1. a) $R_{k+1} = I - AX_{k+1}$

$= I - A(X_k + X_k(I - AX_k))$

$= I - AX_k - AX_k(I - AX_k)$

$= I - AX_k - AX_k + AX_k \cdot AX_k$

$= (I - AX_k) \cdot (I - AX_k)$

$= R_k^2$

$E_{k+1} = A^{-1} - X_{k+1}$

$= A^{-1} - (X_k + X_k(I - AX_k))$

$= A^{-1} - X_k - X_k + X_k A X_k$

$= X_k A X_k - 2X_k + A^{-1}$

Thus, $E_{k+1} = E_k A E_k$

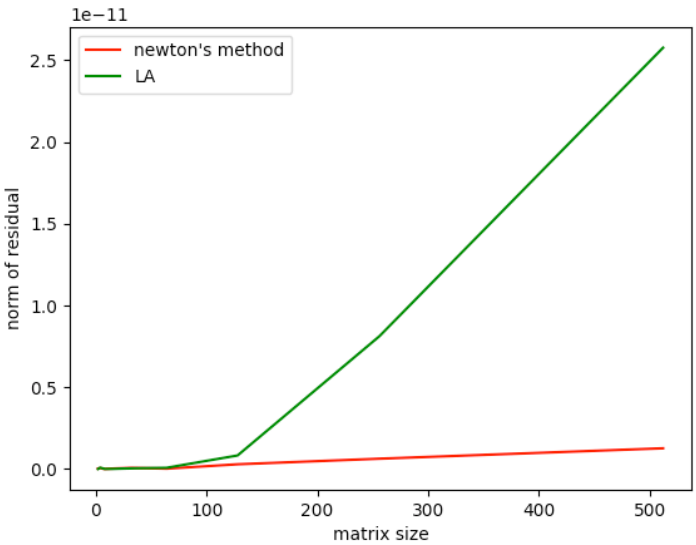
$E_k A E_k$

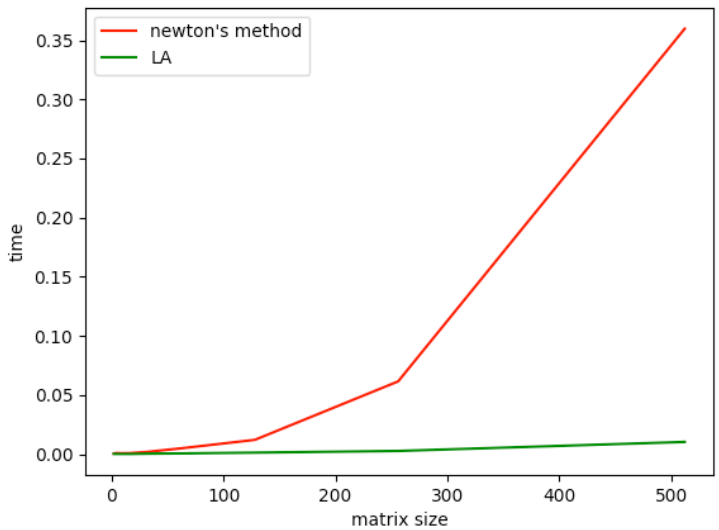
$= (A^{-1} - X_k) \cdot A \cdot (A^{-1} - X_k)$

