Variability Analysis on COVID-19 Interim Trial Data

As at November 2020 there were two sets of publically available interim vaccine trial data.

The BioNTech-Pfizer BNT162b2 vaccine-candidate data (8 Nov 2020) had 38,955 participants, 94 of which showed COVID-19 symptoms, and they quoted an effectiveness of greater than 90%. We calculated this should be taken to mean that 8 of the 94 were unsuccessfully protected, given that half of the participants were given a placebo.

The Moderna mRNA-1273 data, coming in a week later, had more than 30,000 participants. 95 COVID-19 symptomatic cases were reported, 90 from the placebo group, and 5 from the (genuinely) vaccinated group. The reported vaccine efficacy rate was 94.5%, this being calculated (by us) as 100% $\times$ (85/90) = 94.4%.

According to the **National Institutes for Health** News Release webpage:

https://www.nih.gov/news-events/news-releases/promising-interim-results-clinical-trial-nih-moderna-covid-19-vaccine

"The findings are statistically significant, meaning they are likely not due to chance"

A later BioNTech-Pfizer BNT162b2 press release (18 Nov 2020) reported 170 cases of COVID-19, with 162 in the placebo group, and 8 cases in the (genuinely) vaccinated group. This pushes the estimated mean effectiveness up to 95%.

In the absence of professionally evaluated statistics, we offer some analysis based on simple computer modelling, which might reasonably be classed as Monte Carlo analysis

CONTENTS

Model

In the simulation we consider 30,000 simulated trials of 20,000 placebo cases and 20,000 (genuinely) vaccinated people. Each trial consists of randomly exposing the participants, with a low probability of infection.

In order to get adequate resolution on the probability, it was necessary to use a 32-bit random number generator. This generated uniformly distributed random integers in the range 0 to **Int32::MaxValue** = 2,147,483,647. (The standard C-library **rand()** function returns uniformly distributed integers in the range 0 to **RAND_MAX**, which can be only 32,767).

In the initial testing, the random number was converted to a unit-scaled **double**, and the probability was entered as some small range. The vaccinated infection probability was then reduced from that small range. The difficulty found was that the quantisation of the probability generator was not taken into consideration, and was in fact entirely hidden.

In the later code presented here, the probability for the vaccinated case is given as a relatively large integer (**prob**), and the placebo cases are given a much larger probability, calculated from the protection constant, **protect**. This eliminates the quantisation effect, at the expense of having to adjust **prob** when **protect** is changed, in order to maintain the approximate number of cases per day to something around 1.

Each trial is run until 95 cases have been found.[1] Since it is possible for there to be more than one case per day, we can exceed the 95 case threshold. We don't consider such an event to be problematic.

In the results we can readily show two histograms per page. We have included some plots more than once, so that they can be compared more easily on the same page. Each simulation took around 30 minutes, with the 170 & 190 case variants taking around twice as long.

[1] Limits of 170 and 190 cases have also been used in later simulations.

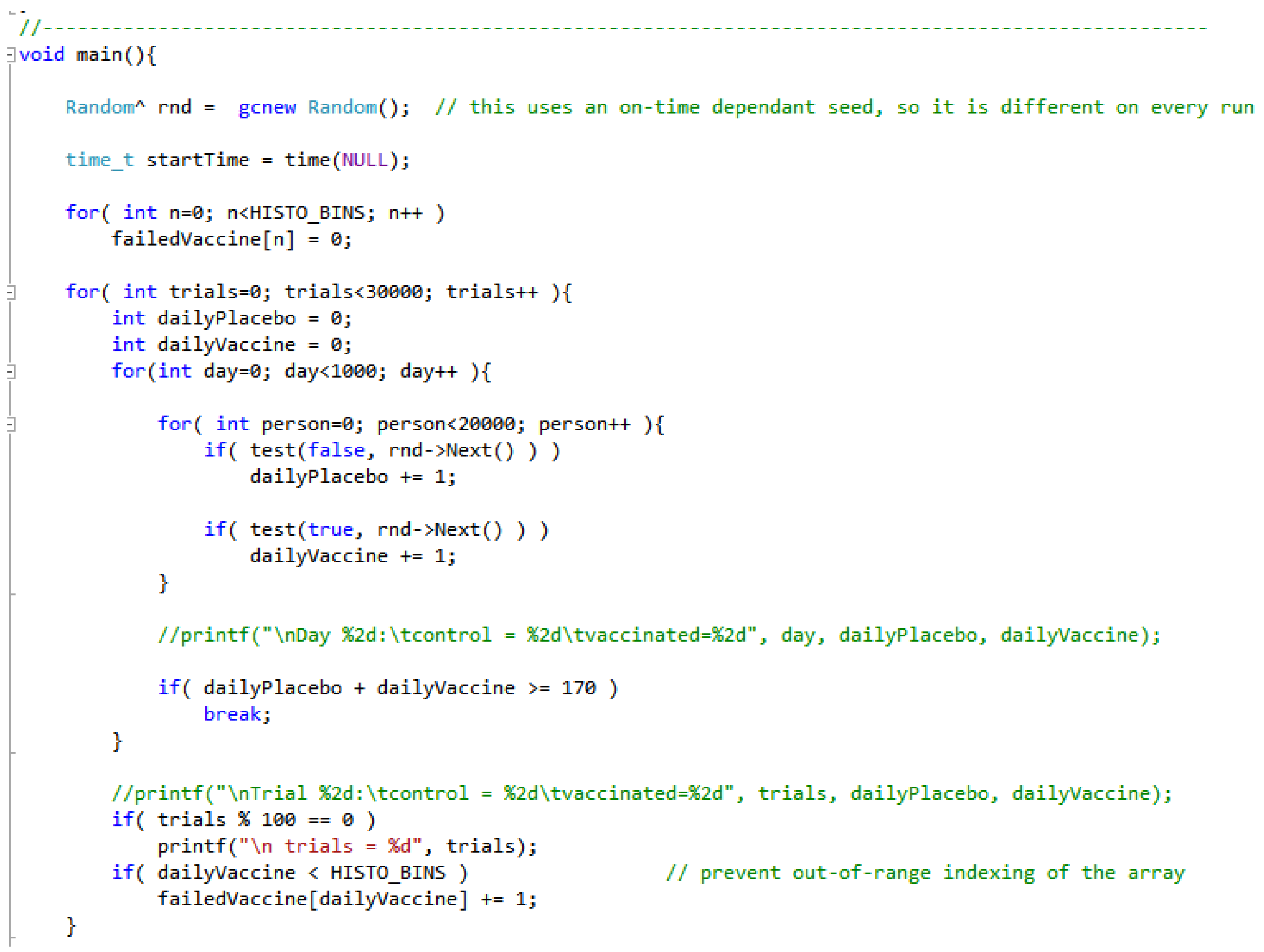
Code

This code is present for reasons of transparency, rather than as an exemplar of excellent programming practice. Whilst written in C++, most is in C, and should be fairly readable to programmers who specialise in a different programming language.

```
// vaccine.cpp : main project file.

#include "stdafx.h"
#include < stdio.h >
#include < stdlib.h >
#include < time.h >

using namespace System;

const double protect = 0.90;        // 0.90 means 90% protection

const int HISTO_BINS = 60;
int failedVaccine[HISTO_BINS];      // for each trial we save the number of failed vaccine cases

//----------------------------------------------------------------------------------------------------
bool test(bool vaccine, int random){

    const int prob = 7000;  // the range of integer counts used to define the vaccinated population risk

    int risk = prob;
    if( !vaccine )
        risk = int( 0.5 + prob/(1.0 - protect) );

    int high = (Int32::MaxValue / 2) + risk;
    int low  = Int32::MaxValue / 2;

    if( random < high && random >= low )
        return true;

    return false;
}
//---------------------------------------------------------------------------------------------------
```

The first assignment line of the program below may seem quite difficult to understand, even for an experienced C-programmer. Unfortunately we had to resort to using the .NET framework to get the required 32-bit random number generator.

