Susceptibility

$$Z = \sum_n e^{-\beta E_n}$$

$$m(B) = \frac{1}{\beta Z}\frac{\partial Z}{\partial B}$$

$$\chi = \lim_{B\to 0}\frac{\partial m}{\partial B}$$

Eigenvalues and eigenstates in presence of $\vec{B}$

The easy ones: $n = 0, 1, 3, 4$

$$\Delta_\pm \equiv \sqrt{v^2 + (\epsilon_d \pm \frac{1}{2}B)^2}$$

$$|0,0\rangle \text{ and } |2,2\rangle \left\{ E_0 = E_{15} = \tfrac{1}{4}k \right.$$

$$\text{between } |\uparrow,0\rangle, |0,\uparrow\rangle \text{ and between } |\uparrow,2\rangle, |2,\uparrow\rangle \begin{cases} E_1 = E_{11} = \frac{1}{2}\left(\epsilon_d + \frac{1}{2}B + \Delta_+\right) \\ E_2 = E_{12} = \frac{1}{2}\left(\epsilon_d + \frac{1}{2}B - \Delta_+\right) \end{cases}$$

$$\text{between } |\downarrow,0\rangle, |0,\downarrow\rangle \text{ and between } |\downarrow,2\rangle, |2,\downarrow\rangle \begin{cases} E_3 = E_{13} = \frac{1}{2}\left(\epsilon_d - \frac{1}{2}B + \Delta_-\right) \\ E_4 = E_{14} = \frac{1}{2}\left(\epsilon_d - \frac{1}{2}B - \Delta_-\right) \end{cases}$$

The easy ones in $n = 2$

$$|\uparrow,\uparrow\rangle : E_5 = \epsilon_d + \frac{1}{4}j + \frac{1}{2}B$$

$$|\downarrow,\downarrow\rangle : E_6 = \epsilon_d + \frac{1}{4}j - \frac{1}{2}B$$

$$|\text{charge triplet } 0\rangle : E_7 = \frac{1}{4}k$$

The remaining subspace

$$\begin{pmatrix} \epsilon_d + \frac{1}{4}j & B & 0 \\ B & \epsilon_d - \frac{3}{4}j & -2v \\ 0 & -2v & -\frac{3}{4}k \end{pmatrix}$$

The basis is: $|\text{spin trip. } 0\rangle, |\text{spin singl.}\rangle, |\text{charge singl.}\rangle$

The next step is to diagonalize this matrix.

In [5]:

```
import itertools
from tqdm import tqdm
```

```python
from time import sleep
from multiprocessing import Pool
import numpy as np
from math import *
import matplotlib
from matplotlib import pyplot as plt

font = {'size'   : 17}

matplotlib.rc('font', **font)
#matplotlib.rcParams['text.usetex'] = True
plt.rcParams["figure.figsize"]= 7, 5
#plt.rcParams['figure.dpi'] = 90
matplotlib.rcParams['lines.linewidth'] = 2
plt.rcParams['axes.grid'] = True
from sympy import *
init_printing(use_unicode=True)
ed,j,k,v,B,E,beta,x = symbols('ed J k v B E beta x')
```

In [6]:

```python
M = Matrix([[ed + j/4, B, 0], [B, ed - 3*j/4, -2*v], [0, -2*v, -3*k/4]])
charp = M.charpoly(x)
sols = solve(charp, x)
```

In [3]:

```python
delta_p = sqrt(v**2 + (ed + B/2)**2)
delta_m = sqrt(v**2 + (ed - B/2)**2)
Es = [[]]*16
Es[0] = Es[15] = k/8 + k/8
Es[1] = Es[11] = (ed + B/2 + delta_p)/2
Es[2] = Es[12] = (ed + B/2 - delta_p)/2
Es[3] = Es[13] = (ed - B/2 + delta_m)/2
Es[4] = Es[14] = (ed - B/2 - delta_m)/2
Es[5] = ed + j/4 + B/2
Es[6] = ed + j/4 - B/2
Es[7] = k/4
Es[8] = sols[0]
Es[9] = sols[1]
Es[10] = sols[2]
```

In [ ]:

```python
Z = 0
for E in Es:
    Z += exp(-beta*E)
m = (1/beta)*diff(Z, B)/Z
chi_B = diff(m, B)
chi_B.subs({ed:0, v: 0, k: 0})
```

In [ ]:

```python
chi = limit(chi_B, B, 0)
```

Alternate way

Okay, so keeping the v and diagonalizing does not seem to be tractable. So we set $v = 0$ and then diagonalize. The resultant susceptibility is

$$\chi(\epsilon_d, k, j, \beta) = \frac{\left[4e^{-\beta\epsilon_d} + 2e^{-\beta\left(\epsilon_d + \frac{j}{2}\right)}\right]\frac{1}{4}\beta + e^{-\beta\left(\epsilon_d - \frac{j}{4}\right)}\sinh\left(\beta\frac{j}{2}\right)\frac{1}{j}}{4 + 3\exp\left\{-\beta\frac{k}{4}\right\} + \exp\left\{\beta\frac{3k}{4}\right\} + 4e^{-\beta\epsilon_d} + 2e^{-\beta\left(\epsilon_d + \frac{j}{2}\right)} + 2e^{-\beta\left(\epsilon_d - \frac{j}{4}\right)}\cosh}$$

This has the behaviour

$$\frac{1}{\beta}\chi \to \frac{1}{8} \text{ when } T \to \infty$$

and

$$\frac{1}{\beta}\chi \to \frac{1}{2j} \text{ when } T \to 0$$

We can plot the susceptibility for the simpler case of $\epsilon_d = k = v = 0$. It looks like

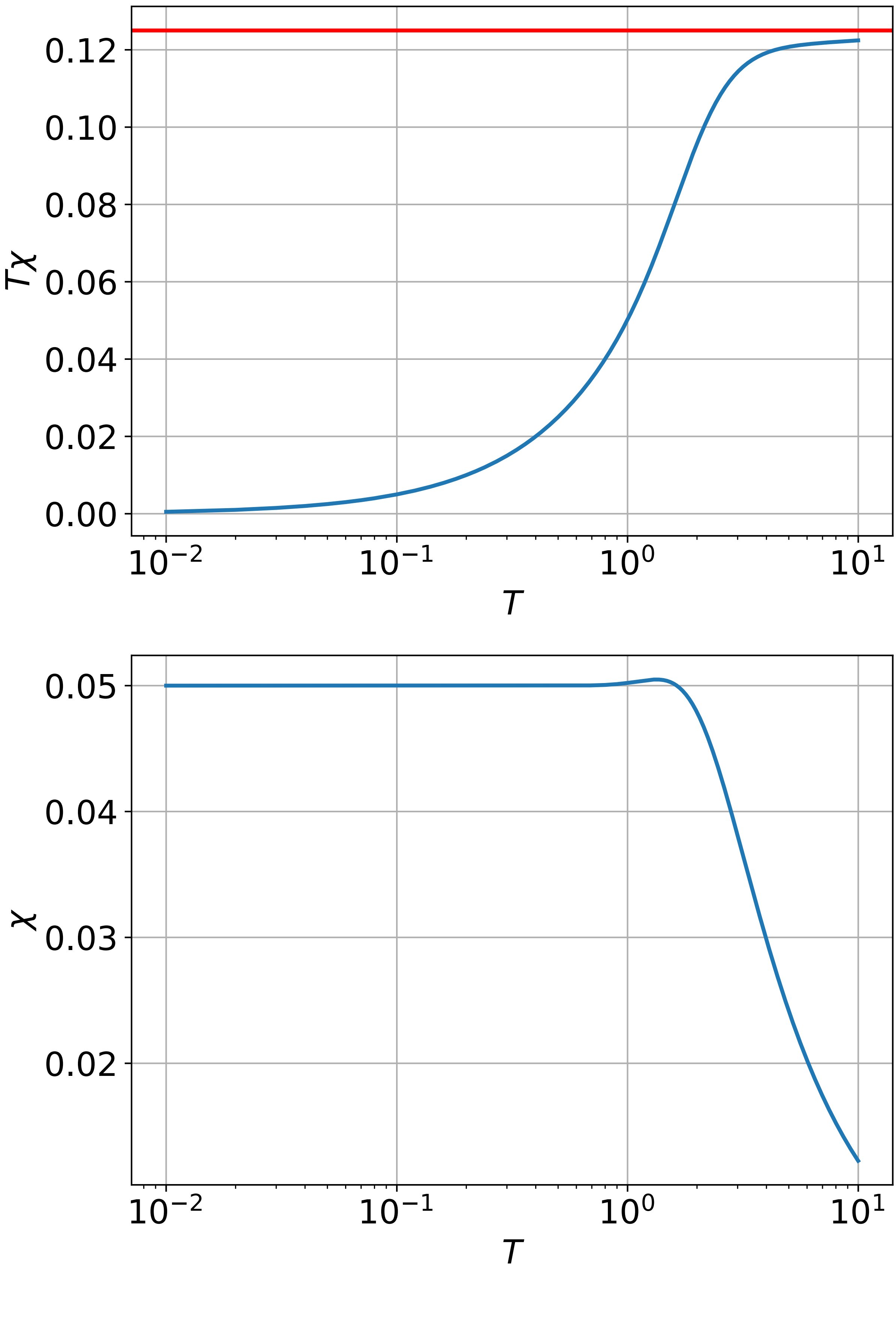
$$\chi(j,\beta) = \frac{\left[4 + 2e^{-\beta\frac{j}{2}}\right]\frac{1}{4}\beta + e^{\beta\frac{j}{4}}\sinh\left(\beta\frac{j}{2}\right)\frac{1}{j}}{12 + 2e^{-\beta\frac{j}{2}} + 2e^{\beta\frac{j}{4}}\cosh\left(\beta\frac{j}{2}\right)}$$

In [7]:

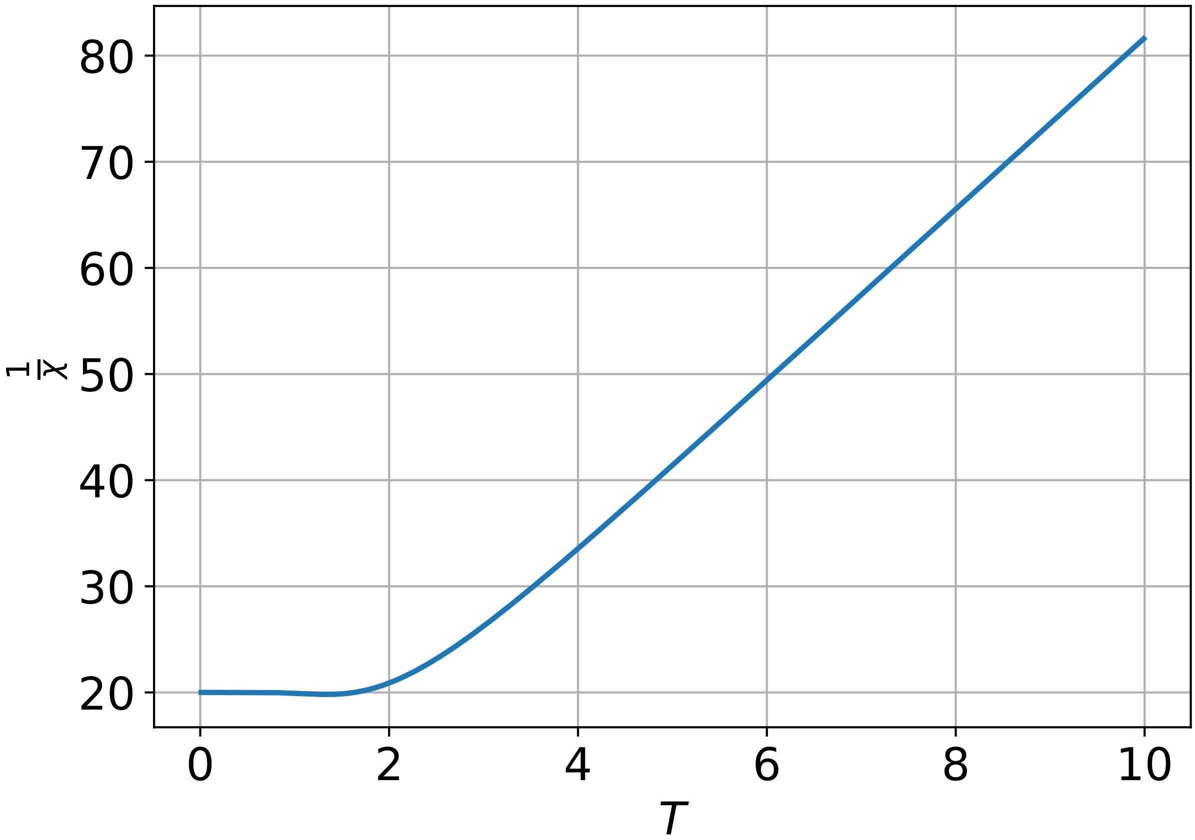
```
def get_chi(ed, j, k, T):
    beta = 1/T
    numerator = (4 * exp(-beta * ed) + 2 * exp(-beta * (ed + j/2))) * beta/4
    denominator = 4 + 3 * exp(-beta * k/4) + exp(beta * 3*k/4) + 4 * exp(-bet
    return numerator/denominator


j = 10
T_range = np.arange(0.01, 10, 0.01)
ed = 0
k = 0
data = itertools.product([ed],[j],[k], T_range)
chi = np.array(Pool(processes=40).starmap(get_chi,data))

plt.axhline(1/8, 0, T_range[-1], color="r")
plt.plot(T_range, T_range * chi)
plt.ylabel(r"$T \chi$")
plt.xlabel(r"$T$")
plt.xscale("log")
plt.show()
smaller_range = np.where(T_range <= 1)
plt.plot(T_range, chi)
plt.ylabel(r"$\chi$")
plt.xlabel(r"$T$")
plt.xscale("log")
plt.show()
plt.plot(T_range, 1/chi)
plt.ylabel(r"$\frac{1}{\chi}$")
plt.xlabel(r"$T$")
plt.show()
```

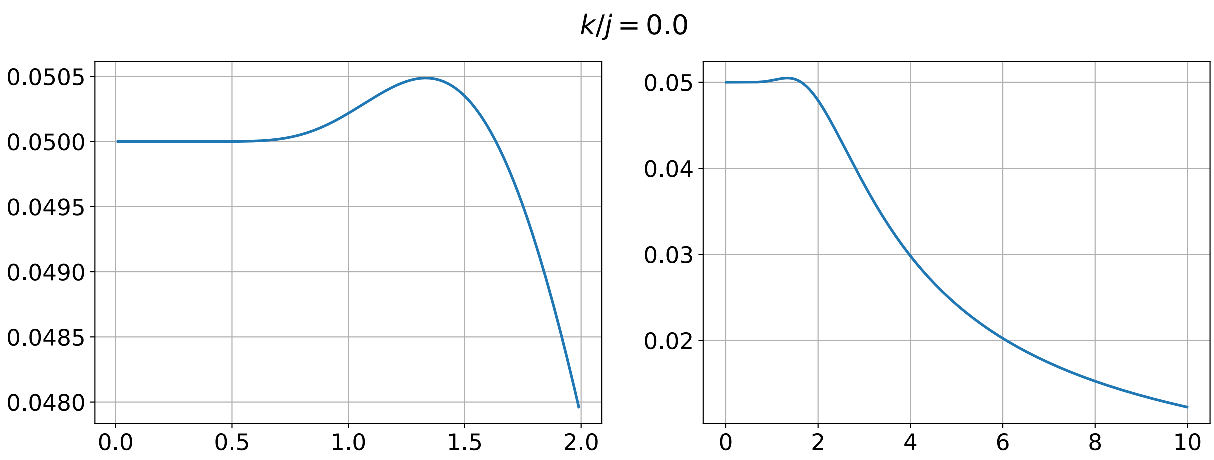


0.12
0.10
0.08
0.06
0.04
0.02
0.00
$T\chi$
10^{-2}
10^{-1}
10^{0}
10^{1}
T
0.05
0.04
0.03
0.02
χ
10^{-2}
10^{-1}
10^{0}
10^{1}
T

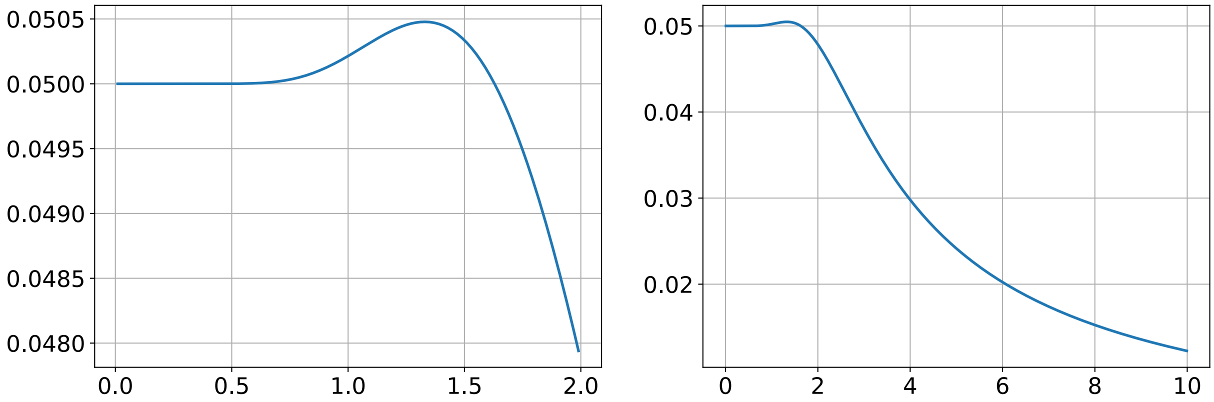


Effect of k

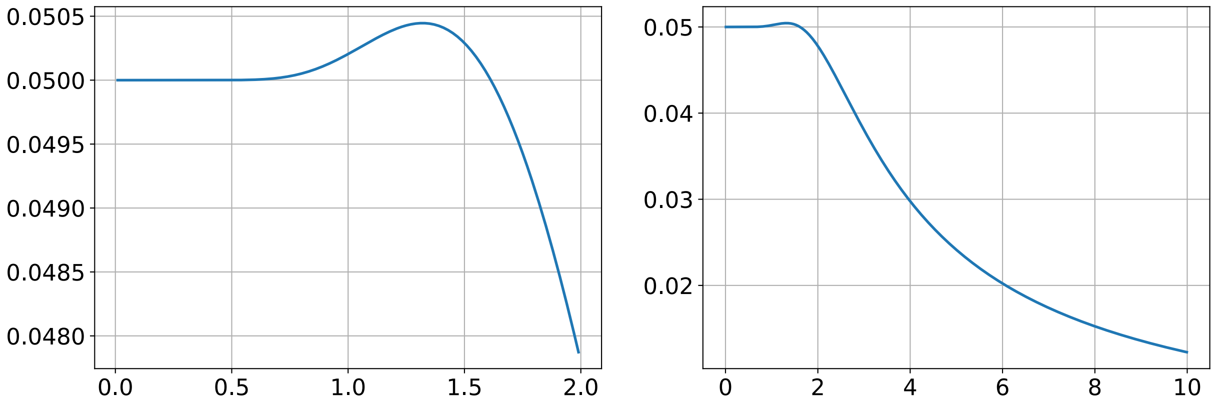
In [10]:
```
j = 10
T_range = np.arange(0.01, 10, 0.01)
ed = 0
for k in j*np.arange(0,1.1,0.05):
    data = itertools.product([ed],[j],[k], T_range)
    chi = np.array(Pool(processes=40).starmap(get_chi,data))
    small_r = np.where(T_range < 2)
    fig, ax = plt.subplots(1,2)
    fig.set_size_inches(15,5)
    ax[0].plot(T_range[small_r], chi[small_r])
    ax[1].plot(T_range, chi)
    plt.suptitle(r"$k/j={:.2}$".format(k/j))
    plt.show()
```



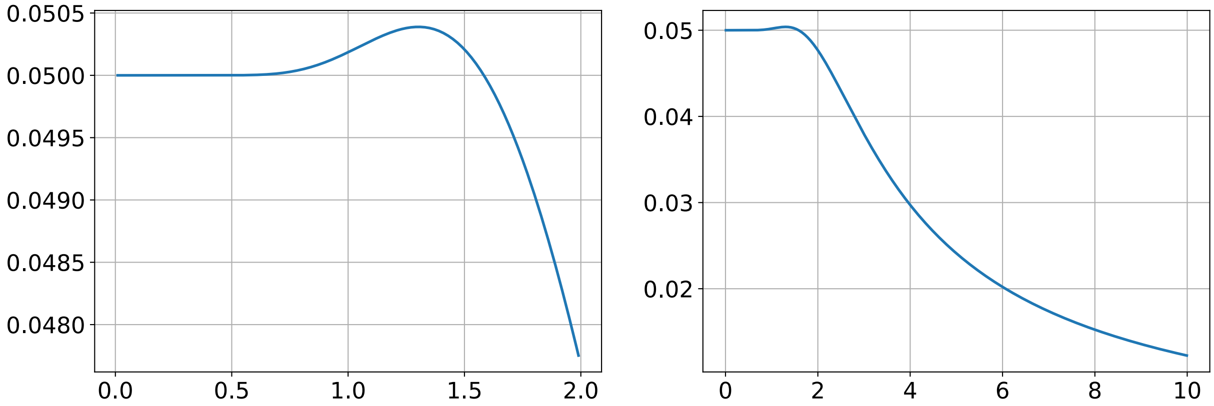
$k/j = 0.05$



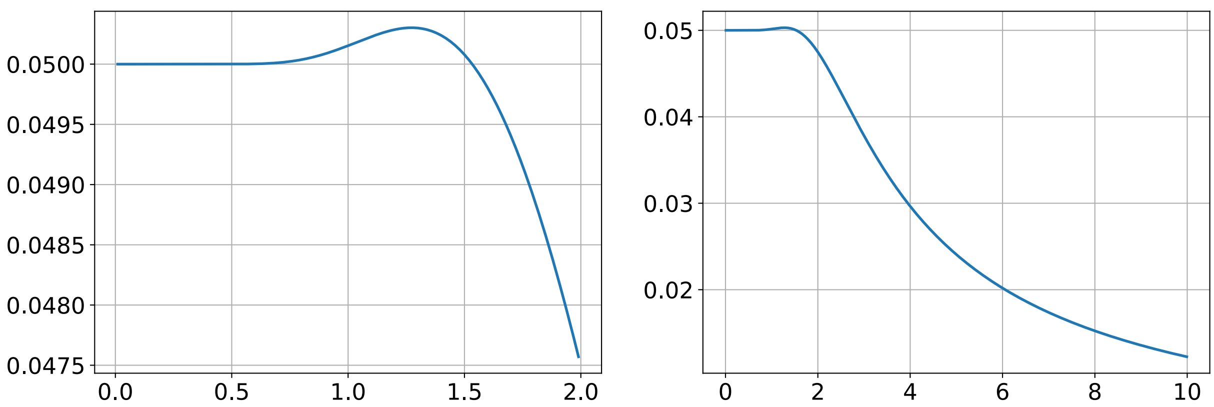
$k/j = 0.1$

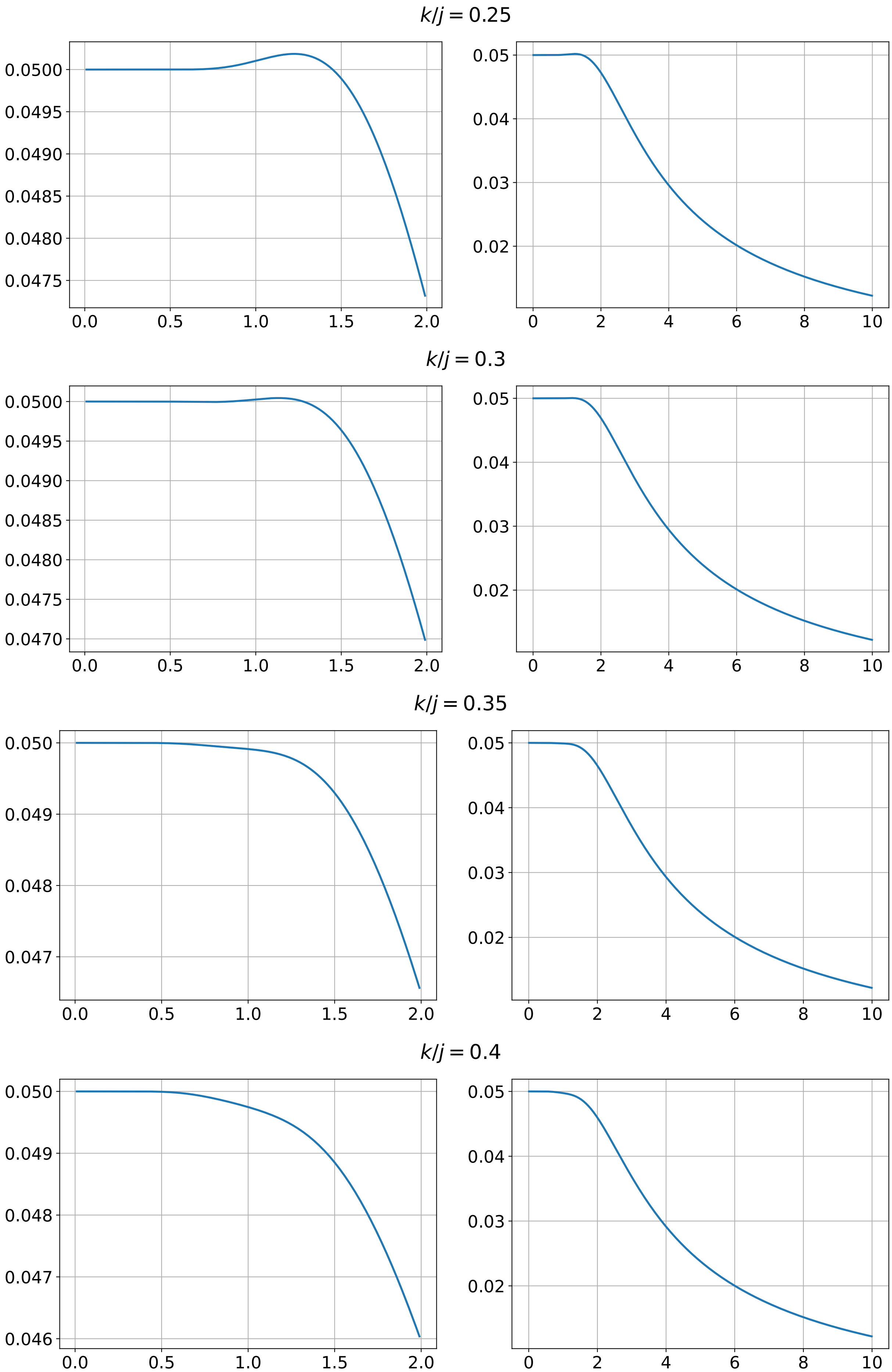


$k/j = 0.15$

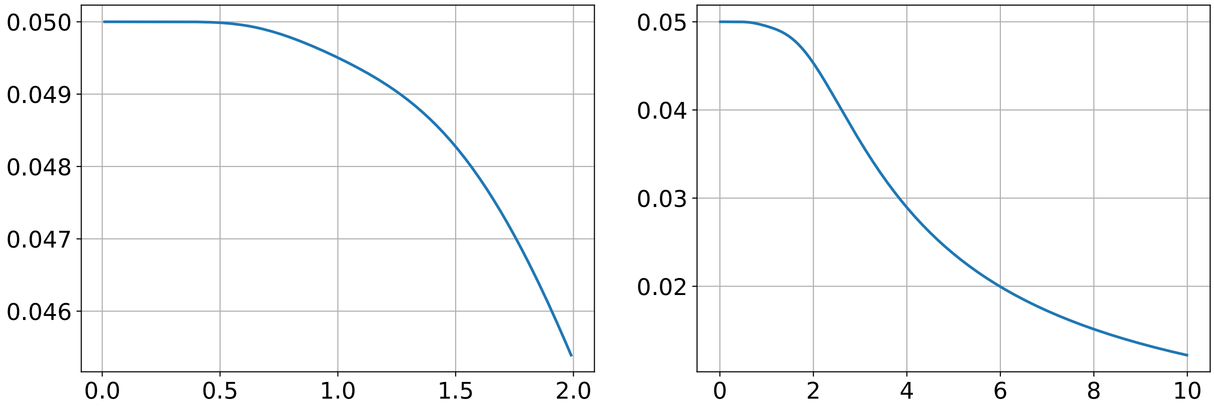


$k/j = 0.2$

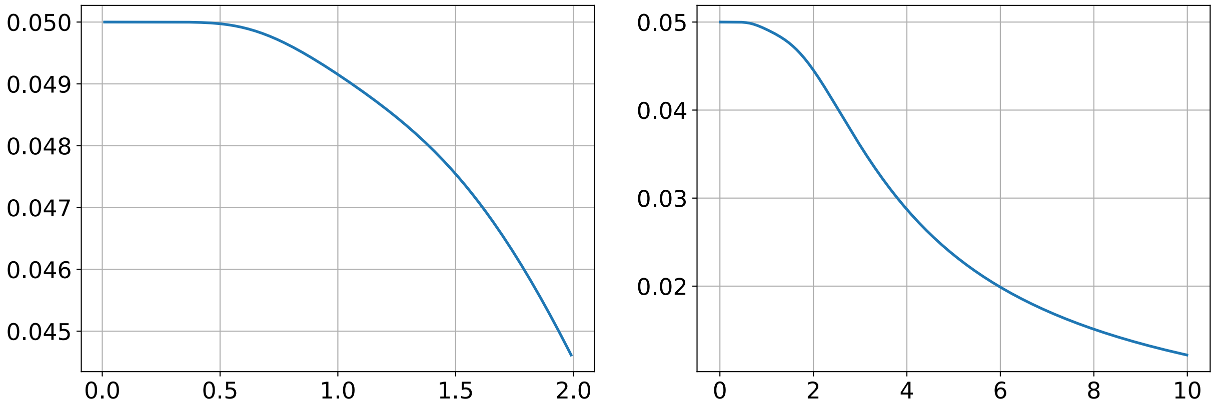



$k/j = 0.25$
0.0500
0.0495
0.0490
0.0485
0.0480
0.0475
0.0 0.5 1.0 1.5 2.0
0.05
0.04
0.03
0.02
0 2 4 6 8 10
$k/j = 0.3$
0.0500
0.0495
0.0490
0.0485
0.0480
0.0475
0.0470
0.0 0.5 1.0 1.5 2.0
0.05
0.04
0.03
0.02
0 2 4 6 8 10
$k/j = 0.35$
0.050
0.049
0.048
0.047
0.0 0.5 1.0 1.5 2.0
0.05
0.04
0.03
0.02
0 2 4 6 8 10
$k/j = 0.4$
0.050
0.049
0.048
0.047
0.046
0.0 0.5 1.0 1.5 2.0
0.05
0.04
0.03
0.02
0 2 4 6 8 10

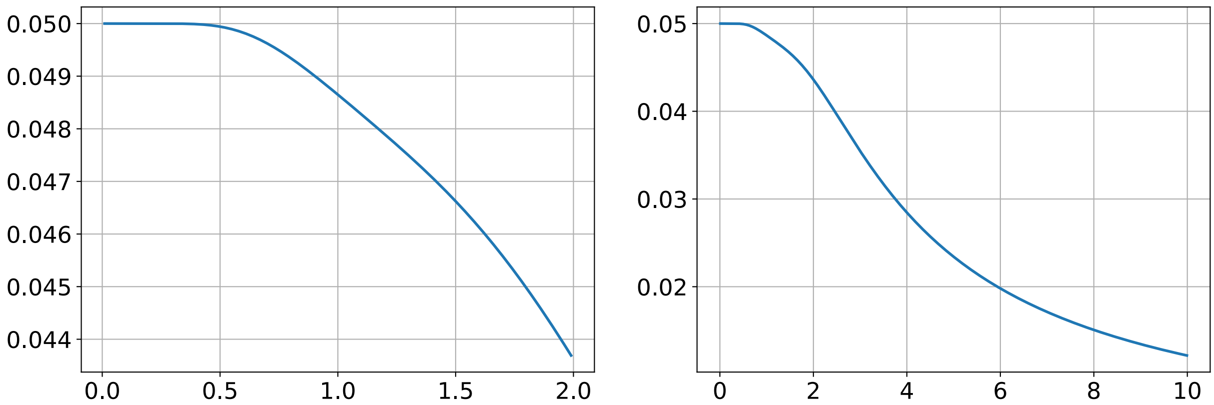
$k/j = 0.45$



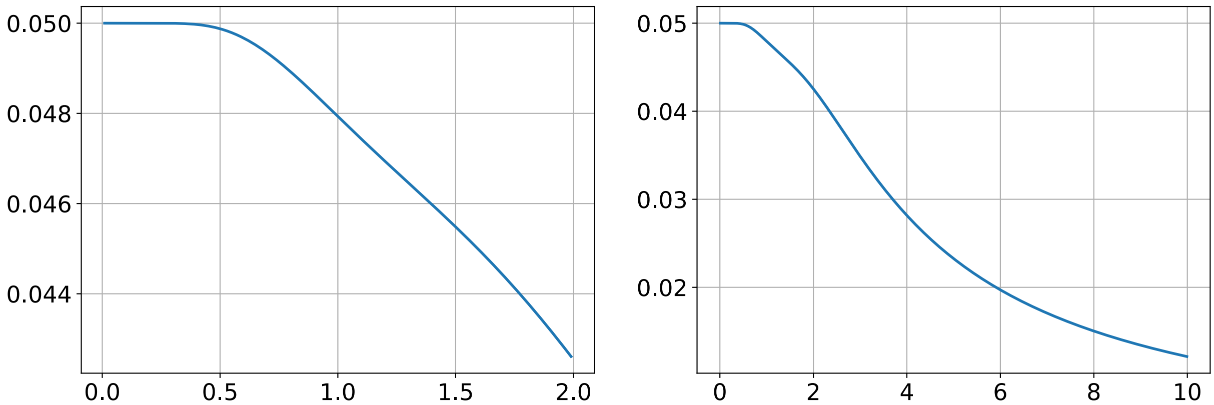
$k/j = 0.5$



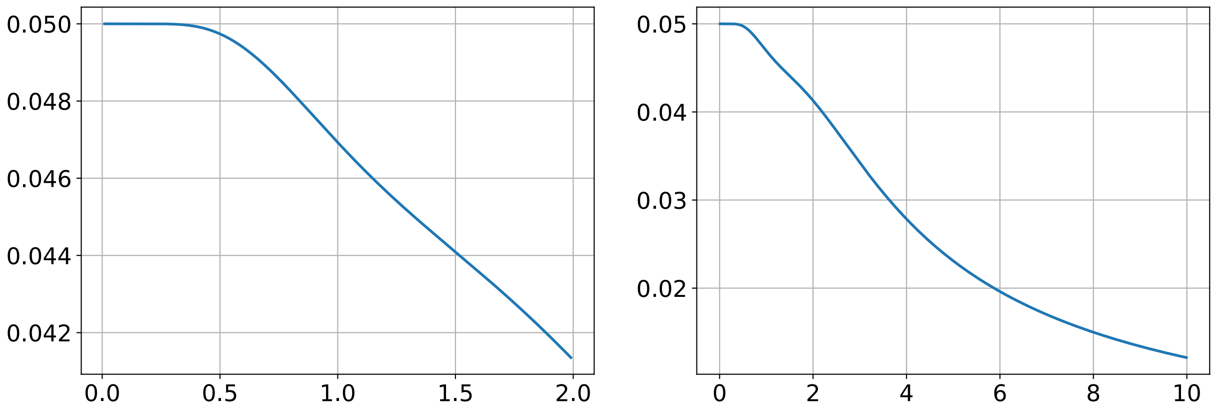
$k/j = 0.55$



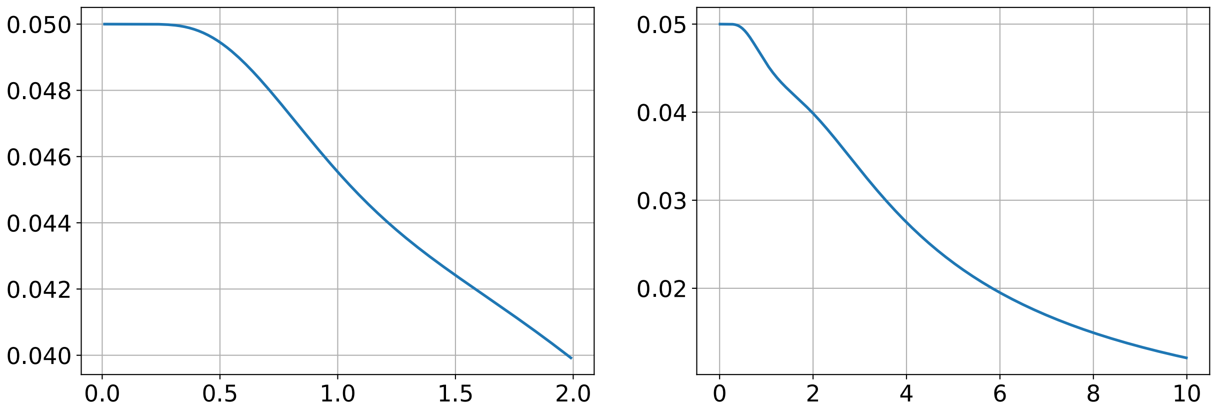
$k/j = 0.6$



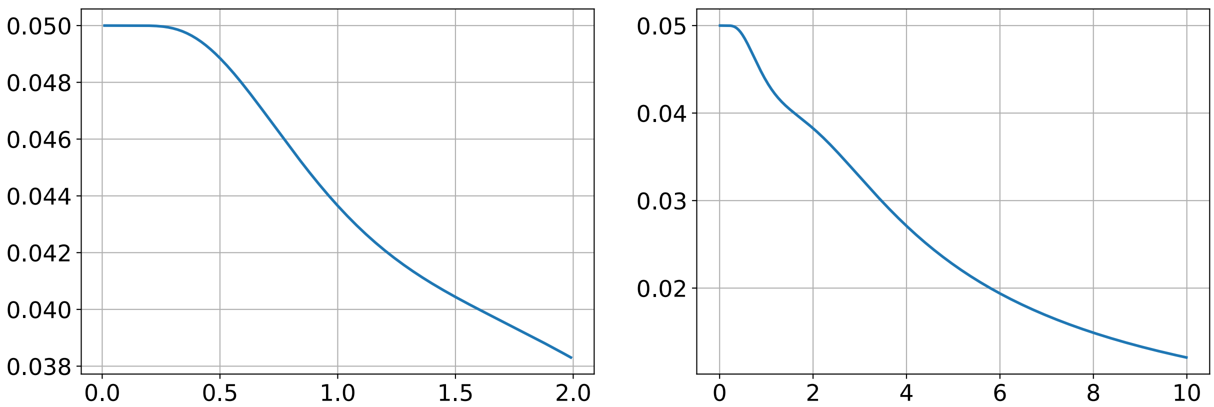
$k/j = 0.65$



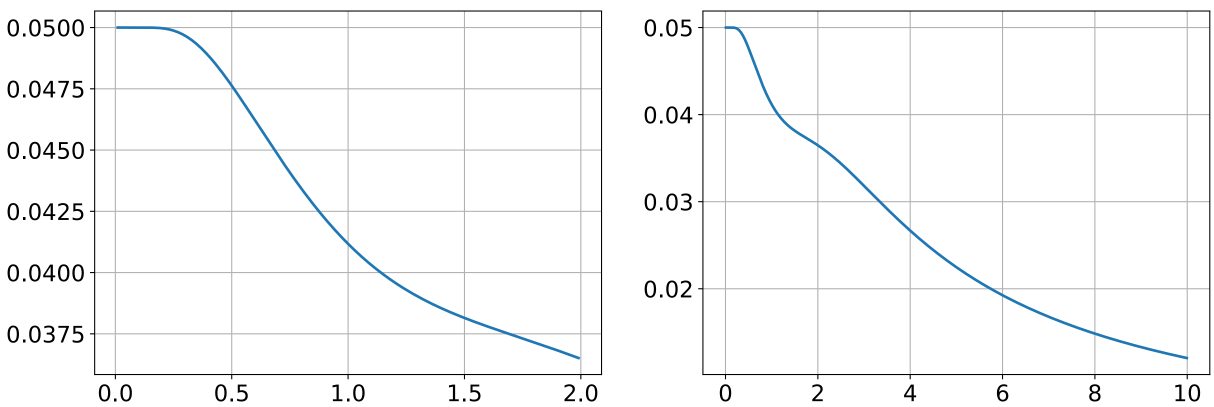
$k/j = 0.7$



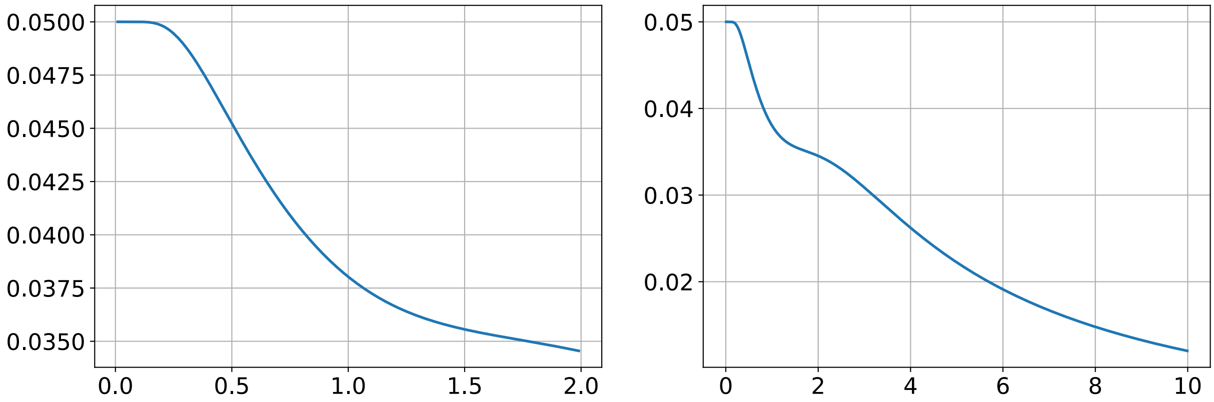
$k/j = 0.75$



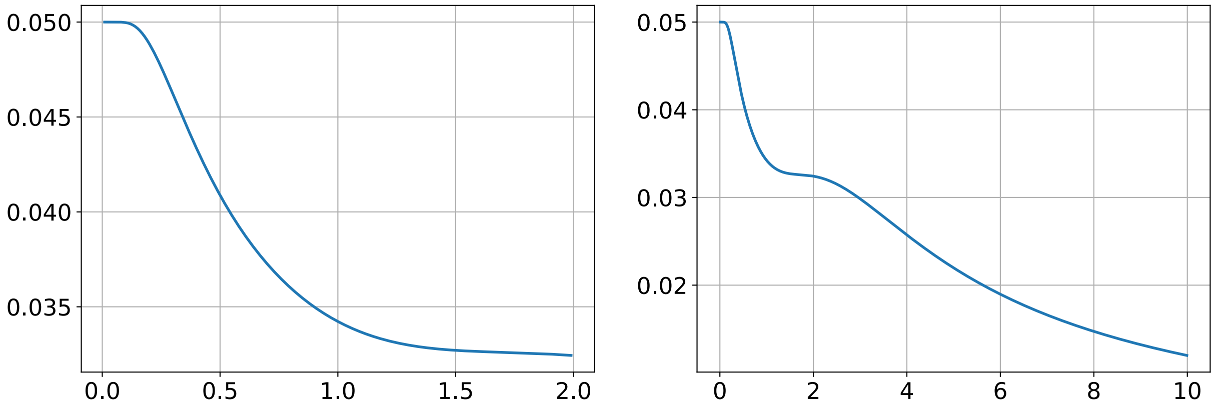
$k/j = 0.8$



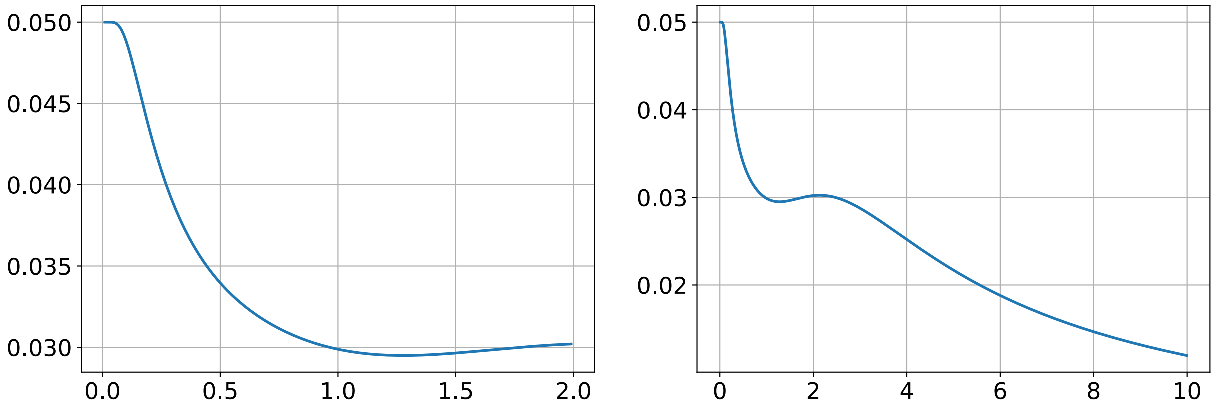
$k/j = 0.85$



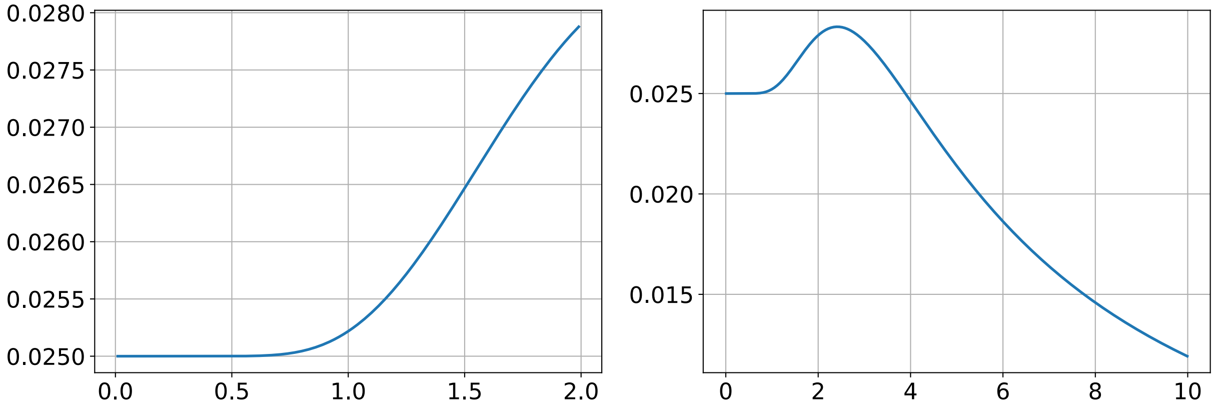
$k/j = 0.9$

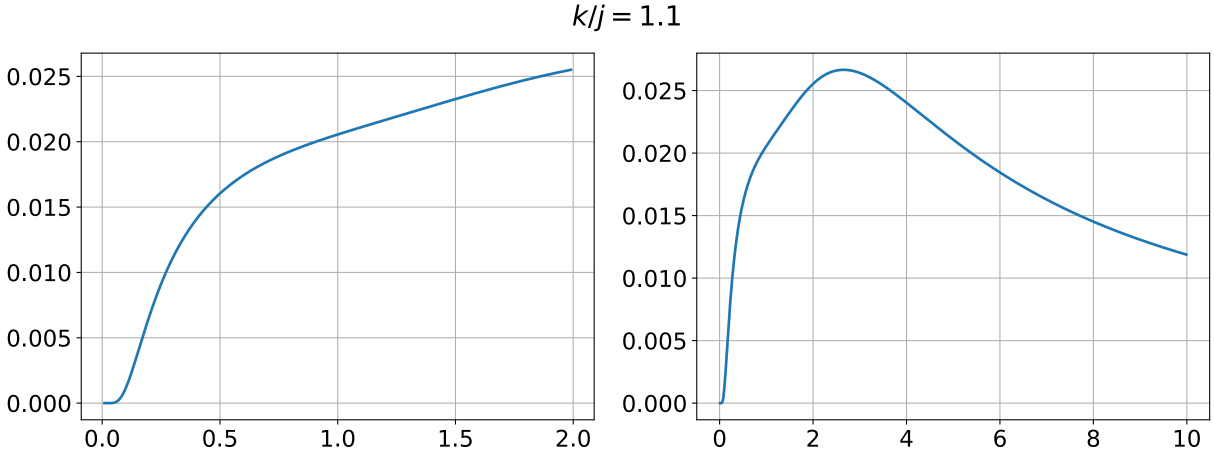


$k/j = 0.95$



$k/j = 1.0$




$k/j = 1.1$
0.025
0.020
0.015
0.010
0.005
0.000
0.0
0.5
1.0
1.5
2.0
0.025
0.020
0.015
0.010
0.005
0.000
0
2
4
6
8
10

In [ ]: