

```
1 #!/opt/local/bin/julia
2 # Solving Bevington example 6.1 with Julia
3 # https://stackoverflow.com/questions/22240581/running-julia-jl-files
4 println( "Bevington Example 6.1" )
5 using Dates
6 println( now( ) )
7
8 # Define mesh
9 A = [ 1 1; 1 2; 1 3; 1 4; 1 5; 1 6; 1 7; 1 8; 1 9 ];
10 println( "" )
11 println( "design matrix:" )
12 println( "A = ", A )
13
14
15 # Define data
16 T = [15.6; 17.5; 36.6; 43.8; 58.2; 61.6; 64.2; 70.4; 98.8];
17 println( "" )
18 println( "data vector:" )
19 println( "T = ", T )
20
21 # least squares solution
22 xls = A \ T
23 println( "" )
24 println( "least squares solution vector:" )
25 println( "xls = A \ T" )
26 println( "xls = ", xls )
27
28 # residual error vector
29 residual = A * xls - T
30 println( "" )
31 println( "residual error vector:" )
32 println( "residual = A * xls - T" )
33 println( "residual = ", residual )
34
35 # load linear algebra package
36 println( "" )
37 println( "using LinearAlgebra" )
38
39 # least total squared error
40 using LinearAlgebra
41 t2 = dot( residual, residual )
42 println( "" )
43 println( "t2 = ", t2 )
44
45 # compute Gram matrix
46 W = transpose( A ) * A
47 println( "" )
48 println( "compute Gram matrix:" )
```

```
49 println( "W = transpose( A ) * A" )
50 println( "W = ", W )
51
52 # invert Gram matrix
53 Winv = inv( W )
54 println( "" )
55 println( "invert Gram matrix:" )
56 println( "Winv = inv( W )" )
57 println( "Winv = ", Winv )
58
59 # harvest diagonal elements
60 values = diag( Winv )
61 println( "" )
62 println( "harvest diagonal elements of Winv:" )
63 println( "values = diag( Winv )" )
64 println( "values = ", values )
65
66 # rows and columns
67 ( m, n ) = size( A )
68 println( "" )
69 println( "matrix dimensions ( rows, cols ):" )
70 println( "( m, n ) = ", ( m, n ) )
71
72 # compute error elements
73 sigma = sqrt.( t2 / ( m - n ) * values )
74 println( "" )
75 println( "compute error elements:" )
76 println( "sigma = sqrt( t2 / ( m - n ) * values )" )
77 println( "sigma = ", sigma )
78
79 println( "" )
80 println( "# # # Compare Julia values to exact values" )
81
82 # compare julia to exact solution
83 # solution values
84 numericError = xls - [ 1733 / 360; 1129 / 120 ]
85 println( "" )
86 println( "error in intercept and slope values" )
87 println( "numericError = xls - [ 1733 / 360; 1129 / 120 ]" )
88 println( "numericError = ", numericError )
89
90 epsError = numericError ./ eps( 1.0 )
91 println( "" )
92 println( "error in intercept and slope values in machine epsilon" )
93 println( "epsError = numericError ./ eps( 1.0 )" )
94 println( "epsError = ", epsError )
95
96 # sigma values
```

```
97 numericError = sigma - sqrt.( [ 108297055; 3419907 ] / 35 ) / 360
98 println( "" )
99 println( "error in intercept and slope sigmas" )
100 println( "numericError = sigma - sqrt.( [ 108297055; 3419907 ] / 35 ) /
... 360" )
101 println( "numericError = ", numericError )
102
103 epsError = numericError ./ eps( 1.0 )
104 println( "" )
105 println( "error in intercept and slope sigmas in machine epsilon" )
106 println( "epsError = numericError ./ eps( 1.0 )" )
107 println( "epsError = ", epsError )
108
109 # dantopa@Quaxolotl.local:least-squares $ julia least-squares.jl
110 # Bevington Example 6.1
111 # 2022-08-29T18:10:15.791
112 #
113 # design matrix:
114 # A = [1 1; 1 2; 1 3; 1 4; 1 5; 1 6; 1 7; 1 8; 1 9]
115 #
116 # matrix dimensions ( rows, cols ):
117 # ( m, n ) = (9, 2)
118 #
119 # data vector:
120 # T = [15.6, 17.5, 36.6, 43.8, 58.2, 61.6, 64.2, 70.4, 98.8]
121 #
122 # least squares solution vector:
123 # xls = A \ T
124 # xls = [4.813888888888871, 9.408333333333335]
125 #
126 # residual error vector:
127 # residual = A * xls - T
128 # residual = [-1.3777777777777924, 6.130555555555542, -3.561111111111124,
... -1.3527777777777885, -6.344444444444463, -0.3361111111111228,
... 6.472222222222214, 9.680555555555543, -9.311111111111117]
129 #
130 # using LinearAlgebra
131 #
132 # t2 = 316.65805555555556
133 #
134 # compute Gram matrix:
135 # W = transpose( A ) * A
136 # W = [9 45; 45 285]
137 #
138 # invert Gram matrix:
139 # Winv = inv( W )
140 # Winv = [0.5277777777777778 -0.08333333333333334; -0.08333333333333333
... 0.016666666666666666]
```

```
141 #
142 # harvest diagonal elements of Winv:
143 # values = diag( Winv )
144 # values = [0.5277777777777778, 0.016666666666666666]
145 #
146 # compute error elements:
147 # sigma = sqrt( t2 / ( m - n ) * values )
148 # sigma = [4.886206312183355, 0.8683016476563611]
149 #
150 # # # Compare Julia values to exact values
151 #
152 # error in intercept and slope values
153 # numericError = xls - [ 1733 / 360; 1129 / 120 ]
154 # numericError = [-1.7763568394002505e-14, 1.7763568394002505e-15]
155 #
156 # error in intercept and slope values in machine epsilon
157 # epsError = numericError ./ eps( 1.0 )
158 # epsError = [-80.0, 8.0]
159 #
160 # error in intercept and slope sigmas
161 # numericError = sigma - sqrt.( [ 108297055; 3419907 ] / 35 ) / 360
162 # numericError = [8.881784197001252e-16, 1.1102230246251565e-16]
163 #
164 # error in intercept and slope sigmas in machine epsilon
165 # epsError = numericError ./ eps( 1.0 )
166 # epsError = [4.0, 0.5]
167
```