In [2]:

```

```

In [4]:

```python
from IPython.display import Image
```

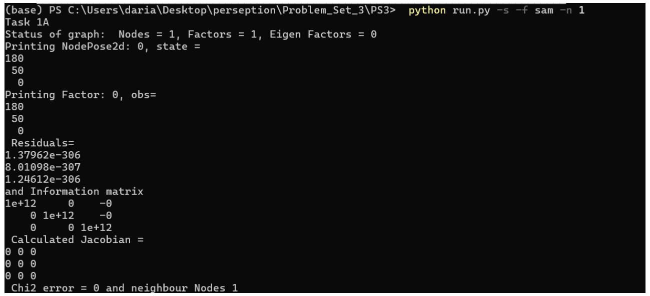
Type *Markdown* and LaTeX: α^2

In [11]:

```python

print('TASK 1 A')
Image("C:/Users/daria\Desktop\perseption\Problem_Set_3\PS3\photos\Task_1_A_res.png", width=320, height=240)
```

TASK 1 A

Out[11]:



```
(base) PS C:\Users\daria\Desktop\perseption\Problem_Set_3\PS3>  python run.py -s -f sam -n 1
Task 1A
Status of graph:  Nodes = 1, Factors = 1, Eigen Factors = 0
Printing NodePose2d: 0, state =
180
 50
  0
Printing Factor: 0, obs=
180
 50
  0
 Residuals=
1.37962e-306
8.01098e-307
1.24612e-306
and Information matrix
1e+12      0     -0
    0 1e+12     -0
    0      0 1e+12
 Calculated Jacobian =
0 0 0
0 0 0
0 0 0
 Chi2 error = 0 and neighbour Nodes 1
```

In [24]:

```python
print('Code 1A')
Image("C:/Users/daria\Desktop\perseption\Problem_Set_3\PS3\photos\code_1A.png", width=320, height=240)
```

Code 1A

Out[24]:

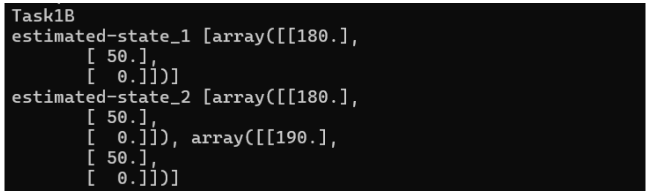
```python
self.graph = mrob.FGraph()
#added part
print('Task 1A')

self.mu = initial_state.mu
self.sigma = initial_state.Sigma
self.sigma = inv(self.sigma)
self.pose = self.graph.add_node_pose_2d(self.mu)
self.graph.add_factor_1pose_2d(self.mu,  *args: self.pose, self.sigma)
self.chi2 = []
self.graph.print(True)

self.landmarks = {}
self.landmarks_node = []
self.skip = []
self.estimate = {}
```

Type *Markdown* and LaTeX: α^2

In [12]:
```
print('TASK 1 B')
Image("C:/Users\daria\Desktop\perseption\Problem_Set_3\PS3\photos\Task_1_B_res.png", width=320, height=240)
```

TASK 1 B

Out[12]:

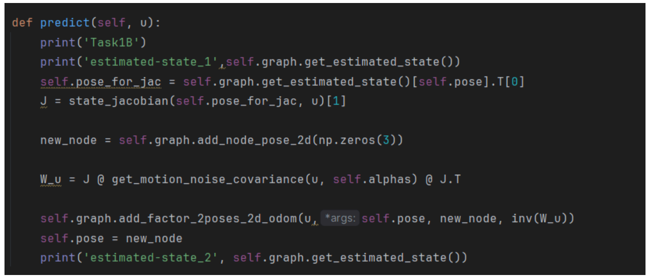


```
Task1B
estimated-state_1 [array([[180.],
       [ 50.],
       [  0.]])]
estimated-state_2 [array([[180.],
       [ 50.],
       [  0.]]), array([[190.],
       [ 50.],
       [  0.]])]
```

In [23]:
```
print('Code 1 B')
Image("C:/Users\daria\Desktop\perseption\Problem_Set_3\PS3\photos\code_1B.png", width=320, height=240)
```