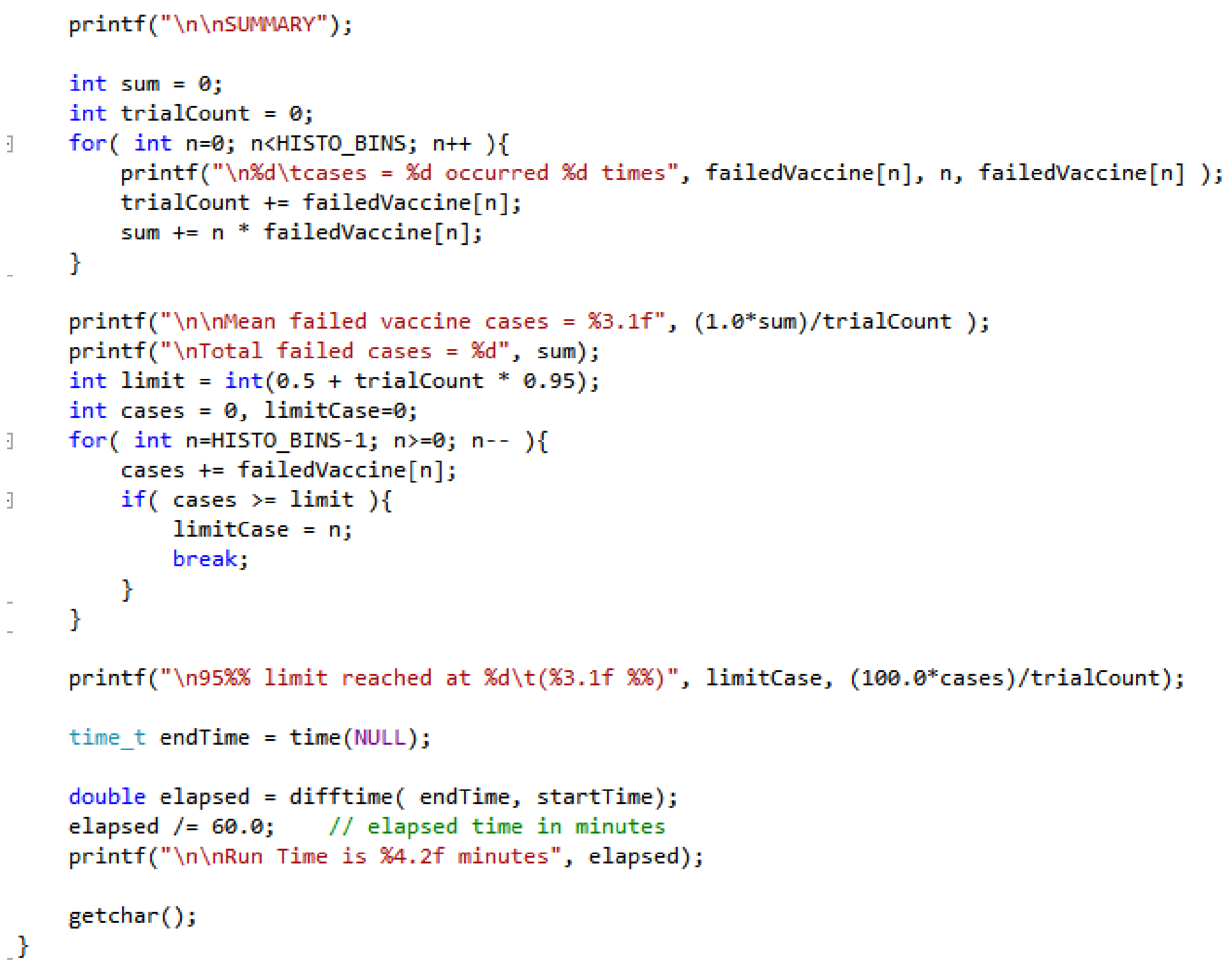
Having created the random number generator, the random value is accessed by using the **Next()** member function.



```
//-----------------------------------------------------------------------------------------------------------
void main(){

    Random^ rnd =  gcnew Random();  // this uses an on-time dependant seed, so it is different on every run

    time_t startTime = time(NULL);

    for( int n=0; n<HISTO_BINS; n++ )
        failedVaccine[n] = 0;

    for( int trials=0; trials<30000; trials++ ){
        int dailyPlacebo = 0;
        int dailyVaccine = 0;
        for(int day=0; day<1000; day++ ){

            for( int person=0; person<20000; person++ ){
                if( test(false, rnd->Next() ) )
                    dailyPlacebo += 1;

                if( test(true, rnd->Next() ) )
                    dailyVaccine += 1;
            }

            //printf("\nDay %2d:\tcontrol = %2d\tvaccinated=%2d", day, dailyPlacebo, dailyVaccine);

            if( dailyPlacebo + dailyVaccine >= 170 )
                break;
        }

        //printf("\nTrial %2d:\tcontrol = %2d\tvaccinated=%2d", trials, dailyPlacebo, dailyVaccine);
        if( trials % 100 == 0 )
            printf("\n trials = %d", trials);
        if( dailyVaccine < HISTO_BINS )                 // prevent out-of-range indexing of the array
            failedVaccine[dailyVaccine] += 1;
    }
```

That completes the main code loop. The first **printf** statement, which has been commented out, is used to setup the **prob** value so that roughly one person gets sick per day. It is then commented out for the main run so you don't unnecessarily print out 3 million daily results.

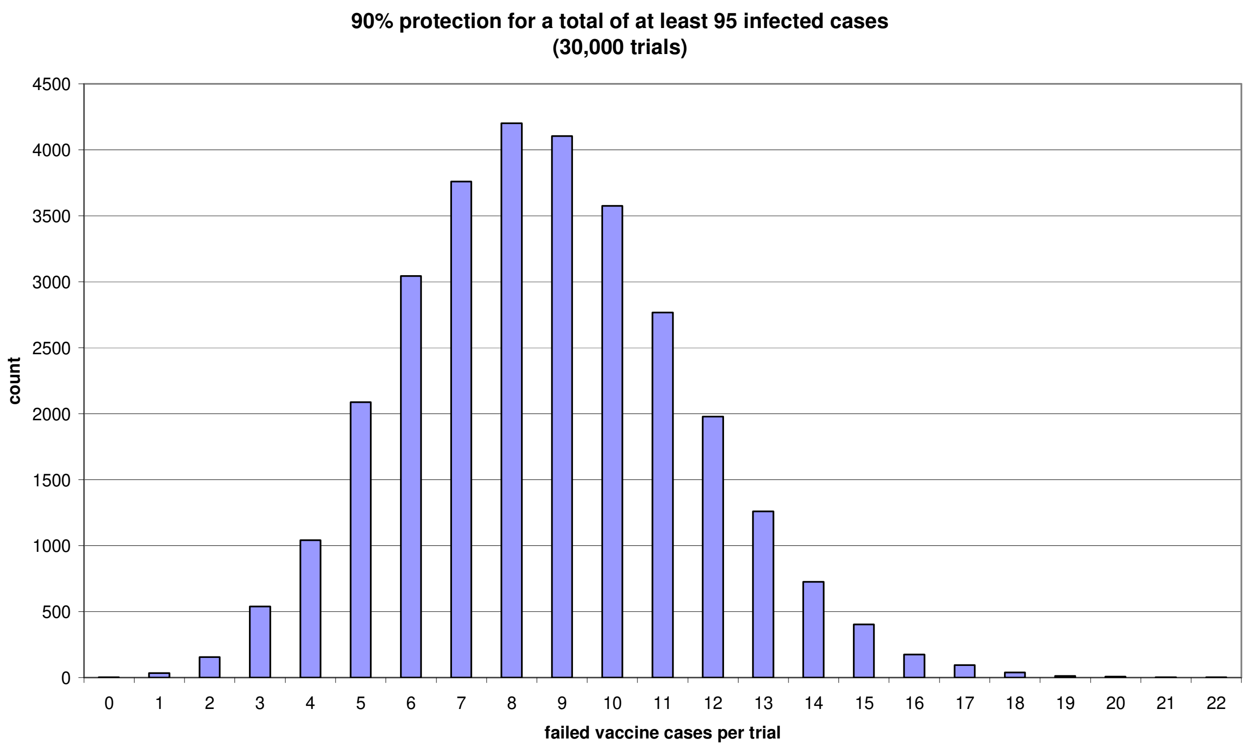
This final section prints out the results in human readable form, as well as leaving a part which is easy to read into an Excel spreadsheet.



