STÆ405G Töluleg greining
Verkefni 2

Andri Freyr Viðarsson og Valgeir Einarsson

Birgir Hrafnkelsson
22.mars 2020

1.

Solve the system (4.37) by using Multivariate Newtons Method. Find the receiver position (x, y, z) near earth and time correction d for known, simultaneous satellite positions (15600,7540,20140), (18760,2750,18610), (17610,14630,13480), (19170,610,18390) in km, and measured time intervals 0.07074,0.07220,0.07690,0.07242 in seconds, respectively. Set the initial vector to be $(x_0, y_0, z_0, d_0) = (0, 0, 6370, 0)$. As a check, the answers are approximately $(x, y, z) = (-41.77271, -16.78919, 6370.0596)$, and $d = -3.201566 \times 10^{-3}$ seconds.

Lausn:

```
import numpy as np
import scipy as sp
import sympy as sy

def F(x, y, z, d):
    F = np.zeros((4, 1))
    for i in range(4):
        F[i] = np.sqrt((x-L[i, 0])**2+(y-L[i, 1])**2+(z-L[i, 2])**2)-c*(t[i]-d)
    return F

def D_F(x, y, z, d):
    DF= np.zeros((4,4))
    # column by column(init)
    for i in range(4):
        DF[i, 0] = (x-L[i, 0])/(np.sqrt((x-L[i, 0])**2+(y-L[i, 1])**2+(z-L[i, 2])**2))
        DF[i, 1] = (y-L[i, 1])/(np.sqrt((x-L[i, 0])**2+(y-L[i, 1])**2+(z-L[i, 2])**2))
        DF[i, 2] = (z-L[i, 2])/(np.sqrt((x-L[i, 0])**2+(y-L[i, 1])**2+(z-L[i, 2])**2))
        DF[i, 3] = c
    return DF

def mult_newt(x_0, max_iter):
    x_i = x_0
    for i in range(max_iter):
        x, y, z, d = x_i[0], x_i[1], x_i[2], x_i[3]
        F_i = F(x, y, z, d)
        DF_i = D_F(x, y, z, d)
        DF_inv = np.linalg.inv(DF_i)
        x_i = x_i - (np.matmul(DF_inv, F_i))
    return x_i

l1 = [15600, 7540, 20140]
l2 = [18760, 2750, 18610]
l3 = [17610, 14630, 13480]
```

```
l4 = [19170, 610, 18390]
L = np.array([l1,l2,l3, l4]).reshape(-1, 3)
t1, t2, t3, t4 = 0.07074, 0.07220, 0.07690, 0.07242
t = np.array([t1, t2, t3, t4]).reshape(-1, 1)
x_0 = np.array([0, 0, 6370, 0]).reshape(-1,1)
c = 299792.458
x_out = mult_newt(x_0, 100)
print('Soulution is: (x,y,z) = (',x_out[0][0],',',x_out[1][0],',',x_out[2][0],')',
      'and d =', x_out[3][0],'seconds')
```

```
> Soulution is: (x,y,z) = ( -41.772709570867974 , -16.789194106538016 , 6370.059559223328 )
and d = -0.0032015658295942535 seconds
```

2.

Write a Matlab program to carry out the solution via the quadratic formula. Hint: Subtracting the last three equations of (4.37) from the first yields three linear equations in the four unknowns $x\vec{u}_x + y\vec{u}_y + z\vec{u}_z + d\vec{u}_d + \vec{w} = 0$, expressed in vector form. A formula for x in terms of d can be obtained from $0 = det[\vec{u}_y|\vec{u}_z|x\vec{u}_x + y\vec{u}_y + z\vec{u}_z + d\vec{u}_d + \vec{w}]$, noting that the determinant is linear in its columns and that a matrix with a repeated column has determinant zero. Similarly, we can arrive at formulas for y and z, respectively, in terms of d, that can be substituted in the first quadratic equation of (4.37), to make it an equation in one variable.

Lausn:

```
# here we will have to use (4.38)
A = L[:,0]
B = L[:, 1]
C = L[:, 2]

# construct necesarry vectors
u_x = -2*np.array([[A[0]-A[1]],[A[0]-A[2]],[A[0]-A[3]] ]).reshape(-1)
u_y = -2*np.array([[B[0]-B[1]],[B[0]-B[2]],[B[0]-B[3]] ]).reshape(-1)
u_z = -2*np.array([[C[0]-C[1]],[C[0]-C[2]],[C[0]-C[3]] ]).reshape(-1)
u_d = 2*c**2*np.array([[t1-t2], [t1-t3], [t1-t4]]).reshape(-1)
w = np.zeros(3)
for i in range(0,3):
    w[i] = A[0]**2-A[i+1]**2+B[0]**2-B[i+1]**2+C[0]**2-C[i+1]**2-c**2*t[0]**2+c**2*t[i+1]**2

# setup determinant for system
x, y, z, d = sy.symbols('x y z d')
matrix_1 = sy.Matrix([u_y, u_z, x*u_x+y*u_y+z*u_z+d*u_d+w]).T
matrix_2 = sy.Matrix([u_x, u_z, x*u_x+y*u_y+z*u_z+d*u_d+w]).T
matrix_3 = sy.Matrix([u_x, u_y, x*u_x+y*u_y+z*u_z+d*u_d+w]).T
det_1 = matrix_1.det()
det_2 = matrix_2.det()
det_3 = matrix_3.det()

# define equations
eq1 = sy.Eq(det_1)
eq2 = sy.Eq(det_2)
eq3 = sy.Eq(det_3)

# solve each equation in terms of x, y, z
x_1 = sy.solve(eq1, x)[0]
y_1 = sy.solve(eq2, y)[0]
z_1 = sy.solve(eq3, z)[0]
```

```
# insert into r1 (4.38)
expr_1 = (x_1-A[0])**2+(y_1-B[0])**2+(z_1-C[0])**2 -(c*(t1-d))**2
eq_d = sy.Eq(expr_1)
print(eq_d)
```

> $-(21207.31847892 - 299792.458d)^2 + (-83788.807142489d - 14038.1958226265)^2 + (-623.677195294773d - 7558.78593770368)^2 + (10.749177881392d - 15641.7382953702)^2 = 0$

Úttakið sýnir annars stigs jöfnu með tilltiti til d

```
# solve for d with "quadratic" solver
d_1 = sy.solve(eq_d, d)
# the quadratic formula has two solutions
# find coordinates
cords = np.zeros((2, 3))

for i, d_i in enumerate(d_1):
    eq_1 = eq1.subs(d, d_i)
    eq_2 = eq2.subs(d, d_i)
    eq_3 = eq3.subs(d, d_i)
    x_i = sy.solve(eq_1, x)
    y_i = sy.solve(eq_2, y)
    z_i = sy.solve(eq_3, z)
    cords[i,:]= np.array([x_i, y_i, z_i]).reshape(1,-1)

# find norm of each solution to determine which one is correct
for j, i in enumerate(cords):
    norm = np.linalg.norm(i)
    print(f'Norm of solution {j+1}:', norm)
```

> Norm of solution 1: 6370.218648080762
> Norm of solution 2: 9414.666041613695

Þar sem lengd lausnar 2 er mun meiri en radíus jarðar getum við ályktað að lausn 1 sé rétta lausnin.

```
cords_0 = list(cords[0])
cords_0.append(d_1[0])
print('Thus the correct solution is: (x,y,z) =',cords_0[0],',',cords_0[1],',',cords_0[2],')',
      'and d =',cords_0[3],'seconds')
```

> Thus the correct solution is: (x,y,z) = (-41.77270957083836 , -16.789194106527116 , 6370.059559223309) and d = -0.00320156582959406 seconds

3.

If the Matlab Symbolic Toolbox is available (or a symbolic package such as Maple or Mathematica), an alternative to Step 2 is possible. Define symbolic variables by using the syms command and solve the simultaneous equations with the Symbolic Toolbox command solve. Use subs to evaluate the symbolic result as a floating point number.

Lausn:

```
# use sy.solve for (4.38) straight up
expr_1 = (x-A[0])**2+(y-B[0])**2+(z-C[0])**2 -(c*(t1-d))**2
eq_1 = sy.Eq(expr_1)
expr_2 = (x-A[1])**2+(y-B[1])**2+(z-C[1])**2 -(c*(t2-d))**2
eq_2 = sy.Eq(expr_2)
expr_3 = (x-A[2])**2+(y-B[2])**2+(z-C[2])**2 -(c*(t3-d))**2
eq_3 = sy.Eq(expr_3)
expr_4 = (x-A[3])**2+(y-B[3])**2+(z-C[3])**2 -(c*(t4-d))**2
eq_4 = sy.Eq(expr_4)
sol = sy.solve((eq_1, eq_2, eq_3, eq_4), (x, y, z, d))
print('Soulution 1: (x,y,z) =(',sol[0][0],',',sol[0][1],',',sol[0][2],') and d =', sol[0][3],'
print('Soulution 1: (x,y,z) =(',sol[1][0],',',sol[1][1],',',sol[1][2],') and d =', sol[1][3],'
```

```
>    Soulution 1: (x,y,z) =( -41.7727095708172 , -16.7891941065184 , 6370.05955922335 ) and d
= -0.00320156582959406 seconds
>    Soulution 1: (x,y,z) =( -39.7478373482207 , -134.274144360683 , -9413.62455373582 ) and
d = 0.185173047095946 seconds
```

Úttakið sýnir að symbolíska lausnin er sú sama og lausnin sem fékkst í lið 2.

4.

Now set up a test of the conditioning of the GPS problem. Define satellite positions (A_i, B_i, C_i) from spherical coordinates (ρ, ϕ_i, θ_i) as

$$A_i = \rho\cos(\phi_i)\cos(\theta_i)$$
$$B_i = \rho\cos(\phi_i)\sin(\theta_i)$$
$$C_i = \rho\sin(\phi_i),$$

where $\rho = 26570$ km is fixed, while $0 \le \phi_i \le \pi/2$ and $0 \le \theta_i \le 2\pi$ for i = 1,...,4 are chosen arbitrarily. The ϕ coordinate is restricted so that the four satellites are in the upper hemisphere. Set x = 0, y = 0, z = 6370, d = 0.0001, and calculate the corresponding satellite ranges $R_i = \sqrt{A_i^2 + B_i^2 + (C_i - 6370)^2}$ and travel times $t_i = d + R_i/c$.

We will define an error magnification factor specially tailored to the situation. The atomic clocks aboard the satellites are correct up to about 10 nanoseconds, or 10^{-8} second. Therefore, it is important to study the effect of changes in the transmission time of this magnitude. Let the backward, or input error be the input change in meters. At the speed of light, $\Delta t_i = 10^{-8}$ second corresponds to $10^{-8}c \approx 3$ meters. Let the forward, or output error be the change in position $||(\Delta x, \Delta y, \Delta z)||_\infty$, caused by such a change in t_i, also in meters.
Then we can define the dimensionless

$$\text{error magnification factor} = \frac{||(\Delta x, \Delta y, \Delta z)||_\infty}{c||(\Delta t_1, ..., \Delta t_m)||_\infty},$$

and the condition number of the problem to be the maximum error magnification factor for all small Δt_i (say, 10^{-8} or less).

Change each t_i defined in the foregoing by $\Delta t_i = +10^{-8}$ or -10^{-8}, not all the same. Denote the new solution of the equations (4.37) by $(\bar{x}, \bar{y}, \bar{z}, \bar{d})$, and compute the difference in position $||(\Delta x, \Delta y, \Delta z)||_\infty$ and the error magnification factor. Try different variations of the Δt_i's. What is the maximum position error found, in meters? Estimate the condition number of the problem, on the basis of the error magnification factors you have computed.

