# Definitions

are number of observations, bootstrap replications, degrees of freedom of hypothesis, fixed effect (FE) dummies, non-FE independent variables

number of groups defined by cluster variable combination

multiplier for cluster variable combination : optional small-sample correction factor

index of bootstrapping cluster combination

wild weights (Rademacher, Webb, etc.)

independent variables data matrix, excluding dummies for the fixed effect variable, if any

dependent variable data matrix

observation weights

true parameter vector

via left-multiplication, sums columns of a data matrix by groups defined in clustering combination like Mata panelsum(); also, left-multiplication by duplicates rows of a matrix of height to height ; and is the indicator matrix whose entry is or according to whether observations and are in the same cluster under clustering combination .

, same for fixed-effect grouping

|  |  |
| --- | --- |
| via left-multiplication, partials out fixed effects. Idempotent. |  |
|  |  |

The operator “:\*” indicates elementwise-multiplication of a matrix by a matrix or of a matrix’s columns or rows by a conformable vector. It has lower precedence than ordinary multiplication. Likewise for “” and “”. If is a column vector, then

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

# Wild bootstrap

|  |  |
| --- | --- |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
| (all wild-weighted residuals) |  |

|  |  |
| --- | --- |
|  |  |
|  |  |

The bootstrap statistics are centered around (which is zero if the null is imposed). So the numerators of the Wald statistic are

|  |  |
| --- | --- |
|  |  |

The last form is designed to run faster by postponing involvement of the large matrix . For the same reason, the term should be computed from right to left in order to work at first only with small matrices. And the computations can be performed without constructing the large matrices, , and . Left multiplication by can be effected by demeaning within fixed effect groups; by and by summing columns clusterwise; and by by performing “” (and storing the observation weights in a column, not a matrix).

The cluster-robust parameter covariance estimate for replication can be written

where is the proxy for under clustering combination , i.e., the matrix whose entry is if observations and and are in different -clusters and otherwise. The Wald denominator for replication is therefore

|  |  |
| --- | --- |
|  |  |

where

|  |  |
| --- | --- |
|  |  |

For one-way clustering, small-sample corrections are superfluous because they apply proportionally to the test statistic and the simulated distribution. I.e., we can set .

# When

If , the terms on both sides of the “” operator in (15) are column vectors. We can then naturally extend (15) to all replications at once:

|  |  |
| --- | --- |
|  |  |

Then, the denominators (14) for all replications are consolidated in

|  |  |
| --- | --- |
|  |  |

The width of (17) derives from that of , which is . For speed, our computational strategy is therefore to delay the involvement of in the denominator too. To develop it, we expand (17) to express it explicitly in terms of , and then rearrange. Starting with the definition of in (16),

|  |  |
| --- | --- |
|  |  |

In words, the columns of the large () matrix are multiplied element-wise by the column vector and its rows elementwise by the row vector . The result is summed row-wise by group and column-wise by group. The resulting small matrix, here labeled , is multiplied by the wild weights .

Note what this says about the mathematical nature of the Wald denominators. For each replication , the denominator component is , the norm of with respect to a particular quadratic form. In multiway clustering, the full denominator is . So another way to write the full set of denominators is

|  |  |
| --- | --- |
|  |  |

The advantage of this computational recipe is that it further defers involvement of , until after the sum over , which thus only involves small matrices.

While this expression for achieves the goal of deferring involvement of as long as possible, it is still a problematic computational recipe because is large. Fortunately, we can avoid explicitly computing it. Expanding (18) with (2),

|  |  |
| --- | --- |
|  |  |

Focus first on the second major term in (20), rewriting it:

Those two factors can be computed without creating width- matrices. So they are computationally manageable. As for the first major term in (20), we rewrite the problematically isolated second instance of as

|  |  |
| --- | --- |
|  |  |

where is a data matrix of the fixed-effect dummies and is a diagonal matrix holding each observation’s weight share within its fixed-effect group (or, if observations are not weighted, the reciprocal of the number of observations in its group). Substituting into that first term and expanding,

In general, if is a column vector, then —the sort of expression found just above in the right term—works out to be a *crosstab*: each entry is the sum of those elements of belonging to -clustering group and fixed effect group . I will symbolize it . It can be computed directly without constructing the large, sparse matrix . The left term too is a crosstab, of with respect to the clustering groups and . Call it .

Drawing these threads together, rewrite (20) as

|  |  |
| --- | --- |
|  |  |

This formulation achieves the goals of avoiding construction of any matrices of width and delaying computational involvement of , of width . If there are no fixed effects, the middle term, which originates in the partialling out of those effects, is not needed.

## Alternative definition of

The middle term is also not needed in a special case…

# When the fixed-effect groups are subsets (or the same as) the bootstrapping clusters

In this case, , which appears in the middle term of (22) has non-zero elements where the sum is within a given fixed effect group. But within a fixed effect group, the average residual is zero. So the middle term of (22) is . Other than computing and (the latter for (9))—i.e., partialling out the fixed effects through demeaning—no special adjustment is needed for fixed effects.

# Incorporating observation weights by multiplying data by

**where is diagonal.**

# Inverting the bootstrap test to construct confidence intervals while imposing the null

Davidson and MacKinnon (2010, section 6) recommend inverting the bootstrap test to construct confidence intervals, at least when . This requires varying in the wild bootstrap test procedure to explore where the resulting *p* value crosses an threshold such as 0.05. This procedure might seem computationally intensive, but here too efficiencies are to be had. A key observation is that affects each *linearly*.