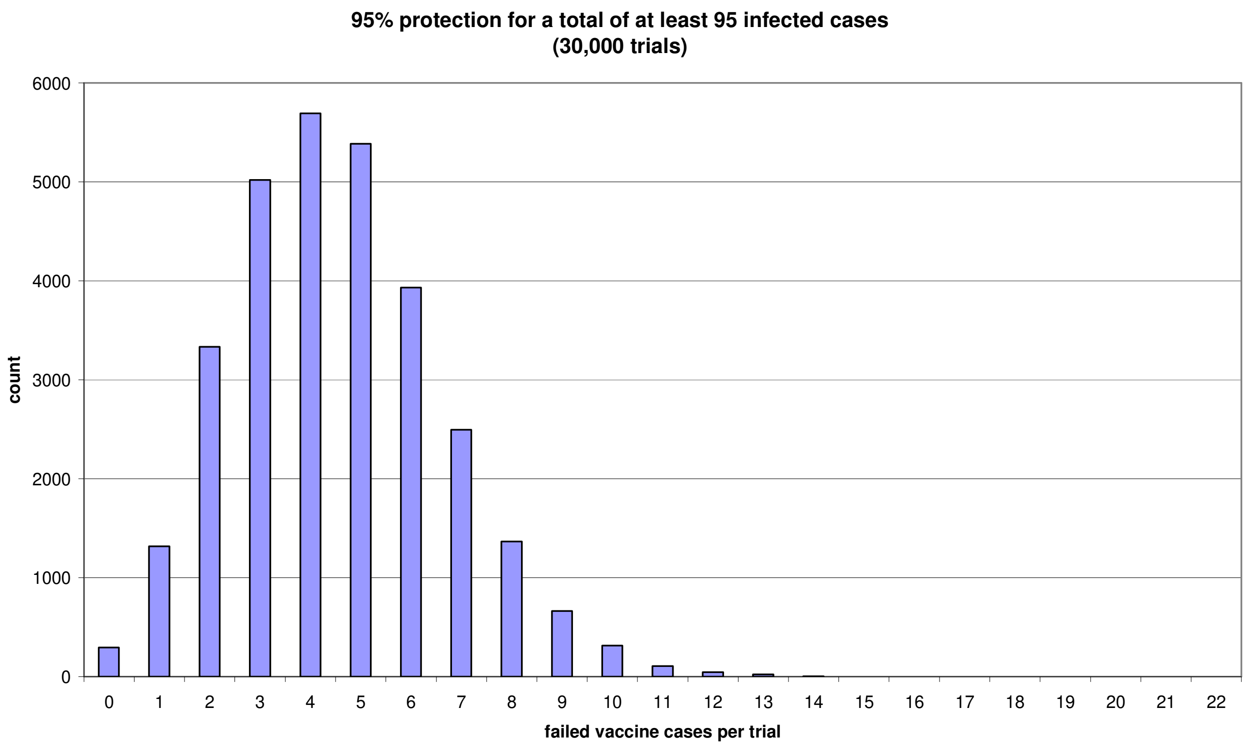
```
    printf("\n\nSUMMARY");

    int sum = 0;
    int trialCount = 0;
    for( int n=0; n<HISTO_BINS; n++ ){
        printf("\n%d\tcases = %d occurred %d times", failedVaccine[n], n, failedVaccine[n] );
        trialCount += failedVaccine[n];
        sum += n * failedVaccine[n];
    }

    printf("\n\nMean failed vaccine cases = %3.1f", (1.0*sum)/trialCount );
    printf("\nTotal failed cases = %d", sum);
    int limit = int(0.5 + trialCount * 0.95);
    int cases = 0, limitCase=0;
    for( int n=HISTO_BINS-1; n>=0; n-- ){
        cases += failedVaccine[n];
        if( cases >= limit ){
            limitCase = n;
            break;
        }
    }

    printf("\n95%% limit reached at %d\t(%3.1f %%)", limitCase, (100.0*cases)/trialCount);

    time_t endTime = time(NULL);

    double elapsed = difftime( endTime, startTime);
    elapsed /= 60.0;     // elapsed time in minutes
    printf("\n\nRun Time is %4.2f minutes", elapsed);

    getchar();
}
```

