```c
#include <R.h>
#include <Rmath.h>
#include <R_ext/BLAS.h>
#include <R_ext/Lapack.h>
#include <R_ext/Utils.h>

void sqmatmul(double *A, double *B, int *n, double *res) {
    double sum;

    for(int i=0; i<*n; i++) {
        for(int j=0; j<*n; j++) {
            sum = 0.0;
            for(int k=0; k<*n; k++) {
                sum = sum + A[i+(*n)*k] * B[k+(*n)*j];
            }
            res[i+(*n)*j] = sum;
        }
    }
}

/* Example of sqmatmul

> A = matrix(rnorm(9), 3)
> B = matrix(rnorm(9), 3)
> x = .C("sqmatmul", as.double(A), as.double(B), as.integer(3), res=as.double(matrix(0, nrow=3, ncol=3)))
> x$res
[1]  0.8598526 -0.5747655  0.5686713 -1.1030245  2.6063841 -2.3720913
[7]  0.9367630 -1.8974657  1.6676321
> matrix(x$res, 3)
           [,1]      [,2]      [,3]
[1,]  0.8598526 -1.103025  0.936763
[2,] -0.5747655  2.606384 -1.897466
[3,]  0.5686713 -2.372091  1.667632
> A %*% B
           [,1]      [,2]      [,3]
[1,]  0.8598526 -1.103025  0.936763
[2,] -0.5747655  2.606384 -1.897466
[3,]  0.5686713 -2.372091  1.667632

*/

// ================================================== \\
```

```
46: void rw2d(int *n, int *xRes, int *yRes) {
47:     GetRNGstate();
48:
49:     int *xRes_p, *yRes_p, lastx, lasty;
50:     xRes_p = xRes;
51:     yRes_p = yRes;
52:     lastx = lasty = *(xRes_p++) = *(yRes_p++) = 0;
53:
54:     for(int i=1; i<*n; i++) {
55:         if(runif(0.0, 1.0) < 0.5) {
56:             lastx = *(xRes_p++) = lastx + sign(runif(-1.0, 1.0));
57:             *(yRes_p++) = lasty;
58:         } else {
59:             *(xRes_p++) = lastx;
60:             lasty = *(yRes_p++) = lasty + sign(runif(-1.0, 1.0));
61:         }
62:     }
63:
64:     PutRNGstate();
65: }
66:
67: /* Example of rw2d
68:
69: > walk = .C("rw2d", as.integer(20), x=as.integer(vector("numeric", 20)), y=as.integer(vector("numeric", 20)))
70: > walk$x
71:  [1]  0  0 -1 -1 -2 -1  0  0  0 -1 -1 -1 -2 -3 -4 -5 -5 -4 -5 -6
72: > walk$y
73:  [1]  0 -1 -1  0  0  0  0 -1  0  0 -1  0  0  0  0  0  1  1  1  1
74: > walk = .C("rw2d", as.integer(10000), x=as.integer(vector("numeric", 10000)), y=as.integer(vector("numeric", 10000
)))
75: > plot(walk$x, walk$y, type="l", col="#0000CC55")
76:
77: */
78:
79: // ================================================== \\
80:
81:
82:
83:
84:
85:
86:
87:
88:
89:
```

```
 90: void linmod(double *X, double *y, int *n, int *p, double *res) {
 91:     double *XtX, *Xty;
 92:     XtX = (double *) R_alloc(*p * *p, sizeof(double));
 93:     Xty = (double *) R_alloc(*p, sizeof(double));
 94:
 95:     // X^t X
 96:     char transa = 'T', transb = 'N';
 97:     double alpha = 1.0, beta = 0.0;
 98:     F77_CALL(dgemm)(&transa, &transb, p, p, n, &alpha, X, n, X, n, &beta, XtX, p);
 99:
100:     // X^t y
101:     int inc = 1;
102:     F77_CALL(dgemv)(&transa, n, p, &alpha, X, n, y, &inc, &beta, Xty, &inc);
103:
104:     // (X^t X)^{-1}
105:     // N.B: LAPACK requires the LU decomposition first, then can do the inverse
106:     // So, LU decompose ...
107:     int *ipiv, info;
108:     ipiv = (int *) R_alloc(*p, sizeof(int));
109:     F77_CALL(dgetrf)(p, p, XtX, p, ipiv, &info);
110:     // ... then invert
111:     // 1st get the optimal workspace size from LAPACK
112:     double *work;
113:     work = (double *) R_alloc(1, sizeof(double));
114:     int lwork = -1;
115:     F77_CALL(dgetri)(p, XtX, p, ipiv, work, &lwork, &info);
116:     // 2nd assign this workspace and do the inverse
117:     lwork = (int) *work;
118:     work = (double *) R_alloc(lwork, sizeof(double));
119:     F77_CALL(dgetri)(p, XtX, p, ipiv, work, &lwork, &info);
120:
121:     // Finally: (X^t X)^{-1} X^t y
122:     F77_CALL(dgemv)(&transb, p, p, &alpha, XtX, p, Xty, &inc, &beta, res, &inc);
123: }
124:
125: /* Example of linmod
126:
127: > # Generate
128: > X = matrix(runif(1000,-5,5), nrow=100, ncol=10)
129: > beta = rexp(10,2)
130: > intercept = rexp(1,4)
131: > y = intercept + X %*% beta + rnorm(100)
132: >
133: > # Fit
134: > reslm = lm(y ~ X)
```

```
  135: > res = .C("linmod", c(rep(1,100), as.double(X)), as.double(y), as.integer(100), as.integer(11), coef=as.double(1:1
1))
  136: >
  137: > # True -vs- fit
  138: > c(intercept, beta)
  139:  [1] 0.24384466 0.02762992 0.46147411 0.57257377 0.13092734 0.01180250
  140:  [7] 0.73116633 0.63927495 0.49887682 0.10780993 0.20213635
  141: > reslm$coefficients
  142: (Intercept)          X1          X2          X3          X4          X5
  143:  0.31205612  0.03818310  0.43924384  0.54862464  0.16904831  0.01388965
  144:          X6          X7          X8          X9         X10
  145:  0.73151975  0.64913967  0.42918589  0.13642267  0.16593451
  146: > res$coef
  147:  [1] 0.31205612 0.03818310 0.43924384 0.54862464 0.16904831 0.01388965
  148:  [7] 0.73151975 0.64913967 0.42918589 0.13642267 0.16593451
  149:
  150: */
```