1. b) When using
x0[4] = {0.0+1e-21, 1.0-1e-21, 2.0+1e-21, 3.0-1e-21};
or similar x0 with deviation smaller than 1e-20 as initial guess my algorithm shows convergence to [0.000000 1.000000 2.000000 3.000000]^T, but this is due to the Jacobi algorithm using 1e-20 as threshold for convergence. For values where the initial deviation from the solution is bigger than the threshold (eg. [0.0+1e-19, 1.0+1e-19, 2.0+1e-19, 3.0+1e-19]) a solution is not found but the x gets to inf and then nan, which causes the loop to terminate.

c) I used Matlab and created the matrix A as
A = [0.000000 1.000000 2.000000 3.000000]
After that I calculated its determinant, inverse matrix and condition number using matlab functions det, inv and cond. These resulted in determinant -16, inverse matrix

-0.0625	-0.1250	0.3125	0.1875
-0.2500	0.5000	0.2500	-0.2500
-0.0625	-0.1250	1.3125	-0.8125
0.4375	-0.1250	-1.1875	0.6875

and condition number 15.3254. The condition number is quite large which means that the problem is ill-conditioned. In this case this means that around 15 digits of accuracy might be lost in addition to the loss from arithmetical precision and the properties of the numerical method used. This makes it quite natural that the solution is not found.

d) I kept x as [0.0+1e-19, 1.0+1e-19, 2.0+1e-19, 3.0+1e-19] and tried different integer values for ω. Runs where ω was at least 16 yielded a result in reasonable time, smaller either gave nan or ran for unacceptably long times (minutes) without producing a result.

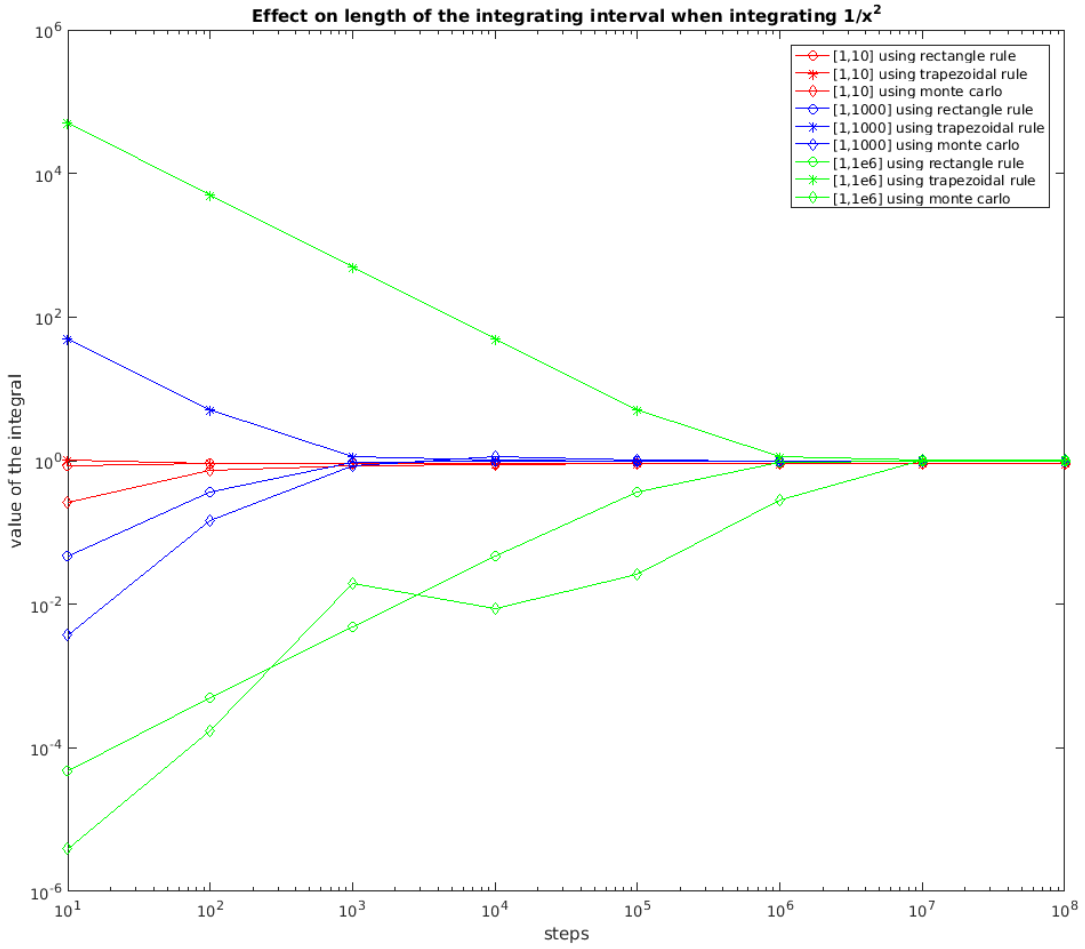
2. I implemented the integration using rectangle rule (using the midpoint of each subinterval as height), trapezoidal rule and Monte Carlo method (as given in http://www.acclab.helsinki.fi/~knordlun/mc/mc5nc.pdf section 5.1.1). Each was integrated the function $1/x^2$ from given start to given end using given number of steps. These functions can be found in file integration.c (rectRule, trapRule and monteCarlo, respectively) and each take start and end points as doubles and number of subintervals/random points as int.

First plot compares the accuracy of different integrators. Red curves integrate over interval [1, 10], blue curves over [1, 1000] and green curves over [1, 1 000 000]. Different integrators are shown with markers, circle for rectangle rule, asterisk for trapezoidal and diamond for monte carlo.

One might be surprised that small upper limit for the integral yields values closer to the analytical solution 1 than larger

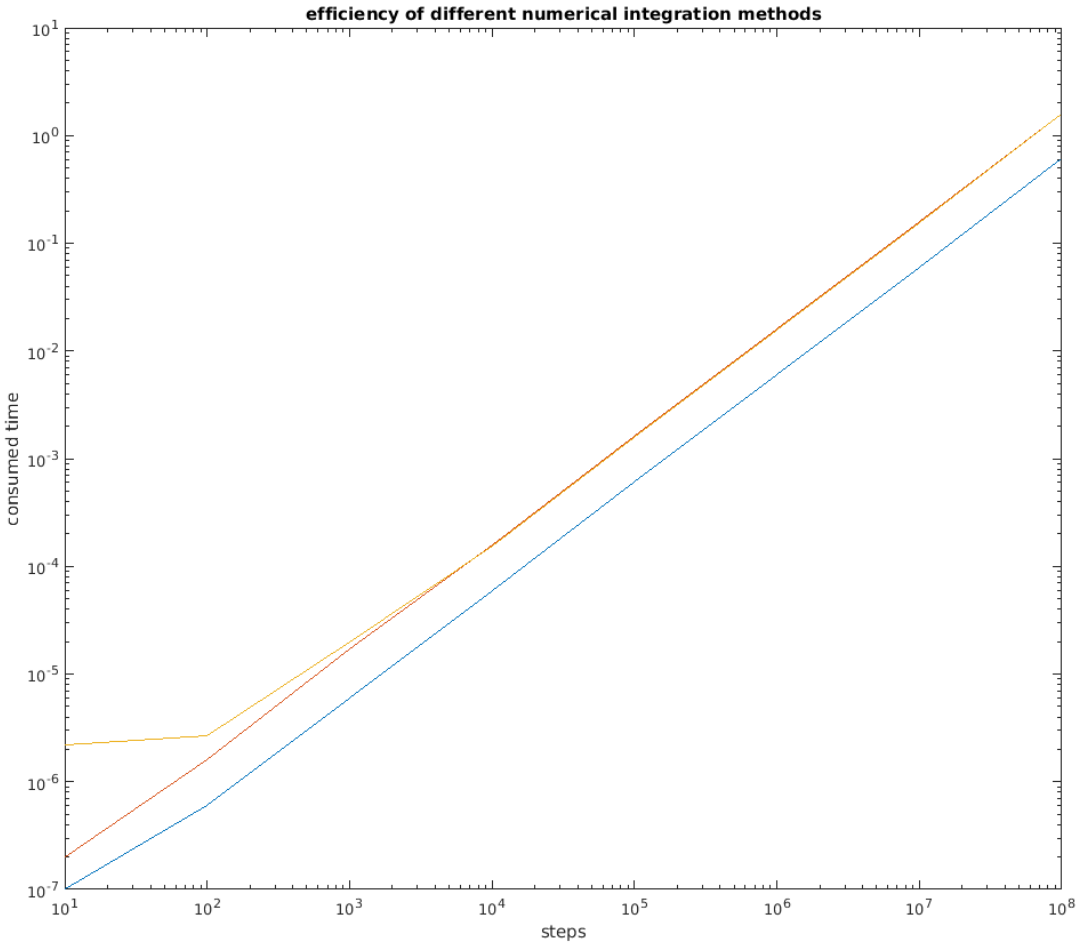
values on all integrators. When looking at the magnitudes of the numbers, this seems more natural, because when for example drawing a trapezoid that has a base of about 1e6/100=1e4 we are vastly exaggerating the area of curve between x=1 and x≈1e4+1 due to $1/x^2$ curving down very steeply near x=1. This makes it easily visible that too long subintervals make results very bad.

When length of subintervals is reasonable, trapezoidal rule seems to work slightly better than rectangle rule, which is expected. For long subintervals, rectangle rule is actually better due to it using the value at the center of the interval and therefore not getting ridiculously huge area on region near x=1 where y starts from 1 and declines steeply. If the left edge of the interval would have been used instead, this would worsen the accuracy of rectangle rule drastically.



In general, monte carlo is somewhat worse than the two other methods. Due to it’s slightly poorer accuracy it is better suited to situations where other methods are challenging or impossible to use, for example when the function is in two or more dimensions or it might not be possible to give it as one function (for example area of an ellipse).

Next image shows the time it took for different integrators to complete the calculations on different numbers of steps. All times were measured as the average of 10 runs. It seems that all are O(n) as one might expect. Trapezoidal rule (red curve) is slightly slower than rectangle (blue curve) due to calculating the area of a trapezoid requiring more operations than calculating the area of a rectangle. Monte Carlo (yellow curve) performs somewhat similarly to trapezoidal rule.



3. a) I used matlab and defined A as
A = [1, 2, 1; 0, 3, 1; 2, 4, 2]
and then called [U, W, V] = svd(A) to find the singular value decomposition:

U =

```
   -0.3922    0.2149   -0.8944
   -0.4804   -0.8770   -0.0000
   -0.7844    0.4297    0.4472
```

W =

```
    6.2108         0         0
         0    1.1941         0
         0         0    0.0000
```

V =

```
   -0.3158    0.8997   -0.3015
   -0.8636   -0.4041   -0.3015
   -0.3931    0.1652    0.9045
```

b) W has a zero value on its diagonal, which means that A is singular and therefore a solution for Ax=0 exists but ax=b does not.