90% vs 95% protection comparison

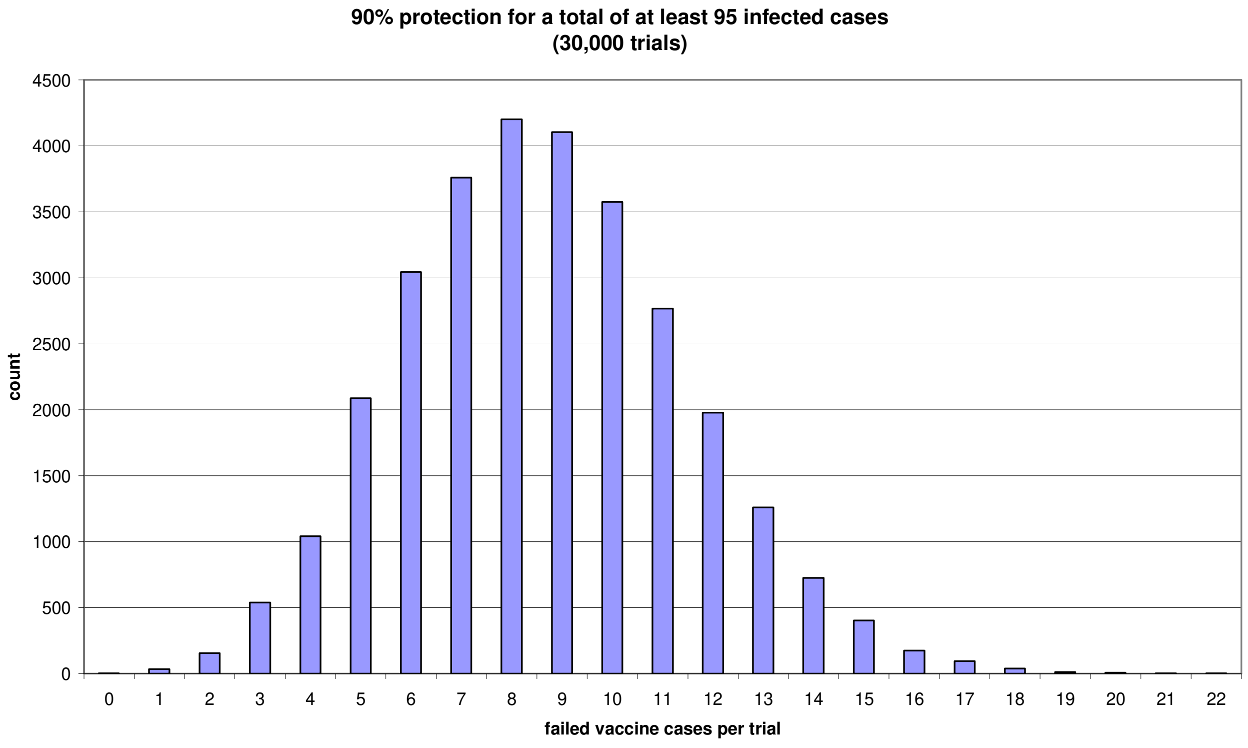
mean failed vaccine cases = 8.7 @ 90% protection



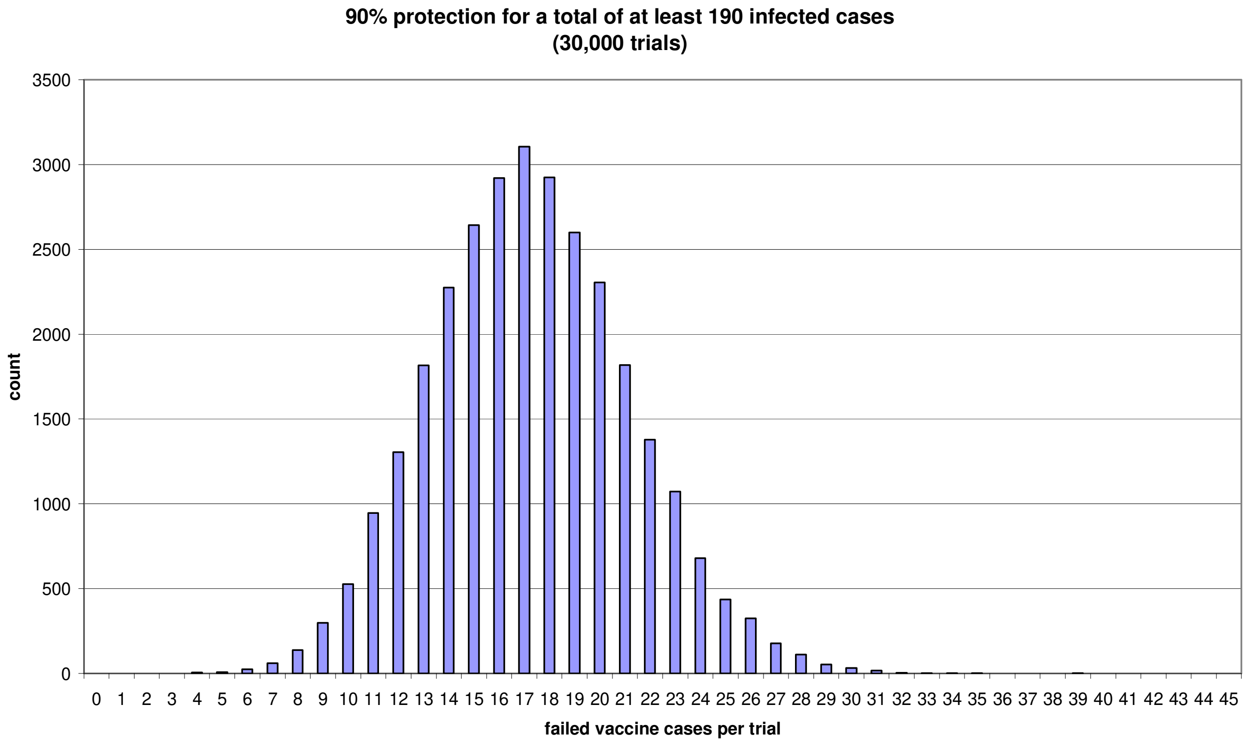
mean failed vaccine cases = 4.5 @ 95% protection



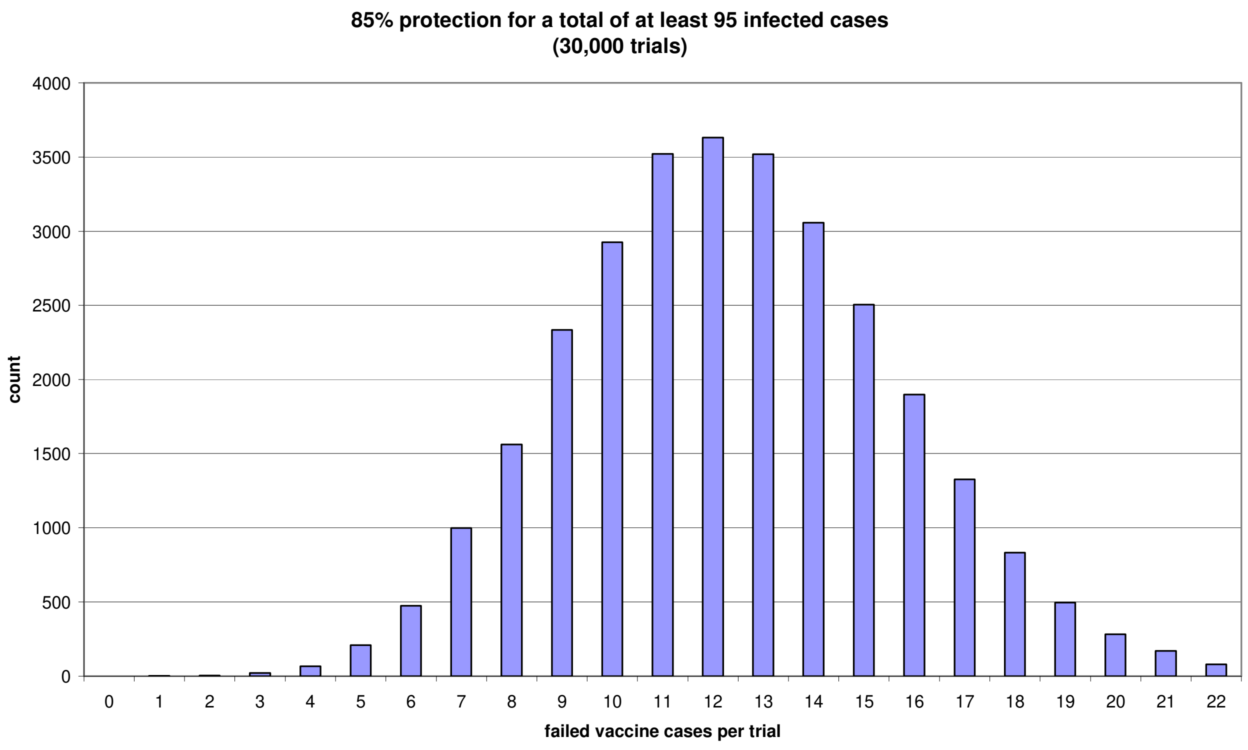
95 cases vs 190 cases comparison (with 90% protection)



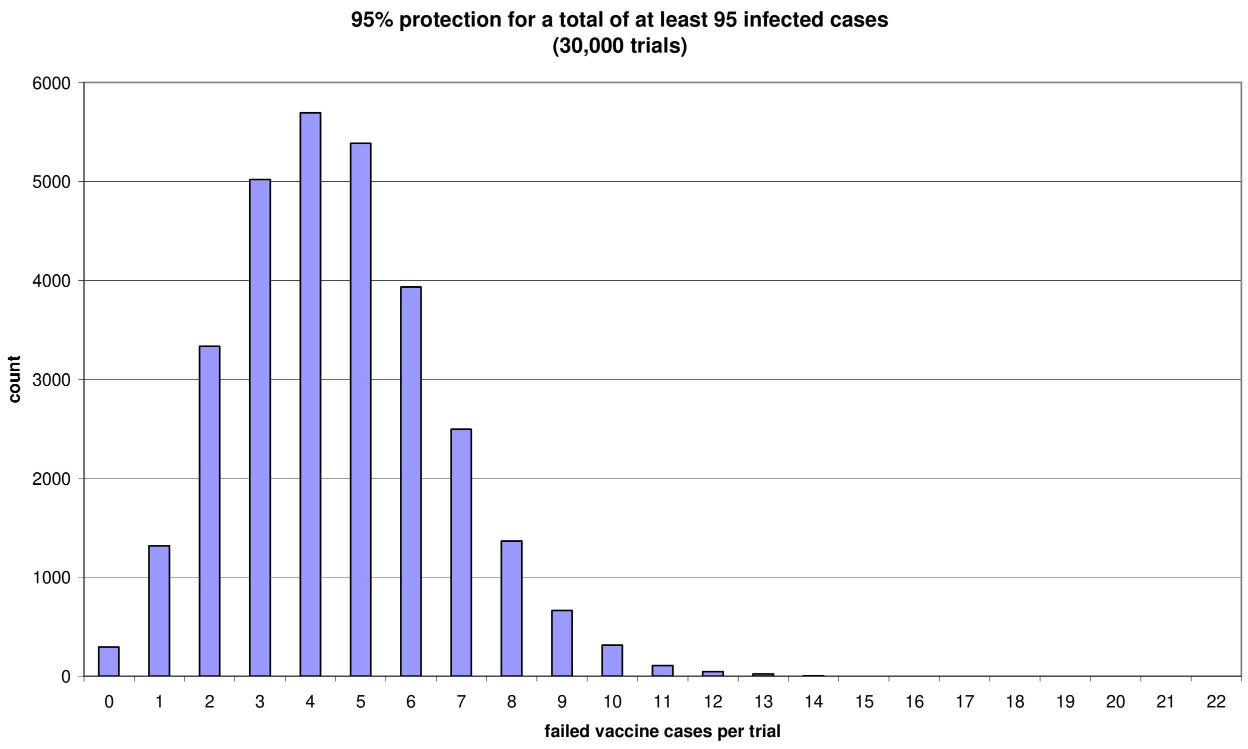
By scaling so the mean is in roughly the same vertically aligned position on the page, it is clear that the relative spread of values has been reduced by waiting longer for the sample size to double (mean of 17.3 failed vaccine cases).



