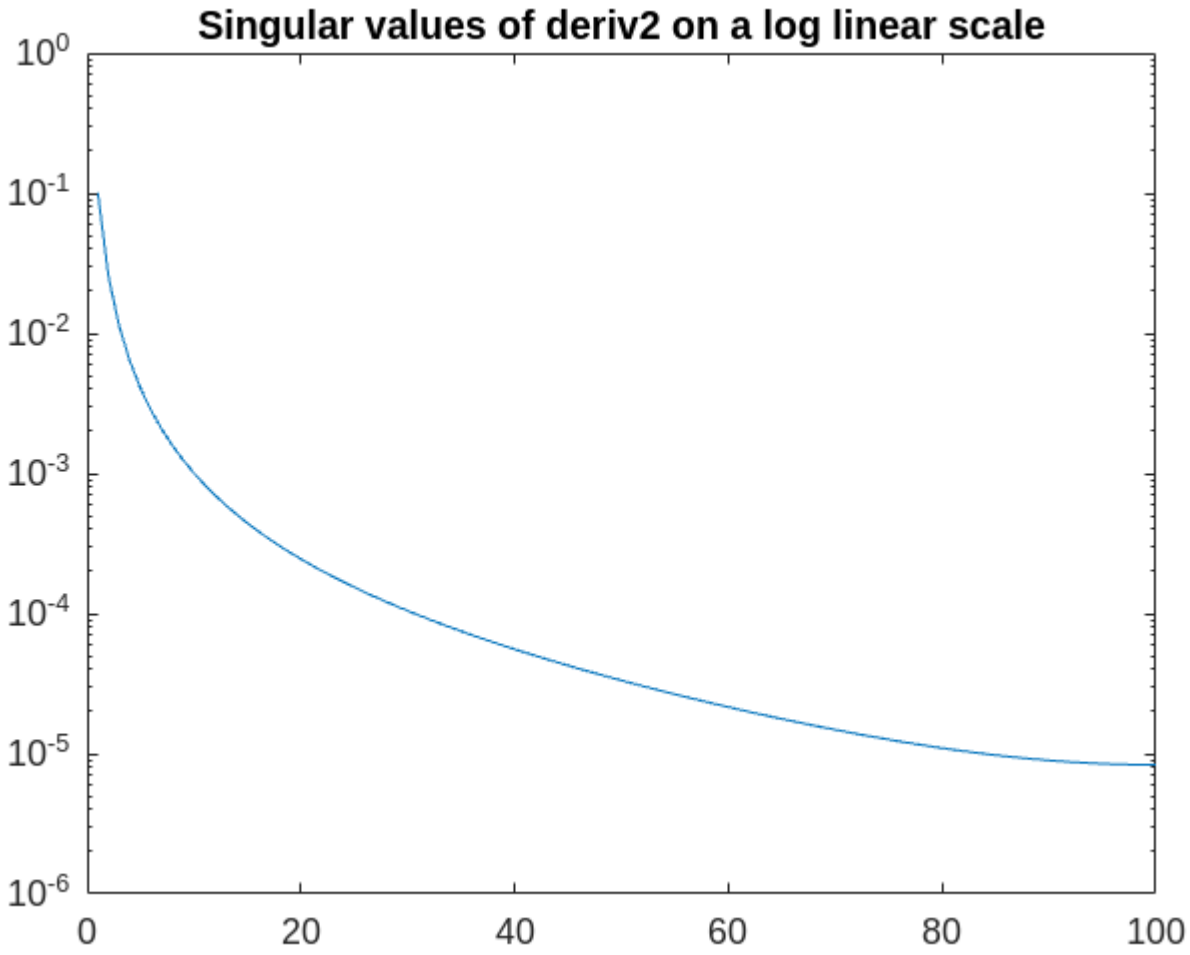
```
plot(x-xtik)
title('Difference between actual and Tikhonov solution')
```



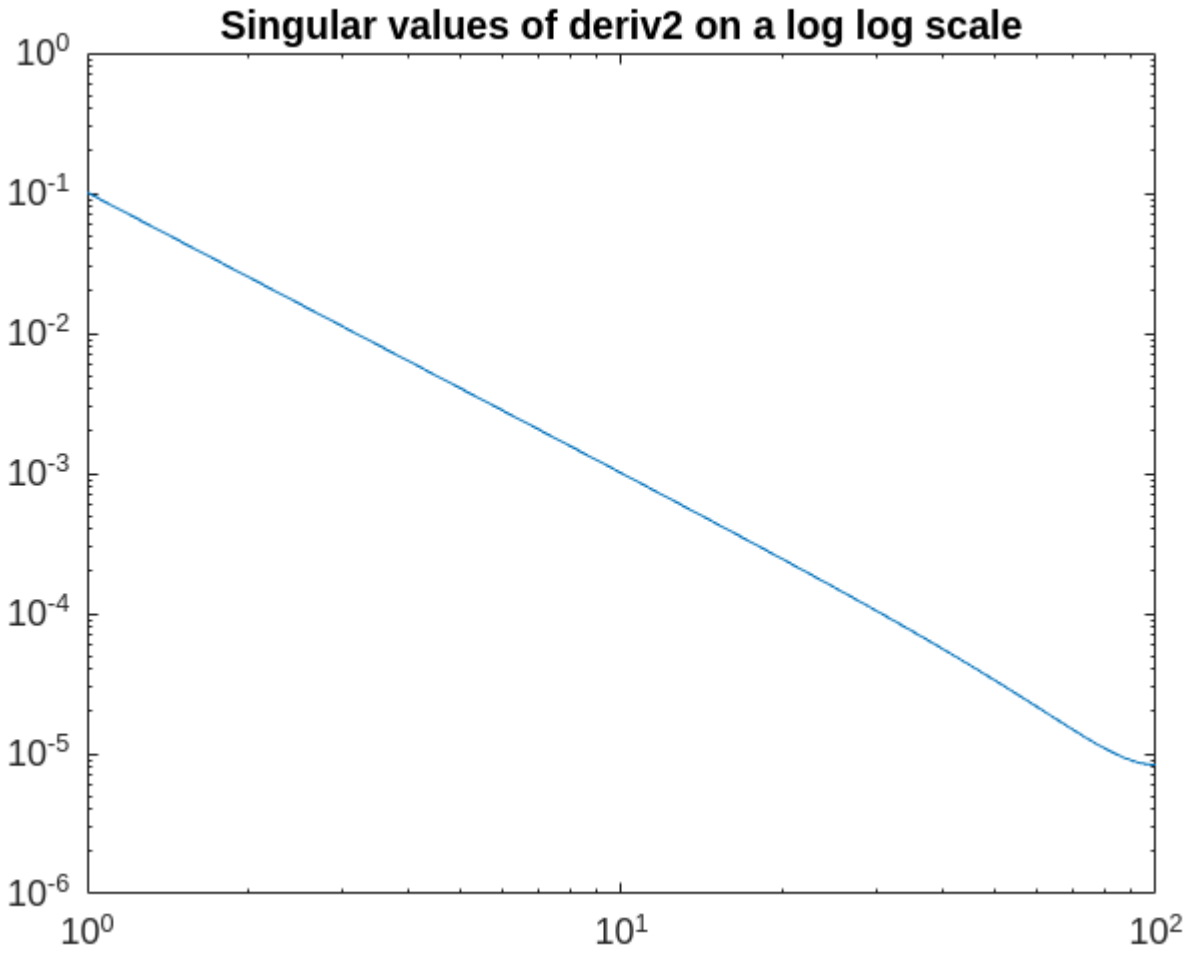
Suggestion: Try changing the Tikhonov factor TikFac. Too small and the solution is jagged, to big and it is over smooth.

Now lets look at the Singular value decomposition. Generally if the singular values decay like a negativ power we think of the problem as mildliy ill-posed, and if they go faster than any negative power (eg exponeltially decaying) we say the problem is severely ill-posed.

```
[U,S,V]=svd(A); %Calculate singular values and vectors
semilogy(diag(S)) % plot
title('Singular values of deriv2 on a log linear scale')
```




```
figure, loglog(diag(S)) %plot
title('Singular values of deriv2 on a log log scale')
```



As the graph of the singular values is close to a straight line on a log log scale we can see that this problem dewriv2 is mildly ill-posed. Roughly numerical differentiation works fine if you assume a bit of smoothness for the solution.

