1) To determine the stability of LUSolve and InvSolve, I use the ill-conditioned system C = [[.835,.667], [.333,.266]] with [b1,b2] = [.168, .067] which has true solution [1,-1] Upon first testing at precision of 3, I found that both solve methods triggered the 'U is not upper triangular' and 'A is singular' error checkers. This was because both solvers use the same rowEform method to get the row echelon form of C, and that method returned a row of zeros at that precision. To avoid the row of zeros, I found it best to start at a precision of 9. Since float can only show up to 13 decimals at a time, I limit the precision up to 13. These are the results:

	LUSolve	InvSolve
6	['1.0', '-1.0']	['1.1', '-1.3']
7	['1.0', '-1.0']	['1.01', '-1.0']
8	['1.0', '-1.0']	['0.996', '-0.998']
9	['1.00066734', '-1.00083542']	['1.0003', '-1.0005']
10	['1.0', '-1.0']	['1.0', '-1.0']
11	['0.99999333004', '-0.99999165004']	['0.99999700001', '-0.99999600001']
12	['1.0', '-1.0']	['0.999999899999', '-0.999999799998']
13	['1.0', '-1.0']	['1.0', '-0.999999990003']

Taking the absolute difference of the calculated solution from the true solution gives this table:

	LUSolve	InvSolve
6	['0.0', '0.0']	['0.1', '0.3']
7	['0.0', '0.0']	['0.01', '0.0']
8	['0.0', '0.0']	['0.004', '0.002']
9	['0.00066734', '0.00083542']	['0.0003', '0.0005']
10	['0.0', '0.0']	['0.0', '0.0']
11	['6.66996e-06', '8.34996e-06']	['2.99999e-06', '3.99999e-06']
12	['0.0', '0.0']	['1.00001e-07', '2.00002e-07']
13	['0.0', '0.0']	['0.0', '9.9972e-09']

Taking the average of the absolute differences with respect to x1 and x2 gives these results:

LUSolve	InvSolve
[8.4251245e-05, 0.000105471245]	[0.014287887498875, 0.03781302624865]

Taking these results, LUSolve deviates from the true solution far less than InvSolve

Since both LUSolve and InvSolve are closest to the true solution at precision 10, I will use that to test the stability of both of them to perturbed versions of C. As expected of an ill-conditioned system, both solutions differed heavily to a slightly perturbed version of C and b, with b2 = .066:

LUSolve	InvSolve
['-666.002668', '834.00334']	['-666.00267', '834.00334']

Testing again with a well condition system this time, I let C = [[.835,.667],[.333,.266]] and [b1,b2] = [4,7] with a true solution of [2,1], i let the perturbed system have a b that is off by .001 times a multiple and tested at precision of 4. the results are as shows:

b	LUSolve	InvSolve
['4.0', '7.0']	['2.0', '1.0']	['2.0', '1.0']
['3.999', '6.999']	['2.001', '0.999']	['2.0', '0.999']
['3.998', '6.998']	['2.002', '0.998']	['2.01', '0.998']
['3.997', '6.997']	['2.003', '0.997']	['2.0', '0.997']
['3.996', '6.996']	['2.004', '0.996']	['2.0', '0.996']
['3.995', '6.995']	['2.005', '0.995']	['2.01', '0.995']
['3.994', '6.994']	['2.006', '0.994']	['2.01', '0.994']
['3.993', '6.993']	['2.007', '0.993']	['2.01', '0.993']
['3.992', '6.992']	['2.008', '0.992']	['2.0', '0.992']
['3.991', '6.991']	['2.009', '0.991']	['2.01', '0.991']

Both solve methods worked as expected for a perturbed well-conditioned system.

Since both LUSolve and InvSolve performed similarly in both the well conditioned and the ill conditioned case, I will focus on the performance of the two with respect to precision. It is apparent from the LUSolve and InvSolve tables with respect to precision that LUSolve is on average closer to the true solution than InvSolve, and is therefore more stable than InvSolve.

