```
import numpy as np
import matplotlib.pyplot as plt
```

Задание №1

```
N = 15 # Номер варианта
n = 6 # Размер матрицы
b = np.full(n, fill_value=N, dtype=float) # вектор свободных членов
A = np.empty((n, n))

# Пункт №1
for i in range(n):    # Заполняю массив
    for j in range(n):
        C = 0.1 * N * (i + 1) * (j + 1)
        A[i, j] = 88.5 / (C + 0.03*(C**2))

x_real = np.linalg.solve(A, b)
print("Решение системы уравнений при помощи встроенной функции: \n x = ", x_real, '\n')

# Пункт №2
cond_value = np.linalg.cond(np.abs(A), p=np.inf)
print("Число обусловленности матрицы A = ", cond_value, '\n')

# Пункт №3
delta = 0.05 # Выбираю произвольную погрешность за 0.05
# x_i_vector = np.empty((n, n)) # i - номер решения. В каждом x_i есть n координат
d_i = np.empty((n))

for i in range(n):
    b_i = b.copy()
    b_i[i] += delta
    x_i = np.linalg.solve(A, b_i)
    d_i[i] = np.linalg.norm(x_real - x_i, ord=np.inf) / np.linalg.norm(x_real, ord=np.inf)
    print("x_", i + 1, " = ", x_i, "\n b_", i + 1, " = ", b_i[i], '\n d_', i+1, " = ", d_i[i])

# Пункт №4
plt.bar(range(1,n+1), d_i)
plt.xlabel('Компоненты вектора')

d_max = np.argmax(d_i)
print("\nКомпонента b_i, которая оказывает наибольшее влияние на погрешность: b_",d_max + 1, " = ", b_i[d_m

# Пункт №5
b_max_vec = b.copy()
b_max_vec[d_max] += delta

delta_rel_b = delta / np.linalg.norm(b_max_vec)
delta_rel_x = cond_value * delta_rel_b
print("Относительная погрешность решения б(x_m) = ", delta_rel_x)
print("Значение практической погрешности d_m = ", d_i[d_max])
```

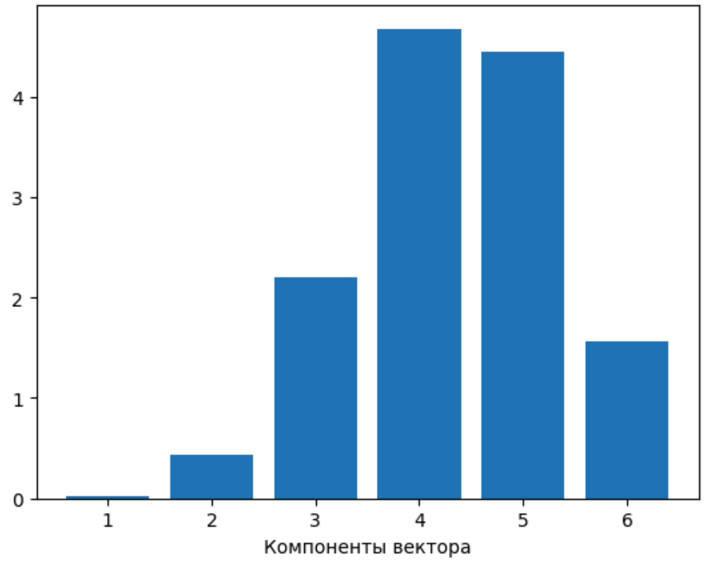
```
Решение системы уравнений при помощи встроенной функции:
 x =  [    192.68325252  -2434.08877772   9813.54467185 -17909.59038672
  15286.8551612   -4961.25854109]

Число обусловленности матрицы A =  625076108.0393722

x_ 1  =  [    190.96416663  -2399.64055993   9624.36746646 -17478.54956502
  14850.7847509   -4799.1386015 ]
 b_ 1  =  15.05
 d_ 1  =  0.024348430136322315
x_ 2  =  [    227.13147031  -3099.19443163  13347.33141296 -25726.60442064
  22987.09222326  -7755.72450348]
 b_ 2  =  15.05
 d_ 2  =  0.43647084411874615
x_ 3  =  [ 3.50604713e+00  1.09969796e+03 -8.45227411e+03  2.15661687e+04
 -2.28310212e+04  8.63477981e+03]
 b_ 3  =  15.05
 d_ 3  =  2.204168730273599
x_ 4  =  [     623.72407421  -10251.1028116    49289.30377419 -101635.2362
   94889.16134713  -32987.40343853]
 b_ 4  =  15.05
 d_ 4  =  4.674905679324354
x_ 5  =  [   -243.38715778   5266.14828429 -28304.331725    61692.71579906
 -59463.38881912  21091.34518245]
 b_ 5  =  15.05
 d_ 5  =  4.444674862290216
x_ 6  =  [    354.8031921   -5228.5547401   23409.58302342 -45935.73528408
  41339.4588847  -13967.94722448]
 b_ 6  =  15.05
 d_ 6  =  1.5648680004499873

Компонента b_i, которая оказывает наибольшее влияние на погрешность: b_ 4  =  15.0

Относительная погрешность решения б(x_m) =  850147.8817303687
Значение практической погрешности d_m =  4.674905679324354
```



Задание №2

```
b_i = np.array([[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]]) # Беру n линейно-независимых векторов b_i
A = np.array([[1,1,1,1],[8,4,2,1],[27,9,3,1],[64,16,4,1]])
x_i = np.empty((4, 4))

# Изначально брала координатные вектора bi, однако результат получался плохим, поэтому решила попробовать
def generate_lin_independent_vectors(k, n):
  vectors = []
  result_vectors = []
  while(len(vectors) != k):
    v = np.random.rand(n)     # Генерация случайного вектора
    vectors.append(v)    # Добавление вектора к списку

    q, r = np.linalg.qr(np.array(vectors).T)     # Выполнение QR-разложения для всех векторов
    lin_indep_vectors = []     # Выбор линейно независимых векторов

    for i in range(len(vectors)):
      # Проверка, не является ли i-й вектор линейной комбинацией предыдущих
      if np.linalg.matrix_rank(r[:, :i+1]) == i+1:
        lin_indep_vectors.append(vectors[i])
    if(len(vectors) == k):
      result_vectors = lin_indep_vectors
  return result_vectors

# Количество векторов и их размерность
k = 4
n = 4

# Генерация k линейно независимых векторов размерности n
b_i_new = generate_lin_independent_vectors(k, n)
print("Сгенерированные линейно независимые векторы:")
for i, v in enumerate(b_i_new):
  print(f"Вектор {i + 1}: {v}")

# Пункт №1
for i in range(4):
    x_i[i] = np.linalg.solve(A, b_i[i])
    print("Решение системы уравнений при b_i - координатные вектора: \n x_", i+1, "= " , x_i[i], '\n')

# Решение при k лнз векторов
x_i_new = np.zeros((k, 4))

for i in range(k):
    x_i_new[i] = np.linalg.solve(A, b_i_new[i])
    print("Решение системы уравнений при помощи генерации b_i: \n x_", i+1, "= " , x_i_new[i], '\n')



# Пункт №2
d_i = np.empty((k))
for i in range(k):
    d_i[i] = np.linalg.norm(x_i[i])/ np.linalg.norm(b_i_new[i])
    print("Отношение d_", i+1, "= " , d_i[i])

# Пункт №3
A_inverse_norm = np.argmax(d_i)
print("\nНорма обратной матрицы, вычисленная через максимум отношений нормы решения к норме вектора \nсво

A_norm_real = np.linalg.norm(np.linalg.inv(A))
print("Норма обратной матрицы, вычисленная через встроенную функцию = ", A_norm_real)

# Пункт №4
A_cond_new = np.linalg.norm(A) *  A_norm_real
print("\nЧисло обусловленности матрицы A = ", A_cond_new, '\n')
```

```
Сгенерированные линейно независимые векторы:
Вектор 1: [0.9424526  0.3218848  0.36262685 0.4536002 ]
Вектор 2: [0.24652158 0.61703741 0.33356197 0.01486061]
Вектор 3: [0.48434699 0.89315482 0.1011223  0.56024925]
```

```
Вектор 4: [0.62324761 0.99137831 0.54099269 0.45498333]
Решение системы уравнений при b_i - координатные вектора:
 x_ 1 =  [-0.16666667  1.5        -4.33333333  4.        ]

Решение системы уравнений при b_i - координатные вектора:
 x_ 2 =  [ 0.5 -4.   9.5 -6. ]

Решение системы уравнений при b_i - координатные вектора:
 x_ 3 =  [-0.5  3.5 -7.   4. ]

Решение системы уравнений при b_i - координатные вектора:
 x_ 4 =  [ 0.16666667 -1.          1.83333333 -1.        ]

Решение системы уравнений при помощи генерации b_i:
 x_ 1 =  [-0.10184642  0.94173346 -2.73284323  2.83540879]

Решение системы уравнений при помощи генерации b_i:
 x_ 2 =  [ 0.10312756 -0.94576098  2.48590588 -1.39675087]

Решение системы уравнений при помощи генерации b_i:
 x_ 3 =  [ 0.40866664 -3.05242001  6.70540139 -3.57730103]

Решение системы уравнений при помощи генерации b_i:
 x_ 4 =  [ 0.19714876 -1.59215074  3.76454157 -1.74629199]

Отношение d_ 1 =  5.280212637765758
Отношение d_ 2 =  16.05268988423255
Отношение d_ 3 =  7.558808834471193
Отношение d_ 4 =  1.6971426855117486

Норма обратной матрицы, вычисленная через максимум отношений нормы решения к норме вектора
свободных коэффициентов, примерно равна =  16.05268988423255
Норма обратной матрицы, вычисленная через встроенную функцию =  16.200137173630463

Число обусловленности матрицы A =  1176.9374570374473
```

Задание №3

```
m = 7
M = 3
n = 40 # Размер матрицы
t = [0.0001, 1, 10000]
it = 0 # Чтобы показать, что решение всеми методами сходятся (в пределах погрешности)

# Пункт №1
def gauss_func(A1, b1):
  A = A1.copy()
  b = b1.copy()
  n = len(b)

  for i in range(n):
    max_el = A[i,i] # Мы проверяем элементы ниже главной диагонали
                    # Ищем среди них максимальный элемент в каждом столбце (среди тех, что ниже a_ii)
    max_index = i
    for j in range(i + 1, n):
      if abs(A[j, i]) > abs(max_el):
        max_index = j
        max_el = A[j, i]  # Ищем максмальный (по модулю) элемент

    if max_index != i:  # Перестановка строк в случае, если максимальный по модулю
                            # элемент не устоит УЖЕ на диагонали. В таком случае ничего не меняем
      A[i, :], A[max_index, :] = A[max_index, :].copy(), A[i, :].copy()
      b[i], b[max_index] = b[max_index].copy(), b[i].copy()

    for j in range(i+1, n):    #Прямой ход
      factor = A[j, i] / A[i, i] # делаем так, чтобы под главной диагональю были 0
      A[j,:] -= factor * A[i, :]
      b[j] -= factor * b[i]

  x = np.zeros(n)   # Вектор решений
  for i in range(n-1, -1, -1):   # Обратный ход
    x[i] = (b[i] - np.dot(A[i, i+1:], x[i+1:])) / A[i, i]
  return x
```

```
def matrix_b(it):
  q_m = 0.993 + ((-1)**M)*M*0.0001
  A = np.empty((n,n))
  b_arr = q_m**(n+1-np.arange(n))
  for i in range(n):
    for j in range(n):
      q_j = q_m**j
      if i == j:
        A[i,j] = q_j + t[it]
      else:
        A[i,j] = q_j
  return A, b_arr

A, b_arr = matrix_b(it)
x_real = np.linalg.solve(A, b_arr)
print("Решение системы уравнений Ax=b при помощи встроенной функции (для проверки):")
print(x_real)

x = gauss_func(A, b_arr)
print("\nРешение системы уравнений Ax=b без округления:")
print(x)

# Пункт №2
def my_round(number, m):
    return int(number * 10 ** m + 0.5) / 10 ** m

def matrix_b_rounded(it):
  q_m = my_round(0.993 + my_round(((-1)**M)*M*0.0001, m), m)
  A1 = np.empty((n,n))
  b1_arr = np.zeros(n) # вектор свободных членов
  for i in range(n):
      for j in range(n):
          q_j = my_round(q_m**j, m)
          b1_arr[j] = my_round(q_m**(n+1-j), m)
          if i == j:
              A1[i,j] = my_round(q_j + my_round(t[it],m), m)
          else:
              A1[i,j] = q_j
  return A1, b1_arr

A1, b1_arr = matrix_b_rounded(it)
x_i_new = gauss_func(A1, b1_arr)
print("\nРешение системы уравнений A1*x=b1:")
print(x_i_new)

# Пункт №3
def gauss_func_rounded(A1, b1):
  A = A1.copy()
  b = b1.copy()
  n = len(b)

  for i in range(n):
    max_el = my_round(A[i,i],m) # Мы проверяем элементы ниже главной диагонали
                    # Ищем среди них максимальный элемент в каждом столбце (среди тех, что ниже a_ii)
    max_index = i
    for j in range(i + 1, n):
      if abs(A[j, i]) > abs(max_el):
        max_index = j
        max_el = my_round(A[j, i],m)  # Ищем максмальный (по модулю) элемент

    if max_index != i:  # Перестановка строк в случае, если максимальный по модулю
                            # элемент не устоит УЖЕ на диагонали. В таком случае ничего не меняем
      A[i, :], A[max_index, :] = my_round(A[max_index, :].copy(),m), my_round(A[i, :].copy(),m)
      b[i], b[max_index] = my_round(b[max_index].copy(),m), my_round(b[i].copy(),m)

    for j in range(i+1, n):    #Прямой ход
      factor = my_round(A[j, i] / A[i, i],m) # делаем так, чтобы под главной диагональю были 0
      b[j] -= my_round(factor * b[i],m)
```

```
      b[j] -= my_round(factor * b[i],m)
      for k in range(0,n):
        A[j,k] -= my_round(factor * A[i,k],m)

  x = np.zeros(n)   # Вектор решений
  for i in range(n-1, -1, -1):   # Обратный ход
    x[i] = my_round((b[i] - np.dot(A[i, i+1:], x[i+1:])) / A[i, i], m)
  return x

x_i_rounded = gauss_func_rounded(A1, b1_arr)
print("\nРешение системы уравнений A1*x=b1 с округлением:")
print(x_i_rounded)
```

```
Решение системы уравнений Ax=b при помощи встроенной функции (для проверки):
[-1.10681208e+03 -1.05235633e+03 -9.97500130e+02 -9.42240536e+02
 -8.86574580e+02 -8.30499275e+02 -7.74011609e+02 -7.17108552e+02
 -6.59787047e+02 -6.02044018e+02 -5.43876366e+02 -4.85280966e+02
 -4.26254675e+02 -3.66794324e+02 -3.06896720e+02 -2.46558648e+02
 -1.85776869e+02 -1.24548120e+02 -6.28691143e+01 -7.36540899e-01
  6.18529357e+01  1.24902675e+02  1.88416063e+02  2.52396507e+02
  3.16847444e+02  3.81772332e+02  4.47174657e+02  5.13057930e+02
  5.79425688e+02  6.46281493e+02  7.13628934e+02  7.81471627e+02
  8.49813214e+02  9.18657363e+02  9.88007770e+02  1.05786816e+03
  1.12824228e+03  1.19913390e+03  1.27054685e+03  1.34248494e+03]

Решение системы уравнений Ax=b без округления:
[-1.10681208e+03 -1.05235633e+03 -9.97500130e+02 -9.42240536e+02
 -8.86574580e+02 -8.30499275e+02 -7.74011609e+02 -7.17108552e+02
 -6.59787047e+02 -6.02044018e+02 -5.43876366e+02 -4.85280966e+02
 -4.26254675e+02 -3.66794324e+02 -3.06896720e+02 -2.46558648e+02
 -1.85776869e+02 -1.24548120e+02 -6.28691143e+01 -7.36540902e-01
  6.18529357e+01  1.24902675e+02  1.88416063e+02  2.52396507e+02
  3.16847444e+02  3.81772332e+02  4.47174657e+02  5.13057930e+02
  5.79425688e+02  6.46281493e+02  7.13628934e+02  7.81471627e+02
  8.49813214e+02  9.18657363e+02  9.88007770e+02  1.05786816e+03
  1.12824228e+03  1.19913390e+03  1.27054685e+03  1.34248494e+03]

Решение системы уравнений A1*x=b1:
[-1.10680125e+03 -1.05234525e+03 -9.97490254e+02 -9.42231254e+02
 -8.86565254e+02 -8.30490254e+02 -7.74003254e+02 -7.17101254e+02
 -6.59780254e+02 -6.02038254e+02 -5.43871254e+02 -4.85276254e+02
 -4.26250254e+02 -3.66790254e+02 -3.06893254e+02 -2.46556254e+02
 -1.85775254e+02 -1.24547254e+02 -6.28692536e+01 -7.37253632e-01
  6.18517464e+01  1.24900746e+02  1.88413746e+02  2.52392746e+02
  3.16843746e+02  3.81767746e+02  4.47168746e+02  5.13051746e+02
  5.79418746e+02  6.46273746e+02  7.13619746e+02  7.81461746e+02
  8.49802746e+02  9.18645746e+02  9.87994746e+02  1.05785475e+03
  1.12822775e+03  1.19911875e+03  1.27053075e+03  1.34246775e+03]

Решение системы уравнений A1*x=b1 с округлением:
[-1.10426195e+03 -1.05126497e+03 -9.96554536e+02 -9.41904100e+02
 -8.87516175e+02 -8.31179203e+02 -7.74380897e+02 -7.16374716e+02
 -6.58717080e+02 -5.99690212e+02 -5.41914559e+02 -4.83471656e+02
 -4.23838308e+02 -3.64958519e+02 -3.06953181e+02 -2.42438832e+02
 -1.84014830e+02 -1.20636420e+02 -5.91748675e+01  7.85250000e-02
  6.22052731e+01  1.27112284e+02  1.94677568e+02  2.58735812e+02
  3.17809003e+02  3.78847694e+02  4.41363161e+02  5.05299973e+02
  5.70590911e+02  6.37158849e+02  7.04917780e+02  7.74010003e+02
  8.44247054e+02  9.15254485e+02  9.87056362e+02  1.05932497e+03
  1.13192465e+03  1.20210202e+03  1.27227527e+03  1.34357632e+03]
```

```
# Осталось сделать обход в цикле по разным значениям t

# Функция для оценки погрешности решения СЛАУ
def estimate_error(x, x_exact):
    return np.linalg.norm(x - x_exact)

# Массивы для хранения погрешностей для каждого способа решения
errors1 = []
errors2 = []
errors3 = []

error_arr = []
# t = [0.0001, 1, 10000]

# Перебор значений параметра t
for it in range(0,3):
    A, b = matrix_b(it)
    A1, b1 = matrix_b_rounded(it)

    # Решение СЛАУ с использованием встроенной функции np.linalg.solve
    x_exact = np.linalg.solve(A, b)  # Замените A и b на вашу матрицу и вектор правой части

    # Решение СЛАУ без округления
    x1 = gauss_func(A, b_arr)
    error1 = estimate_error(x1, x_exact)
    errors1.append(error1)

    # Решение СЛАУ с округлением
    x2 = gauss_func(A1, b1_arr)
    error2 = estimate_error(x2, x_exact)
    errors2.append(error2)

    # Решение СЛАУ с округлением A, b и внуьтри метода Гаусса
    x3 = gauss_func_rounded(A1, b1_arr)
    error3 = estimate_error(x3, x_exact)
    errors3.append(error3)

    error_arr.append(error1)
    error_arr.append(error2)
    error_arr.append(error3)

print('Погрешности при подсчете обычным методом Гаусса                              ', errors1, '\n',
      'Погрешности при подсчете обычным методом Гаусса с округленными A1 и b1       ', errors2,'\n',
      'Погрешности при подсчете "округленным" методом Гаусса с округленными A1 и b1 ', errors3)

plt.plot(t, errors3, label='С округлением A, b и фнутри функции Гаусса', marker='o')
plt.plot(t, errors1, label='Без округления', marker='o')
plt.plot(t, errors2, label='С округлением A и b', marker='o')
# plt.plot(t, errors4, label='С округлением (способ 3)', marker='o')
```