To see this, note first that by the last line of (7), depends linearly on , which, when , is simply the bottom element of . In turn, enters the wild bootstrap only via (8), the equation for the initial fit . Rearranging (8) when the null is imposed,

|  |  |
| --- | --- |
|  |  |

When we are iteratively varying , the two major terms in this equation, outside of itself, need be computed only once.

Likewise rewrite (9),

|  |  |
| --- | --- |
|  |  |

…and the same comment applies. And this technique carries forward to the entirety of (13) and (22)—the Wald numerators and the denominator components .

As a result, we can write and expand the inner sum in (19):

The last form requires less computation since the three sums need be computed just once.

# When

When the null hypothesis is joint, I have found that it is most efficient (in Mata) to perform the calculations above for each row of in turn (and all replications at once). This produces a small set of matrices , where the index corresponds to the rows of and , i.e. the component hypotheses. For each combination , the computation

|  |  |
| --- | --- |
|  |  |

then yields a row vector consisting of all the entries of the denominators, of which there are . Because these denominators are symmetric, the computation (25) needs to be performed times for the combinations of and . Then, the denominator for each replication is constructed by extracting the th elements from the row vectors (25).

# Score bootstrap

For OLS, the score bootstrap (Kline and Santos 2012) computes the score statistic for each replication using the wild-weighted residuals from the initial fit (in which the null is optionally imposed):

The score bootstrap takes the Hessian and scores as its primary inputs. Residuals as a concept are banished. The middle term of (22), which arises from handling fixed effects, refers to the residuals in isolation, so it is not feasible when transplanted to the score bootstrap. So for the score bootstrap, we will assume :

is computed as in (19) and (22), with two modifications. First, since we work with rather than , we do not subtract from the first to obtain the second. This corresponds to dropping the third term of (22). On the other hand, because , unlike **,** is not a vector of residuals from a re-fitted model, and is not a matrix of observation-level scores from a re-fitted model, the latter are demeaned before being used to estimate the variance. Let be the -vector of 1’s, and let be the -vector of 1’s. We demean the scores via

We replace (15) with

is the just the vector whose entries are the share of each -cluster in the weight (or observation) total.

If , then is column vector and we vectorize over the -indexed replications:

|  |  |
| --- | --- |
|  |  |

Much as in (18) and (22), we rearrange :

Substituting into (26),

We handle the case as discussed above.

For models other than OLS, including nonlinear, ML-fitted models, we replace above with , which is algebraically equivalent but where “ as a unit represents the observation-level scores from the model fit and is the model-based parameter covariance matrix, i.e., the one derived directly from the Hessian.

# When is large

If is large enough or some fixed , it is best to go back to (16) and (17) and apply them more directly, at the potential expense of having to explode to .

Generalizing to , elements of the -indexed components of the denominators are gathered in

In the extreme but common case (the “robust” case), this simplifies to

which we can implement more quickly as

with (assuming we’re not score-bootstrapping, so that is well-defined in isolation). The latter performs the same number of FLOPs, but cross() is faster than colsum( :\* ).

To analyze the computational trade-offs in this alternate approach, let’s assume , and there are no fixed effects or weights. In the main approach, documented earlier, we compute , where

By (37) in the paper,

where and

We take , as pre-computed, since they’re needed in both methods. Also, both multiply a matrix by so that can be discarded. Then the multiply-adds for the first method is approximately .

For the method, we need to factor in the cost of the rightmost matrix multiplication in computing . The multiply-add count for is then about .

Favor “granular” method if

Generally the granular method does not look good because its distinctive runtime components are linear in . Maybe it’s better for small , but how much do we care about that? But, conditioning on special handling for the “robust” case, generalizing to granular doesn’t complicate the code much.

**Optimization for classical tests— is all 1s**

Optimize (63) when and third term of (62) drops out.

Actually the second term drops out too because is just the sum of the FE-demeaned errors within each FE group.

And we need to demean. So we have

Likewise for numerators, eq (56) becomes

where means colsum().

**Direct computation of cross-over points for each bootstrap stat?**

We can think of each bootstrap replication as producing a value of 0 or 1. Overall value is the mean of each of those. So all we need from each bootstrap replication is the two cross-over points from 0 to 1 for each replication. Consider case

where

Looking at the construction of in (62), doesn’t seem like there’s a shortcut/benefit to building this stacked object explicitly. The problem is that we can’t make a third dimension. If we could, then it would all click. So once again it appears best to make the manually constructed dimension.

Can write

If the and are linear in then each test statistic and replication thereof is a ratio of quartic expressions in the . Equating them and solving for the path seems hard. Instead, just work to interpolate denominator entries

In first version, need to compute the three sums (, , ) for each . Can vectorize these objects over . For the third, store in a matrix whose rows are coefficients on , ,

version of that is

But we need to be able to handle version because an AR test on one endogenous variable will have number of instruments.

Vectorize over :

where , first block for unit increase in , second for unit increase in .

version of that is

With regard to analytically solving cross-over point, focus on case

In addition the roots must be real, and the two sides of must have the same sign (since we really solved ):

If , the bootstrap while and the real static will lie fully outside the distribution at the extremes. is probably not guaranteed for multiway clustering.