Lets try linear regression to get the slope and intercept

```
s=diag(S);
ls=log(s);
nn=1:n; ln =log(nn);
%Now regress ls against ln
regress = [ones(n,1),reshape(ln,n,1)]\reshape(ls,n,1);
slope = regress(2)
```

```
slope =
  -2.119106386283447
```

```
intercept= regress(1)
```

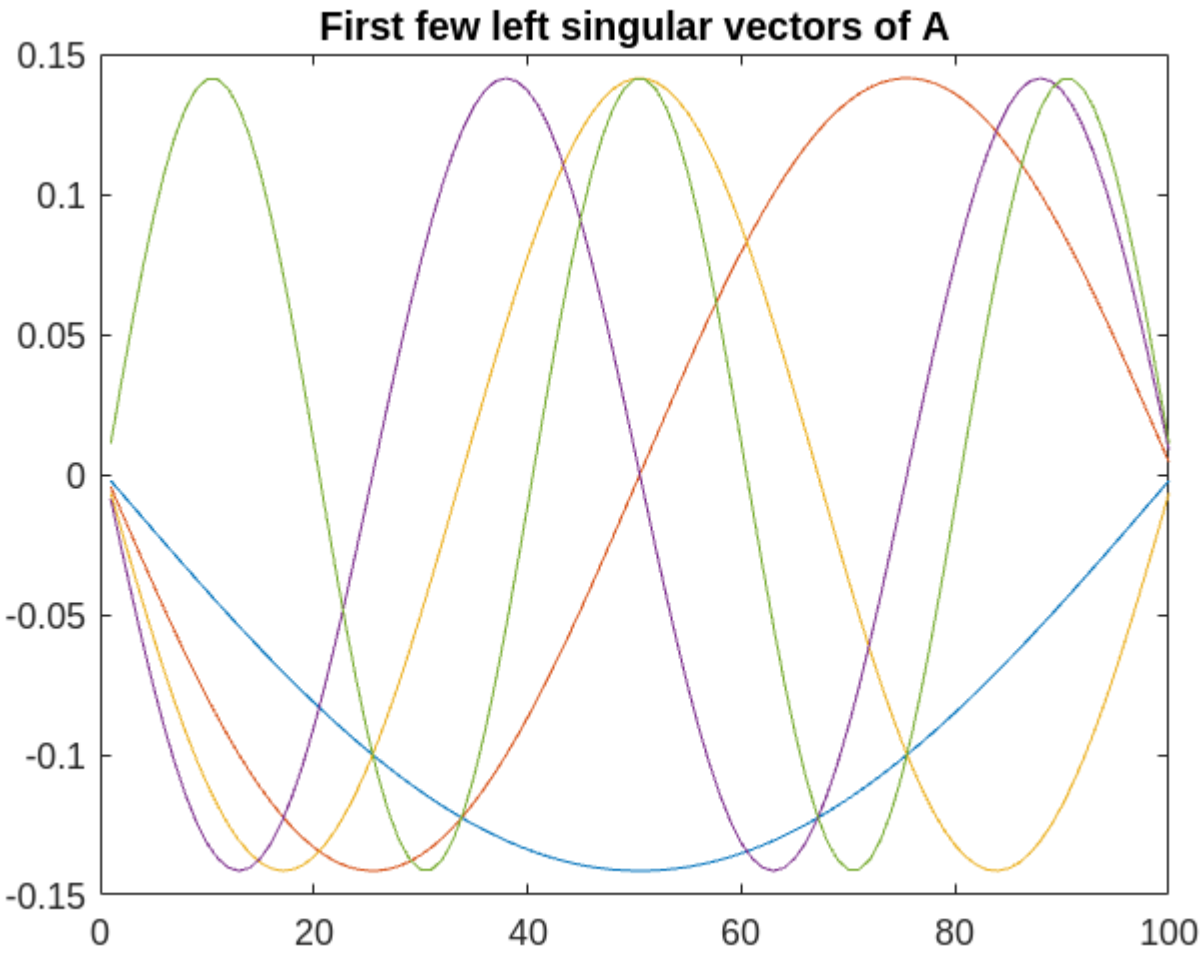
```
intercept =
  -2.042525314368363
```

So the slope is approximately -2 which is consistent with the idea A implements `integrating twice'.

Suggestion: If you have not seen linear regression done like this with backslash check how this works.

If we look at some singular vectors (columns of U or V) we see they are discretizations of the Fourier basis.

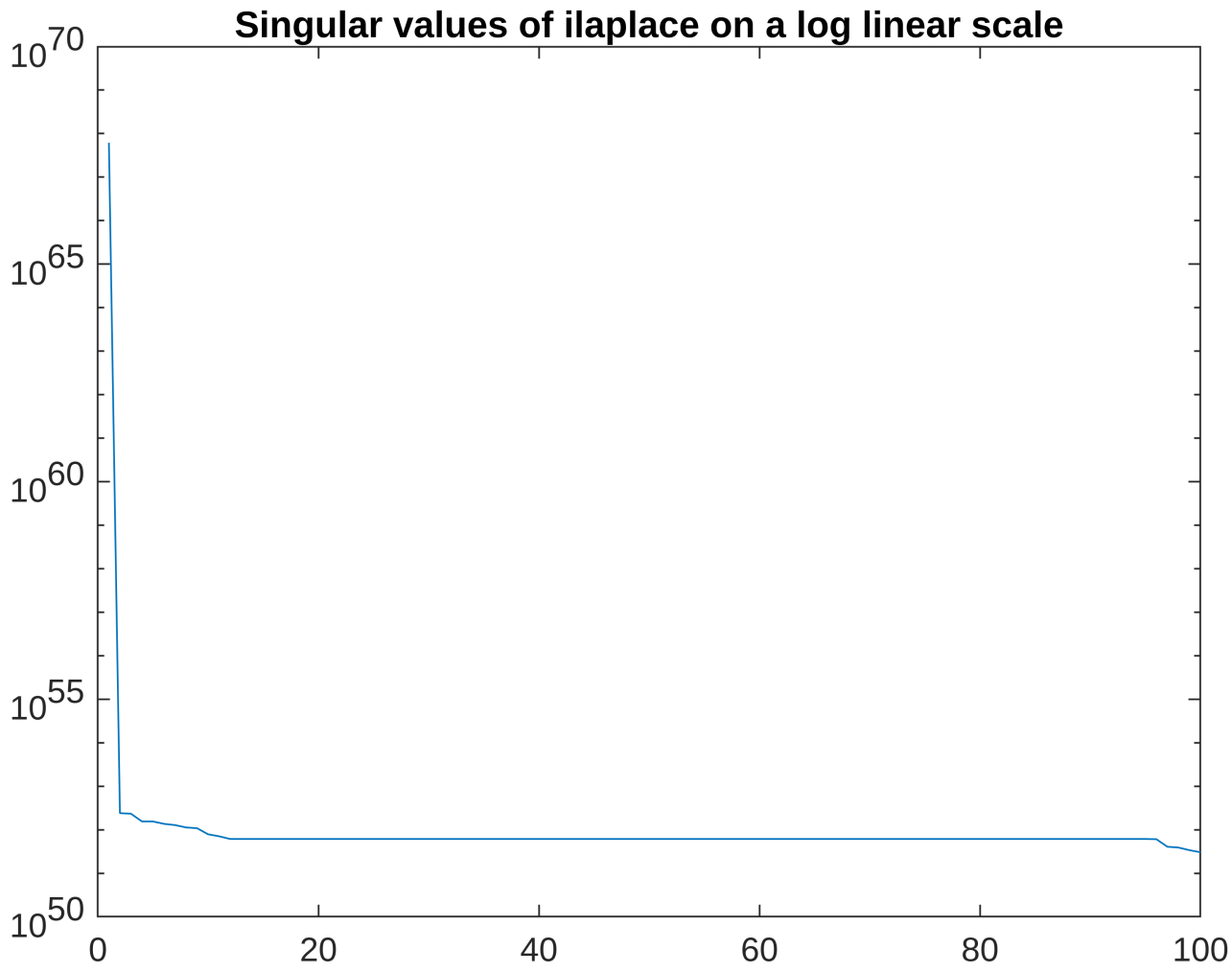
```
plot(U(:,1:5))
title('First few left singular vectors of A')
```



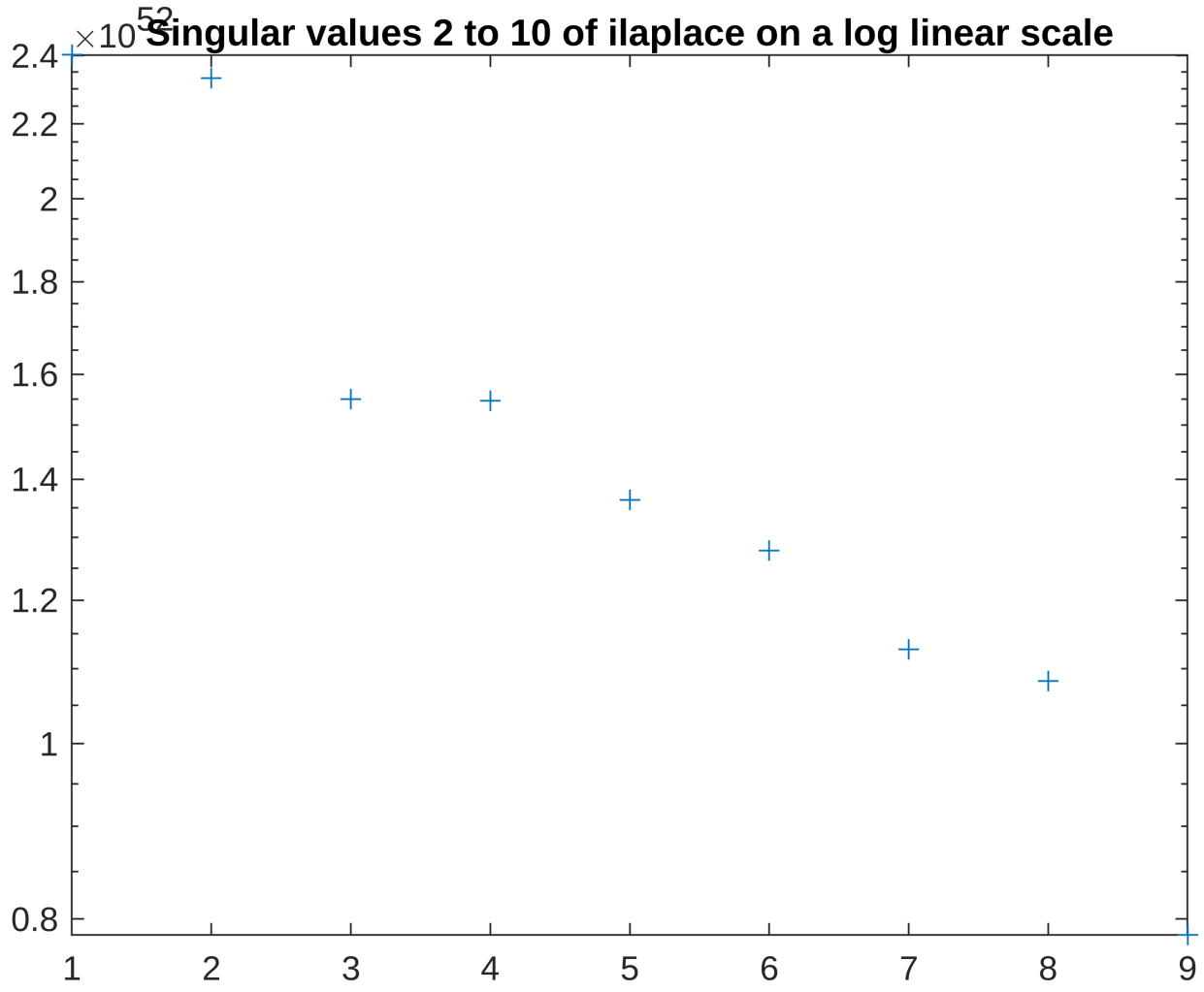
Notice in the Foruer domain 'differentiate twice' becomes 'multiply by freqency squared' (up to constants).

Now lets have a look at a *severly* ill-posed problem. The backward heat equation `heat` and the nverse truncated Laplace transform `i_laplace` are examples in reg tools. We ill look at singular values

```
[A,x,b]= i_laplace(n,example);
[U,S,V]=svd(A); %Calculate singular values and vectors
s= diag(S);
semilogy(s) % plot
title('Singular values of ilaplace on a log linear scale')
```



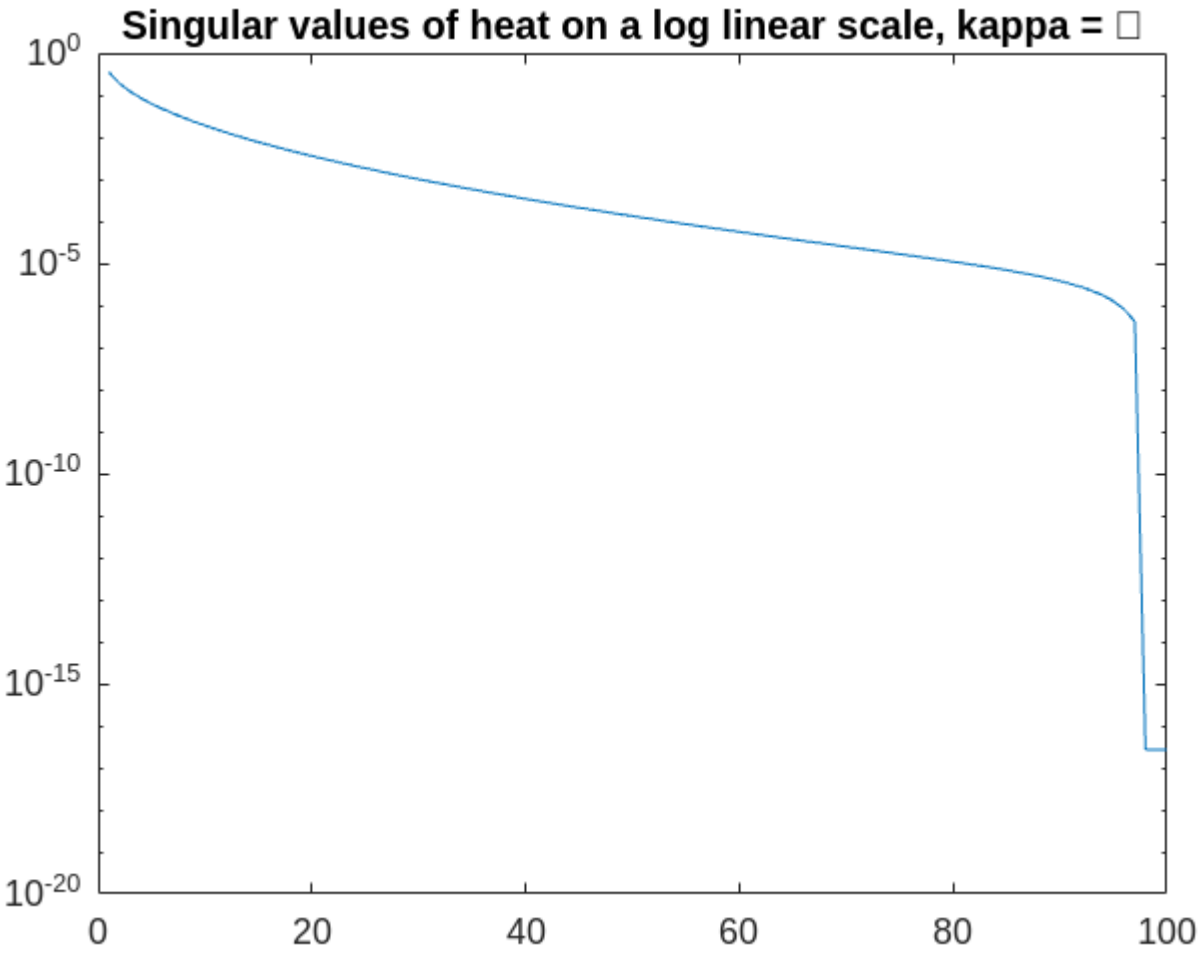

```
semilogy(s(2:10),'+')
title('Singular values 2 to 10 of ilaplace on a log linear scale')
```



The first singular value is 6×10^{67}, then the next 9 are around 10^{52} decreasing roughly in pairs exponentially, quitel likely the rest are lost in the rounding error of the calculation.

Now for the backward heat equation which is not quite as strange.

```
kappa=1.0;
[A,x,b]= heat(n,kappa);
[U,S,V]=svd(A); %Calculate singular values and vectors
s= diag(S);
semilogy(s) % plot
title(['Singular values of heat on a log linear scale, kappa = ',kappa])
```

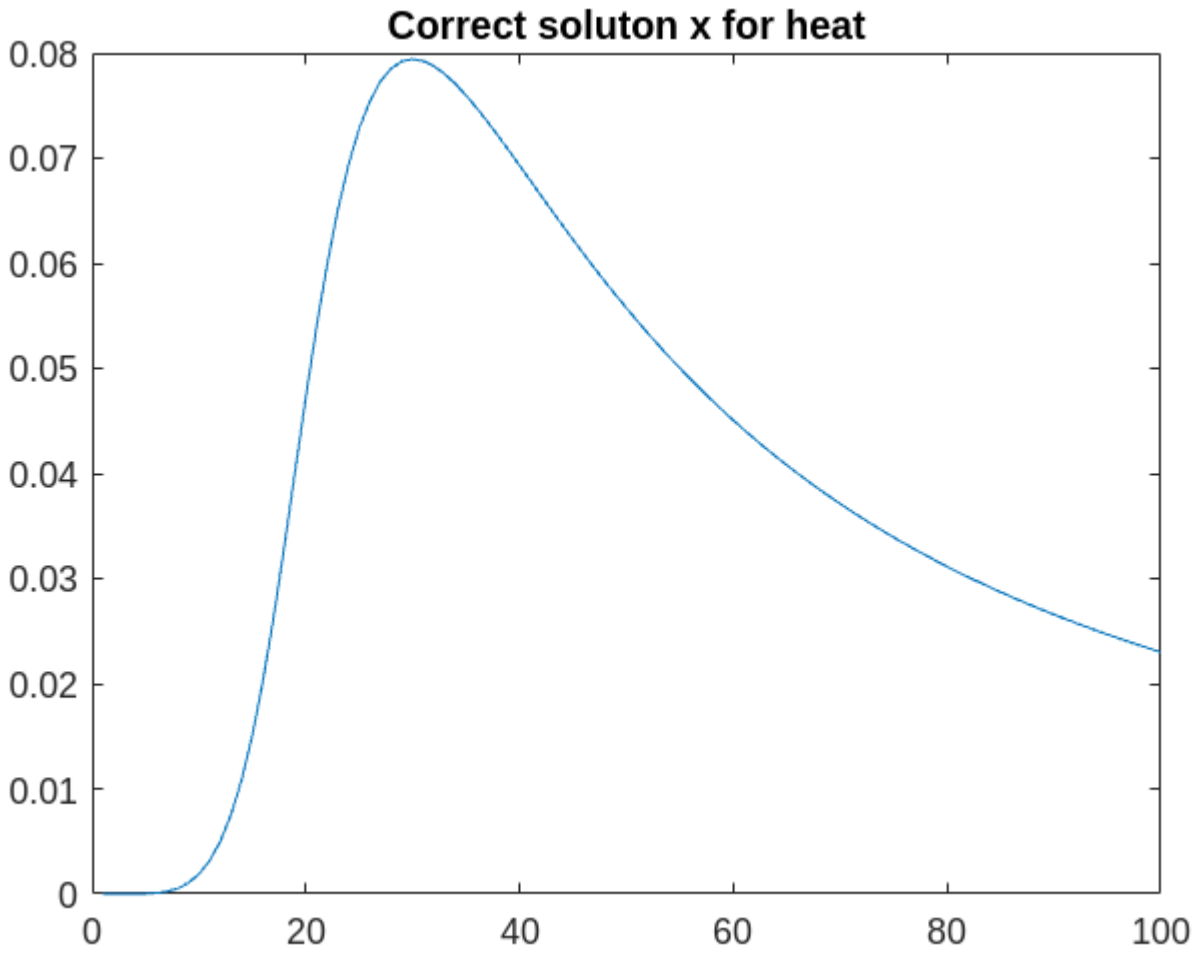


Notice that the last three singular values are around 10^{-17}which we can take numerically to be zero.So there appears to be a actuall null space for this operator. The log-linear plot reveals that before the dramatic fall off the singular values are decreasing slightly faster than an exponential.

