In [6]:
```
import numpy as np
from matplotlib import pyplot as plt
from numpy import linalg as LA
```

1.1 Матрица плотности подпространства квантовой системы

In [25]:
```
# get_rho - функция для задачи матрицы плотности рандомной квантов
ой системы размерности (d,k) и
# вычисленя матрицы плотности подсистемы заданной квантовой систем
ы размерноси d

# get_ly_pure - вычисляет собственные значения матрицы плотности,
а так же чистоту.

def get_rho(d,k):
    C_list = np.random.normal(0, 1, size=(d, k)) + 1j*np.random.no
rmal(0, 1, size=(d, k))
    C_list = C_list/(np.sqrt(np.sum(C_list*np.conjugate(C_list))))
    C_list = np.reshape(C_list, [1,d,k,1])
    g_rho = np.tensordot(C_list, np.conjugate(C_list), [0,3])
    g_rho = np.reshape(g_rho, (d,k,d,k))
    rho = np.trace(g_rho, axis1=1, axis2=3)
    return rho

def get_ly_pure(rho,d,k):
    ly = np.linalg.eigvals(rho)
    purity = np.trace(np.dot(rho, rho))
    return ly, purity
```

Зададим 1000 случайных квантовых систем и расчиитаем статистику получанных частот и собственных значений

In [26]:
```
d = 5
k = 4
ly_list = []

for i in range(1000):
    rho = get_rho(d,k)
    ly, purity = get_ly_pure(rho,d,k)
    ly_list.extend(ly)
```

In [27]:
```
# Построим гистограмму распределения С3
fig, ax = plt.subplots()
ax.set_title('Распределение С3')
ax.hist(ly_list, bins = 200)
plt.plot()
```

```
/home/stas/anaconda3/lib/python3.7/site-packages/numpy/lib/histogr
ams.py:854: ComplexWarning: Casting complex values to real discard
s the imaginary part
  indices = f_indices.astype(np.intp)
/home/stas/anaconda3/lib/python3.7/site-packages/matplotlib/transf
orms.py:796: ComplexWarning: Casting complex values to real discar
ds the imaginary part
  points = np.array(args, dtype=float).reshape(2, 2)
/home/stas/anaconda3/lib/python3.7/site-packages/matplotlib/transf
orms.py:1959: ComplexWarning: Casting complex values to real disca
rds the imaginary part
  x, y = float(x), float(y)
```

Out[27]:
```
[]
```

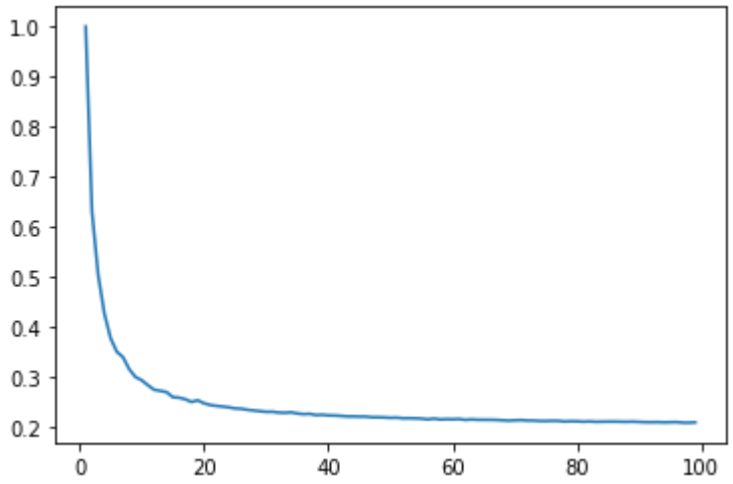
Проведём расчёты и построим зависимость Чистоты полученного состояния от размерности вспомогательной подсистемы:

In [28]:
```
d = 5
N = 100
ly_list = []
pure_list = []
k_list = []
purity_avg = 0

p_l = []
for k in range(1,100):
    for i in range(N):
        rho = get_rho(d,k)
        ly, purity = get_ly_pure(rho,d,k)
        ly_list.extend(ly)
        p_l.append(purity)
    purity_avg = np.mean(p_l)
    p_l = []
    k_list.append(k)
    pure_list.append(purity_avg)
```

In [29]:
```
plt.plot(k_list, pure_list)
```

```
/home/stas/anaconda3/lib/python3.7/site-packages/numpy/core/_asarr
ay.py:85: ComplexWarning: Casting complex values to real discards
the imaginary part
  return array(a, dtype, copy=False, order=order)
```

Out[29]:
```
[<matplotlib.lines.Line2D at 0x7fa7b1db5310>]
```



1.2 Частичный след и разложение Шмидта:

In [86]:
```
class Qstate():
    def __init__(self, N = 2):
        self.A_dims = []
        self.B_dims = []
        self.vec = np.zeros(tuple([2]*N))
        self.N = N
        self.AB_matrix = None


    def build_random_state(self): # строит случайное состояние зад
анной размерности
        vec = np.random.randn(*([2]*self.N)) + 1j*np.random.randn
(*([2]*self.N))
        vec = vec/np.sqrt(np.sum(vec*vec.conj()))
        self.vec = vec

    def build_def_state(self):  # строит состояние соответствующее
заданному в варианте 5
        x = np.zeros((2,2,2,2))
        x[[0],[0],[1],[1]] += 3/8
        x[[0],[1],[1],[0]] += np.sqrt(6)/8
        x[[1],[1],[0],[0]] -= 7/8
        self.vec = x

    def split_system(self, a_dims, b_dims): # записывает состояние
в виде матрицы,
        dim_list = a_dims.copy()            # разделяющей систему
на два подпространства
        dim_list.extend(b_dims)
        self.AB_matrix = np.reshape(self.vec.transpose(dim_list),
(2**len(a_dims),2**len(b_dims)), order = 'F')

    def HS_decomposition(self, rho_matrics = False): # Разложение
Шмидта по заданным подпространствам
        u, s, v = LA.svd(self.AB_matrix.T)

        if rho_matrics == True:
            sh0 = self.AB_matrix.shape[1]
            sh1 = self.AB_matrix.shape[0]

            sa = np.zeros(sh0)
            sb = np.zeros(sh1)

            sh_min = min(sh0,sh1)

            sa[:sh_min] = s[:sh_min]
            sb[:sh_min] = s[:sh_min]


            rho_A_svd = np.dot(u, np.dot(np.diag(sa) ** 2, np.conj
ugate(u).T))
            rho_B_svd = np.dot(np.conjugate(v).T, np.dot(np.diag(s
b) ** 2, v)).T

            return rho_A_svd, rho_B_svd, u,s,v
        return u, s, v

    def Trace_decomposition(self): # Частичный след по заданным по
дпространствам
```

```
        sh0 = self.AB_matrix.shape[0]
        sh1 = self.AB_matrix.shape[1]
        m = np.reshape(self.AB_matrix, [1, sh0, sh1, 1])
        dense_m = np.reshape(np.tensordot(m,m, [0,3]), [sh0,sh1,sh
0,sh1])
        A_matrix = np.trace(dense_m, axis1 = 0, axis2 = 2)
        B_matrix = np.trace(dense_m, axis1 = 1, axis2 = 3)

        return A_matrix, B_matrix

    def get_vec(self): # возвращаем вектор
        return self.vec

    def get_AB_matrix(self): # возвращаем матрицу состояния раздел
яющую на подсистемы AB
        return self.AB_matrix
```

In [87]:
```
# зададим рандомное чистое квантовое состояние

state = Qstate(N = 4)
state.build_random_state()
print(state.get_vec())

# зададим квантовое состояние из условия (Вариант 5)

state.build_def_state()
print(state.get_vec())
```

```
[[[[ 0.05828316-0.01785938j  0.01656728-0.16242182j]
   [-0.04804861+0.04125682j -0.24128226-0.5441396j ]]

  [[-0.24894998-0.25241672j  0.10987102+0.05544226j]
   [ 0.00437783-0.08990344j  0.10132072-0.08071269j]]]


 [[[-0.22044555-0.16901584j  0.03393008-0.03330614j]
   [-0.04798895-0.24924008j -0.10167516+0.07960747j]]

  [[-0.02281229-0.03036828j  0.30104792-0.39634371j]
   [ 0.04656884-0.11427617j  0.01653183+0.14286796j]]]]
[[[[ 0.          0.        ]
   [ 0.          0.375     ]]

  [[ 0.          0.        ]
   [ 0.30618622  0.        ]]]


 [[[ 0.          0.        ]
   [ 0.          0.        ]]

  [[-0.875       0.        ]
   [ 0.          0.        ]]]]
```

In [88]:
```
# Запишем состояние в виде матрицы разделяющей состояние на два по
дпространства по заданным наборам осей

state.split_system(a_dims = [0,1,2],b_dims = [3])
state.get_AB_matrix()
```

Out[88]:
```
array([[ 0.        ,  0.        ],
       [ 0.        ,  0.        ],
       [ 0.        ,  0.        ],
       [-0.875     ,  0.        ],
       [ 0.        ,  0.375     ],
       [ 0.        ,  0.        ],
       [ 0.30618622,  0.        ],
       [ 0.        ,  0.        ]])
```

In [89]:
```
# Проведём svd разложение и получим из него матрицы плотности подс
мстем

rho_A_svd, rho_B_svd, u, s, v = state.HS_decomposition(rho_matrics
= True)
print('SVD разложение:')
print(u, s, v)
print()

print("_____________________________________________________________
__________")
print('матрицы плотности подсистем через svd:')
print(rho_A_svd, rho_B_svd)
print()
```

```
SVD разложение:
[[1. 0.]
 [0. 1.]] [0.92702481 0.375     ] [[ 0.          0.          0.
-0.94387981  0.          0.
   0.33028913  0.        ]
 [ 0.          0.          0.          0.          1.          0.
   0.          0.        ]
 [ 0.          0.          1.          0.          0.          0.
   0.          0.        ]
 [ 0.94387981  0.          0.          0.10909091  0.          0.
   0.31175324  0.        ]
 [ 0.         -1.          0.          0.          0.          0.
   0.          0.        ]
 [ 0.          0.          0.          0.          0.          1.
   0.          0.        ]
 [-0.33028913  0.          0.          0.31175324  0.          0.
   0.89090909  0.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.          1.        ]]

_____________________________________________________________________
___
матрицы плотности подсистем через svd:
[[0.859375 0.      ]
 [0.       0.140625]] [[ 0.          0.          0.          0.
0.          0.
   0.          0.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.          0.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.          0.        ]
 [ 0.          0.          0.          0.765625    0.          0.
  -0.26791294  0.        ]
 [ 0.          0.          0.          0.          0.140625    0.
   0.          0.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.          0.        ]
 [ 0.          0.          0.         -0.26791294  0.          0.
   0.09375     0.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.          0.        ]]
```

In [90]:
```
# Проверим правильность разложения Шмидта:

l = max(len(u),len(v))

psi = state.get_AB_matrix().T

s_full = np.zeros(l)
u_full = np.zeros((l,l))
v_full = np.zeros((l,l))
init_state = np.zeros((l,l))

s_full[:len(s)] = s
u_full[:len(u), :len(u)] = u
v_full[:len(v),:len(v)] = v
init_state[:len(u),:len(v)] = state.get_AB_matrix().T

print('Различие между изначальным состоянием и восстановленным:')
init_state - np.dot(u_full, np.dot(np.diag(s_full), v_full))
```

```
Различие между изначальным состоянием и восстановленным:
```

Out[90]:
```
array([[0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.]])
```

In [91]:
```
# Создадим списки состояний в подсиситмах A и B в базисах, заданны
х разложением Шмидта:

list_vec_A = [u.T[0], u.T[1]]
list_vec_B = [v[0], v[1], v[2], v[3], v[4], v[5], v[6], v[7]]
```

In [92]:
```
# Восстановим изначальный вектор из разложения на собственные вект
оры:

psi_new = np.zeros((1, 2, 8, 1))
for k in range(len(s)):
    psi_a_k = np.reshape(list_vec_A[k], (1, 2, 1))
    psi_b_k = np.reshape(list_vec_B[k], (1, 8, 1))
    psi_new += s[k] * np.tensordot(psi_a_k, psi_b_k, axes=([2],
[0]))
psi_new = np.reshape(psi_new, (2, 8))
print('Различие между изначальным состоянием и восстановленным по
СВ и СЗ разложения Шмидта:')
print()
print(psi_new - psi)
```

```
Различие между изначальным состоянием и восстановленным по СВ и СЗ
разложения Шмидта:

[[0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]]
```

In [93]:
```
# Получим матрицы плотности систем через взятие частичного следа

rho_A_tr, rho_B_tr = state.Trace_decomposition()
rho_A_tr, rho_B_tr
```

Out[93]:
```
(array([[0.859375, 0.      ],
        [0.      , 0.140625]]),
 array([[ 0.        ,  0.        ,  0.        ,  0.        ,  0.
,
          0.        ,  0.        ,  0.        ],
        [ 0.        ,  0.        ,  0.        ,  0.        ,  0.
,
          0.        ,  0.        ,  0.        ],
        [ 0.        ,  0.        ,  0.        ,  0.        ,  0.
,
          0.        ,  0.        ,  0.        ],
        [ 0.        ,  0.        ,  0.        ,  0.765625  ,  0.
,
          0.        , -0.26791294,  0.        ],
        [ 0.        ,  0.        ,  0.        ,  0.        ,  0.14
0625  ,
          0.        ,  0.        ,  0.        ],
        [ 0.        ,  0.        ,  0.        ,  0.        ,  0.
,
          0.        ,  0.        ,  0.        ],
        [ 0.        ,  0.        ,  0.        , -0.26791294,  0.
,
          0.        ,  0.09375   ,  0.        ],
        [ 0.        ,  0.        ,  0.        ,  0.        ,  0.
,
          0.        ,  0.        ,  0.        ]]))
```

In [94]:
```
# Сравним матрицы плотности, полученные двумя способами:

loss = (np.sum(np.sqrt((rho_A_svd - rho_A_tr)**2))+ np.sum(np.sqrt
((rho_B_svd - rho_B_tr)**2)))
print(loss)
```

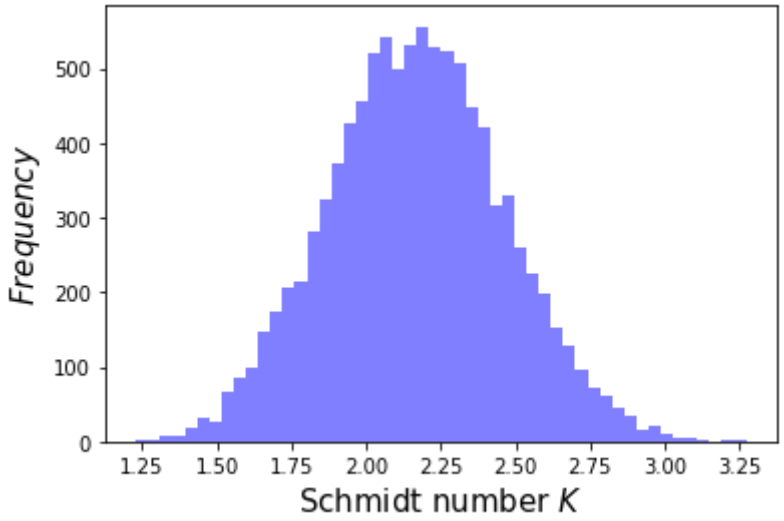
```
1.249000902703301e-16
```

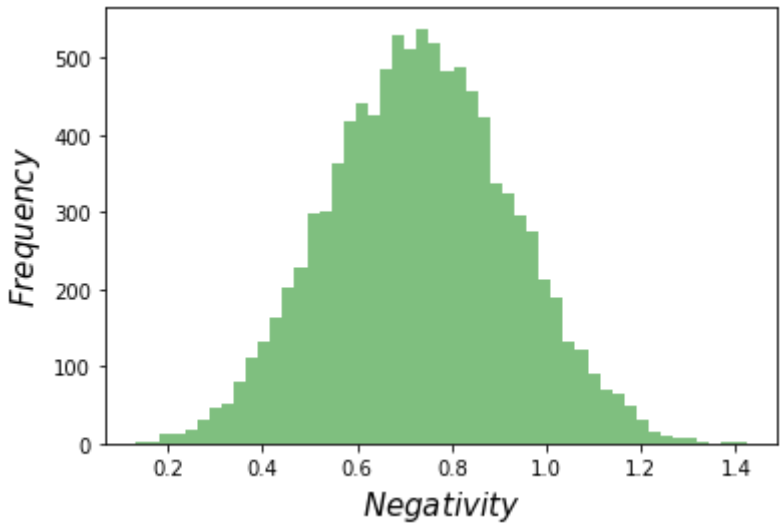
In [95]:
```
# Вычислим число Шмидта для заданного состояния

K = 1 / np.sum(s ** 4)
print('K = ', K)
print('Norma = ', np.sum(s ** 2))
```

```
K =  1.3187379265936896
Norma =  1.0
```

In [99]:
```
# Зададим случайное состояние
state = Qstate(N = 4)
state.build_random_state()
state.split_system(a_dims = [0,1], b_dims = [2,3])

# Возьмём частичный след по подпространствам A и B

A_matrix, B_matrix = state.Trace_decomposition()

# Найдём собственные значения матриц плотности посистем A и B
eig_AB = np.linalg.eig(state.get_AB_matrix())[0]
eig_A = np.linalg.eig(A_matrix)[0]
eig_B = np.linalg.eig(B_matrix)[0]

# Посчитаем negativity

negativity_AB = 0.5 * np.sum(np.abs(eig_AB) - eig_AB)
negativity_A = 0.5 * np.sum(np.abs(eig_A) - eig_A)
negativity_B = 0.5 * np.sum(np.abs(eig_B) - eig_B)

print('Negativity системы A:')
print(negativity_A)
print('Negativity системы B:')
print(negativity_B)
print('Negativity системы AB:')
print(negativity_AB)
print()
print('Заметно, что Negativity подсистем A и B сильно меньше в сра
внении с Negativity полной системы')

K_list = []
Negativity_list = []

for i in range(10000):
    state = Qstate(N = 4)
    state.build_random_state()
    state.split_system(a_dims = [0,1], b_dims = [2,3])

    u, s, vh = state.HS_decomposition()
    K_list.append(1 / np.sum(s ** 4))

#     A_matrix, B_matrix = state.Trace_decomposition()

    eig_AB = np.linalg.eig(state.get_AB_matrix())[0]
    negativity_AB = 0.5 * np.sum(np.abs(eig_AB) - eig_AB)
    Negativity_list.append(negativity_AB)

K_list = np.array(list(map(lambda x: x.real, K_list)))
Negativity_list = np.array(list(map(lambda x: x.real, Negativity_l
ist)))

plt.hist(K_list, bins=50, color='blue', alpha=0.5)
plt.xlabel(r'Schmidt number $K$', fontsize=15)
plt.ylabel(r'$Frequency$', fontsize=15)
plt.show()

plt.hist(Negativity_list, bins=50, color='green', alpha=0.5)
plt.xlabel(r'$Negativity$', fontsize=15)
plt.ylabel(r'$Frequency$', fontsize=15)
```

```
plt.show()
Negativity системы A:
(0.29867629058155953-0.09330889855034806j)
Negativity системы B:
(0.29867629058155926-0.09330889855034818j)
Negativity системы AB:
(0.8521400011195885+0.1600724814344526j)

Заметно, что Negativity подсистем A и B сильно меньше в сравнении
с Negativity полной системы
```





In [ ]:

In [ ]: