Sample Size Calculation for Pilot Trials in Stata

Xiangji Ying

August 6, 2025

This document is to help researchers calculate sample sizes for external pilot randomized controlled trials using Stata. It serves as a companion application to the forthcoming article titled "Determining sample size for pilot trials: A tutorial" (Ying X, et al. BMJ 2025;390:e083405. doi:10.1136/bmj-2024-083405). For details on the methods, examples, and sample size tables, please refer to the paper. Unless otherwise specified, all calculations refer to the total number of participants that will need to be enrolled in the pilot trial.

1 Estimate a feasibility parameter (Box 1)

1.1 Example 1

When estimating a binary feasibility parameter (e.g., recruitment uptake), the pilot sample size is calculated to achieve a specified confidence interval width for a single proportion at a stated confidence level. While multiple methods exist for estimating confidence intervals, the Wilson method is recommended for small sample sizes (n < 40).

[1]:
```
// * Wilson method * //

*program drop propci_wilson
program define propci_wilson
    syntax , Proportion(real) Width(real) [ Level(real 95) ONEsided ]

    // Parse input
    local p = `proportion'                // Estimated proportion
    local desired_width = `width'         // Desired CI width
    local level = `level'                 // Confidence level

    // Determine if one-sided or two-sided CI
    if "`onesided'" != "" {
        // One-sided CI uses different z-value
        local z = invnormal(1 - (1 - `level'/100))
        local ci_type = "one-sided"
    }
    else {
        // Two-sided CI (default)
        local z = invnormal(1 - (1 - `level'/100) / 2)
        local ci_type = "two-sided"
    }
```

```
    // Initial guess using Wald approximation
    local n = (`z' * sqrt(`p' * (1 - `p')) / (`desired_width' / 2))^2

    // Iterative solution for Wilson method
    local tolerance = 0.0000000001
    local max_iter = 100
    local iter = 0
    local converged = 0

    while (`iter' < `max_iter' & !`converged') {
        local p1 = `p' + (`z'^2)/(2*`n')
        local p2 = sqrt((`p'*(1-`p')/`n') + (`z'^2)/(4*`n'^2))
        local denom = 1 + `z'^2/`n'
                local current_width = 2*`z'*`p2'/`denom'

        if (abs(`current_width' - `desired_width') < `tolerance') {
            local converged = 1
        }
        else {
            // Adjust n proportionally
            local n = `n' * (`current_width' / `desired_width')^2
        }

        local ++iter
    }

    // Display results
    if (`converged') {
        display as text "Sample size calculation for Proportion (Wilson CI)"
        display as text "--------------------------------------"
        display as text "Expected proportion: " as result %6.3f `p'
        display as text "Desired CI width: " as result %6.3f `desired_width'
        display as text "Confidence level: " as result %6.1f `level' "%"
        display as text "CI type: " as result "`ci_type'"
        display as text "Required sample size: " as result %12.4f `n'
        display as text "Achieved CI width: " as result %6.4f `current_width'
    }
    else {
        display as error "No optimal sample size found within the range."
    }
end
```

[2]:

```
// Example usage:
propci_wilson, proportion(0.1) width(0.2) level(95) onesided
```

```
Sample size calculation for Proportion (Wilson CI)
-------------------------------------
Expected proportion:  0.100
Desired CI width:  0.200
Confidence level:   95.0%
CI type: one-sided
Required sample size:      25.7589
Achieved CI width: 0.2000
```

1.2 Example 2

When estimating a Poisson feasibility parameter (e.g., recruitment rate), the pilot sample size is calculated to achieve a specified confidence interval width for a single Poisson rate at a stated confidence level. While several methods exist for estimating confidence intervals for Poisson rates, two approaches are commonly recommended: (1) The Score method: Generally preferred for most situations; (2) The Exact method: Most reliable for small Poisson rate (<= 5).