As the documentation suggests increasing the value f kappa makes the problem less ill-posed

The soution in this example is as follows (note it is smooth).

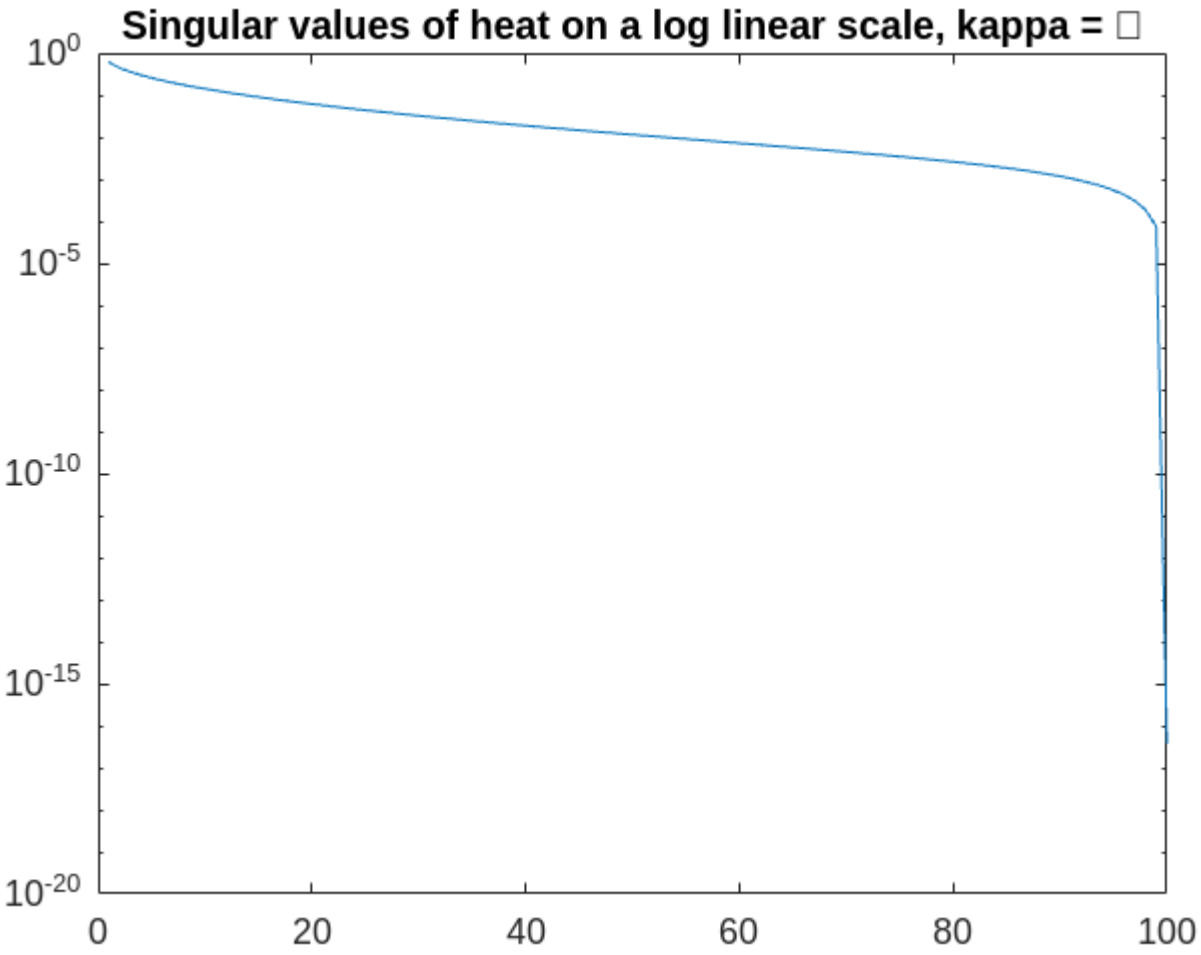
```
plot(x);title('Correct soluton x for heat')
```



We can of course make data for any x just by multiplying by A.

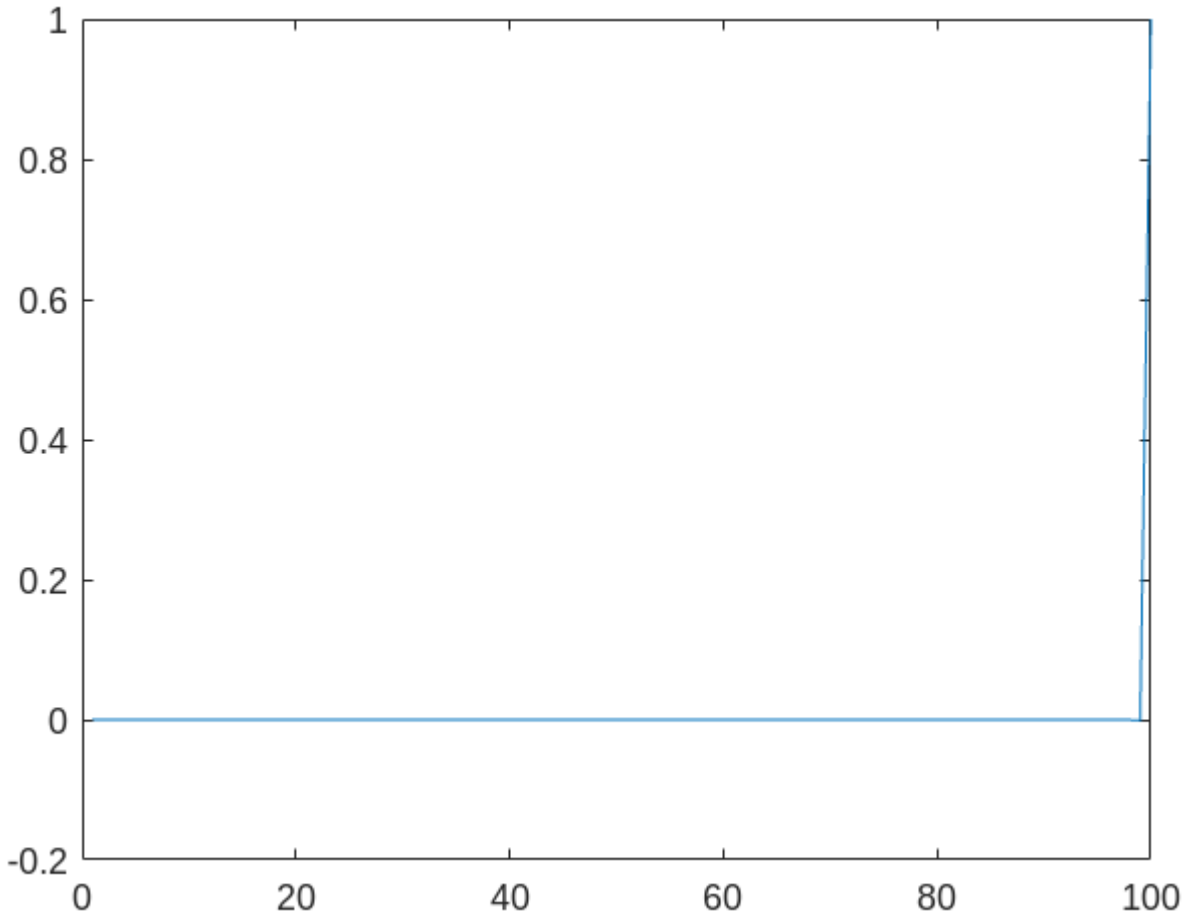
Trying a bigger value of kappa

```
kappa=2.0;
[A,x,b]= heat(n,kappa);
[U,S,V]=svd(A); %Calculate singular values and vectors
s= diag(S);
semilogy(s) % plot
title(['Singular values of heat on a log linear scale, kappa = ',kappa])
```



A little bit of experimentation shows that with this inverse crime Tikhonov regularization gives a reasonable solution. We see that while the singular values still decay exponentially they are not as fast and that only the last singular value is zero. This means there is a null space of A, to see what it is look at the last right singular vector

```
plot(V(:,end))
```



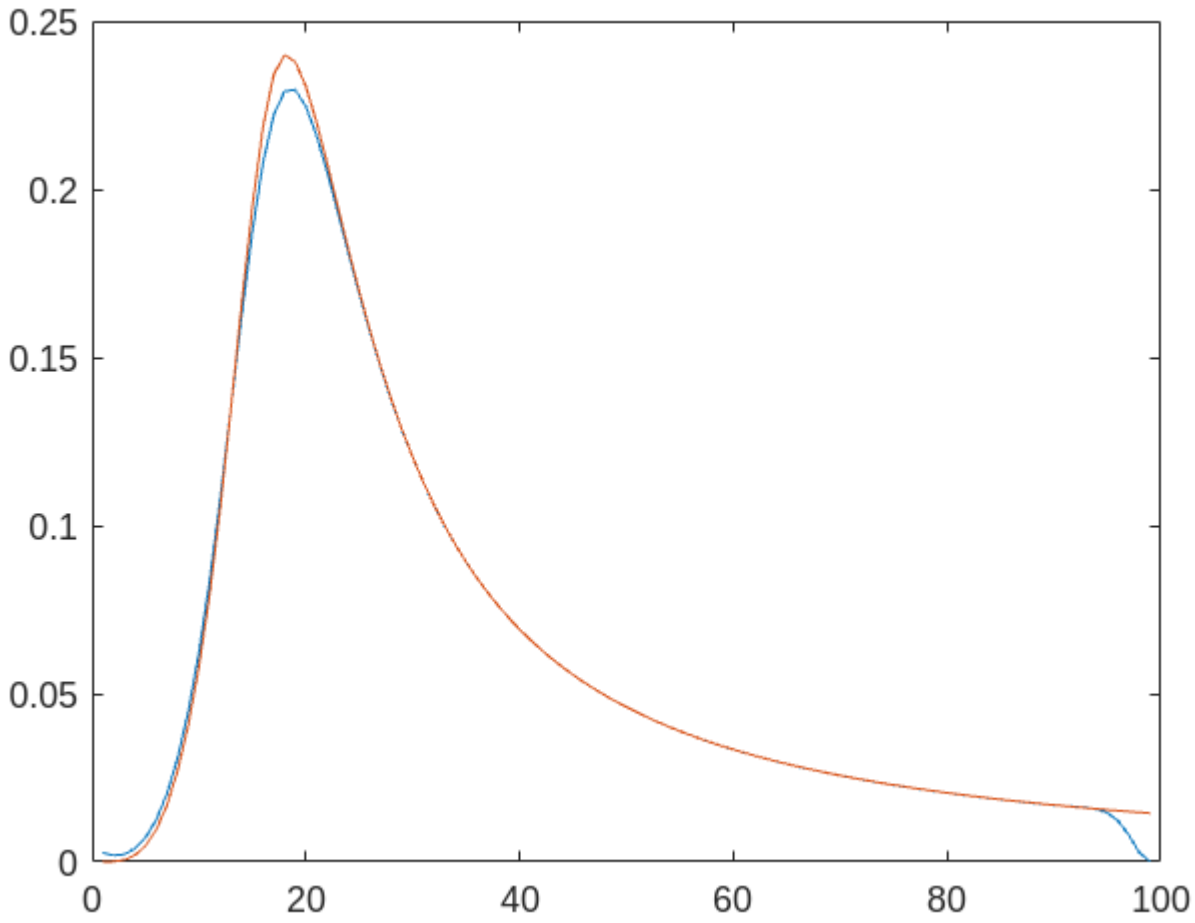

```
```

which shows that it is x(n) that we cannot find! Lets just chop off the last row and column (OK that is a bit of a hack!) and try Tikhonov.

```
Ar=A(1:(end-1), 1:(end-1));
xr=x(1:(end-1));
br = Ar *xr;
TikFac=0.05
```

```
TikFac =
   0.050000000000000
```

```
plot((Ar'*Ar + (TikFac)^2 *eye(n-1))\Ar'*br)
hold on
plot(xr)
hold off
```



Suggestion: 1) Try different Tikhonov factors. 2) Does this hack make any sense? Can you think of a better approach?

Although our examples with n=100 can be done quickly this way for larger problems forming A'*A explicitly is unecessary and is likely to use more memory than is necessary. Tikhonov regularizaton can be formulated as

$ x_{\mathrm{Tik} = \mbox{arg min} || A x -b ||^2 + \apha^2 ||L x||^2 $

where α is the Tikhonov factor and L is a (typically sparse) penalty matrix, eg a finite difference approximation to a differentla operator. We can solve this least squares problem using backslash without form in A'*A. Let us illistrate using derive again.

```
n=100
```

```
n =
```

```
   100
```

```
example=3;
TikFac=5e-3
```

```
TikFac =
   0.005000000000000
```

```
[A,b,x]=deriv2(n,example);
```

We form the augmented matrix using the identity matrix for L

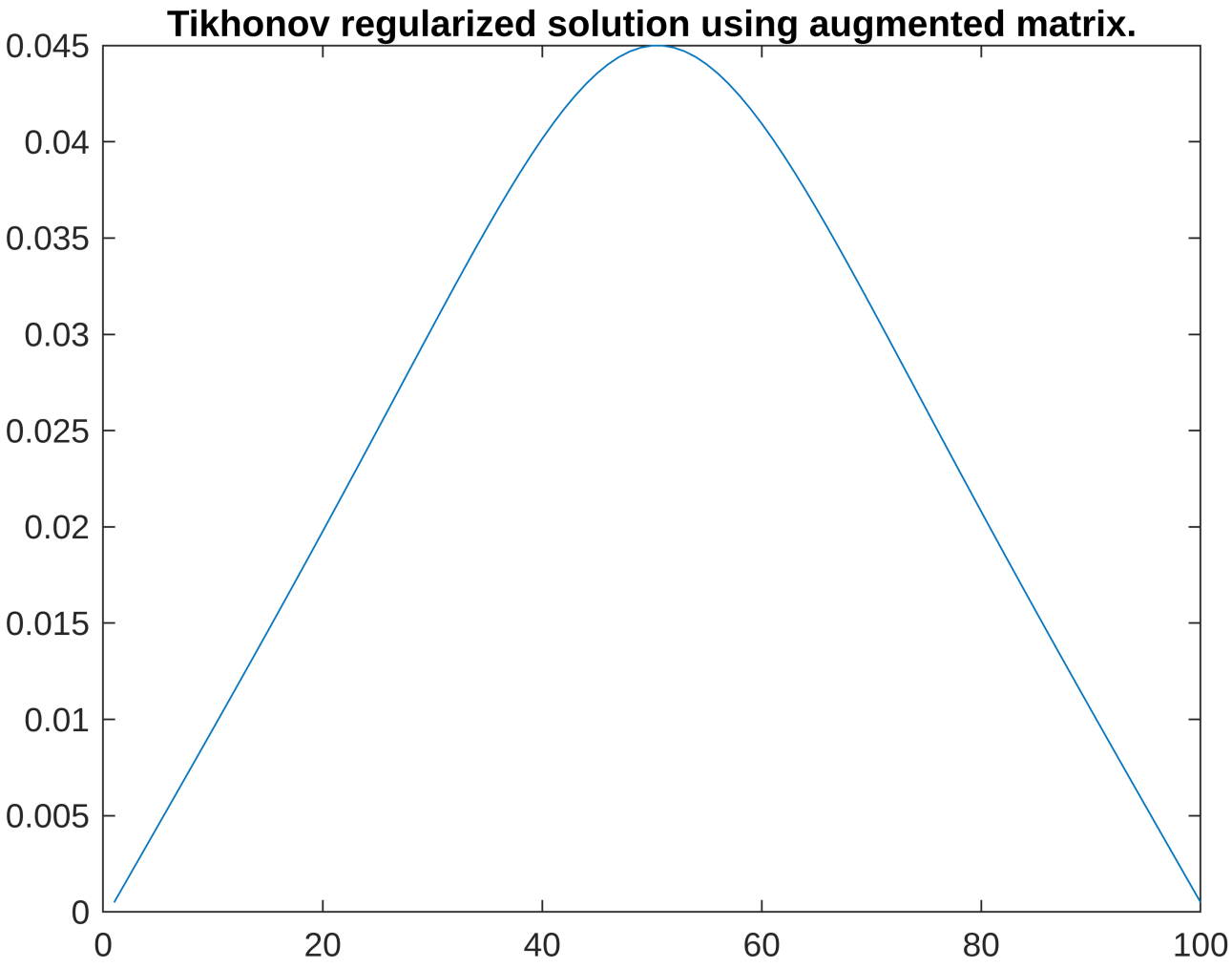
```
Aa = [A; TikFac* eye(n)];
size(Aa)
```

```
ans = 1×2
   200   100
```

```
ba = [b;zeros(n,1)];
```

Now we just minimize the 2-norm of Aa*x - ba

```
xTik = Aa\ba;
plot(xTik)
title('Tikhonov regularized solution using augmented matrix.')
```



with the same result as before. Often A and L are both sparse matrices. If A is too big to store or solve directly iterative solvers can be used in which A*x and A'*y are calculated on the fly without the elements of A being stored.We will see this later in tomography problems.

Suggestion: Look at some of the other examples in reg tools and see if they are severely ill-posed or mildly ill-posed.

Suggestion: These examples are all discretizations of integral operators using some form of numerical quadrature. In many cases the operator has a discrete spectrum. Experiment with increasing n for some examples in reg tools and see if the singular values appear to be converging. How large a matrix can you solve on your system?