Code 1 B

Out[23]:



```
def predict(self, u):
    print('Task1B')
    print('estimated-state_1',self.graph.get_estimated_state())
    self.pose_for_jac = self.graph.get_estimated_state()[self.pose].T[0]
    J = state_jacobian(self.pose_for_jac, u)[1]

    new_node = self.graph.add_node_pose_2d(np.zeros(3))

    W_u = J @ get_motion_noise_covariance(u, self.alphas) @ J.T

    self.graph.add_factor_2poses_2d_odom(u, *args: self.pose, new_node, inv(W_u))
    self.pose = new_node
    print('estimated-state_2', self.graph.get_estimated_state())
```

TASK 1 C

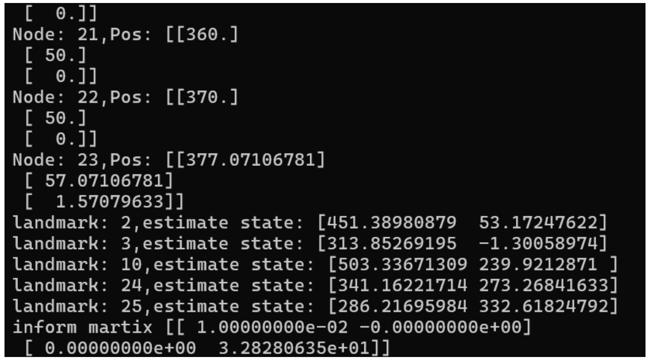
In [13]:
```
Image("C:/Users\daria\Desktop\perseption\Problem_Set_3\PS3\photos/task1_c_1.png", width=320, height=240)
```

Out[13]:

```
Node: 0,Pos: [[180.]
 [ 50.]
 [  0.]]
Node: 1,Pos: [[190.]
 [ 50.]
 [  0.]]
Node: 4,Pos: [[200.]
 [ 50.]
 [  0.]]
Node: 5,Pos: [[210.]
 [ 50.]
 [  0.]]
Node: 6,Pos: [[220.]
 [ 50.]
 [  0.]]
Node: 7,Pos: [[230.]
 [ 50.]
 [  0.]]
Node: 8,Pos: [[240.]
 [ 50.]
 [  0.]]
Node: 9,Pos: [[250.]
 [ 50.]
 [  0.]]
Node: 11,Pos: [[260.]
 [ 50.]
 [  0.]]
Node: 12,Pos: [[270.]
 [ 50.]
 [  0.]]
Node: 13,Pos: [[280.]
 [ 50.]
 [  0.]]
Node: 14,Pos: [[290.]
 [ 50.]
 [  0.]]
Node: 15,Pos: [[300.]
 [ 50.]
 [  0.]]
```

In [14]:
```
Image("C:/Users\daria\Desktop\perseption\Problem_Set_3\PS3\photos/task_1_c_2.png", width=320, height=240)
```

Out[14]:

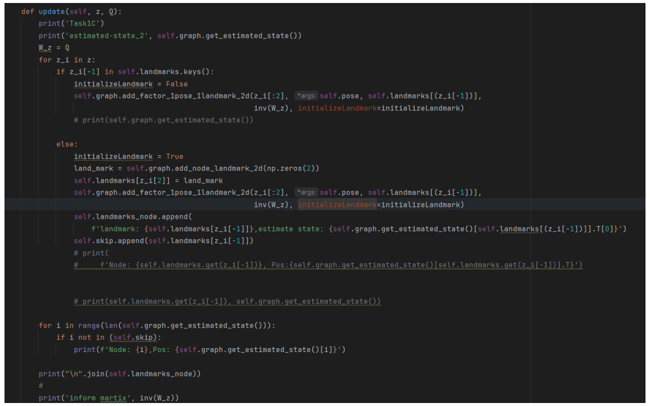


```
 [  0.]]
Node: 21,Pos: [[360.]
 [ 50.]
 [  0.]]
Node: 22,Pos: [[370.]
 [ 50.]
 [  0.]]
Node: 23,Pos: [[377.07106781]
 [ 57.07106781]
 [  1.57079633]]
landmark: 2,estimate state: [451.38980879  53.17247622]
landmark: 3,estimate state: [313.85269195  -1.30058974]
landmark: 10,estimate state: [503.33671309 239.9212871 ]
landmark: 24,estimate state: [341.16221714 273.26841633]
landmark: 25,estimate state: [286.21695984 332.61824792]
inform martix [[ 1.00000000e-02 -0.00000000e+00]
 [ 0.00000000e+00  3.28280635e+01]]
```