$= I A^{-1} - X_k A \cdot A^{-1} - I X_k + X_k A X_k$

$= X_k A X_k - 2X_k + A^{-1}$

```
matrix size: 2x2 , LA.inv runtime: 2.4335e-04 ; Newton's runtime: 4.4325e-04 , LA.inv residual norm: 5.5511e-17 ; Newton's residual norm: 2.5704e-14
matrix size: 4x4 , LA.inv runtime: 1.0952e-04 ; Newton's runtime: 9.1250e-04 , LA.inv residual norm: 8.3621e-14 ; Newton's residual norm: 5.5846e-14
matrix size: 8x8 , LA.inv runtime: 1.1396e-04 ; Newton's runtime: 7.3952e-04 , LA.inv residual norm: 4.2894e-15 ; Newton's residual norm: 1.2862e-15
matrix size: 16x16 , LA.inv runtime: 1.1053e-04 ; Newton's runtime: 8.2218e-04 , LA.inv residual norm: 7.9604e-15 ; Newton's residual norm: 3.0715e-14
matrix size: 32x32 , LA.inv runtime: 3.4971e-04 ; Newton's runtime: 1.8577e-03 , LA.inv residual norm: 4.0531e-14 ; Newton's residual norm: 6.8761e-14
matrix size: 64x64 , LA.inv runtime: 5.9828e-04 ; Newton's runtime: 5.0773e-03 , LA.inv residual norm: 7.4772e-14 ; Newton's residual norm: 2.0889e-14
matrix size: 128x128 , LA.inv runtime: 1.2438e-03 ; Newton's runtime: 1.2053e-02 , LA.inv residual norm: 8.2844e-13 ; Newton's residual norm: 2.8892e-13
matrix size: 256x256 , LA.inv runtime: 2.6502e-03 ; Newton's runtime: 6.1502e-02 , LA.inv residual norm: 8.1244e-12 ; Newton's residual norm: 6.2991e-13
matrix size: 512x512 , LA.inv runtime: 1.0284e-02 ; Newton's runtime: 3.5984e-01 , LA.inv residual norm: 2.5774e-11 ; Newton's residual norm: 1.2665e-12
```





By comparing Netwon's method and the scipy.linalg.inv, we found that the time required to compute matrix inv for Netwon's method increases exponentially, while the scipy build in method seems to have a constant time. However, the higher cost of Netwon's method leads to a higher accuracy. The norm of residual for scipy.linalg.inv increases much more rapidly than the norm of residual for Netwon's method.

2.

a) We have a system: $Ad = b + f$

Assume f_n is known, we can have:

$$A^{-1} \cdot Ad = A^{-1}b + A^{-1}f$$

$$d = A^{-1}b + A^{-1}f$$

Since we only want to know the last element of d, then:

$$d_n = A^{-1}[-1]\,b[-1] + A^{-1}[-1]\,f_n$$

d) $d_n = A^{-1}[-1]\,b + A^{-1}[-1]\,f_n = f(f_n)$

e) $\frac{d\,d_n}{d\,f_n} = A^{-1}[-1][-1]$

then $f_{n+1} = f_{n0} - \frac{f(f_{n0})}{f'(f_{n0})} = f_{n0} - \frac{A^{-1}[-1]\,b + f_{n0}\,A^{-1}[-1][-1]}{A^{-1}[-1][-1]}$

f) The fn value from my calculation seems to converge to 0.375103623599399, so we may assume the true value of fn is 0.375103623599399. We will compare the relative error for different n. From the table, we can see the relative error for all three methods are decreasing initially, since the numbers are converging to the point that we observed previously. However, the relative errors bounced up suddenly at n = 32768 and continuous increases as n increases. This is probably because of the ill conditioning in the linear system. We more or less used different solvers in each method, and additional errors are generated. Overall, three methods have very similar performance, while the newton's method's result is slightly (0.000001) different from other two results. As for the speed, newton's method requires least time to compute and fsolve requires the most. As n gets largers, newton's method's performance is significant better than other two, since it only iterates once, and we are computing inv efficiently. Therefore, I will choose the newton's method, since it only requires the least time but have very similar performance as other two methods.

g) The exact value of fn should be around 0.375103623599399.