The augmented matrix is well conditioned with respect to inversion for a big enough Tikhonov factor.

```
cond(Aa)
```

```
ans =
  20.287203000682400
```

Iterative methods that are designed to converge to the least squares solution will work well on such overdetermined systems of equation. One of the best is conjugate gradient least squares Here we specify 10 iterations

```
[X,rho,eta] = cgls(Aa,ba,10);
```

The variable X is returned with all 10 iterations while eta gives the norm of the solution (which should tend to the norm of the Tikhenov regularized solution) and rho returns the norm of the residual, ie norm(Aa*xk -ba) where xk is the k-th iteration. As the system is overdetermined we expect the least squares solution to have a non-zero residual, but it should converge quickly.

```
format long; norm(Aa*xTik-ba)
```

```
ans =
   0.001439100005445
```

This is the residual we expect rho to tend to

```
norm(ba)
```

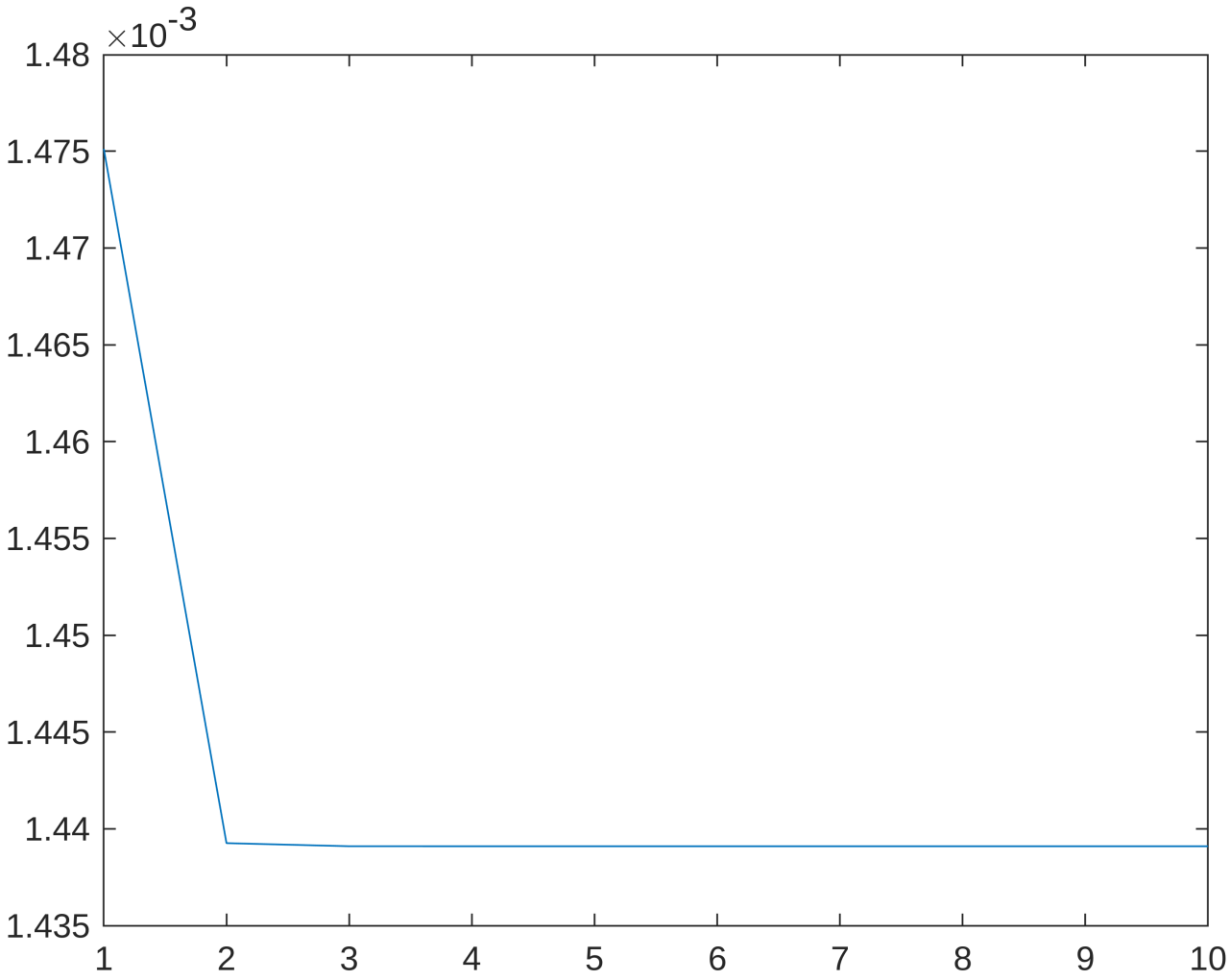
```
ans =
   0.029037639816692
```

```

```

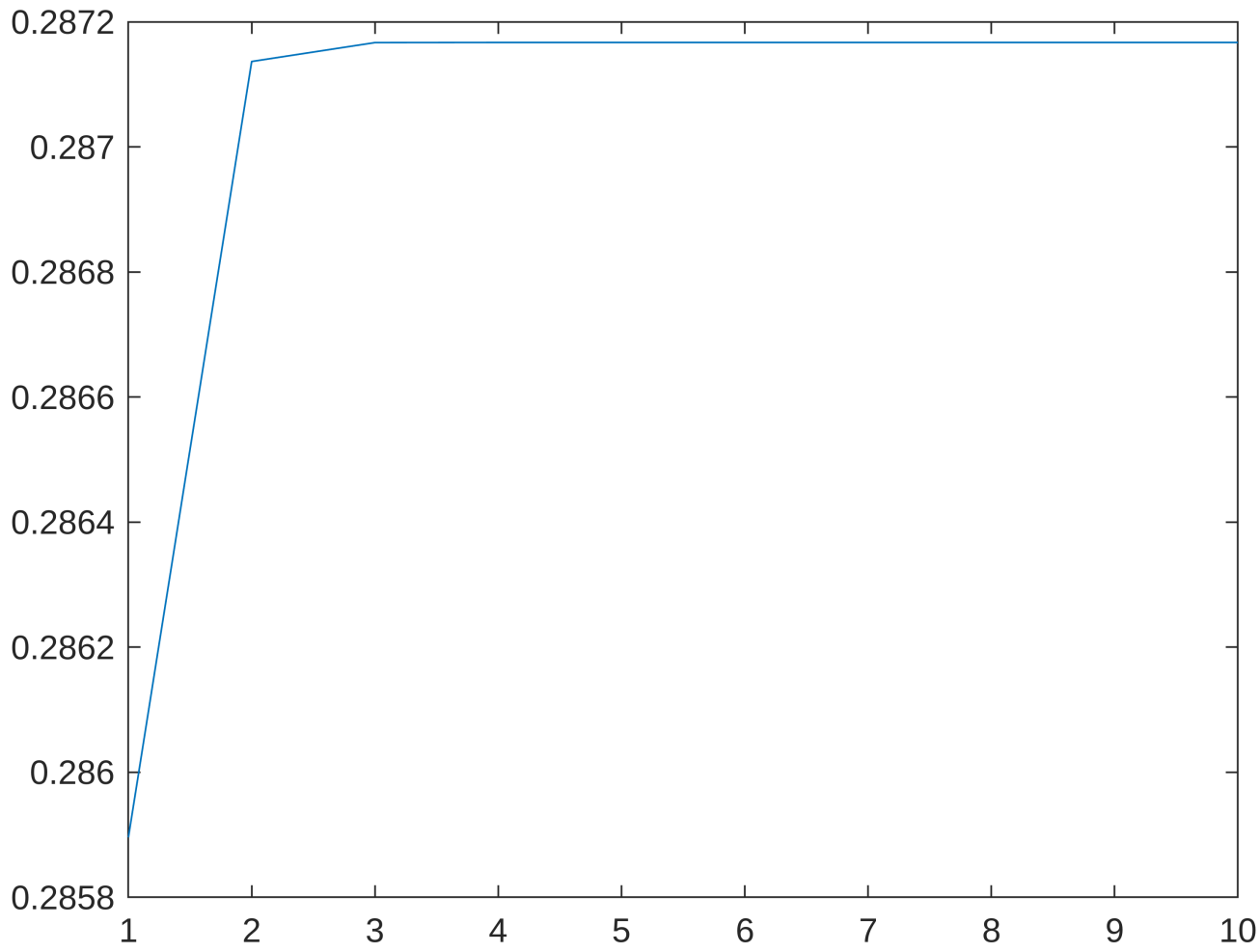
This is the residual for the starting point of a zero vector for x.

```
plot(rho)
```

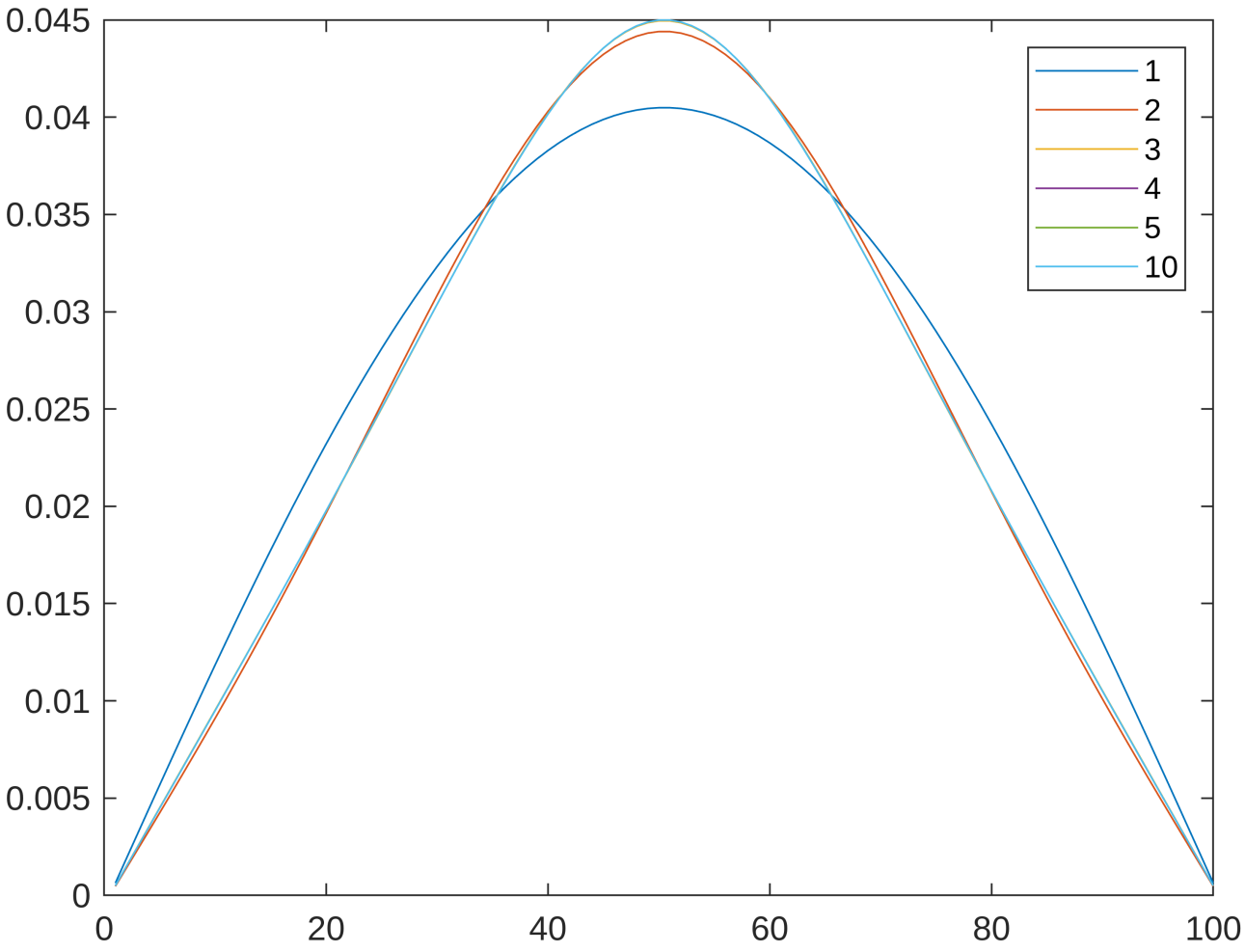


Note that the frst iteration has alrady significantly reduced the residual andafter the second iteration we do not see much improvement in the residual. CGLS is a fast converging iterative scheme with still a relatively low cost for each step. As well as looking at the residuals we should look at the norms of the solutions

```
plot(eta)
```




```
plot(X(:,[1:5,10]));legend('1','2','3','4','5','10' )
```



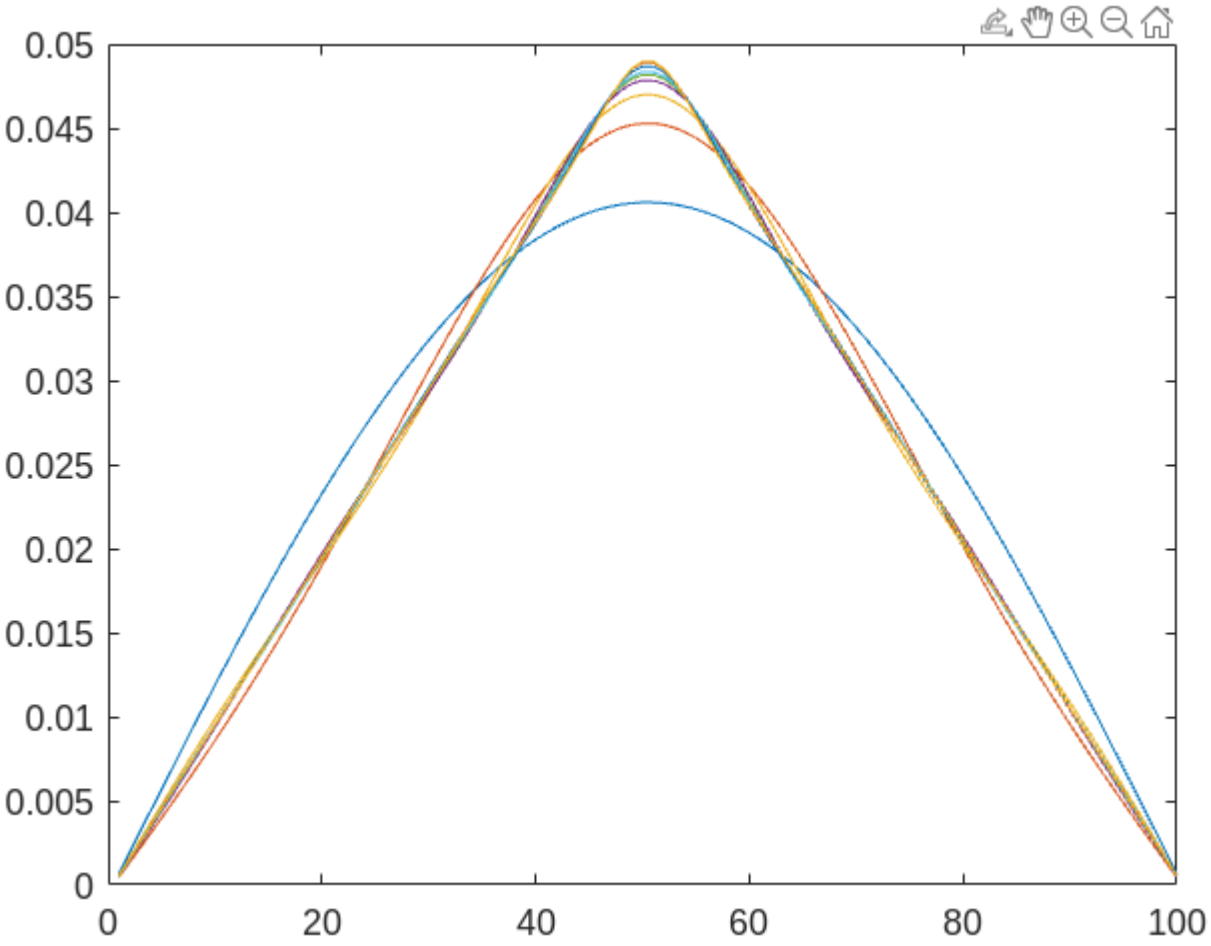
We see only a small improvement after iteration 2, but still a small improvement in the 'piontedness'. Sometimes iterative methods stopped early are used

as a substitute for more specific regularization. A problem with this is the the regularization should repressent an explicit a priori assumption. Using iterative methods for regularization means that the

a priori assumptions about the solution x depends on the measurement matrix A, which is generally not the case.

Never the less we can try CGLS on the matrix A rather than the augmented matrix Aa

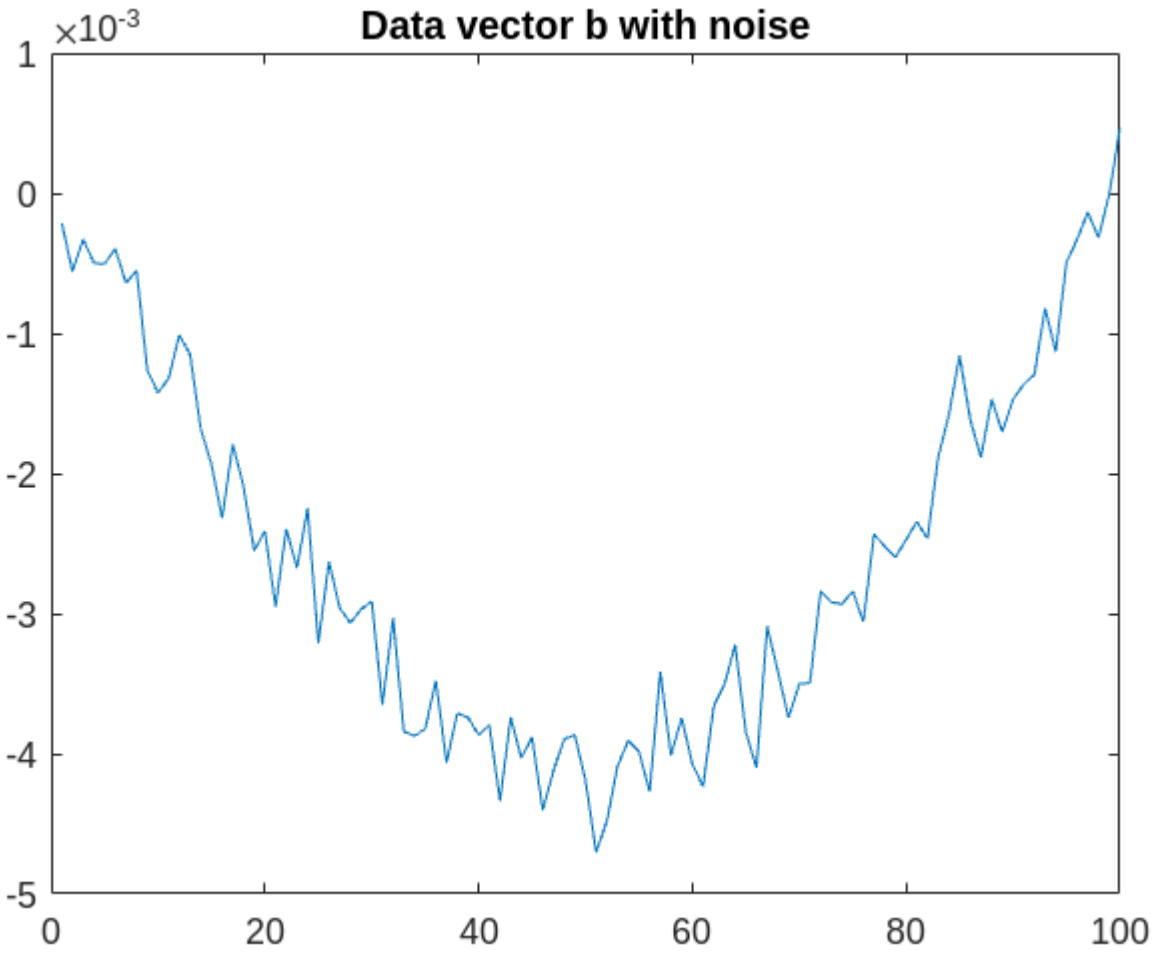
```
[X,rho,eta] = cgls(A,b,10);
 plot(X(:,1:10))
```



