Complex Wishart sampling

Ray Hinton

Expectations

In the following two examples, the sample mean and true mean are compared when two different methods are used to generate complex Wishart matrices.

First, the method using a product of matrices is shown. The true mean and sample mean are close in Frobenius distance, and their entries appear similar.

```
# Frobenius distance
norm(df*Sigma - avgW, "F")
```

```
[1] 0.9593366
```

```
# true mean
round(df*Sigma, 3)
```

```
                [,1]            [,2]            [,3]            [,4]
[1,]  51.918+ 0.000i -11.404-16.387i -32.573-37.767i  16.797+ 7.337i
[2,] -11.404+16.387i  14.281+ 0.000i  24.559+10.457i  -2.703+ 1.385i
[3,] -32.573+37.767i  24.559-10.457i 173.866+ 0.000i -19.671-30.567i
[4,]  16.797- 7.337i  -2.703- 1.385i -19.671+30.567i  19.827+ 0.000i
```

```
# sample mean
round(avgW, 3)
```

```
                [,1]            [,2]            [,3]            [,4]
[1,]  51.849+ 0.000i -11.389-16.387i -32.461-37.573i  16.800+ 7.319i
[2,] -11.389+16.387i  14.280+ 0.000i  24.482+10.410i  -2.710+ 1.399i
[3,] -32.461+37.573i  24.482-10.410i 173.001+ 0.000i -19.628-30.421i
[4,]  16.800- 7.319i  -2.710- 1.399i -19.628+30.421i  19.778+ 0.000i
```

Next, the method based on the Cholesky decomposition is shown. Again, the true means and sample means are similar.

```
# Frobenius distance
norm(df*Sigma - avgW, "F")
```

```
[1] 0.5050177
```

```
# true mean
round(df*Sigma, 3)
```

```
                [,1]            [,2]            [,3]            [,4]
[1,]  51.918+ 0.000i -11.404-16.387i -32.573-37.767i  16.797+ 7.337i
[2,] -11.404+16.387i  14.281+ 0.000i  24.559+10.457i  -2.703+ 1.385i
[3,] -32.573+37.767i  24.559-10.457i 173.866+ 0.000i -19.671-30.567i
[4,]  16.797- 7.337i  -2.703- 1.385i -19.671+30.567i  19.827+ 0.000i
```

```
# sample mean
round(avgW, 3)
```

```
                [,1]            [,2]            [,3]            [,4]
[1,]  52.055+ 0.000i -11.452-16.411i -32.754-37.697i  16.931+ 7.375i
[2,] -11.452+16.411i  14.307+ 0.000i  24.564+10.455i  -2.722+ 1.407i
[3,] -32.754+37.697i  24.564-10.455i 174.058+ 0.000i -19.824-30.655i
[4,]  16.931- 7.375i  -2.722- 1.407i -19.824+30.655i  19.950+ 0.000i
```

Speed

There are two parts of the algorithm that can be slow. First, the matrix square root of the Σ parameter is needed. This can be computed just using the eigendecomposition via `eigen`. Alternatively, `EigenR::Eigen_sqrt` is faster in some cases. Second, sampling the Cholesky decomposition of the complex Wishart matrix is generally faster as the degrees of freedom increases.

For small matrices and low degrees of freedom, finding $\Sigma^{1/2}$ with `EigenR::Eigen_sqrt` greatly speeds up the sampler. Using the Cholesky decomposition sampling technique does not make as much of a difference.

The following times are in microseconds for `p = 8`.

```r
df <- 100
p <- 8

Sigma <- rcomplex_wishart(p + 1, diag(p))

mb_res <- microbenchmark(
    rcomplex_wishart(df, Sigma, useEigenR = FALSE, byCholesky = FALSE),
    rcomplex_wishart(df, Sigma, useEigenR = FALSE, byCholesky = TRUE),
    rcomplex_wishart(df, Sigma, useEigenR = TRUE, byCholesky = FALSE),
    rcomplex_wishart(df, Sigma, useEigenR = TRUE, byCholesky = TRUE),
    times = 1000
) |> summary()

mb_res$expr <- c("FALSE, FALSE", "FALSE, TRUE", "TRUE, FALSE", "TRUE, TRUE")
names(mb_res)[1] <- c("EigenR, byChol")

cat(paste0("df = ", df, ", p = ", p))
```

```
df = 100, p = 8
```

```r
knitr::kable(mb_res, digits = 3)
```

EigenR, byChol	min	lq	mean	median	uq	max	neval
FALSE, FALSE	338.928	416.659	648.890	511.708	774.944	4642.590	1000
FALSE, TRUE	306.442	387.097	611.252	481.089	740.568	5735.023	1000
TRUE, FALSE	80.678	90.098	136.938	101.998	168.229	685.329	1000
TRUE, TRUE	42.682	56.654	101.319	71.406	131.447	2360.553	1000

With a larger degrees of freedom, `df = 1000`, using the Cholesky decomposition also results in a speed up, regardless of how the matrix square root is found.

```
df = 1000, p = 8
```

EigenR, byChol	min	lq	mean	median	uq	max	neval
FALSE, FALSE	783.915	870.781	1175.097	953.114	1303.448	4581.723	1000
FALSE, TRUE	309.284	397.941	545.834	455.643	629.077	1809.689	1000
TRUE, FALSE	517.809	538.770	741.213	558.704	798.628	6263.800	1000

EigenR, byChol	min	lq	mean	median	uq	max	neval
TRUE, TRUE	42.937	58.036	90.606	71.321	100.697	1654.859	1000

With larger matrices, `p = 64` below, the `EigenR::Eigen_sqrt` function is slower than using `eigen` to compute the matrix square root. Using the Cholesky decomposition is faster in either case.

The following are in milliseconds.

`df = 100, p = 64`

EigenR, byChol	min	lq	mean	median	uq	max	neval
FALSE, FALSE	2.611	2.738	3.503	2.982	3.994	8.456	100
FALSE, TRUE	2.443	2.531	3.115	2.645	3.359	7.280	100
TRUE, FALSE	3.530	3.585	4.139	3.652	3.908	10.599	100
TRUE, TRUE	3.319	3.429	3.837	3.466	3.674	7.572	100

`df = 1000, p = 64`

EigenR, byChol	min	lq	mean	median	uq	max	neval
FALSE, FALSE	6.425	6.653	9.528	7.870	11.292	53.597	100
FALSE, TRUE	2.420	2.614	3.162	2.809	3.554	5.796	100
TRUE, FALSE	7.310	7.497	10.056	8.892	11.550	22.147	100
TRUE, TRUE	3.444	3.542	4.135	3.627	4.423	7.657	100

With even higher degrees of freedom, the Cholesky decomposition is clearly faster.

`df = 10000, p = 64`

EigenR, byChol	min	lq	mean	median	uq	max	neval
FALSE, FALSE	47.935	52.195	59.754	56.873	63.654	95.190	100
FALSE, TRUE	2.493	3.004	3.921	3.407	4.330	12.022	100
TRUE, FALSE	48.263	51.104	59.340	56.608	63.326	97.877	100
TRUE, TRUE	3.398	3.750	5.009	3.964	4.307	40.335	100