[3]:
```
// * Score method * //

*program drop poissonci_score

program define poissonci_score
    syntax , Rate(real) /// Expected Poisson rate (events per time unit)
             Width(real) /// Desired total CI width
             [ Level(real 95) ] /// Confidence level (default 95%)
             [ ONEsided ] /// One-sided CI (default is two-sided)

    // Adjust alpha for one-sided CI
    if "`onesided'" != "" {
        local alpha = (100-`level')/100
    }
    else {
        local alpha = (100-`level')/200 // Split alpha for two-sided CI
    }

    // Convert confidence level to chi-square value
    local chi2 = invchi2(1, 1-`alpha'*2)

    // Initialize search for required time units
    // Start with rough normal approximation as initial guess
    local z = invnormal(1-`alpha')
    local T = (2*`z'/`width')^2 * `rate'

    // Iterative solution for T using score interval
    local done = 0
    local iter = 0
    local maxiter = 100
    local tolerance = 0.0000000001
```

```
    while (!`done' & `iter' < `maxiter') {
        // Calculate score interval bounds for current T
        local lambda = `rate'*`T'
        local L = (`lambda' + `chi2'/2 - sqrt(`chi2'^2/4 + `chi2'*`lambda'))/`T'
        local U = (`lambda' + `chi2'/2 + sqrt(`chi2'^2/4 + `chi2'*`lambda'))/`T'

        // Check if width matches desired width
        local current_width = `U' - `L'
        local diff = abs(`current_width' - `width')

        if (`diff' < `tolerance') {
            local done = 1
        }
        else {
            // Adjust T proportionally
            local T = `T' * (`current_width'/`width')
        }
        local ++iter
    }

    // Display results
    if (`done') {
        display as text "Observation units calculation for Poisson rate (Score␣
↪CI)"
        display as text "------------------------------------------------"
        display as text "Expected rate (events per unit): " as result %6.3f␣
↪`rate'
        display as text "Desired CI width: " as result %6.3f `width'
        display as text "Required units of observation: " as result %12.4f `T'
        display as text "Achieved CI width: " as result %6.3f `current_width'
        if "`onesided'" != "" {
            display as text "One-sided CI: " as result "Yes"
        }
        else {
            display as text "One-sided CI: " as result "No"
        }
    }
    else {
        display as error "No optimal units found within the range."
    }
end
```

[4]:
```
// Example usage:

poissonci_score, rate(10) width(6) level(95)
*poissonci_score, rate(10) width(6) level(95) onesided
```

```
Observation units calculation for Poisson rate (Score CI)
------------------------------------------------
Expected rate (events per unit): 10.000
Desired CI width:  6.000
Required units of observation:       4.3623
Achieved CI width:  6.000
One-sided CI: No
```

[5]:
```
// * Exact method * //

*program drop poissonci_exact
program define poissonci_exact, rclass
    syntax, rate(real) width(real) level(real)

    // Define input parameters
    local rate = `rate'          // Mean rate (events per time unit)
    local width = `width'        // Desired CI width
    local level = `level'        // Confidence level

    // Initialize variables to hold the optimal T and smallest difference
    local optimal_T = -1
    local min_diff = .

    // Loop over potential values for T
    forvalues T = 1(0.01)100 {

        // Calculate expected events
        local events = round(`rate' * `T')

        // Calculate CI
        quietly cii means `T' `events', poisson level(`level')
        local lower = r(lb)
        local upper = r(ub)
        local current_width = `upper' - `lower'

        // Calculate the difference between current_width and desired width
        local diff = abs(`current_width' - `width')

        // Check if this is the smallest difference found so far
        if (`min_diff' == .) | (`diff' < `min_diff' & `current_width' <=␣
↪`width') {
            local min_diff = `diff'
            local optimal_T = `T'
        }
    }

    // Display the optimal T with the smallest difference
```

```
    if `optimal_T' != -1 {
        display as text "Observation units calculation for Poisson rate (Exact␣
↪CI)"
        display as text "------------------------------------------------"
        display as text "Expected rate (events per unit): " as result %6.3f␣
↪`rate'
        display as text "Desired CI width: " as result %6.3f `width'
        display as text "Required units of observation: " as result %12.4f␣
↪`optimal_T'
        display as text "Achieved CI width: " as result %6.3f `current_width'
    }
    else {
        display "No optimal units found within the range."
    }
end
```

[6]:
```
// Example usage:

poissonci_exact,  rate(10) width(6) level(95)
* poissonci_exact,  rate(10) width(6) level(90) // one-sided 95%CI
```

```
Observation units calculation for Poisson rate (Exact CI)
------------------------------------------------
Expected rate (events per unit): 10.000
Desired CI width:  6.000
Required units of observation:       4.6200
Achieved CI width:  1.250
```

2 Test feasibility progression criteria (Box 2)

2.1 Example 1

To test progression criteria based on binary feasibility parameters (e.g., recruitment uptake), the pilot sample size is calculated to determine whether a population proportion (goal threshold, H1) significantly differs from a hypothesized value (minimum threshold, H0).

To determine if a proportion is less than or greater than a specified value, two commonly recommended tests are the normal approximation proportion test with continuity correction and the binomial exact test. Both methods are considered conservative compared to normal approximation without continuity correction. The binomial exact test often results in larger sample sizes compared to the normal approximation proportion test with continuity correction.

In Stata, the calculation based on proportion test can be conveniently applied using the `power oneproportion` command. To specify the use of continuity correction, simply include the "continuity" option.

[7]:
```
// Proportion test with continuity correction
power oneproportion 0.2 0.5, alpha(0.05) beta(0.05) onesided continuity
```

```
Estimated sample size for a one-sample proportion test
Score z test
Ho: p = p0  versus  Ha: p > p0

Study parameters:

        alpha =    0.0500
         beta =    0.0500
        delta =    0.3000
           p0 =    0.2000
           pa =    0.5000

Estimated sample size:

            N =        28
```

2.2 Example 2

To test progression criteria based on Poisson feasibility parameters (e.g., recruitment rate), the pilot sample size is calculated to assess whether a population Poisson rate (goal threshold, (H1)) significantly differs from a hypothesized value (minimum threshold, (H0)).

To determine if a Poisson rate is less than or greater than a specified value, one can use the exact test for Poisson rate.

[8]:
```
*program drop poisson_exact

program define poisson_exact, rclass
    syntax, lambda1(real) lambda0(real) [Alpha(real 0.05) Beta(real 0.1)]

    // Initialize parameters
    local d = 1
    local found = 0

    // Loop until we find valid n
    while `found' == 0 {
        // Calculate bounds using chi-square distribution
        local lower = invchi2tail(2 * `d', `beta') / (2 * `lambda1')
        local upper = invchi2tail(2 * `d', 1 - `alpha') / (2 * `lambda0')

        // Check if interval contains at least one integer
        if ceil(`lower') <= floor(`upper') {
            local n = `lower'
            local found = 1
        }
        else {
            local d = `d' + 1
```

```
        }

        // Add safety break to prevent infinite loop
        if `d' > 1000 {
            display as error "No solution found within 1000 iterations"
            exit 498
        }
    }

    // Return and display results
    local n_ceil = ceil(`n')
    display as text "Required number of observation units (n) = " as result␣
↪`n_ceil'
    *display as text "Degree of freedom = " as result `d'
    display as text "Note: Result rounded up to the nearest integer."

    return scalar n = `n_ceil'
    return scalar d = `d'
    return scalar lower = `lower'
    return scalar upper = `upper'
end
```

[9]:
```
// Example usage
poisson_exact, lambda1(10) lambda0(6) alpha(0.05) beta(0.1)
```

```
Required number of observation units (n) = 5
Note: Result rounded up to the nearest integer.
```

3 Detect a feasibility problem (Box 3)

To observe at least one occurrence of a feasibility problem during the pilot trial, the pilot trial sample size can be calculated using the following formula:

$$n = \frac{\ln(1-\gamma)}{\ln(1-\pi)}$$

Where n represents the pilot trial sample size, π is the probability that a problem may occur, and γ denotes the level of confidence interval that researchers want to have with observing at least one occurrence of those problems.

[10]:
```
// Set parameters
local gamma = 0.95 // Confidence level
local pi = 0.1 // Anticipated probability of the event of interest

// Calculate sample size
local sample_size = log(1 - `gamma') / log(1 - `pi')
```

[11]:
```
// Display the calculated sample size
display "Required sample size: " `sample_size'
```

```
Required sample size: 28.433159
```

4 Minimize total sample size of pilot and definitive trials (Box 4)

This method is to find the pilot trial sample size that minimizes the combined sample size required for both pilot and definitive trials. As we did not find existing packages for these calculations, we developed functions to perform them based on information from relevant literature. The sample size for the definitive trial is calculated using the `power` function.

