

Hilbert matrices serve as test data for numerical software performing matrix inversion because they are so nearly singular, the more so as dimensions increase, and yet their determinants and inverses are computable accurately for comparison purposes from comparatively simple formulas. These applications will be illustrated hereunder by MATLAB programs.

Hilbert matrices arise from (weighted) least-squares fitting of polynomials $p(x)$ to arbitrary functions $y(x)$ over the interval $[0, 1]$ as follows (using MATLAB's notation):

Given the function $y(x)$, choose integers $K \geq 0$ (for the *weight*) and $N > 0$ (for the *degree*), and then find the column-vector $\mathbf{x} := [x_1; x_2; \dots; x_N]$ of coefficients x_j of the unique polynomial

$p(x) := \sum_{j=1}^N x_j \cdot x^{j-1}$ of degree N that minimize

$$\int_0^1 w(x) \cdot (p(x) - y(x))^2 \cdot dx.$$

(Most often $K = 0$.) The minimizing column vector \mathbf{x} satisfies the

$$\text{Normal Equations} \quad \mathbf{H}_N \cdot \mathbf{x} = \mathbf{b}_N$$

in which column $\mathbf{b}_N := [b_1; b_2; \dots; b_N]$ has elements $b_i := \int_0^1 w(x) \cdot x^{i-1+K} \cdot y(x) \cdot dx$, and matrix \mathbf{H}_N has elements $h_{ij} := \int_0^1 w(x) \cdot x^{i+j+K-2} \cdot dx = 1/(i+j+K-1)$ in row $\#i$ and column $\#j$. This \mathbf{H}_N is an N -by- N *Hilbert* matrix. MATLAB functions to generate $\mathbf{H} = \text{hilb}(N)$ in the most common case $K = 0$ have been programmed by Cleve Moler (his comes with MATLAB) and by Nicholas Higham, but their programs are not the best to generate test data. Here is why:

Hilbert matrix \mathbf{H}_N becomes "ill conditioned" because it approaches singular matrices rapidly as N increases. Consequently tiny perturbations, such as occur when reciprocals $1/(i+j-1)$ are rounded off, cause the computed inverse of a perturbed \mathbf{H}_N to change drastically, and more so as N increases. Hypersensitivity to perturbations is explored in class notes posted on the web at

www.cs.berkeley.edu/~wkahan/MathH110/GI1ite.pdf, [.../Jacobi.pdf](http://www.cs.berkeley.edu/~wkahan/MathH110/Jacobi.pdf),
[.../LstSqrs.pdf](http://www.cs.berkeley.edu/~wkahan/MathH110/LstSqrs.pdf), [.../NORMlite.pdf](http://www.cs.berkeley.edu/~wkahan/MathH110/NORMlite.pdf), and [.../~wkahan/Math128/FailMode.pdf](http://www.cs.berkeley.edu/~wkahan/Math128/FailMode.pdf).

To avoid perturbing the data my program `hilbl(N, K)` supplied below computes $\mathbf{Y} := L \cdot \mathbf{H}_N$ for the smallest integer $L := \text{LCM}(\{K+1, K+2, \dots, 2N+K-1\}) > 0$ whose quotients $L/(i+j+K-1)$ are all integers computed exactly, except when this L gets so big that it has to be rounded off.

```
function [Y, L] = hilbl(N,K)
%HILBL scaled N-by-N Hilbert matrix Y(i,j) = L/(i+j-1+K) of integers.
% Y = hilbl(N, K) = L*H is a scaled N-by-N Hilbert matrix whose elements
% Y(i,j) = L/(i+j-1+K) are all integers computed exactly because K is a
% nonnegative integer which, if omitted, defaults to K = 0.
% [Y, L] = hilb(N, K) delivers both Y = L*H and integer scale-factor L.
% Restriction: If N or K is too big, hilbl(N,K) should balk rather
% than deliver elements of Y wrong because of roundoff.
% See also DETHILBL, HILB and INVHILB. Needs GCD and LCM, q.v.,
% and maybe r0und all as modified by W. Kahan, 1996 - 22 Oct. 2008.

if ( nargin < 2 ), K = 0 ; end
if ( K < 0 ) | ( K ~= round(K) ), K
    error( ' hilbl(N, K) requires a nonnegative integer K .' )
end
```

```

% First compute the scale-factor L = lcm([K+1:2N-1+K]') accurately:
L = lcm([K+1: 2*N-1+K]') ; %... works only with W.K.'s version of lcm(...)
if ( isinf(L) & (K > 0) )
    L = lcm(lcm([K+1: 2*N-2+K]'), 2*N-1+K) ; end %... maybe rounded
if isinf(L)
    N_K = [N, K]
    error(' N or K is too big to compute hilbl(N,K) accurately.')
end

x = [1:N] ; Y = x(ones(N,1),:) ; Y = round(L./( Y + Y' + (K-1) )) ;
% Replace buggy round by r0und in 386-Matlab 3.5 & PC-Matlab 4.2 !