3. a) $S(t_{i+1}) = S(t_i) - h\frac{\beta}{N} S(t_{i+1}) I(t_{i+1})$

$S(t_{i+1}) + h\frac{\beta}{N} S(t_{i+1}) I(t_{i+1}) = S(t_i)$

$S(t_{i+1})(1 + h\frac{\beta}{N} I(t_{i+1})) = S(t_i)$

$$S(t_{i+1}) = \frac{S(t_i)}{1 + h\frac{\beta}{N} I(t_{i+1})}$$

$$E(t_{i+1}) = E(t_i) + h\frac{\beta}{N} S(t_{i+1}) I(t_{i+1}) - \omega h E(t_{i+1})$$

$E(t_{i+1}) + \omega h E(t_{i+1}) = E(t_i) + h\frac{\beta}{N} S(t_{i+1}) I(t_{i+1})$

$E(t_{i+1})(1 + \omega h) = E(t_i) + h\frac{\beta}{N} S(t_{i+1}) I(t_{i+1})$

$$E(t_{i+1}) = \frac{E(t_i) + h\frac{\beta}{N} S(t_{i+1}) I(t_{i+1})}{1 + \omega h}$$

$$I(t_{i+1}) = I(t_i) + \omega h E(t_{i+1}) - h\gamma I(t_{i+1})$$

$I(t_{i+1}) + h\gamma I(t_{i+1}) = I(t_i) + \omega h E(t_{i+1})$

$I(t_{i+1})(1 + h\gamma) = I(t_i) + \omega h E(t_{i+1})$

$$I(t_{i+1}) = \frac{I(t_i) + \omega h E(t_{i+1})}{1 + h\gamma}$$

$$R(t_{i+1}) = R(t_i) + h\gamma I(t_{i+1})$$

$$\Rightarrow \begin{bmatrix} S(t_i) - (1 + h\frac{\beta}{N} I(t_{i+1})) S(t_{i+1}) = 0 \\ \frac{E(t_i) + h\frac{\beta}{N} S(t_{i+1}) I(t_{i+1})}{1 + \omega h} - E(t_{i+1}) = 0 \\ \frac{I(t_i) + \omega h E(t_{i+1})}{1 + h\gamma} - I(t_{i+1}) = 0 \\ R(t_i) + h\gamma I(t_{i+1}) - R(t_{i+1}) = 0 \end{bmatrix}$$

Jacobian:

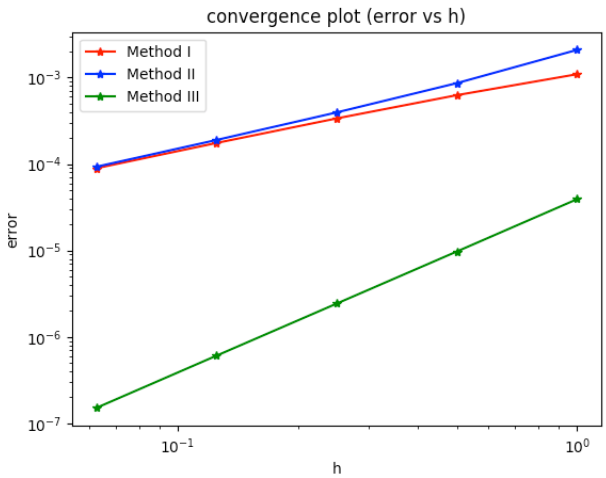
$$\begin{bmatrix} -1-h\frac{\beta}{N}I(t_{i+1}) & 0 & -h\frac{\beta}{N}S(t_{i+1}) & 0 \\ \frac{h\frac{\beta}{N}I(t_{i+1})}{1+wh} & -1 & \frac{h\frac{\beta}{N}S(t_{i+1})}{1+wh} & 0 \\ 0 & \frac{wh}{1+hr} & -1 & 0 \\ 0 & 0 & hr & -1 \end{bmatrix}$$

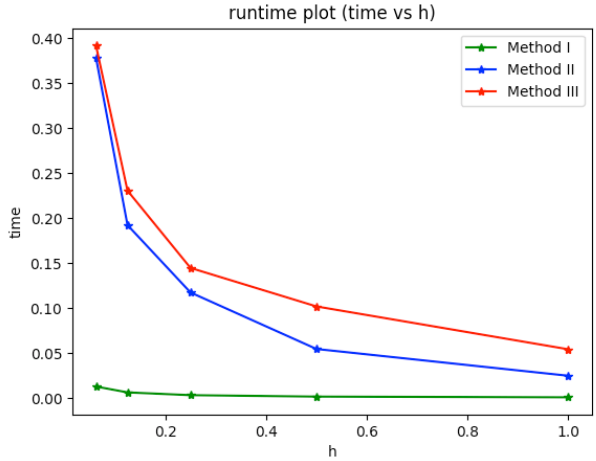
From method II to method III:

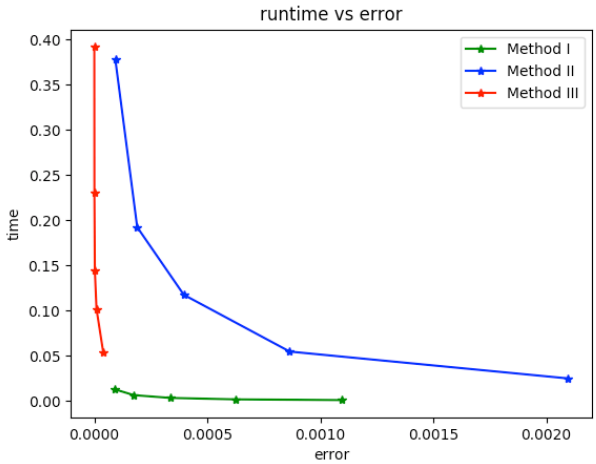
In method II, we choose $\eta = t_{i+1}$, but in method III, we consider both $\eta_1 = t_i$ and $\eta_2 = t_{i+1}$, and then take the midpoint.

b) F_jac $= \begin{bmatrix} -\frac{\beta}{N}I(t) & 0 & -\frac{\beta}{N}S(t) & 0 \\ \frac{\beta}{N}I(t) & -w & \frac{\beta}{N}S(t) & 0 \\ 0 & w & -r & 0 \\ 0 & 0 & r & 0 \end{bmatrix}$

c)







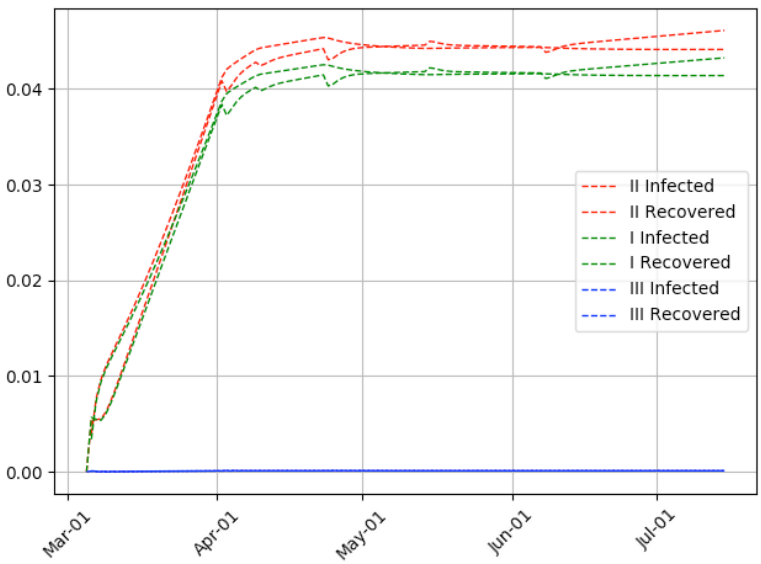
From the first graph, we can observe that all three methods' accuracy performances are getting worse when n increases, where method I and II has similar performance and method III performs much better.

From the second graph, we can see that if n gets larger, the time required by method II and III decreases dramatically at the beginning, and getting slower as n continue increases, but it seems that changing has no effect on time required for method I. Since from code we can see that method I shouldn't depend on h, it only calls a multiplication operation that involves h. As for the convergence rate, method I and II converges almost linearly, with method II's convergence rate is a little bit larger that method I's. We can see from graph 1 that the line for method II is little bit steeper. Method III converges

quadratically and we can see the line on graph 1 is much steeper than the other two. The converge rate seems to have no effect on runtime since we are iterating when curr_time == 0 until curr_time == end. Finally, the third graph told us that method I has least $\frac{\text{runtime}}{\text{error}}$ ratio, since its runtime is nearly independent from h.

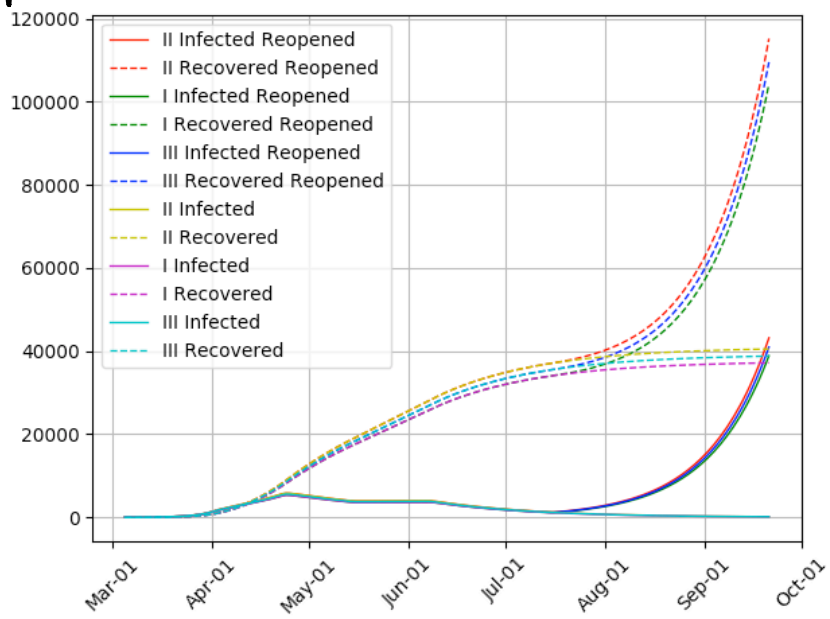
Method III has most $\frac{\text{runtime}}{\text{error}}$ ratio, since it converges quadratically $\Rightarrow$ its error is most sensitive to h, while its runtime's sensitivity to h is similar to method II. Therefore, method II is sitting in between method I and III.

d)



From the graph, we observe that the performance of method III is significantly better than others. It almost has no relative error, while the other two methods originally have less relative error, but more iterations are predicted, their relative errors are getting larger and converges to somewhere above 0.4. Since for method III, we are taking the midpoint between t_i and t_{i+1}, where method I takes t_n and method II takes t_{n+1} only. The trade off for doing more computation and increase runtime for method III is the decrease of the truncation error and rounding error when computing Jacobian and fsolve, which ultimately lead the significantly lower relative error.

e) Ontario is entering reopening process right now. Restaurants are opening for dine in. I want to find how will the trend be affected by the reopening in the next few month. I will have two sets of data, both of them are added 200 time units to the end, but one of them will have the last beta factor stays the same as previous one, the other will be added a 0.5 to the end to represent the reopen process.



The result from the graph is pretty resonable. Both reopened and non-reopened shares the same trend until Aug, since they have the same parameters previously. However starting from august, the amount of infected people in reopen model goes up quadratically while the data continue decreases in the non reopened model, since the restaurants are opened, people are getting closer to each other and we can not wear

masks when we are eating. Thus the amount of infected people is expected to increase. With more people are infected, sure there will be more people are recovered.