In [22]:
```
print('code 1C')
Image("C:/Users\daria\Desktop\perseption\Problem_Set_3\PS3\photos/code_1C.png", width=320, height=240)
```

code 1C

Out[22]:



```
def update(self, z, Q):
    print('Task1C')
    print('estimated-state_2', self.graph.get_estimated_state())
    W_z = Q
    for z_i in z:
        if z_i[-1] in self.landmarks.keys():
            initializeLandmark = False
            self.graph.add_factor_1pose_1landmark_2d(z_i[:2], *args: self.pose, self.landmarks[(z_i[-1])],
                                                     inv(W_z), initializeLandmark=initializeLandmark)
            # print(self.graph.get_estimated_state())

        else:
            initializeLandmark = True
            land_mark = self.graph.add_node_landmark_2d(np.zeros(2))
            self.landmarks[z_i[2]] = land_mark
            self.graph.add_factor_1pose_1landmark_2d(z_i[:2], *args: self.pose, self.landmarks[(z_i[-1])],
                                                     inv(W_z), initializeLandmark=initializeLandmark)
            self.landmarks_node.append(
                f'landmark: {self.landmarks[z_i[-1]]},estimate state: {self.graph.get_estimated_state()[self.landmarks[(z_i[-1])]].T[0]}')
            self.skip.append(self.landmarks[z_i[-1]])
            # print(
            #     f'Node: {self.landmarks.get(z_i[-1])}, Pos:{self.graph.get_estimated_state()[self.landmarks.get(z_i[-1])].T}')


            # print(self.landmarks.get(z_i[-1]), self.graph.get_estimated_state())

    for i in range(len(self.graph.get_estimated_state())):
        if i not in (self.skip):
            print(f'Node: {i},Pos: {self.graph.get_estimated_state()[i]}')

    print("\n".join(self.landmarks_node))
    #
    print('inform martix', inv(W_z))
```

New function gives to us more information. Gauss-Newton add more ditails about estimation and poses of our robot

TASK 1 D-

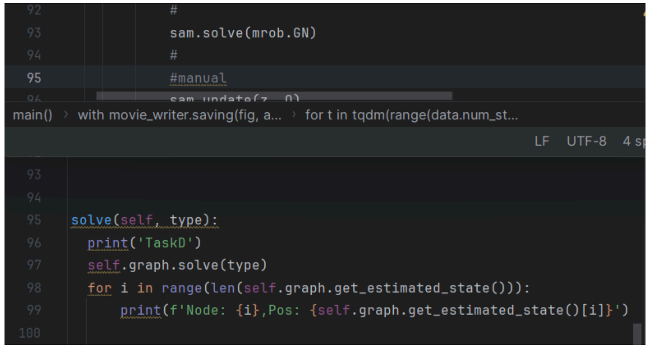
In [20]:
```
Image("C:/Users\daria\Desktop\perseption\Problem_Set_3\PS3\photos/Task_1_D.png", width=320, height=240)
```

Out[20]:
```
TaskD
Node: 0,Pos: [[1.80000000e+02]
 [5.00000000e+01]
 [1.18460566e-13]]
Node: 1,Pos: [[1.90000000e+02]
 [5.00001185e+01]
 [2.36921134e-05]]
Node: 2,Pos: [[467.20476224]
 [  0.88213239]]
Node: 3,Pos: [[317.49146288]
 [  7.73966435]]
Node: 4,Pos: [[2.00045301e+02]
 [5.00186572e+01]
 [3.62517477e-03]]
Node: 5,Pos: [[2.10054954e+02]
 [5.00671383e+01]
 [5.97725355e-03]]
Node: 6,Pos: [[2.20033540e+02]
 [5.01331359e+01]
 [7.16992733e-03]]
Node: 7,Pos: [[2.30048079e+02]
 [5.02048981e+01]
 [7.16144571e-03]]
Node: 8,Pos: [[2.40049923e+02]
 [5.02713796e+01]
 [6.11955049e-03]]
Node: 9,Pos: [[2.50064066e+02]
 [5.03258702e+01]
 [4.78200064e-03]]
Node: 10,Pos: [[2.60165377e+02]
 [5.03802453e+01]
 [6.06344313e-03]]
Node: 11,Pos: [[2.70208281e+02]
```

