```
#!/opt/local/bin/julia
# Solving Bevington example 6.1 with Julia
# https://stackoverflow.com/questions/22240581/running-julia-jl-files
println( "Bevington Example 6.1" )
using Dates
println( now( ) )

# Define mesh
A = [ 1 1; 1 2; 1 3; 1 4; 1 5; 1 6; 1 7; 1 8; 1 9 ];
println( "")
println( "design matrix:")
println( "A = ", A )


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

# least squares solution
xls = A \ T
println( "")
println( "least squares solution vector:" )
println( "xls = A \\ T" )
println( "xls = ", xls )

# residual error vector
residual = A * xls - T
println( "" )
println( "residual error vector:" )
println( "residual = A * xls - T" )
println( "residual = ", residual )

# load linear algebra package
println( "" )
println( "using LinearAlgebra" )

# least total squared error
using LinearAlgebra
t2 = dot( residual, residual )
println( "" )
println( "t2 = ", t2 )

# compute Gram matrix
W = transpose( A ) * A
println( "" )
println( "compute Gram matrix:" )
```

```
println( "W = transpose( A ) * A" )
println( "W = ", W )

# invert Gram matrix
Winv = inv( W )
println( "" )
println( "invert Gram matrix:" )
println( "Winv = inv( W )" )
println( "Winv = ", Winv )

# harvest diagonal elements
values = diag( Winv )
println( "" )
println( "harvest diagonal elements of Winv:" )
println( "values = diag( Winv )" )
println( "values = ", values )

# rows and columns
( m, n ) =  size( A )
println( "")
println( "matrix dimensions ( rows, cols ):" )
println( "( m, n ) = ", ( m, n ) )

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

println( "" )
println( "#  #  #  Compare Julia values to exact values" )

# compare julia to exact solution
#   solution values
numericError = xls - [ 1733 / 360; 1129 / 120 ]
println( "" )
println( "error in intercept and slope values" )
println( "numericError = xls - [ 1733 / 360; 1129 / 120 ]" )
println( "numericError = ", numericError )

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

#   sigma values
```

```
numericError = sigma - sqrt.( [ 108297055; 3419907 ] / 35 ) / 360
println( "" )
println( "error in intercept and slope sigmas" )
println( "numericError = sigma - sqrt.( [ 108297055; 3419907 ] / 35 ) /
360" )
println( "numericError = ", numericError )

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

# dantopa@Quaxolotl.local:least-squares $ julia least-squares.jl
# Bevington Example 6.1
# 2022-08-29T18:10:15.791
#
# design matrix:
# A = [1 1; 1 2; 1 3; 1 4; 1 5; 1 6; 1 7; 1 8; 1 9]
#
# matrix dimensions ( rows, cols ):
# ( m, n ) = (9, 2)
#
# data vector:
# T = [15.6, 17.5, 36.6, 43.8, 58.2, 61.6, 64.2, 70.4, 98.8]
#
# least squares solution vector:
# xls = A \ T
# xls = [4.813888888888871, 9.408333333333335]
#
# residual error vector:
# residual = A * xls - T
# residual = [-1.3777777777777924, 6.130555555555542, -3.561111111111124,
-1.3527777777777885, -6.344444444444463, -0.3361111111111228,
6.472222222222214, 9.680555555555543, -9.311111111111117]
#
# using LinearAlgebra
#
# t2 = 316.65805555555556
#
# compute Gram matrix:
# W = transpose( A ) * A
# W = [9 45; 45 285]
#
# invert Gram matrix:
# Winv = inv( W )
# Winv = [0.5277777777777778 -0.08333333333333334; -0.08333333333333333
0.016666666666666666]
```

```
#
# harvest diagonal elements of Winv:
# values = diag( Winv )
# values = [0.5277777777777778, 0.0166666666666666]
#
# compute error elements:
# sigma = sqrt( t2 / ( m - n ) * values )
# sigma = [4.886206312183355, 0.8683016476563611]
#
# #  #  #  Compare Julia values to exact values
#
# error in intercept and slope values
# numericError = xls - [ 1733 / 360; 1129 / 120 ]
# numericError = [-1.7763568394002505e-14, 1.7763568394002505e-15]
#
# error in intercept and slope values in machine epsilon
# epsError = numericError ./ eps( 1.0 )
# epsError = [-80.0, 8.0]
#
# error in intercept and slope sigmas
# numericError = sigma - sqrt.( [ 108297055; 3419907 ] / 35 ) / 360
# numericError = [8.881784197001252e-16, 1.1102230246251565e-16]
#
# error in intercept and slope sigmas in machine epsilon
# epsError = numericError ./ eps( 1.0 )
# epsError = [4.0, 0.5]

```