In [36]:

```
import numpy as np
```

In [43]:

```
# вычисление 3 нормы матрицы
def norm_3(A):
  return np.sqrt(max(abs(np.linalg.eigvals(np.dot(A, A.transpose())))))
```

0) Инициализация матрицы

In [38]:

```
n = 20
# init A matrix
InputMatrix = np.zeros((n, n))
for i in range(n):
    InputMatrix[i][i] = 10
for i in range(n - 1):
    InputMatrix[i+1][i] = InputMatrix[i][i+1] = 1
for i in range(n - 2):
    InputMatrix[i+2][i] = InputMatrix[i][i+2] = 0.1
# init f vector
inputF = np.arange(1., n + 1)
```

1) прямой метод Гаусса

1.1) Реализация

In [39]:

```
def gauss(ACopy, fCopy):
    A = ACopy.copy()
    f = fCopy.copy()
    n = A.shape[0]

    # straight pass
    for i in range(n):
        maxValue = 0
        maxIdx   = i # current idx
        for m in range(i, n):
            if abs(A[m][i]) > abs(maxValue):
                maxIdx = m; maxValue = A[m][i]

        if maxValue == 0: continue
        A[i], A[maxIdx] = A[maxIdx], A[i]
        f[i], f[maxIdx] = f[maxIdx], f[i]

        for j in range(i + 1, n):
            coef = A[j][i] / A[i][i]
            A[j] = A[j] - A[i] * coef
            f[j] = f[j] - f[i] * coef

    # revert pass
    for i in range(n):
        for j in range(i + 1, n):
            coef = A[n - 1 - j][n - i - 1] / A[n - 1 - i][n - 1 - i]
            A[n - 1 - j] =  A[n - 1 - j] - A[n - 1 - i] * coef
            f[n - 1 - j] =  f[n - 1 - j] - f[n - 1 - i] * coef

    for i in range(n):
        f[i] = f[i] / A[i][i]
```

```
    return f
```

In [41]:

```
solution = gauss(InputMatrix, inputF)
solution
```

Out[41]:

```
array([0.08113926, 0.16401722, 0.24590172, 0.32786801, 0.40983615,
       0.49180328, 0.57377049, 0.65573771, 0.73770492, 0.81967213,
       0.90163934, 0.98360656, 1.06557377, 1.14754118, 1.2295064 ,
       1.31147392, 1.39363506, 1.47365191, 1.5557294 , 1.82969054])
```

1.2) Невязка

In [50]:

```
v = InputMatrix.dot(solution) - inputF
print (np.sqrt(v.dot(v)))
```

```
4.218847493575595e-15
```

2) итерационный метод Гаусса-Зейделя

Условия Завершения: $||Ax_k - b|| < \epsilon$

2.1) Реализация

In [31]:

```
def seidel(A, b, eps):
    n = A.shape[0]
    x = np.zeros(n) # start point

    while True:
        x_next = np.copy(x)
        for i in range(n):
            s1 = s2 = 0
            for j in range(i):
                s1 += A[i][j] * x_next[j]
            for j in range(i + 1, n):
                s2 += A[i][j] * x[j]
            x_next[i] = (b[i] - s1 - s2) / A[i][i]

        if np.sqrt(sum((x_next[i] - x[i]) ** 2 for i in range(n))) <= eps:
            break

        x = x_next

    return x
```

In [51]:

```
solution = seidel(InputMatrix, inputF, 0.01)
solution
```

Out[51]:

```
array([0.08153652, 0.16446335, 0.24642773, 0.32847407, 0.41052195,
       0.49256885, 0.57461584, 0.65666282, 0.73870981, 0.82075679,
       0.90280378, 0.98485076, 1.06689775, 1.14894473, 1.23099172,
       1.3130387 , 1.39489552, 1.47316827, 1.55578489, 1.82968983])
```

2.2) Невязка

In [52]:

In [52]:

```
v = InputMatrix.dot(solution) - inputF
print (np.sqrt(v.dot(v)))
```

```
0.05215206867059179
```

3) **max, min** собственные значения. Число обусловленности.

3.1) Число обусловленности.

In [55]:

```
InputMatrixInv = np.linalg.inv(InputMatrix)
u = norm_3(InputMatrix) * norm_3(InputMatrixInv)
u
```

Out[55]:

```
1.481553466967567
```

3.2) **max** собственное значение

Вычисление проведены с помощью степенного метода

In [62]:

```
def Poly(A, iters):
    n = A.shape[0]
    x = np.ones(n) # start point

    u = 0
    for i in range(iters):
        x_next = A.dot(x)
        u      = x.dot(x_next) / x.dot(x)

    return u
```

In [64]:

```
l_max = Poly(InputMatrix, 1000)
l_max
```

Out[64]:

```
12.080000000000002
```

3.3) **min** собственное значение

С учётом того, что матрица симметричная:

In [66]:

```
l_min = l_max / u
l_min
```

Out[66]:

```
8.153603814734584
```

In [ ]:

```

```