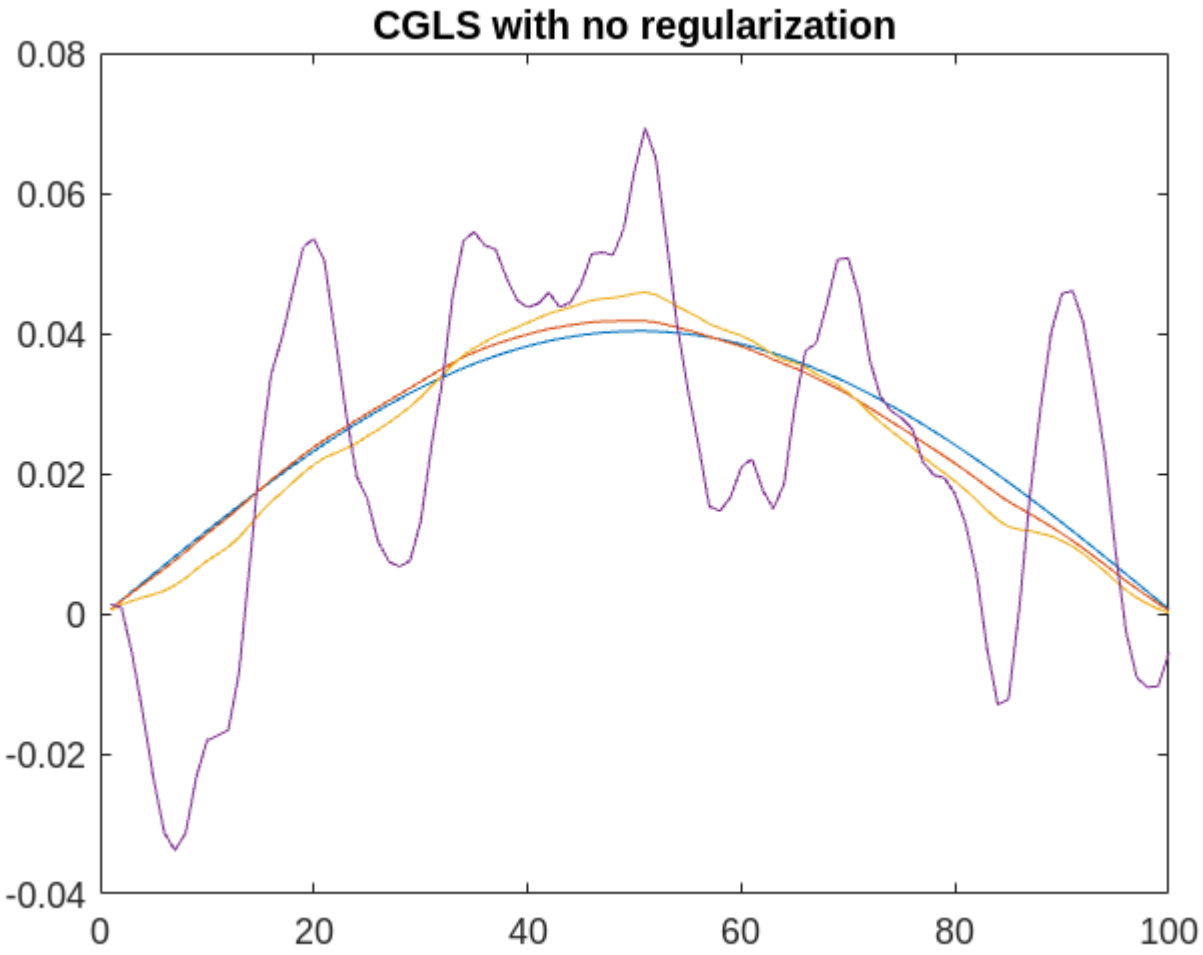
We see in this case not much difference fro Tikhonov regularization (and remember this is mildly ill posed so the regularization does not have as much effect as for a severely ill-posed problem), Also this is with the inverse cirime of no noise in the data.

Now let us try with noise

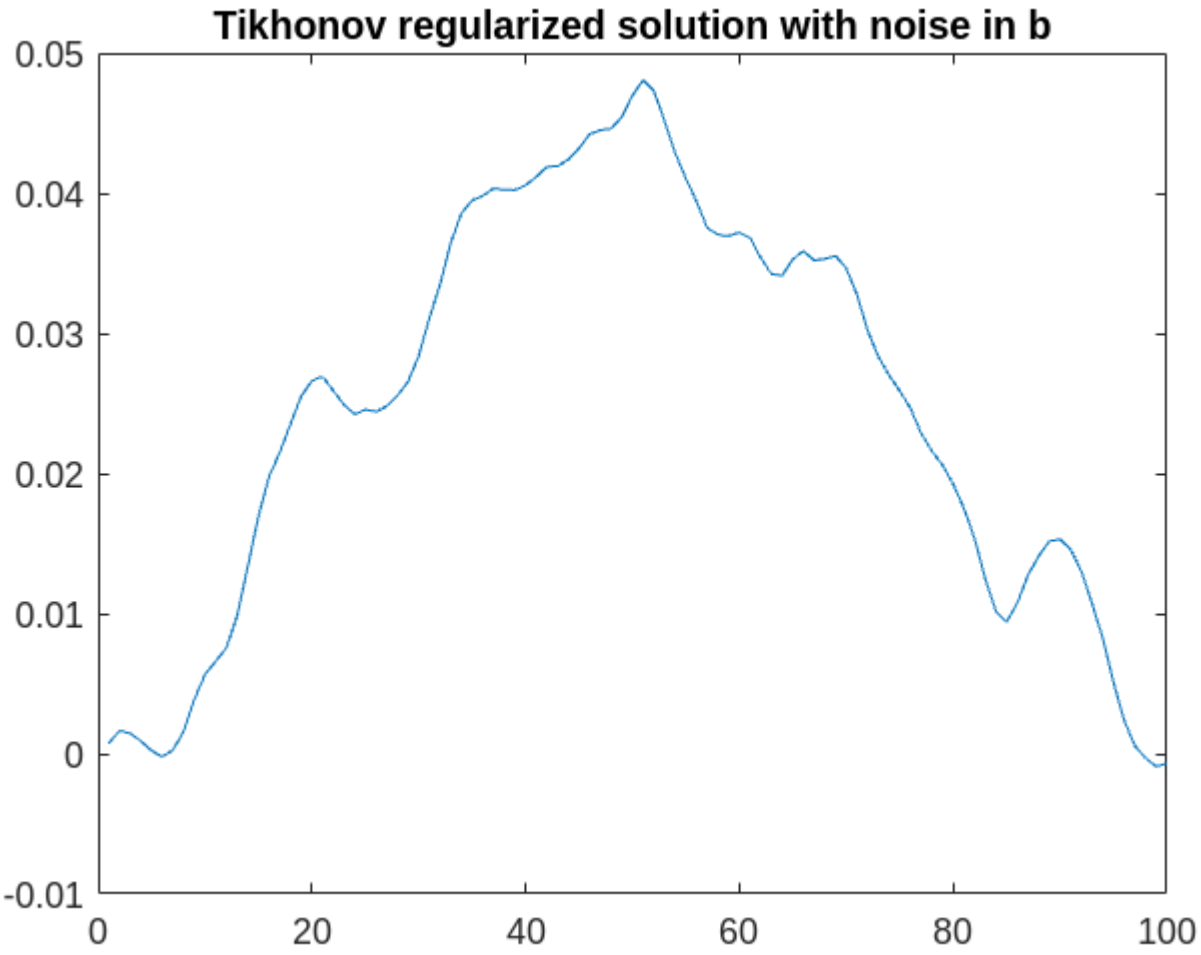
```
 bn = b + 0.01*norm(b)*randn(n,1);
[X,rho,eta] = cgls(A,bn,10);
plot(bn);title('Data vector b with noise')
```




```
plot(X(:,1:4));title('CGLS with no regularization')
```




```
bna=[bn;zeros(n,1)];
xtn= Aa\bna;
plot(xtn);
title('Tikhonov regularized solution with noise in b')
```



We see in this case with 1% noise or Tikhonov factor may be too small (look in the reg tools manual for the L-curve method for finding the Tikhnov factor). CGLS used for regularization

results in a gighly smooth solution. To understand this note CGLS is a Krylov subspace method and as the singular vectors of A are essentially a Fourier basis the CGLS algorith can be seen as solving for progressively higher frequency components.

Suggestion: Can the results of CGLS with no regularization be reproduced approximately by Tikhonov regularization with a larger Tihonov factor?

For more on iterative regularization see IRtools http://people.compute.dtu.dk/pcha/IRtools/

And for tutorials applying these methods to tomographic problems see Per Christian Hansen's " Lectures on Algebraic Iterative Reconstruction Methods - Theory and Experience" http://www2.compute.dtu.dk/~pcha/AIRtoolsII/Tutorial/ (using AIRTools).