In [21]:
```
print('Code 1D')
Image("C:/Users\daria\Desktop\perseption\Problem_Set_3\PS3\photos/code_1D.png", width=320, height=240)
```

```
Code 1D
```

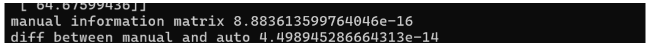
Out[21]:



Type *Markdown* and LaTeX: α^2

In [16]:
```
Image("C:/Users\daria\Desktop\perseption\Problem_Set_3\PS3\photos/Task_1_EF.png", width=320, height=240)
```

Out[16]:



We can see that both results of calculation really very closed. So we can say that using automatic functions or manually it is on you

Type *Markdown* and LaTeX: α^2

In [18]:
```python
print('Code task 1E-F')
Image("C:/Users\daria\Desktop\perseption\Problem_Set_3\PS3\photos/code_1_EF.png", width=320, height=240)
```

Code task 1E-F

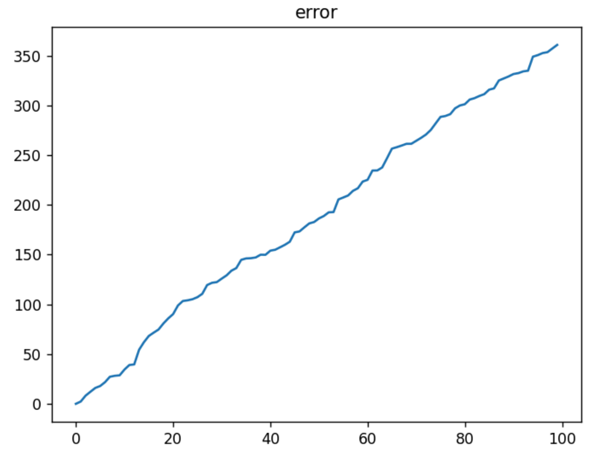
Out[18]:
```python
#manual
sam.update(z, Q)

state_1 = []
state_2 = []
st = sam.graph.get_estimated_state()

for s in st:
    flat = list(s.flatten())
    state_1 = state_1 + flat
state_1 = np.array(state_1)

sam.solve(mrob.GN)

st = sam.graph.get_estimated_state()

for s in st:
    flat = list(s.flatten())
    state_2 = state_2 + flat
state_2 = np.array(state_2)

A = sam.graph.get_adjacency_matrix()
W = sam.graph.get_W_matrix()
I = sam.graph.get_information_matrix()

inf_matrix = A.todense().T @ W.todense() @ A.todense
norm_inf_m = norm(I - inf_matrix)

b = sam.graph.get_vector_b()

d_x = inv(inf_matrix) @ b
norm_d_x = norm(d_x - (state_1 - state_2))

print('manual information matrix',norm_inf_m_)
print('diff between manual and auto', norm_d_x)
```

TASK2

In [26]:
```
print('TASK 2A')
Image("C:/Users\daria\Desktop\perseption\Problem_Set_3\PS3\photos/error.png", width=320, height=240)
```

TASK 2A

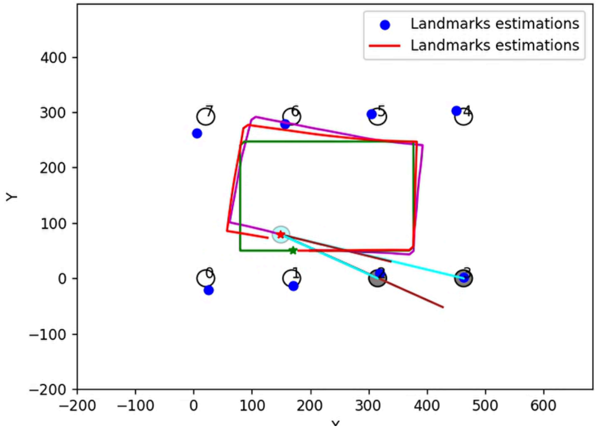
Out[26]:



In [28]:
```
print('TASK 2B')
Image("C:/Users\daria\Desktop\perseption\Problem_Set_3\PS3\photos/task2b.png", width=320, height=240)
```

TASK 2B

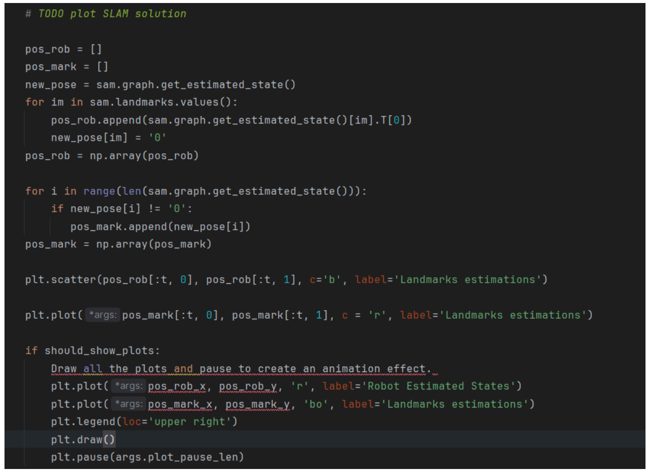
Out[28]:



In [27]:
```
print('Code 2B')
Image("C:/Users\daria\Desktop\perseption\Problem_Set_3\PS3\photos/code2b.png", width=320, height=240)
```

Code 2B

Out[27]:

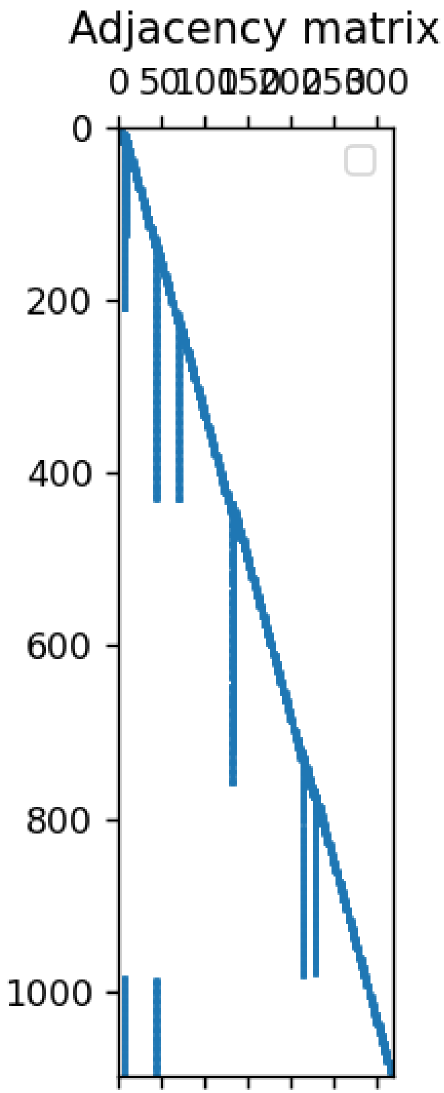


```
# TODO plot SLAM solution

pos_rob = []
pos_mark = []
new_pose = sam.graph.get_estimated_state()
for im in sam.landmarks.values():
    pos_rob.append(sam.graph.get_estimated_state()[im].T[0])
    new_pose[im] = '0'
pos_rob = np.array(pos_rob)

for i in range(len(sam.graph.get_estimated_state())):
    if new_pose[i] != '0':
        pos_mark.append(new_pose[i])
pos_mark = np.array(pos_mark)

plt.scatter(pos_rob[:t, 0], pos_rob[:t, 1], c='b', label='Landmarks estimations')

plt.plot( *args: pos_mark[:t, 0], pos_mark[:t, 1], c = 'r', label='Landmarks estimations')

if should_show_plots:
    Draw all the plots and pause to create an animation effect.
    plt.plot( *args: pos_rob_x, pos_rob_y, 'r', label='Robot Estimated States')
    plt.plot( *args: pos_mark_x, pos_mark_y, 'bo', label='Landmarks estimations')
    plt.legend(loc='upper right')
    plt.draw()
    plt.pause(args.plot_pause_len)
```

In [30]:
```
print('TASK 2C')
Image("C:/Users\daria\Desktop\perseption\Problem_Set_3\PS3\photos/matrix_1.png", width=320, height=240)
```