Lausn:

```
# step out of the symbolic universe
rho = 26570
x, y, z, d = 0, 0, 6370, 0.0001
xyz = [x, y, z]

R_i = np.zeros(4)
t_i = np.zeros(4)

# setup
# choose
phi_i = np.linspace(0, np.pi/2,4)
theta_i = np.linspace(0, 2*np.pi, 4)
```

```
A = np.zeros(4)
B = np.zeros(4)
C = np.zeros(4)

for i in range(4):
    A[i] = rho*np.cos(phi_i[i])*np.cos(theta_i[i])
    B[i] = rho*np.cos(phi_i[i])*np.sin(theta_i[i])
    C[i] = rho*np.sin(phi_i[i])
    R_i[i] = np.sqrt(A[i]**2+B[i]**2+(C[i]-6370)**2)
    t_i[i] = d+(R_i[i]/c)

delta_t = 1e-8

# find all different combinations of t_i except when all are the same
from itertools import combinations
from itertools import chain

indices = [0, 1, 2, 3]
combs = []
for i in range(1,4):
    combs.append(list(combinations(indices, i)))
merged_combs = list(chain(*combs)) #2^4 -2

# solve symbolic each time
#base equations
x1, y1, z1, d1, t = sy.symbols('x y z d t')
expr_1 = (x1-A[0])**2+(y1-B[0])**2+(z1-C[0])**2 -(c*(t-d1))**2
expr_2 = (x1-A[1])**2+(y1-B[1])**2+(z1-C[1])**2 -(c*(t-d1))**2
expr_3 = (x1-A[2])**2+(y1-B[2])**2+(z1-C[2])**2 -(c*(t-d1))**2
expr_4 = (x1-A[3])**2+(y1-B[3])**2+(z1-C[3])**2 -(c*(t-d1))**2

exps = [expr_1, expr_2, expr_3, expr_4]
sols = []
forward_errs = []
magn_errs = []
for comb in merged_combs:
    t_temp = t_i[:] # copy by value
    exprs = [[] for i in range(4)]
    change = [True for i in range(4)]
    eqs = [[] for i in range(4)]
    for i in range(4):
        if i in comb:
```

```
            change[i] = False # change downwards
    for i in range(4):
        if change[i]:
            t_temp[i]+=delta_t
        else:
            t_temp[i]-=delta_t
        exprs[i] = exps[i].subs(t, t_temp[i]) # not in place so fine
        eqs[i] = sy.Eq(exprs[i])
    sol = sy.solve((eqs[0],eqs[1], eqs[2], eqs[3]), (x1, y1, z1, d1))
    sols.append(list(sol[1])) # correct solution
    xyz_bar = list(sol[1][0:3])
    comp = np.array(xyz_bar).reshape(-1,1)
    real = np.array(xyz).reshape(-1,1)
    forward_err = max(np.abs(comp-real))
    magn_err = forward_err[0]/(c*delta_t)
    forward_errs.append(forward_err)
    magn_errs.append(magn_err)

max_err = max(forward_errs)*10**3
cond_num = max(magn_errs)
print('Max error is:',max_err[0],' meters and condition number is:', cond_num)
```

```
>    Max error is: 11.9401958163508 meters and condition number is: 3.98282061396980
```

5.

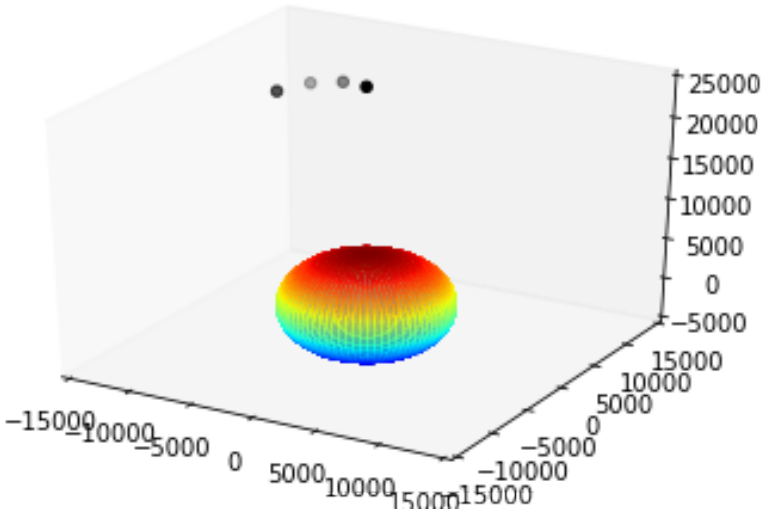
Now repeat Step 4 with a more tightly grouped set of satellites. Choose all ϕ_i within 5 percent of one another and all θ_i within 5 percent of one another. Solve with and without the same input error as in Step 4. Find the maximum position error and error magnification factor. Compare the conditioning of the GPS problem when the satellites are tightly or loosely bunched.
Lausn:

```
theta_step = 0.049* (2*np.pi)
phi_step = 0.049 *(0.5*np.pi)

# choose positon
theta_int = np.pi - 1.5*theta_step
phi_int = 0.5*np.pi
phi = [phi_int - phi_step*i for i in range(4)]
theta = [theta_int +theta_step*i for i in range(4)]

#plot satillites with respect to Earth
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d.axes3d as axes3d
theta_list, phi_list = np.linspace(0, 2 * np.pi, 80), np.linspace(0, np.pi, 80)
THETA, PHI = np.meshgrid(theta_list, phi_list)
R = 6370
X = R * np.sin(PHI) * np.cos(THETA)
Y = R * np.sin(PHI) * np.sin(THETA)
Z = R * np.cos(PHI)
x_list = rho*np.cos(phi)*np.cos(theta)
y_list = rho*np.cos(phi)*np.sin(theta)
z_list = rho*np.sin(phi)

fig = plt.figure()
ax = fig.add_subplot(1,1,1, projection='3d')
plot = ax.plot_surface(
    X, Y, Z, rstride=1, cstride=1, cmap=plt.get_cmap('jet'),
    linewidth=0, antialiased=False, alpha=0.5)
ax.scatter(x_list, y_list, z_list, color='black')
ax.grid(False)
ax.set_xlim3d(-15000, 15000)
ax.set_ylim3d(-15000,15000)
ax.set_zlim3d(-5000,25000)
plt.show()
```



Myndin hér á undan sýnir staðsetningu gervitunglana með tilliti til jarðarinnar.

```
# repeat step 4
# setup
R_i = np.zeros(4)
t_i = np.zeros(4)
phi_i = np.array(phi).reshape(-1)
theta_i = np.array(theta).reshape(-1)

A = np.zeros(4)
B = np.zeros(4)
C = np.zeros(4)
for i in range(4):
    A[i] = rho*np.cos(phi_i[i])*np.cos(theta_i[i])
    B[i] = rho*np.cos(phi_i[i])*np.sin(theta_i[i])
    C[i] = rho*np.sin(phi_i[i])
    R_i[i] = np.sqrt(A[i]**2+B[i]**2+(C[i]-6370)**2)
    t_i[i] = d+(R_i[i]/c)

delta_t = 1e-8

# find all different combinations of t_i except when all are the same

indices = [0, 1, 2, 3]
combs = []
for i in range(1,4):
    combs.append(list(combinations(indices, i)))
merged_combs = list(chain(*combs)) #2^4 -2
```

```
# solve symbolic each time
#base equations
x1, y1, z1, d1, t = sy.symbols('x y z d t')
expr_1 = (x1-A[0])**2+(y1-B[0])**2+(z1-C[0])**2 -(c*(t-d1))**2
expr_2 = (x1-A[1])**2+(y1-B[1])**2+(z1-C[1])**2 -(c*(t-d1))**2
expr_3 = (x1-A[2])**2+(y1-B[2])**2+(z1-C[2])**2 -(c*(t-d1))**2
expr_4 = (x1-A[3])**2+(y1-B[3])**2+(z1-C[3])**2 -(c*(t-d1))**2

exps = [expr_1, expr_2, expr_3, expr_4]
sols = []
forward_errs = []
magn_errs = []
for comb in merged_combs:
    t_temp = t_i[:] # copy by value
    exprs = [[] for i in range(4)]
    change = [True for i in range(4)]
    eqs = [[] for i in range(4)]
    for i in range(4):
        if i in comb:
            change[i] = False # change downwards
    for i in range(4):
        if change[i]:
            t_temp[i]+=delta_t
        else:
            t_temp[i]-=delta_t
        exprs[i] = exps[i].subs(t, t_temp[i]) # not in place so fine
        eqs[i] = sy.Eq(exprs[i])
    sol = sy.solve((eqs[0],eqs[1], eqs[2], eqs[3]), (x1, y1, z1, d1))
    sols.append(list(sol[1])) # correct solution
    xyz_bar = list(sol[1][0:3])
    comp = np.array(xyz_bar).reshape(-1,1)
    real = np.array(xyz).reshape(-1,1)
    forward_err = max(np.abs(comp-real))
    magn_err = forward_err[0]/(c*delta_t)
    forward_errs.append(forward_err)
    magn_errs.append(magn_err)

max_err = max(forward_errs)*10**3
cond_num = max(magn_errs)
print('Max error is:',max_err[0],'meters and condition number is:', cond_num)
```

```
>    Max error is: 4625.36426610495 meters and condition number is: 1542.85544638516
```

Sjáum að skekkjan er miklu hærri fyrir þétta dreifingu gervihnatta en fyrir víða dreifingu, ástandstala verkefnisins hækkar einnig þegar hnettirnir eru færðir nær hvor öðrum.

6.

Decide whether the GPS error and condition number can be reduced by adding satellites. Return to the unbunched satellite configuration of Step 4, and add four more. (At all times and at every position on earth, 5 to 12 GPS satellites are visible.) Design a Gauss–Newton iteration to solve the least squares system of eight equations in four variables (x, y, z, d). What is a good initial vector? Find the maximum GPS position error, and estimate the condition number. Summarize your results from four unbunched, four bunched, and eight unbunched satellites. What configuration is best, and what is the maximum GPS error, in meters, that you should expect solely on the basis of satellite signals?