4.1 Non-central t Distribution (NCT) Method

[12]:
```
// *  NCT method * //

clear

* Set the number of observations to the maximum of pilot trial sample size␣
 ↪per group
quietly set obs 1000

* Generate a variable for the pilot trial sample size per group (Step 1)
generate n_p= _n+1

* Calculate definitive trial sample size per group (Step 2.1)

// Use a standardized effect size of 0.4, a two-sided type I error of 0.05,␣
 ↪and a type II error of 0.2
quietly power twomeans 0 0.4, alpha(0.05) beta(0.2) nfrac

scalar n_m = r(N1) // Store the definitive trial sample size per group

* Calculate the inflated definitive trial sample size per group (Step 2.2)

gen alpha=0.05
gen beta=0.2
gen d=0.4

// Calculate the inflated definitive trial sample size per group
gen infl_n_m=2*invnt(2 * n_p- 2,invt(2*n_m-2,1-alpha/2),1-beta)^2/d^2

* Calculate the overall definitive trial sample size per group (Step 3)

gen N = n_p + infl_n_m

* Sort to find the minimum overall sample size (Step 5)
```

```
sort N
```

[13]:
```
* Display the results corresponding to the minimum value of overall sample␣
↪size
disp "Optimal Pilot Sample Size: " n_p[1]*2
disp "Definitive Trial Sample Size: " infl_n_m[1]*2
disp "Overall Sample Size: " N[1]*2
```

```
Optimal Pilot Sample Size: 22

Definitive Trial Sample Size: 215.51312

Overall Sample Size: 237.51312
```

4.2 Upper Confidence Limit (UCL) Method

[14]:
```
// * UCL method * //

clear

* Set the number of observations to the maximum of pilot trial sample size␣
↪per group
quietly set obs 1000

* Generate a variable for the pilot trial sample size per group (Step 1)
generate n_p= _n+1


* Calculate definitive trial sample size per group (Step 2.1)

// Use a standardized effect size of 0.4, a two-sided type I error of 0.05,␣
↪and a type II error of 0.2
quietly power twomeans 0 0.4, alpha(0.05) beta(0.2) nfrac

scalar n_m = r(N1) // Store the definitive trial sample size per group

* Calculate the inflated definitive trial sample size per group (Step 2.2)
gen std_conf_level=0.8 // Use 80% UCL method
*gen std_conf_level=0.95 // Use 95% UCL method

// Calculate the inflated definitive trial sample size per group
gen infl_n_m=n_m * (2*n_p - 2) / invchi2(2 * n_p- 2, 1 - std_conf_level)


* Calculate the overall definitive trial sample size per group (Step 3)
```

```
gen N = n_p + infl_n_m

* Sort to find the minimum overall sample size (Step 5)
sort N
```

[15]:
```
* Display the results corresponding to the minimum value of overall sample␣
 ↪size
disp "Optimal Pilot Sample Size: " n_p[1]*2
disp "Definitive Trial Sample Size: " infl_n_m[1]*2
disp "Overall Sample Size: " N[1]*2
```

```
Optimal Pilot Sample Size: 36

Definitive Trial Sample Size: 250.10747

Overall Sample Size: 286.10748
```

5 Rule out interventions unlikely to produce clinically important effects (Box 5)

This method calculates sample sizes for pilot trials aiming to exclude, with high confidence, interventions unlikely to yield clinically meaningful effects. The sample size estimation is based on the confidence interval width for differences between two groups.

For continuous outcomes, calculation can be made in Stata by using the function `ciwidth`, assuming a variance of 1 and a probability of 0.5 for the confidence interval width.

For binary outcomes, the Newcombe and Agresti-Caffo methods are recommended for sample sizes (fewer than 30 per group). The Wald method is acceptable for samples with 100 or more in each group.

5.1 Continuous Outcomes

[16]:
```
// * Continuous outcomes * //

ciwidth twomeans, width(0.6) sd(1) level(60) nfractional knownsds
```

```
Estimated sample sizes for a two-means-difference CI
Normal two-sided CI assuming sd1 = sd2 = sd

Study parameters:

        level =     60.00
        width =    0.6000
           sd =    1.0000
```

```
Estimated sample sizes:

            N =   31.4812
  N per group =   15.7406
```

5.2 Binary Outcomes

[17]:

```
// * Newcombe method * //

*program drop propdiffci_newcombe
program define propdiffci_newcombe, rclass
    syntax, p1(real) p2(real) conf_width(real) level(real)

    // Define input parameters
    local alpha = 1 - `level'
    local z = invnormal(1 - `alpha')
    local p1 = `p1'
    local p2 = `p2'
    local conf_width = `conf_width'

    // Initialize variables to hold the optimal n1 and smallest difference
    local optimal_n1 = -1
    local min_diff = .

    // Loop over potential values for n1
    forvalues n1 = 1(1)1000 {
        // Set n2 equal to n1
        local n2 = `n1'

        // Calculate Wilson confidence intervals for p1
        local ci1_lwr = (`p1' + `z'^2 / (2 * `n1') - `z' * sqrt((`p1' * (1 -␣
↪`p1') / `n1') + (`z'^2 / (4 * `n1'^2)))) / (1 + `z'^2 / `n1')
        local ci1_upr = (`p1' + `z'^2 / (2 * `n1') + `z' * sqrt((`p1' * (1 -␣
↪`p1') / `n1') + (`z'^2 / (4 * `n1'^2)))) / (1 + `z'^2 / `n1')

        // Calculate Wilson confidence intervals for p2
        local ci2_lwr = (`p2' + `z'^2 / (2 * `n2') - `z' * sqrt((`p2' * (1 -␣
↪`p2') / `n2') + (`z'^2 / (4 * `n2'^2)))) / (1 + `z'^2 / `n2')
        local ci2_upr = (`p2' + `z'^2 / (2 * `n2') + `z' * sqrt((`p2' * (1 -␣
↪`p2') / `n2') + (`z'^2 / (4 * `n2'^2)))) / (1 + `z'^2 / `n2')

        // Calculate confidence width
        local cw = sqrt((`p1' - `ci1_lwr')^2 + (`ci2_upr' - `p2')^2) +␣
↪sqrt((`ci1_upr' - `p1')^2 + (`p2' - `ci2_lwr')^2)

        // Calculate the difference between cw and conf_width
        local diff =  abs(`conf_width' - `cw')
```

```
        // Check if this is the smallest difference found so far
        if (`min_diff' == .) | (`diff' < `min_diff' & `conf_width' > `cw') {
            local min_diff = `diff'
            local optimal_n1 = `n1'
        }
    }

    // Display the optimal n1 with the smallest difference
    if `optimal_n1' != -1 {
        display "Pilot sample size per group: `optimal_n1'"
        return scalar optimal_n1 = `optimal_n1'
    }
    else {
        display "No optimal n1 found within the range."
    }
end
```

[18]:
```
propdiffci_newcombe, p1(0.5) p2(0.5) conf_width(0.2) level(0.8)
```

```
Pilot sample size per group: 35
```

[19]:
```
// * Wald method * //

* Set parameters
scalar oneside_ci_level = 0.8 // confidence level
scalar alpha = (1-oneside_ci_level)*2 // alpha corresponding to the confidence␣
 ↪limit
scalar p1 = 0.5 // proportion of the participants with the outcome in control␣
 ↪group
scalar p2 = 0.5 // proportion of the participants with the outcome in␣
 ↪treatment group
scalar w = 0.1*2 // width of the confidence interval or twice of the␣
 ↪clinically meaningful difference

* Calculate pilot trial sample size per group
scalar  Pilot_Size=4*(p1*(1-p1)+p2*(1-p2))*invnormal(alpha/2)^2/(w)^2 // Wald␣
 ↪test
```

[20]:
```
display "Pilot sample size per group: " Pilot_Size
```

```
Pilot sample size per group: 35.416315
```

[ ]:
```

```