```

Ideally, if N or K is too big for $\text{hilbl}(N,K)$ to be computed accurately, the program should stop at an error message. It will after the command `system_dependent('setprecision', 64)` has been executed by PC MATLAB 6.5 to enable extra-precise accumulation of sufficiently small matrix products. Otherwise some versions of MATLAB on some hardware may fail to discover when $\text{hilbl}(N,K)$ is wrong. This is due to an unavoidable failure of `lcm` explained in the web posting www.cs.berkeley.edu/~wkahan/Math128/GCD5.pdf whence `gcd` and `lcm` were obtained. To preclude such failures, invoke $\text{hilbl}(N,K)$ only with integers N and K that are not too big. Tabulated below under each listed K is the biggest N that is not too big:

K	0	1	2	3	4	5	6	7	8	9
max N	21	21	21	20	20	19	19	18	18	17
K	10	11	12	13	14	15	16	17	18	19
max N	17	16	16	15	15	14	14	13	13	12
K	20	21	22	23	24	25	26	27	28	29
max N	12	11	11	10	10	11	10	10	9	9
K	30	31	32	33	34	35	36	37	38	39
max N	9	10	9	9	9	8	8	8	10	9
K	40	41	42	43	44	45	46	47	48	49
max N	9	9	8	8	8	7	8	8	7	7
K	50	51	52	53	54	55	56	57	58	59
max N	9	8	8	8	8	7	7	7	7	7
K	60	61	62	63	64	65	66	67	68	69
max N	7	7	7	7	6	7	6	6	7	7
K	70	71	72	73	74	75	76	77	78	79
max N	6	6	6	7	7	6	7	6	6	7
K	80	81	82	83	84	85	86	87	88	89
max N	6	6	6	6	6	6	6	7	7	6
K	90	91	92	93	94	95	96	97	98	99
max N	6	6	6	6	6	6	6	6	6	6

Let $Y_{N,K}$ and $L_{N,K}$ denote the results produced exactly by $[Y, L] = \text{hilbl}(N,K)$, and set $H_{N,K} := Y_{N,K}/L_{N,K}$; this N -by- N matrix, with elements $1/(i+j+K-1)$ in its row $\#i$ and column $\#j$, is a segment of the bigger Hilbert matrix H_{N+K} , and turns out to have only integer element in its inverse. Our next program will compute this inverse fairly accurately.

Computing the Inverse and Determinant of $H_{N,K}$:

In principle, $H^{-1} = L \cdot (L \cdot H)^{-1}$ could be computed using Matlab's `inv(...)` function, but this incurs rounding errors that cause at least about as much damage as would rounding off H 's elements. To avoid that damage, MATLAB provides a special function `invhilb(N)` for use instead of `inv(hilb(N))` to get an accurate inverse by computing a diagonal matrix D of integers for which $H^{-1} = D \cdot H \cdot D$. The program `invhilbl` here takes K into account too:

```
function W = invhilbl(N, K)
%INVHILBL accurate inverse of a segment of the Hilbert matrix
% invhilbl(N, K) is the inverse of an N-by-N matrix H whose elements
% are H(i,j) = 1/(i+j-1+K) EXACTLY. (Rounded elements may be obtained
% from [Y,L] = hilbl(N,K) via H = Y/L.) If omitted, K = 0; then
% H is the notoriously ill-conditioned Hilbert matrix. Like invhilb,
% invhilbl is computed not from Matlab's inv(H) but from an elegant
% (and faster) formula published by Sam Schechter in MTAC (1959) to
% compute the diagonal matrix D from which we get invhilbl = D*H*D.
% Its result is accurate despite ill-condition. If K is a nonnegative
% integer, all elements of abs(invhilbl(N, K)) are positive integers,
% except perhaps for roundoff that must interfere if (N,K) lies beyond
% (12,0) - (12,2) - (11,5) - (10,8) - (9,10) - (8,15) - (7,27) -
% (6,39) - (5,73) - (4,195) - (3,1287) - (2,262142) - (1, 2^53)
% for Matlab's 53-sig.-bits IEEE 754 floating-point arithmetic.
% See also hilb, invhilb, hilbl and dethilbl.
%
% W. Kahan, 1994 - 28 Oct. 2008
if ( nargin < 2 ), K = 0; end
u = K+1;
if (N == 1), W = u; return, end
if (N == 2), %... avoid unnecessarily big intermediate integers
    p = (u+1)*(u+1); q = -u*(u+1)*(u+2);
    W = [p*u, q; q, p*(u+2)]; return, end
x = [0:N-1]; y = (x+u).';
if ( K == 0 ),
    p = N; % ... The most common case is the fastest:
elseif ((0 < K) & (K == round(K)) & (K < N)),
    p = N*(N+1); for j = 2:K, p = (p/j)*(N+j); end
else, % ... The most general case is the slowest:
    p = u; for j = 1:N-1, p = (p/j)*(u+j); end
end
u = [ p, zeros(1,N-1) ];
for j = 1:N-1, u(j+1) = (( u(j)/(j+K))*(j-N) )/j*(N+j+K); end
W = (u.'*u)./( x(ones(N,1),:) + y(:,ones(1,N))) );
if ((0 <= K) & (K == round(K))), W = round(W); end
```

Roundoff poses the same threat to $\det(H)$ as to $\text{inv}(H)$, and the threat is avoided the same way, namely by computing the diagonal matrix D of integers that figures in the formula $H^{-1} = D \cdot H \cdot D$, whence we get integers $\det(H^{-1}) = \det(D)$ and $\det(Y) = \det(L \cdot H) = \det(L \cdot D^{-1})$:

```

function [dy, L, dhi] = dethilbl(N, K)
%DETHILBL integer determinants of a scaled Hilbert matrix and inverse
% [dy, L, dhi] computes integer-valued determinants related to the
% N-by-N scaled Hilbert matrix Y produced by [Y,L] = hilbl(N,K) :
% dy = det(Y) and dhi = det(inv(Y/L)) except that dethilb uses,
% instead of Matlab's det and inv, a method far less vulnerable
% to roundoff based upon a formula published by Sam Schechter in
% MTAC (1959). K must be a nonnegative integer which, if omitted,
% defaults to K = 0, and then H = Y/L is the familiar Hilbert
% matrix with elements H(i,j) = 1/(i+j-1). The integer scale factor
% L = lcm([K+1:2*N+K-1]) is computed accurately only if N and K
% are not too big; otherwise L may be wrong but dhi should be
% approximately right and then dy = (L^N)/dhi. See also INVHILBL.
% Needs W. K.'s modified gcd and lcm, and maybe r0und too.
%
% W. Kahan, 1966 - 26 Oct. 2008

if ( nargin < 2 ), K = 0 ; end
if ( K < 0 ) | ( K ~= round(K) ), N__K = [N, K]
    error( ' dethilbl(N,K) requires a nonnegative integer K .' ), end

u = K+1 ; m = 2*N+K-1 ; L = lcm([u: m]') ; %... Correct if finite.
if isinf(L) %... compute instead an approximate L :
    L = u*(u+1) ;
    for j = u+2:m , L = j*(L/gcd(L,j)) ; end
    % Now L = lcm(K+1, K+2, ..., 2*N+K-1) but for roundoff if too big.
    N__K__L = [N, K, L]
disp('WARNING: N and/or K are so big in [dy,L,dgi] = dethilbl(N, K) that')
disp('L may be wrong because of roundoff though dhi is approximately right.')
% This is the best I can do without IEEE Standard 754's Inexact Flag.
end
% x=[1:N]; H=x(ones(N,1),:); hilbl(N,K) = round(L./( H+H' + (K-1) )) ;

if ( K == 0 ) ,
    p = N ; % ... The most common case is the fastest:
elseif ( K < N ),
    p = N*(N+1) ; for j = 2:K , p = (p/j)*(N+j) ; end
else, % ... The most general case is the slowest:
    p = u ; for j = 1:N-1, p = (p/j)*(u+j) ; end
end
u = [ p, zeros(1,N-1) ] ;
for j = 1:N-1, u(j+1) = (( u(j)/(j+K))*(N-j) )/j*(N+j+K) ; end
dhi = round(prod(u)) ; dy = round(prod(L./u)) ;
% Replace buggy round in 386-Matlab 3.5 & PC-Matlab 4.2 by r0und .

```

Tests of hilbl, invhilbl, dethilbl

Before these programs were used to generate data to test other programs their own correctness had to be assayed. A test of $[Y, L] = \text{hilbl}(N, K)$ confirmed that all the elements of $V = L./Y$ were the correct small integers, the reciprocals of the elements of $H_{N,K}$, except when some of the biggest values of the scale factor L got rounded back to 53 sig. bits, in which case some of the entries in V differed from integers in their 53rd sig. bit.

$W = \text{invhilbl}(N, K)$ was tested by computing $\text{norm}([Y, L \cdot \text{eye}(N)] * [W; -\text{eye}(N)])$ to obtain $\|Y \cdot W - L \cdot I\|$ as one matrix product accumulated extra-precisely in 64 sig. bits on an old 68040-based Macintosh Quadra 950. Both norms vanished for small values of N and K , but beyond these 64 sig. bits were too few to hold the intermediate products of elements of W and Y , so roundoff contaminated $Y \cdot W$. Almost the same results were obtained from MATLAB 6.5 on an IBM PC after the command `system_dependent('setprecision', 64)`, without which roundoff contaminated results sooner and worse. Another test better indicative of (in)correctness in `invhilbl` was needed; it was constructed from the following observation:

Let column N -vector $\mathbf{u} := [1, 1, 1, \dots, 1]'$; then $s_{N,K} := \mathbf{u}' \cdot H_{N,K}^{-1} \cdot \mathbf{u} = N \cdot (N+K)$ after massive cancellation, the more so as N and K increase, and the computation of $s_{N,K}$ generates no intermediate sums bigger in magnitude than the biggest elements of $H_{N,K}^{-1}$ because their signs alternate. If $\mathbf{u}' \cdot W \cdot \mathbf{u} = N \cdot (N+K)$ then surely $W = H_{N,K}^{-1}$. This is how roundoff's interference was inferred for the documentation of `invhilbl`.

`[dy, L, dhi] = dethilbl(N, K)` was tested for a few small integers N and K by comparing its outputs with values of the determinants computed exactly by the automated algebra system DERIVE 4.1. run on an IBM PC. The outputs matched perfectly until they got so big that only their rounded values displayed by MATLAB to 15 sig. dec. could be compared; these matched in all but at worst the last digit displayed.

Tests of MATLAB's `inv`:

The accuracy of any estimate M of H_N^{-1} , no matter how MATLAB computes it, can now be assessed using arithmetic no more accurate than was used to compute M by comparing it with $W = \text{invhilbl}(N)$. What measure of (in)accuracy is suitable? One possibility is *elementwise*:

First compute $R = 2 \cdot (M - W) ./ (M + W)$, the array of symmetric differences from 1 of ratios of respective elements of M and W ; then $m = -\log(\max(\text{eps}/2, \max(\text{abs}(R(:)))))/\log(2)$ is the least number m of matching sig. bits between respective elements of M and W . No more than 53 can match; this is why `eps/2` appears there. See the first graph below.

This elementwise measure m is appropriate for inverses of Hilbert matrices because they have no elements much closer to zero than their neighbors. There are matrices for which elementwise measures of (in)accuracy are not appropriate; for these a better measure may be *normwise*, using $r := 2 \cdot \|M - W\| / \|M + W\|$ in place of $\max|R(:)|$ above. The choice of norm $\|\dots\|$ matters little for the inverses of Hilbert matrices but can matter crucially for others; for some examples see

www.cs.berkeley.edu/~wkahan/Math128/FailMode.pdf.

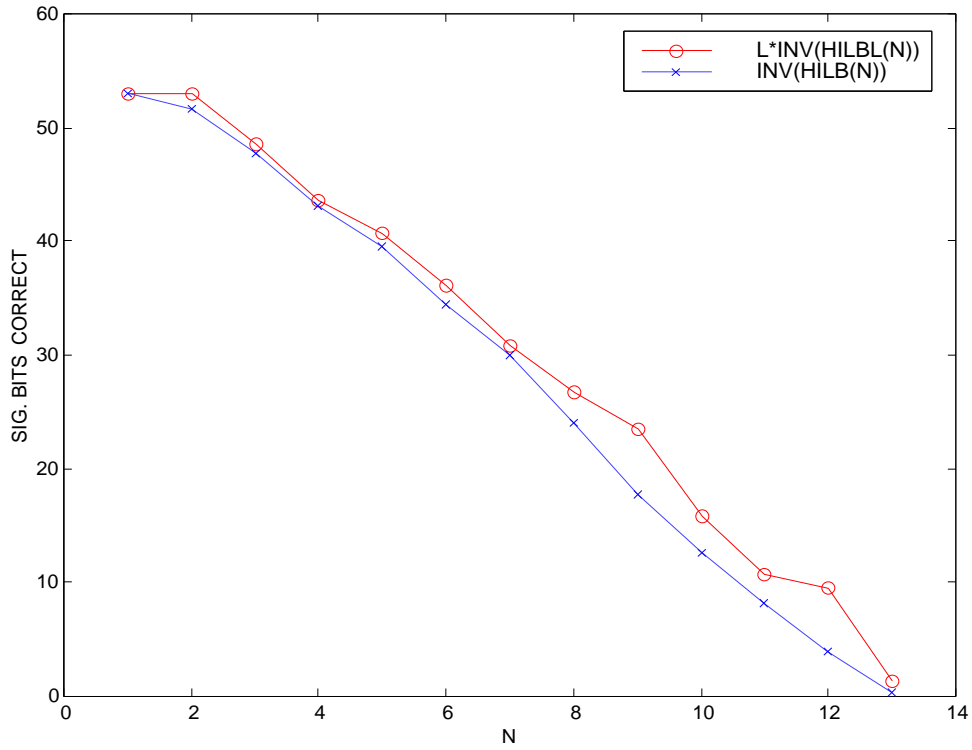
MATLAB's own `norm` was chosen for the second graph below; it plots

$$-\log(\max(\text{eps}/2, 2 \cdot \text{norm}(M - W) / \text{norm}(M + W)))/\log(2)$$

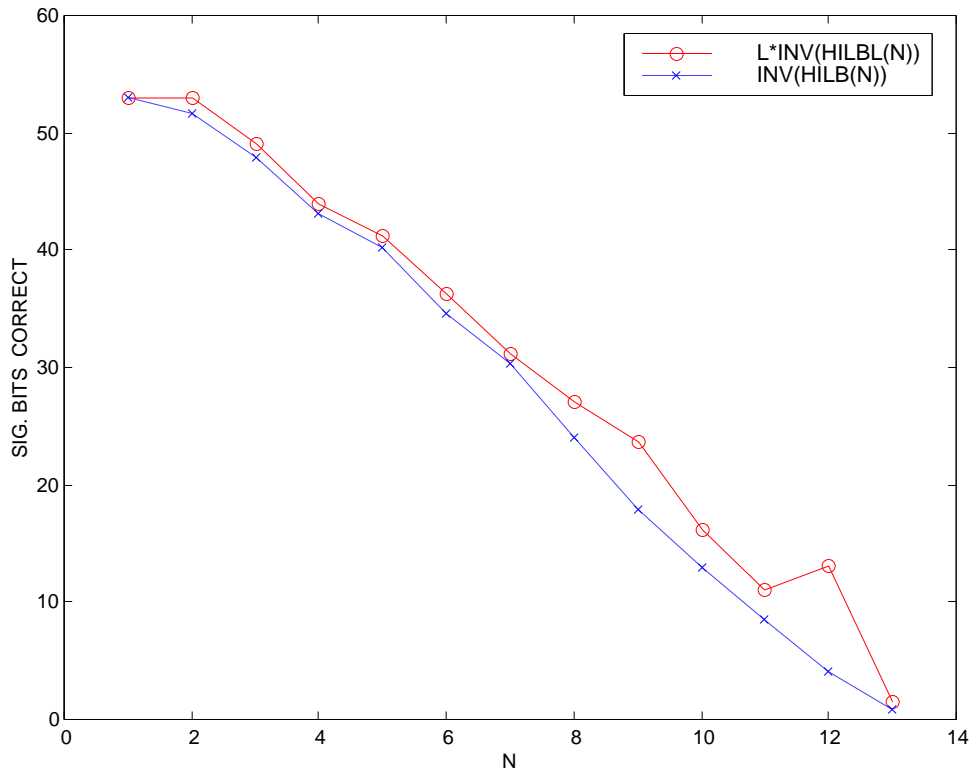
against N for $1 \leq N \leq 13$, beyond which `hilb(N)` is too nearly singular for MATLAB to invert.

Both graphs suggest that the effect of roundoff inside `inv(...)` is comparable with the damage done by rounding the fractional elements of H_N to MATLAB's working precision of 53 sig. bits.

Elementwise Accuracies of $L \cdot \text{inv}(\text{hilbl}(N))$ and $\text{inv}(\text{hilb}(N))$



Normwise Accuracies of $L \cdot \text{inv}(\text{hilbl}(N))$ and $\text{inv}(\text{hilb}(N))$



Exercise: Use tests like those above to compare with `invhilbl(N)` diverse estimates for H_N^{-1} computed by MATLAB from expressions like

```
inv(hilb(N)), round(inv(hilb(N)), L*inv(hilbl(N))), round(L*inv(hilbl(N))),
flipud(inv(fliplr(hilb(N))), round(flipud(inv(fliplr(hilb(N))))), ...
```

for $N = 1, 2, 3, \dots$ in turn to determine when each kind of estimate breaks down, and how much good or harm is done by different sources of roundoff, epilog, column order, ...

Exercise: Compare the output `dhi` of `[dy, L, dhi] = dethilbl(N)` with $1/\det(\text{hilb}(N))$ and its variations analogous to the previous exercise.

Variations Among Computed Inverses of Near-Singular Matrices

Recall that the singular matrix S nearest H is distant from it by $\|H-S\| = 1/\|H^{-1}\|$. (This is proved as Theorem 7 in the note "Huge Generalized Inverses of Rank-Deficient Matrices", <www.cs.berkeley.edu/~wkahan/MathH110/GI1ite.pdf>). The norm $\|\dots\|$ here is Matlab's `norm(...)`. Even a rough computation of H^{-1} reveals how near H is to singular; any estimate $M \approx H^{-1}$ provides an estimate $\|H-S\| \approx \|M\|$. How widely can estimates M vary?

Programs like Matlab's `inv(...)` compute inverses by Gaussian Elimination or, equivalently, triangular factorization in which roundoff causes `inv(H)` to differ from H^{-1} by not much more than $(H+H)^{-1}$ can differ if $\|H\| \approx N \cdot \|\text{roundoff}\|$, although often `inv(H) \approx (H+H)^{-1}` for any such small perturbation H . Here $\epsilon = \text{eps} = 1/2^{52}$ is Matlab's roundoff threshold. In other words, although a computed $M = \text{inv}(H)$ is not necessarily the inverse of any matrix $H+H$ differing from H by at most about N rounding errors in each element of H , the computed inverse M is not much farther from H^{-1} than the inverse $(H+H)^{-1}$ of such a perturbed H . And

$$(H+H)^{-1} - H^{-1} = -H^{-1} \cdot H \cdot (H+H)^{-1}, \text{ so}$$

$$\|(H+H)^{-1} - H^{-1}\| \approx \|H^{-1}\| \cdot \|H\| \cdot \|(H+H)^{-1}\| \approx N \cdot \|\text{roundoff}\| \cdot \|H\| \cdot \|(H+H)^{-1}\|.$$

Therefore any estimate M of H^{-1} obtained from `inv(...)` should satisfy roughly

$$\|M - H^{-1}\| \approx N \cdot \|\text{roundoff}\| \cdot \|M\|^2 \text{ provided } N \cdot \|\text{roundoff}\| \ll 1/\|M\| \approx \|H-S\|.$$

The last proviso says "H is much farther than N rounding errors from its nearest singular matrix S ."

But if $\|H-S\|$ is not much bigger than $N \cdot \|\text{roundoff}\|$, then $M = \text{inv}(H)$ may well be computed far too inaccurately for the bound upon $\|M - H^{-1}\|$ above to be trusted. Usually Matlab issues a

Warning: Matrix is close to singular or badly scaled.

Results may be inaccurate. RCOND = 3.572099e-17."

The displayed value `RCOND = 1/(||H||*||M||)` is MATLAB's estimate of $\|H-S\|/\|H\|$, but it may be wrong by an order of magnitude because $\|H+H-S\|$ may vary by that much or more as H varies by several rounding errors per element of H . Consequently $\|(H+H)^{-1}\| = 1/\|H+H-S\|$ may vary that much, and therefore so may $\|M\|$, and consequently so may M , depending upon the rounding errors occurring during its computation.

The last “consequently ...” follows from the observation that the norm of a matrix can change by no more than the norm of its change: $\|M+ \Delta M\| - \|M\| \leq \|\Delta M\|$; can you see why?

In summary, if H is too nearly singular, different computations may yield estimates M of H^{-1} that seem very different. However, provided all the estimates are no worse than $(H+ \Delta H)^{-1}$ for roundoff-like perturbations ΔH , the diverse estimates M usually have this in common:

They are all nearly scalar multiples one of another.

This is explained on pp. 3-4 of the note “Jacobi's Formula for the Derivative of a Determinant”,
<www.cs.berkeley.edu/~wkahan/MathH110/jacobi.pdf> .

An alternative explanation in terms of the singular values of $H+ \Delta H$ is more illuminating but requires a little more technical machinery; it's a story for another day.

Exercise: Use different formulas, like those in the previous two exercises, to compute diverse estimates M_1, M_2, \dots for inverses of N -by- N Hilbert matrices H with N barely big enough to elicit MATLAB's “close to singular” warning. To see how nearly two such estimates, say M and W , come to scalar multiples of each other, first estimate a scalar multiple z by computing, say, $z := \text{trace}(W \cdot M) / \text{trace}(W \cdot W)$, and then compute $t := \|M - z \cdot W\| / \|M + z \cdot W\|$. If N is so big that t is not tiny compared with $1 - z$, can you explain why? Reviewing the hypotheses from which the aforementioned note's conclusions were drawn may help, or ask MATLAB to estimate $N - \text{rank}(H)$.

Further Reading

“Tricks or Treats with the Hilbert Matrix” by Man-Duen Choi, pp. 301-312 in *Amer. Math. Monthly* **80** #5, May 1983, summarizes much of the lore about the Hilbert matrix, including its infinite-dimensional limit, plus an extensive list of references.