2) To check the stability of matrix vector multiplication, I will test matrixProduct in terms of both perturbed versions of C, b and precision.

Testing with precision between 1 and 13, the results of C*x for C = [[.835,.667], [.333,.266]] and x = [1,-1] with true solution b = [0.168, 0.067] is as follows:

t precision	matrixProduct(C,x,t)	t precision	matrixProduct(C,x,t)
1	['0.1', '0.0']		
2	['0.16', '0.06']		
3	['0.168', '0.067']		
4	['0.168', '0.067']		
5	['0.168', '0.067']		
6	['0.168', '0.067']		
7	['0.168', '0.067']		
8	['0.168', '0.067']		
9	['0.168', '0.067']		
10	['0.168', '0.067']		
11	['0.168', '0.067']		
12	['0.168', '0.067']		
13	['0.168', '0.067']		

The solution for C*x stays constant and is equal to the true solution for precision 3 and above. From this we can conclude that matrix multiplication is stable under precision bigger than or equal to the largest number of significant digits in C or b.

Testing with perturbation of x - .001*scalar at precision 3:

perturbed x	b solution
[1.0, -1.0]	[['0.168', '0.067']]
[0.999, -1.001]	[['0.166', '0.067']]
[0.998, -1.002]	[['0.165', '0.065']]
[0.997, -1.003]	[['0.163', '0.065']]
[0.996, -1.004]	[['0.162', '0.065']]
[0.995, -1.005]	[['0.161', '0.064']]
[0.994, -1.006]	[['0.159', '0.063']]
[0.993, -1.007]	[['0.157', '0.063']]
[0.992, -1.008]	[['0.156', '0.062']]
[0.991, -1.009]	[['0.154', '0.062']]

Surprisingly, matrix product is stable even when an ill conditioned system is slightly perturbed. From this we can conclude that matrix vector product is very stable.

3) To check the stability of matrix inverse calculation, I will test Invert in terms of both perturbed versions of C and precision.
Start precision at 6 as noted in 1)

Letting C = [[.835,.667], [.333,.266]] and varying precision gives these results:

precision 6	precision 7	precision 8
['-318562.0', '798799.0']	['-265469.0', '665668.7']	['-265469.07', '665668.66']
['398802.0', '-1000000.0']	['332335.3', '-833333.3']	['332335.33', '-833333.33']

precision 9	precision 10
['-266134.399', '667337.005']	['-266001.0639', '667002.6678']
['333168.25', '-835421.888']	['333001.332', '-835003.34']

precision 11	precision 12
['-265998.84289', '666997.09855']	['-265999.95345', '666999.883276']
['332998.55146', '-834996.36777']	['332999.941725', '-834999.853875']

precision 13
['-265999.997872', '666999.994664']
['332999.997336', '-834999.99332']

The resulting inverses from varying precision differ greatly but become more constant as the precision increases, but this brings to question the stability of the inverse

Testing again at precision 6, and varying the top left element in C by .001*scalar

first element = 0.836
['1003.72', '-2516.86']
['-1256.54', '3154.57']

first element = 0.837
['500.474', '-1254.94']
['-626.533', '1574.8']

first element = 0.838
['333.778', '-836.952']
['-417.848', '1051.52']

first element = 0.839
['250.232', '-627.464']
['-313.26', '789.266']

first element = 0.84
['200.17', '-501.928']
['-250.587', '632.111']

The inverse of perturbed C varies greatly.

With conclusions in mind, the calculation for the matrix inverse is unstable.

4) The conclusions from 2) and 3) explain why InvSolve is more unstable than LUSolve. While matrix product is very stable, the instability from the matrix inverse still carries over in the InvSolve calculation. LUSolve uses front and back solve, which are less prone to instability due to perturbation and precision than the matrix inverse.