```
# Solving Bevington example 6.1 with Octave

printf( "Bevington Example 6.1\n" )
strftime ("%Y-%m-%d %H:%M:%S", localtime (time ()))

# design matrix
printf( "design matrix:\n" )
A = [ 1 1; 1 2; 1 3; 1 4; 1 5; 1 6; 1 7; 1 8; 1 9 ]

# Define data
printf( "data vector:\n" )
T = [15.6; 17.5; 36.6; 43.8; 58.2; 61.6; 64.2; 70.4; 98.8]

# solve least squares problem
printf( "least squares solution\n" )
printf( "xls = A \\ T:\n" )
xls = A \ T

# residual error vector
residual = A * xls - T;
printf( "residual error vector = A * xls - T\n" )

t2 = dot( residual, residual );
printf( "least total squared error t2 = residual . residual = %d\n", t2 )

# compute Gram matrix
W = transpose( A ) * A;

# invert Gram matrix
Winv = inv( W );
values = diag( Winv );

# measure design matrix
# m = rows, n = columns
[ m, n ] = size ( A );
g = sprintf('%d ', [ m, n ]);
fprintf('matrix dimensions: %s\n', g);

printf( "\ncompute error elements:\n" )
printf( "sigma = sqrt( t2 / ( m - n ) * values ):\n" )
sigma = sqrt( t2 / ( m - n ) * values )

printf( "\n#  #  #  Compare Octave values to exact values\n" )
printf( "\nFit parameters\n" )

printf( "\nerror in intercept and slope values\n" )
printf( "numericError = xls - [ 1733 / 360; 1129 / 120 ]\n" )
numericError = xls - [ 1733 / 360; 1129 / 120 ]
```

```

printf( "\nerror in intercept and slope values in machine epsilon\n" )
printf( "epsError = numericError ./ eps( 1.0 )\n" )
epsError = numericError ./ eps( 1.0 )

printf( "\nError parameters\n" )
printf( "\nintercept and slope sigmas\n" )
printf( "numericError = sigma - sqrt( [ 108297055; 3419907 ] / 35 ) / 360\n" )
numericError = sigma - sqrt( [ 108297055; 3419907 ] / 35 ) / 360

printf( "\nerror in intercept and slope sigmas in machine epsilon\n" )
printf( "epsError = numericError ./ eps( 1.0 )\n" )
epsError = numericError ./ eps( 1.0 )

## dantopa@Quaxolotl.local:least-squares $ pwd
## /Volumes/T7-Touch/repos/github/jop/octave/genesis/least-squares
## dantopa@Quaxolotl.local:least-squares $ octave-cli wtf.m
## Bevington Example 6.1
## ans = 2022-09-06 20:41:57
## design matrix:
## A =

##    1   1
##    1   2
##    1   3
##    1   4
##    1   5
##    1   6
##    1   7
##    1   8
##    1   9

## data vector:
## T =

##    15.600
##    17.500
##    36.600
##    43.800
##    58.200
##    61.600
##    64.200
##    70.400
##    98.800

## least squares solution
## xls = A \ T:
```

```
## xls =

##    4.8139
##    9.4083

## residual error vector = A * xls - T
## least total squared error t2 = residual . residual = 316.658
## matrix dimensions: 9 2

## compute error elements:
## sigma = sqrt( t2 / ( m - n ) * values ):
## sigma =

##    4.8862
##    0.8683


## #  #  #  Compare Octave values to exact values

## Fit parameters

## error in intercept and slope values
## numericError = xls - [ 1733 / 360; 1129 / 120 ]
## numericError =

##   -1.0658e-14
##    1.7764e-15


## error in intercept and slope values in machine epsilon
## epsError = numericError ./ eps( 1.0 )
## epsError =

##   -48
##     8


## Error parameters

## intercept and slope sigmas
## numericError = sigma - sqrt( [ 108297055; 3419907 ] / 35 ) / 360
## numericError =

##    8.8818e-16
##    1.1102e-16


## error in intercept and slope sigmas in machine epsilon
## epsError = numericError ./ eps( 1.0 )
## epsError =

##    4.0000
##    0.5000

```