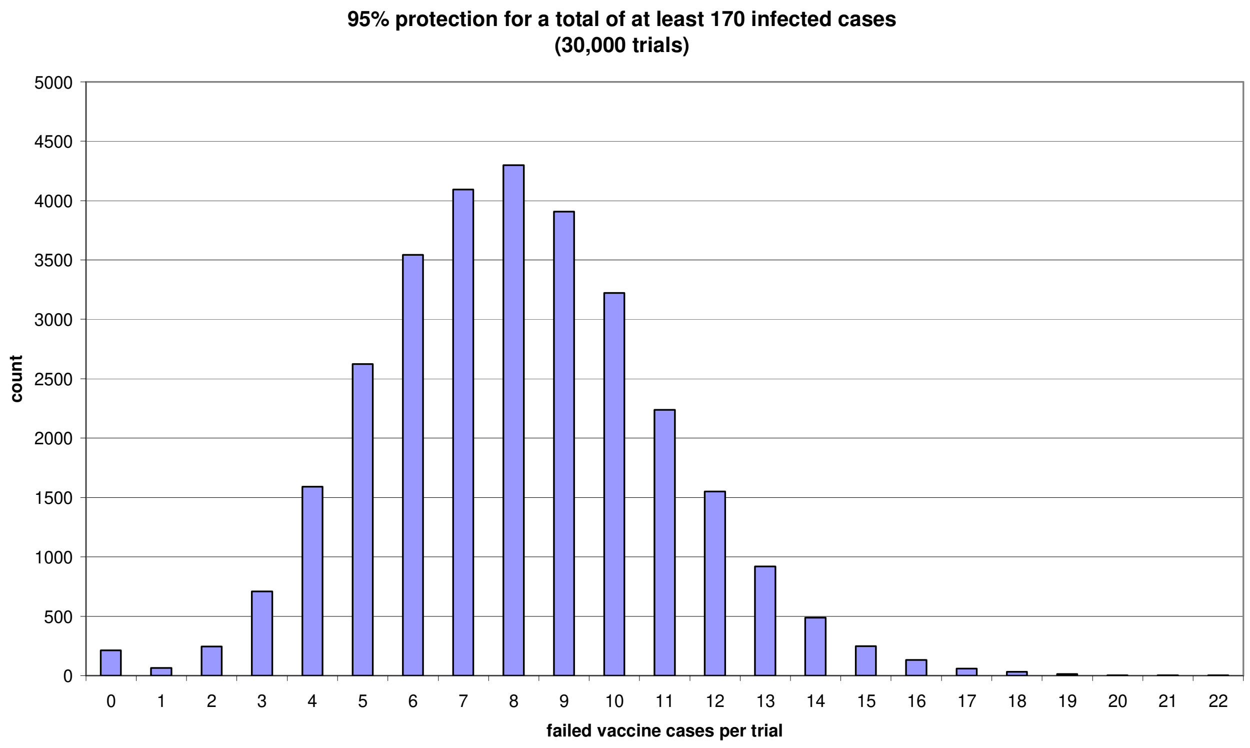
85% vs 95% protection comparison



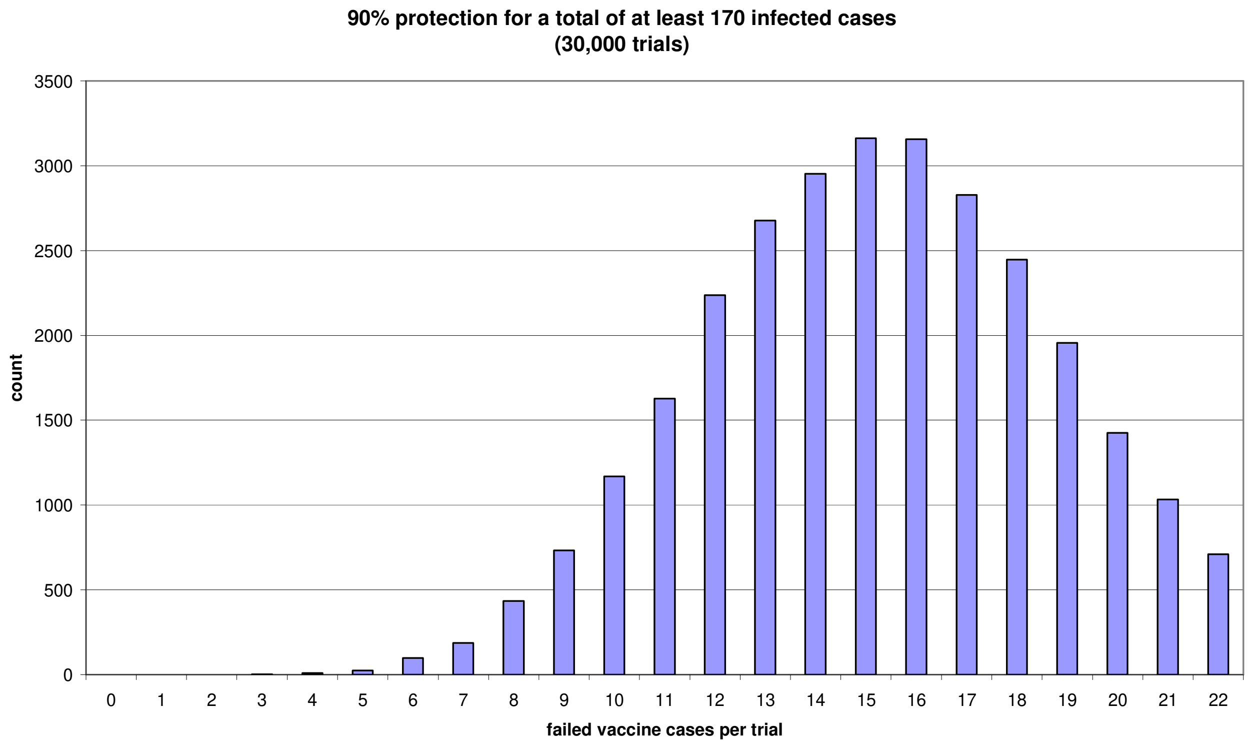
'5 failed vaccine cases' could be due to an 85% protection level, but is much more likely indicative of a higher protection level.



170 infected cases – protection comparisons



Using the same horizontal scale for the 90% test below shows how well shifted the data is. Even though 7, 8, and 9 failed vaccine cases at 90% protection look significant, a summation shows that 95% of trials give 10 failed cases or more.



Discussion of Results

When the total number of infected cases is at the level of 95, it is not possible to say with any great certainty that a result of 5 or 8 (genuinely) vaccinated participants means that the vaccine is definitely 90%, or definitely 95% effective. The spread of possible results is too great.

Going up to 170 cases (or more) reduces the relative spread of the results. What we really want to know is: given 8 failed vaccination cases in a total of 170 symptomatic participants, what is the least possible vaccine effectiveness we can state with high confidence? This is a more difficult question to answer.

Now things get a bit tricky. We will be talking about a "90% protection level" in the same sentence as a "95% confidence interval". The 95% confidence interval is to be understood in this way: We did 30,000 simulated trials. In 95% of these simulated trials ($0.95 \times 30{,}000 = 28{,}500$) the failed vaccine cases per trial result was larger than some particular value. For example, looking at the data for the plot at a 90% protection level, the number of trials which showed 10 or more failed vaccine cases per trial was 28,514.

For that simulation, less than 10 failed cases correspond to at least a 90% protection level (with 95% confidence level). We then have to step the protection level up slightly to see if we can get less than 9 failed cases, but still with a 95% confidence level. This is tiresome because each computer run takes 50 minutes. Worse still, every time you do a computer run, you get a different answer.

With a 92% protection level we got a 97.1% chance of 7 or more failed cases.

Back to 90% protection level gives a 95.3% chance of 10 or more failed cases.

The answer of course is to do more simulated trials. Maybe 300,000 trials with a run time of 8 hours?

But it should now be clear that we are really trying to over-analyse this one trial result.

The reported effectiveness of the BioNTech vaccine is at least 90% with a good level of confidence. What happens in practice, when freshly recruited people are used to deliver a vaccine which has to be shipped at -70°C is less certain. Human nature is such that people will be loathe to admit that they messed up transporting it, so it got damaged by heat (eg -30°C), and therefore is now ineffective. Throwing that shipment away, at the cost of thousands of dollars, is more than many people (and companies) would be willing to own up to!

Version History

v1.00: First Release 18 November 2020, on http://lesliegreen.byethost3.com