Lausn:

```
phi_i = np.linspace(0, np.pi/2,8)
theta_i = np.linspace(0, 2*np.pi, 8)
A = np.zeros(8)
B = np.zeros(8)
C = np.zeros(8)
t_i =[[] for i in range(8)]
R_i = np.zeros(8)

for i in range(8):
    A[i] = rho*np.cos(phi_i[i])*np.cos(theta_i[i])
    B[i] = rho*np.cos(phi_i[i])*np.sin(theta_i[i])
    C[i] = rho*np.sin(phi_i[i])
    R_i[i] = np.sqrt(A[i]**2+B[i]**2+(C[i]-6370)**2)
    t_i[i] = d+(R_i[i]/c)

t= t_i
L = np.zeros((8, 3))
L[:,0] = A.reshape(-1)
L[:,1] = B.reshape(-1)
L[:, 2] = C.reshape(-1)

def F(x, y, z, d, t):
    F = np.zeros((8, 1))
    for i in range(8):
        F[i] = np.sqrt((x-L[i, 0])**2+(y-L[i, 1])**2+(z-L[i, 2])**2)-c*(t[i]-d)
    return F

def D_F(x, y, z, d):
    DF= np.zeros((8,4))
    # column by column(init)
    for i in range(8):
```

```
        DF[i, 0] = (x-L[i, 0])/(np.sqrt((x-L[i, 0])**2+(y-L[i, 1])**2+(z-L[i, 2])**2))
        DF[i, 1] = (y-L[i, 1])/(np.sqrt((x-L[i, 0])**2+(y-L[i, 1])**2+(z-L[i, 2])**2))
        DF[i, 2] = (z-L[i, 2])/(np.sqrt((x-L[i, 0])**2+(y-L[i, 1])**2+(z-L[i, 2])**2))
        DF[i, 3] = c
    return DF

def gauss_new(max_iter, x_0, t):
    x_k = x_0
    for i in range(max_iter):
        A = D_F(*x_k)
        r_k = F(*x_k, t)
        left = np.matmul(A.T, A)
        right = -np.matmul(A.T,r_k)
        try:
            vec = np.linalg.solve(left, right)
        except Exception:
            break
        x_temp = np.array(x_k).reshape(-1,1) +vec
        x_k = x_temp
    return x_k

# slightly modified version of 4
indices = [i for i in range(8)]
combs = []
xyz = np.array([0, 0, 6370]).reshape(-1,1)
for i in range(1,8):
    combs.append(list(combinations(indices, i)))
merged_combs = list(chain(*combs)) #2^8 -2

forward_errors = []
magn_errors = []
for comb in merged_combs:
    t_temp = t[:]
    change = [True for i in range(8)]
    for i in range(8):
        if i in comb:
            change[i] = False # change downwards
    for i in range(8):
        if change[i]:
            t_temp[i]+=delta_t
        else:
```

```
            t_temp[i]-=delta_t
    sol = gauss_new(100, x_0, t_temp)
    forward_err = max(np.abs(sol[0:3,:]-xyz))
    magn_err = forward_err[0]/(c*delta_t)
    forward_errors.append(forward_err)
    magn_errors.append(magn_err)

max_err = max(forward_errors)*10**3
cond_num = max(magn_errors)
print('Max error is:',max_err[0],'meters and condition number is:', cond_num)
```

```
> Max error is: 8.300222503748955 meters and condition number is: 2.7686562094063607
```

Úttakið að ofan sýnir að fjölgun gervitungla lækkar skekkjuna og ástandstöluna þó nokkuð. Val á upphafsgildi virðist ekki hafa nein áhrif á skekkjuna. Þegar teknar eru saman niðurstöður úr liðum 4,5,6 kemur í ljós að fjölgun gervihnatta sem eru ekki nálægt hvor öðrum eykur nákvæmni.

```
mean_err = np.mean(forward_errors)*10**3
print('Mean error is:',mean_err,'meters')
```

```
> Mean error is: 3.8564074627218794 meters
```

Væntanleg skekkja ef aðeins er tekið tillit til skekkju í merki frá gervihnetti ætti að vera háð skekkju í mismuni klukkunnar á jörðu og í gervihnetti, sem er $\Delta t_i \cdot c \approx 3\text{m}$. Ef meðalskekkjan er reiknuð þá sést að hún er um það bil 3m sem styður við þá kenningu að væntanleg skekkja er 3m.

Andri Freyr Viðarsson xxx
Valgeir Einarsson xxx