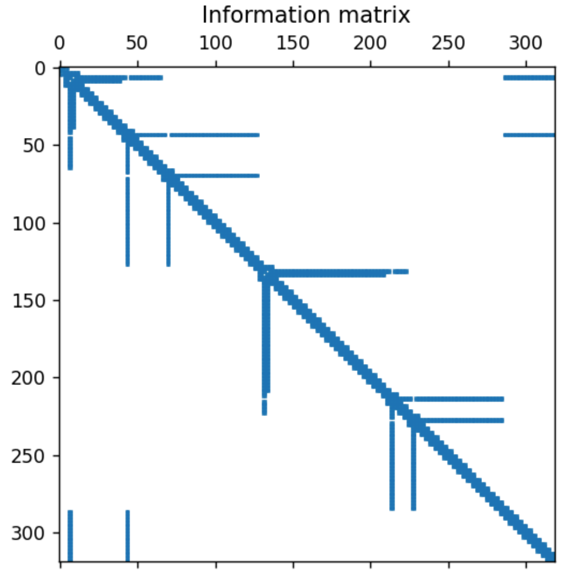
```
TASK 2C
```

Out[30]:



In [31]:
```python
Image("C:/Users\daria\Desktop\perseption\Problem_Set_3\PS3\photos/matrix_2.png", width=320, height=240)
```

Out[31]:



In [32]:
```python
print('Code 2C')
Image("C:/Users\daria\Desktop\perseption\Problem_Set_3\PS3\photos/code2c.png", width=320, height=240)
```

```
Code 2C
```

Out[32]:

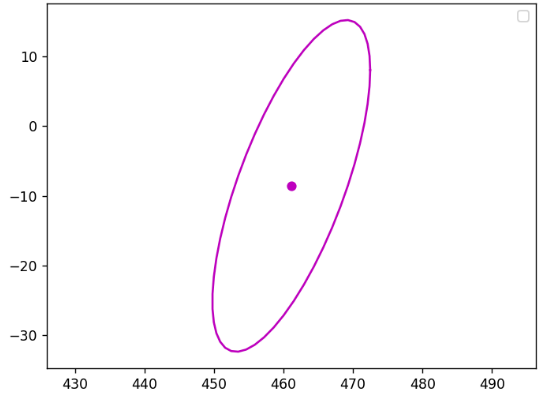
```python
plt.title("Adjacency matrix")
plt.spy(sam.graph.get_adjacency_matrix())
plt.show(True)

plt.title("Information matrix")
plt.spy(sam.graph.get_information_matrix())
plt.show(True)
```

We can see relationships between states and factors. We can see that information matrix has more relations

In [33]:
```
print('TASK 2D')
Image("C:/Users\daria\Desktop\perseption\Problem_Set_3\PS3\photos/iso.png", width=320, height=240)
```

TASK 2D

Out[33]:



In [35]:
```
print('Code 2D')
Image("C:/Users\daria\Desktop\perseption\Problem_Set_3\PS3\photos/code2d.png", width=320, height=240)
```

Code 2D

Out[35]:

```
#iso
plot2dcov(sam.graph.get_estimated_state()[-1].T[0, :-1], scipy.sparse.inv(sam.graph.get_information_matrix()[-3:-1, -3:-1].todense(), 'm', 3))
plt.axis('equal')
plt.show(block=True)
```

In [37]:
```
print('TASK 2E')
Image("C:/Users\daria\Desktop\perseption\Problem_Set_3\PS3\photos/task2e.png", width=320, height=240)
```

TASK 2E

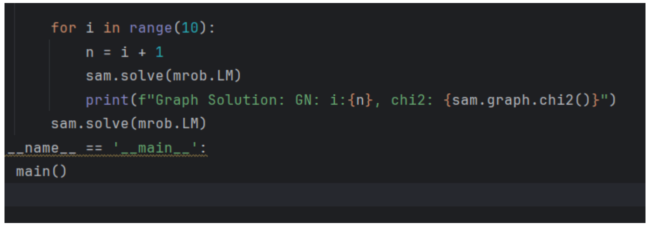
Out[37]:

```
Graph Solution: GN: i:3, chi2: 19.651626371764653
```

In [36]:
```python
print('Code 2E')
Image("C:/Users\daria\Desktop\perseption\Problem_Set_3\PS3\photos/code2e.png", width=320, height=240)
```

Code 2E

Out[36]:



```python
    for i in range(10):
        n = i + 1
        sam.solve(mrob.LM)
        print(f"Graph Solution: GN: i:{n}, chi2: {sam.graph.chi2()}")
    sam.solve(mrob.LM)
__name__ == '__main__':
 main()
```

In [ ]: