```
In [1]: %reload_ext autoreload
        %autoreload 2

        import numpy as np
        import matplotlib.pyplot as plt
        from scipy import constants as c

        from wave_propagation import *
```

3. Wave propagation

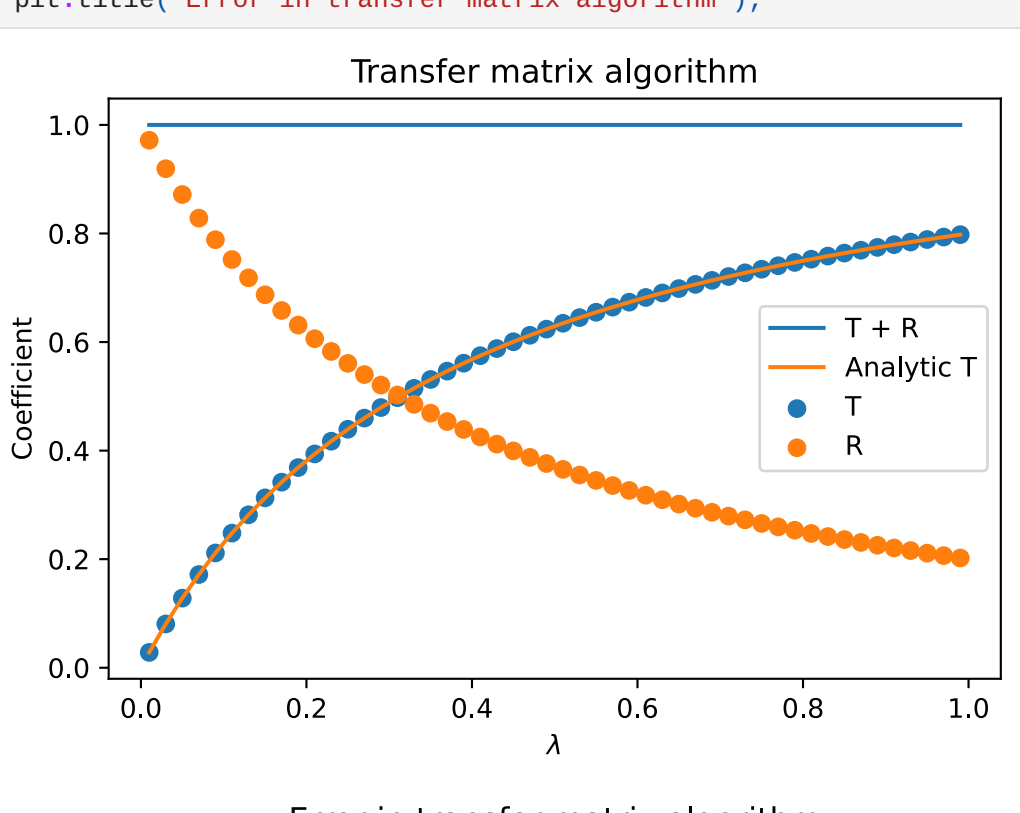
Kevin Vonk

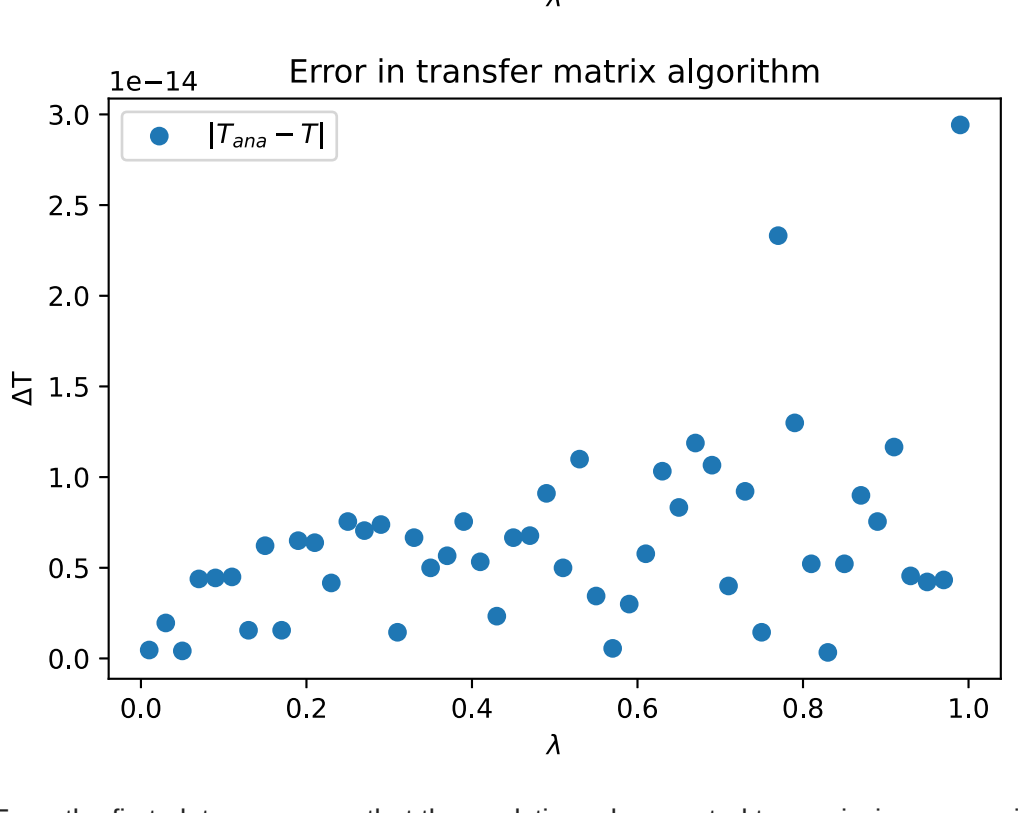
Mar - Apr 2020 & Mar 2021

Section 1: A square barrier

a)

```
In [2]: pot = Potential(-0.5, 0.5, 100, expr = lambda y: 1)
        lambda_ = np.linspace(0.01, 0.99, 50)

        T = []
        R = []

        for l in lambda_:
            tf = TransferMatrix(pot, l)
            T.append(tf.transmission())
            R.append(tf.reflection())

        plt.scatter(lambda_, T, label="T")
        plt.scatter(lambda_, R, label="R")
        plt.plot(lambda_, [i + j for i, j in zip(T, R)], label="T + R")
        plt.plot(lambda_, T_ana(lambda_), label="Analytic T")
        plt.xlabel(r"$\lambda$")
        plt.ylabel("Coefficient")
        plt.title("Transfer matrix algorithm")
        plt.legend()

        plt.figure()
        plt.scatter(lambda_, np.abs(T_ana(lambda_) - T), label=r"$| T_{ana} - T |$")
        plt.xlabel(r"$\lambda$")
        plt.ylabel(r"$\Delta$T")
        plt.legend()
        plt.title("Error in transfer matrix algorithm");
```



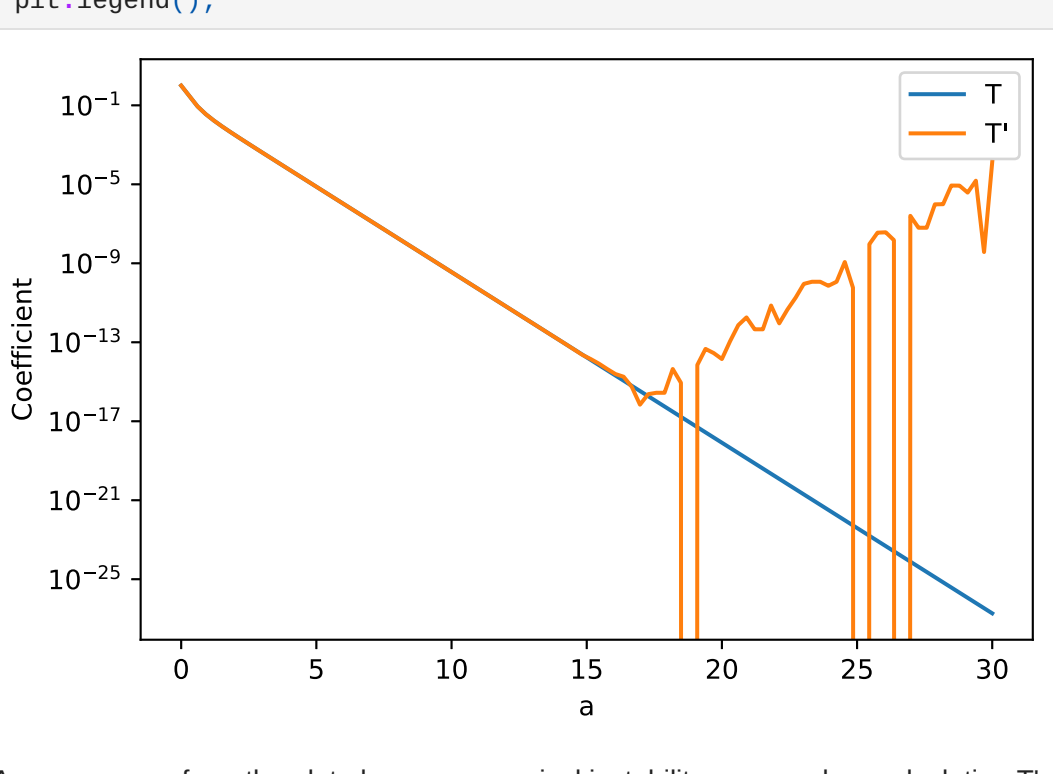


From the first plot we can see that the analytic and computed transmissions are quite close together. The reflection and transmission coefficients sum to 1 as expected.

Looking at the error plot, we see that the accuracy is in the order of roughly 10^{-14}.

b)

```
In [3]: lambda_ = 0.01
        A = np.linspace(0, 30, 100)

        T = []
        Tp = []

        for a in A:
            pot = Potential(-0.5*a, 0.5*a, 100, expr = lambda y: 1)
            tf = TransferMatrix(pot, lambda_)

            T.append(tf.transmission())
            Tp.append(tf.transmission(dir="left"))

        plt.semilogy(A, T, label="T")
        plt.semilogy(A, Tp, label="T'")
        plt.xlabel("a")
        plt.ylabel("Coefficient")
        plt.legend();
```



As we can see from the plot above, a numerical instability occurs when calculating T' when the barrier thickness a becomes too large. If the numeric solution were to be accurate, one would expect T and T' to be identical. This is because the properties of a medium are independent of the directionality of the incoming wave in this case.

The reason that this numerical instability occurs has to do with the way a computer stores numbers. Decimal numbers are stored as floats: floating point numbers. On a 64-bit machine, 52 bits are used to store the fraction, 11 bits for the exponent and 1 bit for the sign, resulting in the following expression for the number:

$$n = (-1)^{\text{sign}} * \text{fraction} * 10^{\text{exponent}} \tag{1}$$

(The actual number conversion is slightly different from the representation here given by IEEE 754, but that is beyond the scope of this report.)

Because we have a finite amount of bits to represent every number, we cannot represent every number exactly. This results in a grid or a number line of values we can actually represent. For example, say that a float can represent every $0, 1n$ steps, where $n \in \mathbb{Z}$. If we want to have the number $0, 17$ represented in this system, it must be rounded up to $0, 2$ in order to be stored.

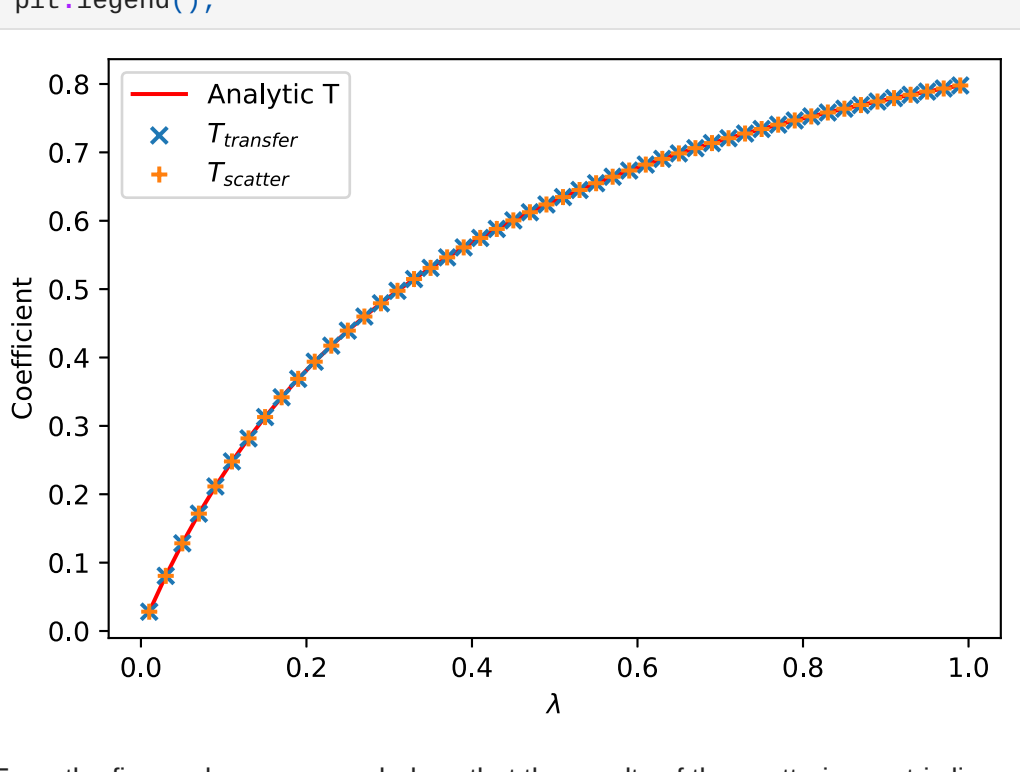
This problem compounds itself when doing arithmatic upon two floats. If the exponents are similar in their orders of magnitude, the accuracy lost due to floating point rounding errors is limited. However, if there is too much variation in the exponents, the largest one will take precedence. In order for arithmatic to occur, the float with the smaller exponent must realign itself onto the grid of the larger exponent. In the worst case, the closest point to which the float can align is zero, and thus the smaller float is lost completely in the arithmatic operation.

This problem then is what we see in the figure above. The coefficients become smaller and smaller, up to a point where there simply aren't enough bits of accuracy to complete the arithmatic "correctly" (floating point math is never really "correct", it is always an approximation). This results in the upwards slope of T'. The zero results are also due to the same inaccuracies, as the small differences equate to two identical numbers, cancelling each other in the process. Since we only have 11 bits of exponent accuracy, it is fairly easy to run into this problem.

c)

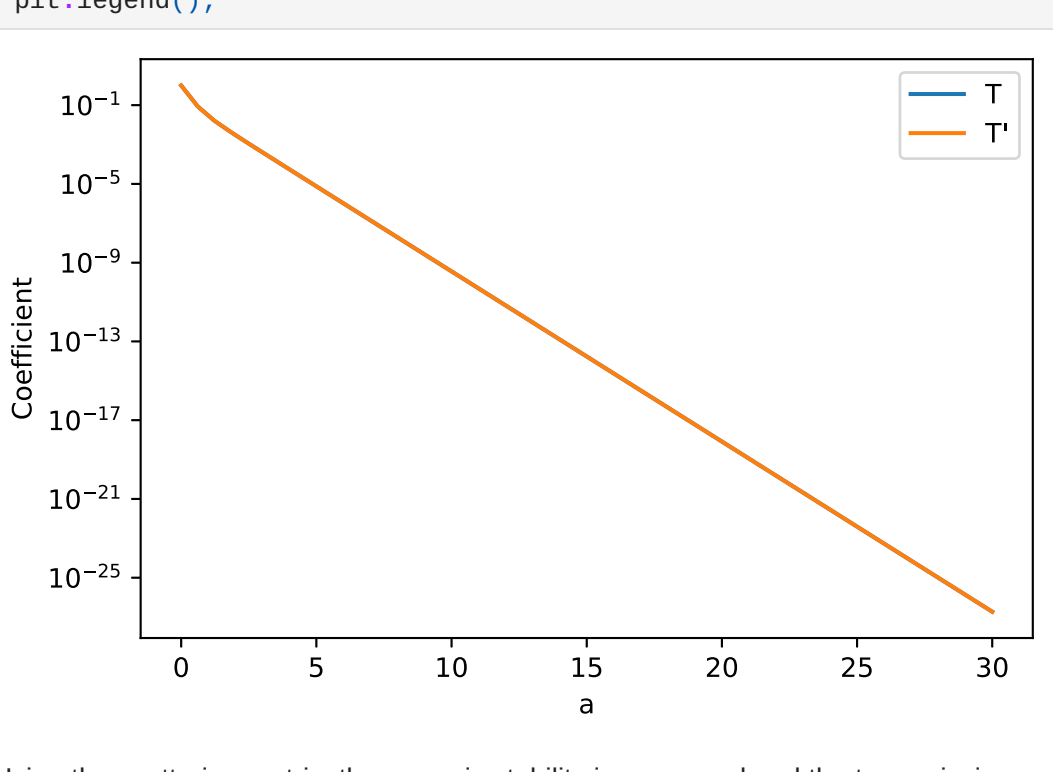
As a sanity check, let us first compare the results of 1a) to our scattering matrix result. If everything has gone well, the results should be the same.

```
In [4]: pot = Potential(-0.5, 0.5, 100, expr = lambda y: 1)
        lambda_ = np.linspace(0.01, 0.99, 50)

        Tt = []
        Ts = []

        for l in lambda_:
            tm = TransferMatrix(pot, l)
            sm = ScatterMatrix(pot, l)
            Tt.append(tm.transmission())
            Ts.append(sm.transmission())

        plt.plot(lambda_, T_ana(lambda_), "-r", label="Analytic T", zorder=-100)
        plt.scatter(lambda_, Tt, label=r"$T_{transfer}$", marker="x")
        plt.scatter(lambda_, Ts, label=r"$T_{scatter}$", marker="+")
        plt.xlabel(r"$\lambda$")
        plt.ylabel("Coefficient")
        plt.legend();
```



From the figure above we can deduce that the results of the scattering matrix line up with the results of both the transfer matrix algorithm and the analytic expression. Now then, let us redo the experiment of 1b) with the scattering matrix.

```
In [5]: lambda_ = 0.01
        A = np.linspace(0, 30, 50)

        T = []
        Tp = []

        for a in A:
            pot = Potential(-0.5*a, 0.5*a, 100, expr = lambda y: 1)
            sm = ScatterMatrix(pot, lambda_)

            T.append(sm.transmission())
            Tp.append(sm.transmission(dir="left"))

        plt.semilogy(A, T, label="T")
        plt.semilogy(A, Tp, label="T'")
        plt.xlabel("a")
        plt.ylabel("Coefficient")
        plt.legend();
```

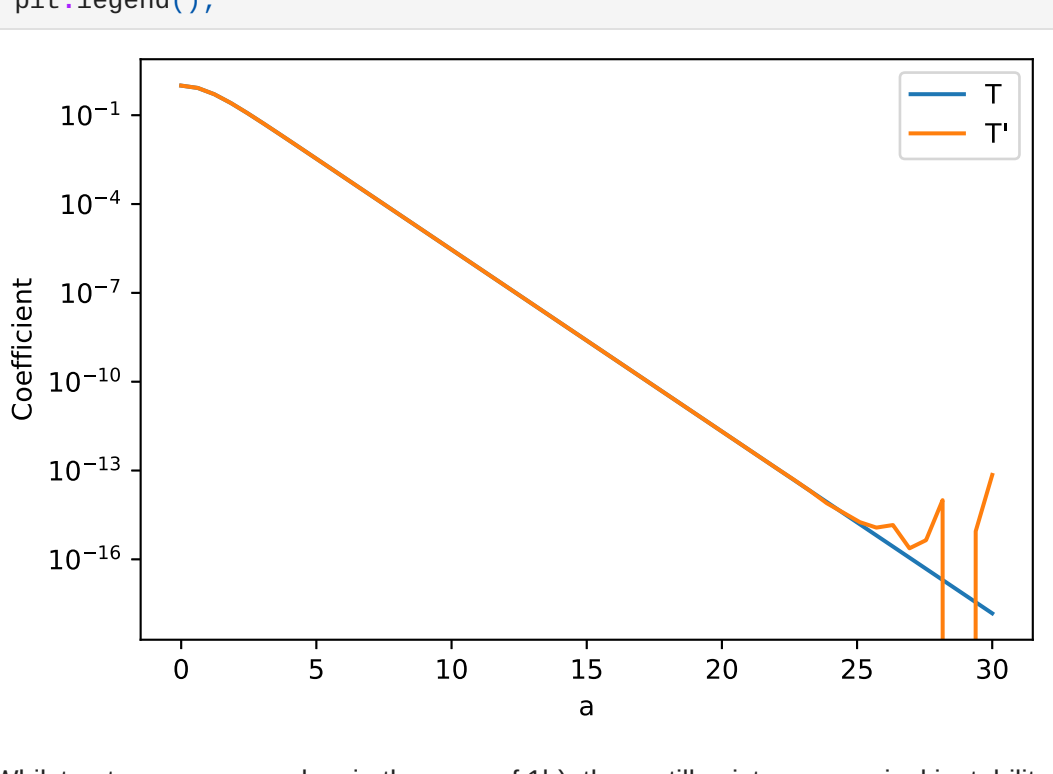


Using the scattering matrix, the numeric stability is preserved and the transmission coefficient from either direction is identical. The reason that we do not see the floating point rounding errors in T' here, is simply because propagation from the left and from the right are treated as equal, instead of forced to originate from one direction.

d)

Taking the code from 1b) and setting $\lambda = 0, 5$:

```
In [6]: lambda_ = 0.5
        A = np.linspace(0, 30, 50)

        T = []
        Tp = []

        for a in A:
            pot = Potential(-0.5*a, 0.5*a, 100, expr = lambda y: 1)
            tf = TransferMatrix(pot, lambda_)

            T.append(tf.transmission())
            Tp.append(tf.transmission(dir="left"))

        plt.semilogy(A, T, label="T")
        plt.semilogy(A, Tp, label="T'")
        plt.xlabel("a")
        plt.ylabel("Coefficient")
        plt.legend();
```



Whilst not as pronounced as in the case of 1b), there still exists a numerical instability when using the transfer matrix for increased barrier thickness. The reason for the instability occuring at a larger thickness compared to 1b) is because there is more energy for the wave to propagate through the barrier, meaning that the decrease in the transmission coefficient is less drastic. However, once we near the region where we do not have enough bits of accuracy, we start seeing the numerical instability yet again.

All in all, the scattering matrix algorithm seems like the more stable algorithm. Furthermore, it is not less accurate than the transfer matrix algorithm in areas where the numerical stability of the transfer matrix is a non-issue. Additionally, the numerical complexity O is the same order for both algorithms. Hence, I recommend the scattering matrix algorithm over the transfer matrix algorithm.

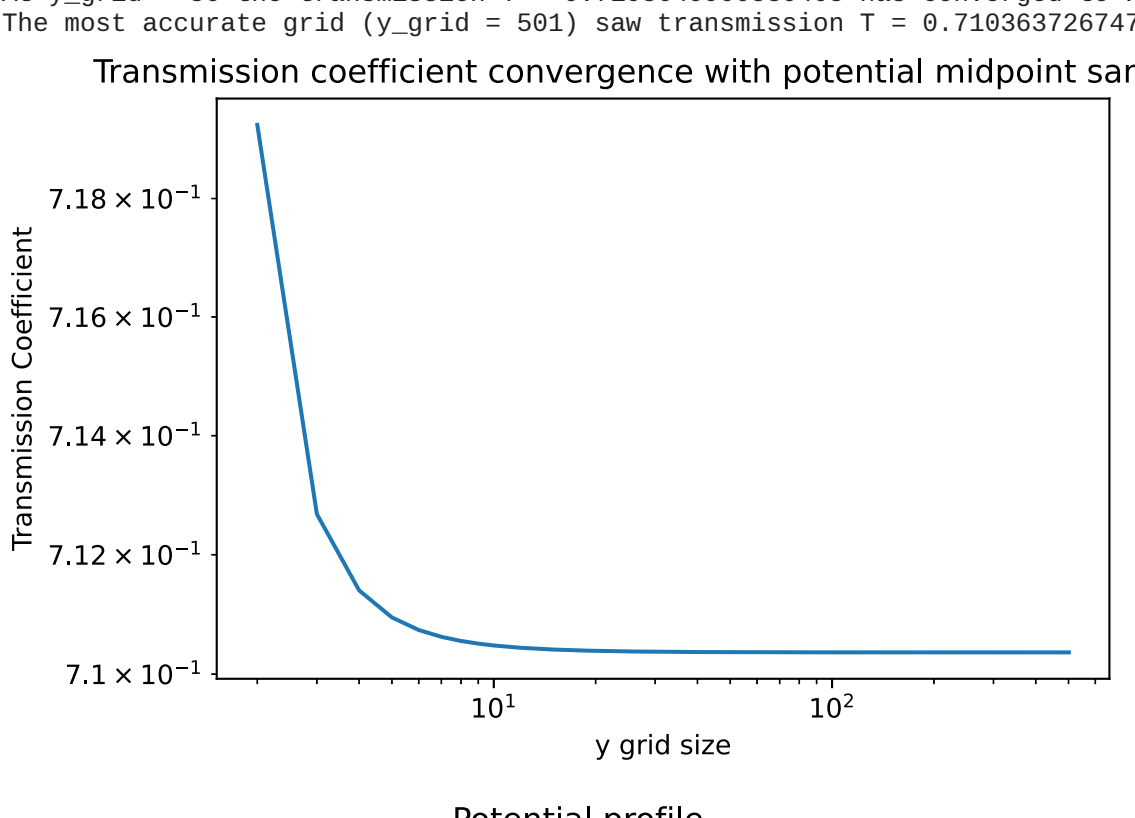
Section 2: A distorted barrier

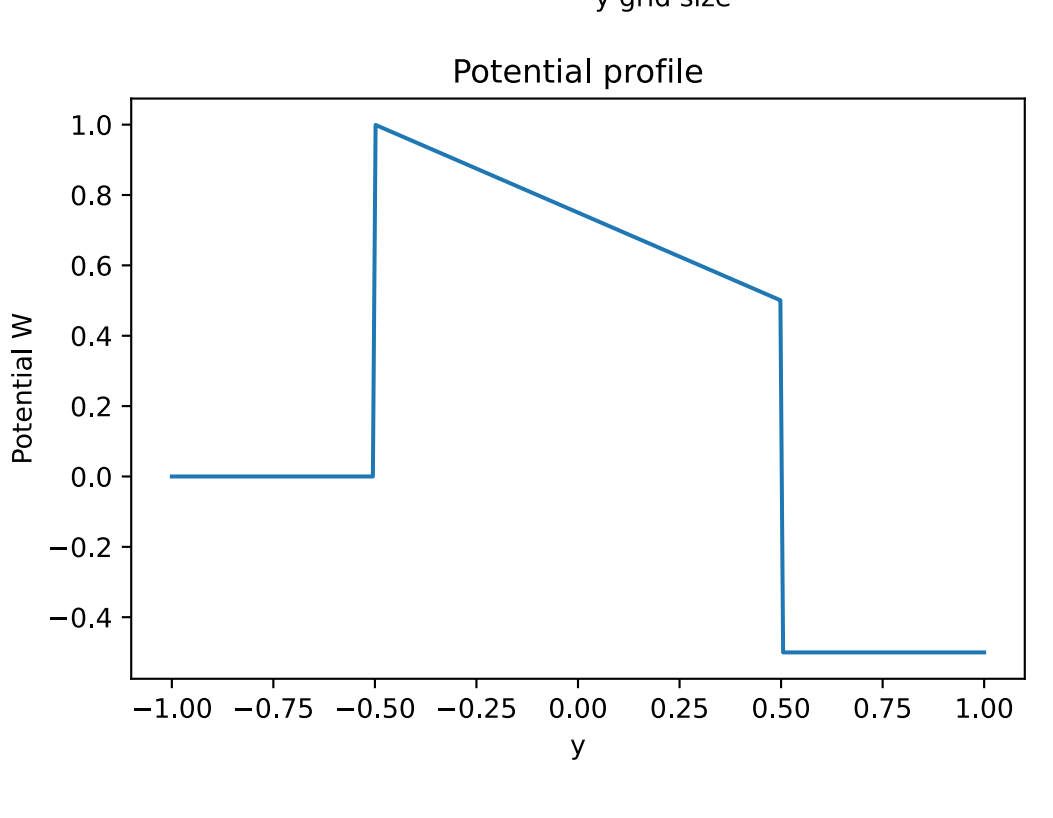
a)

In order to determine the minimal amount of grid points required to describe the transmission coefficient with the requested accuracy, I will iterate over a large amount of grid points. We can assume the transmission coefficient be fully converged when the amount of grid points nears infinity. Thus, a large amount of grid points already consitutes a "fair" accuracy.

In this instance, we calculated the transmission coefficient for a range between 2 and 501 grid points. The convergence of 10^{-5} will be compared to the most accurate grid (with 501 points). From the results below we can see that we need 86 grid points to obtain the required accuracy. Discarding digits not within the accuracy window, we obtain $T = 0, 71036$, which is the same for 86 and 501 grid points. Thus, 86 grid points is accuracte enough.

```
In [7]: V1 = 1
        a = 1
        dV = 0.5
        lambda_ = 0.5
        epsilon = dV / a
        conv = 1E-5

        start = 2
        end = start + 500

        T = np.zeros(end - start)

        for i, grid in enumerate(range(start, end)):
            pot = Potential(-0.5*a, 0.5*a, grid, \
                            expr = lambda y: V1 - epsilon * (y + 0.5*a), \
                            outside_range = (lambda _: 0, lambda _: -epsilon * a))
            sm = ScatterMatrix(pot, lambda_)
            sm.midpoint = True

            T[i] = sm.transmission()

        conv_keys = [i for i, _ in enumerate(T[:-1]) if int(round(T[i] / conv)) == int(round(
        print(f"At y_grid = {conv_keys[0] + start} the transmission T = {T[conv_keys[0]]} has
        print(f"The most accurate grid (y_grid = {len(T) - 1 + start}) saw transmission T = {

        plt.loglog(range(start, end), T)
        plt.xlabel("y grid size")
        plt.ylabel("Transmission Coefficient")
        plt.title("Transmission coefficient convergence with potential midpoint sampling")

        plt.figure()
        plt.plot(np.linspace(-1, 1, 300), [pot[i] for i in np.linspace(-1, 1, 300)])
        plt.xlabel("y")
        plt.ylabel("Potential W")
        plt.title("Potential profile");

At y_grid = 86 the transmission T = 0.7103649900530405 has converged to within 1e-05
The most accurate grid (y_grid = 501) saw transmission T = 0.710363726747062
```





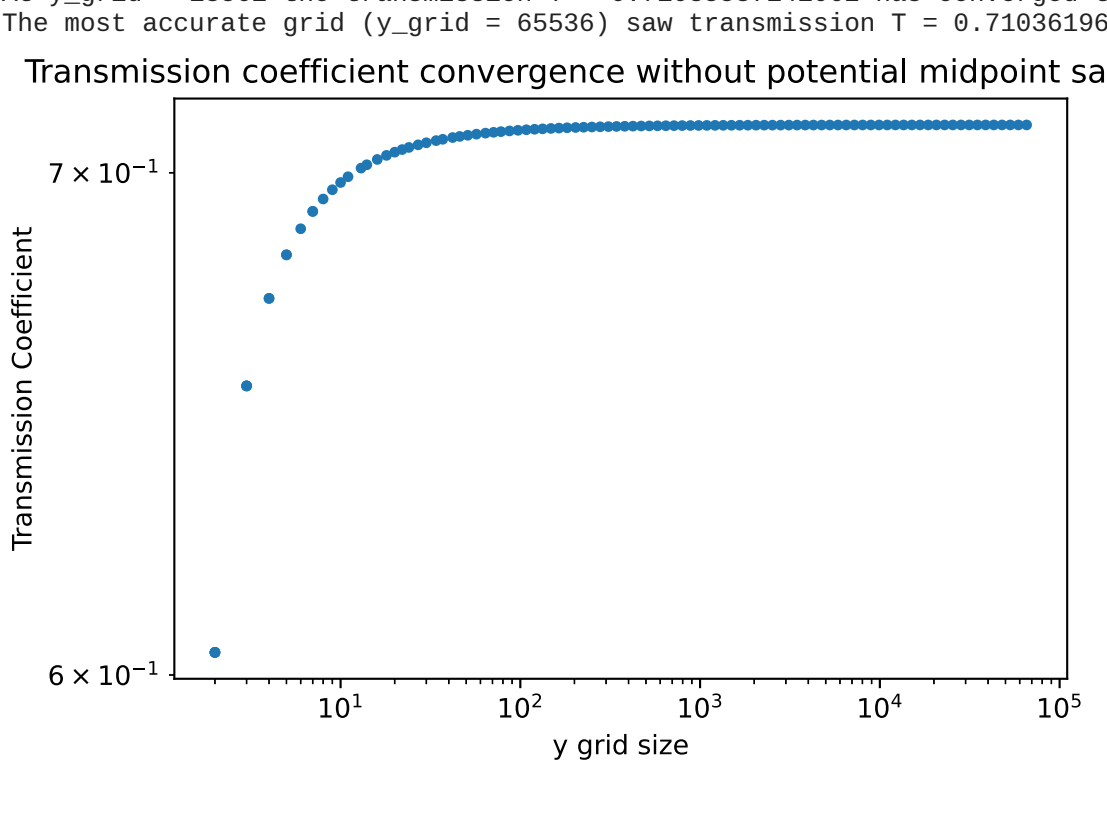
b)

The code below is mostly copy-pasted from 2a), but now not taking midpoints when sampling the potential. This time, we will consider various grid sizes between 2 and $65536 = 2^{16}$.

The resulting figure shows that the convergence is much slower when compared to the results of a). To obtain a convergence of 10^{-5}, we need at least 13561 grid points, compared to 86 when using the midpoint rule. Even though this method is much slower, we do eventually obtain convergence.

From these results it is clear that the grid resulting from the midpoint rule is far more favourable, and is thus the best grid to use.

```
In [8]: grid_range = 2**np.linspace(1, 16, num=100)
        grid_range = grid_range.astype(np.int_)

        T = np.zeros(grid_range.size)

        for i, grid in enumerate(grid_range):
            pot = Potential(-0.5*a, 0.5*a, grid, \
                            expr = lambda y: V1 - epsilon * (y + 0.5*a), \
                            outside_range = (lambda _: 0, lambda _: -epsilon * a))
            sm = ScatterMatrix(pot, lambda_)
            sm.midpoint = False

            T[i] = sm.transmission()

        conv_keys = [i for i, _ in enumerate(T[:-1]) if int(round(T[i] / conv)) == int(round(
        print(f"At y_grid = {grid_range[conv_keys[0]]} the transmission T = {T[conv_keys[0]]}
        print(f"The most accurate grid (y_grid = {grid_range[-1]}) saw transmission T = {T[-1

        plt.loglog(grid_range, T, '.')
        plt.xlabel("y grid size")
        plt.ylabel("Transmission Coefficient")
        plt.title("Transmission coefficient convergence without potential midpoint sampling")

At y_grid = 13561 the transmission T = 0.71035537141062 has converged to within 1e-05
The most accurate grid (y_grid = 65536) saw transmission T = 0.7103619681016985
```



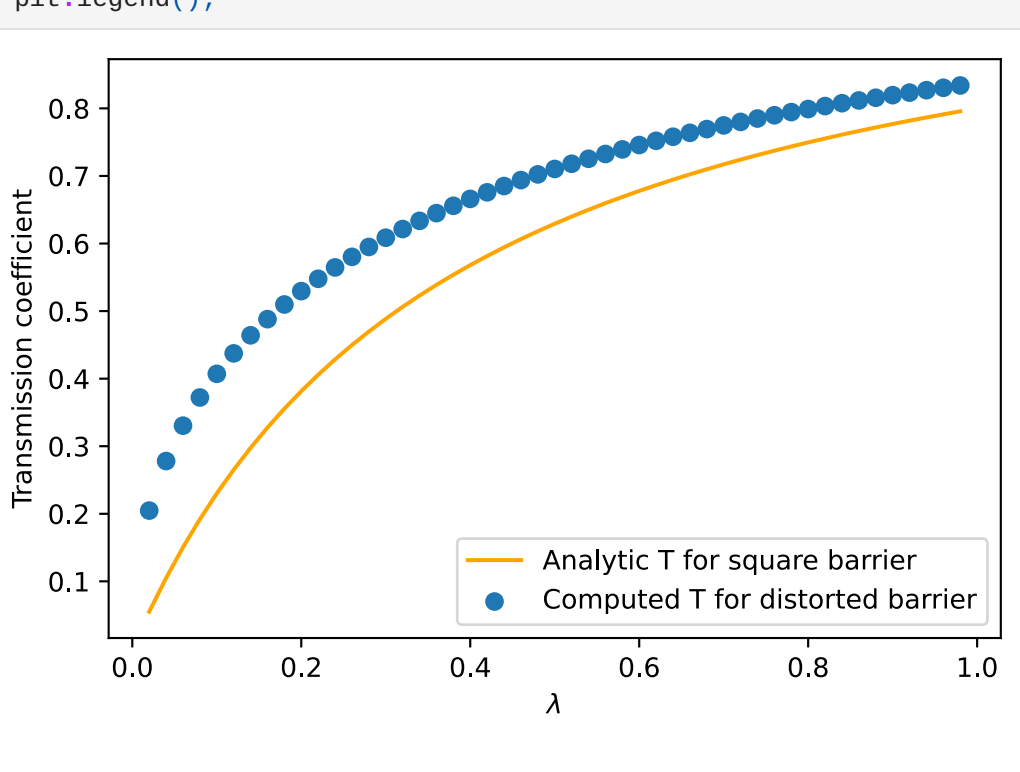
c)

The plot of the analytic transmission coefficient for the square potential and the computed transmission coefficient for the triangularly distorted potential is found below. From this plot we gather that the transmission for the distorted barrier is higher than that of the square barrier. This is especially the case for lower energies, as for higher energies the transmission coefficient will converge towards 1.

The main reason for this is because the average potential of the distorted barrier is lower than that of the square barrier. Additionally, the potential on the right hand side of the barrier is lower than that on the left side. This makes it energetically favourable for the waves to be on the right side of the barrier. This increases the transmission for this system.

The profile of the potential itself will also influence the transmission. Most notably a higher potential means a faster decay of transmission given increasing barrier thickness. In the case of the distorted potential, the decreasing potential profile means that the transmission coefficient decays less quickly when an incident wave passes through. This however is harder to derive from the figure below.

```
In [9]: lambda_ = np.linspace(0, V1, 50, endpoint=False)[1:]
        grid = 86

        T = []

        for l in lambda_:
            pot = Potential(-0.5*a, 0.5*a, grid, \
                            expr = lambda y: V1 - epsilon * (y + 0.5*a), \
                            outside_range = (lambda _: 0, lambda _: -epsilon * a))
            sm = ScatterMatrix(pot, l)
            sm.midpoint = True

            T.append(sm.transmission())

        plt.scatter(lambda_, T, label="Computed T for distorted barrier")
        plt.plot(lambda_, T_ana(lambda_), label="Analytic T for square barrier", color="orange
        plt.xlabel(r"$\lambda$")
        plt.ylabel("Transmission coefficient")
        plt.legend();
```

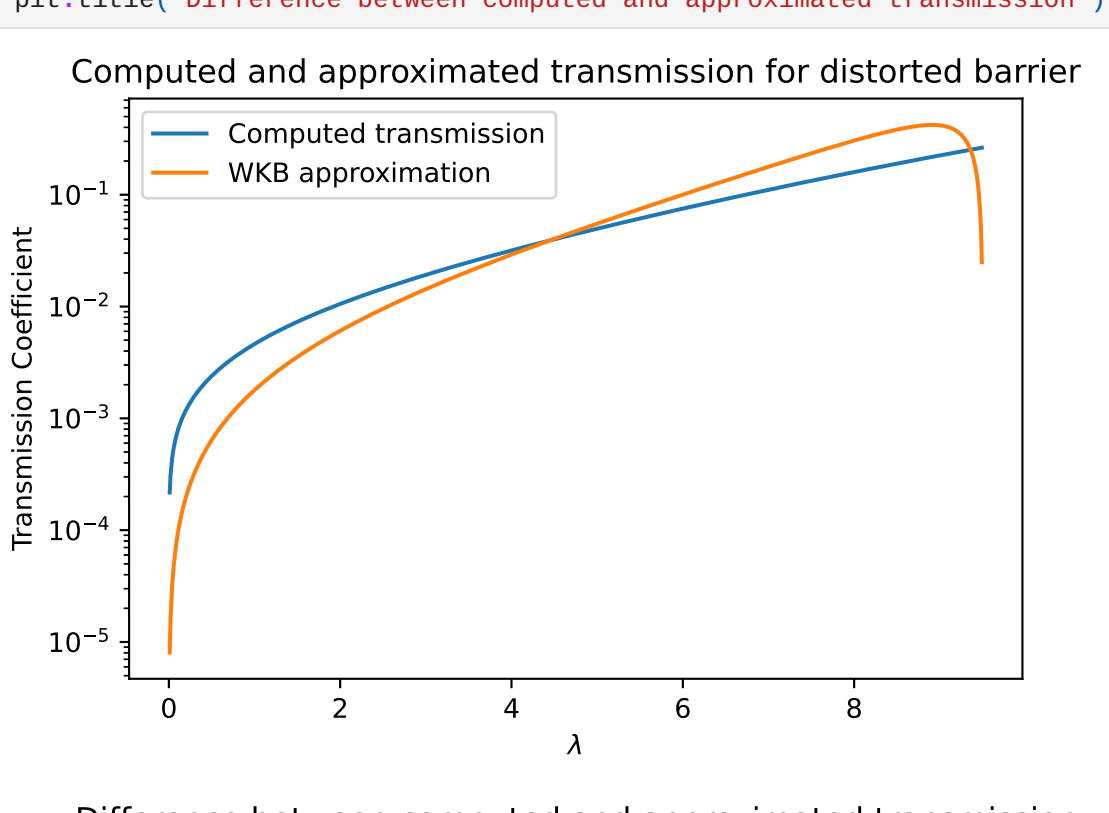


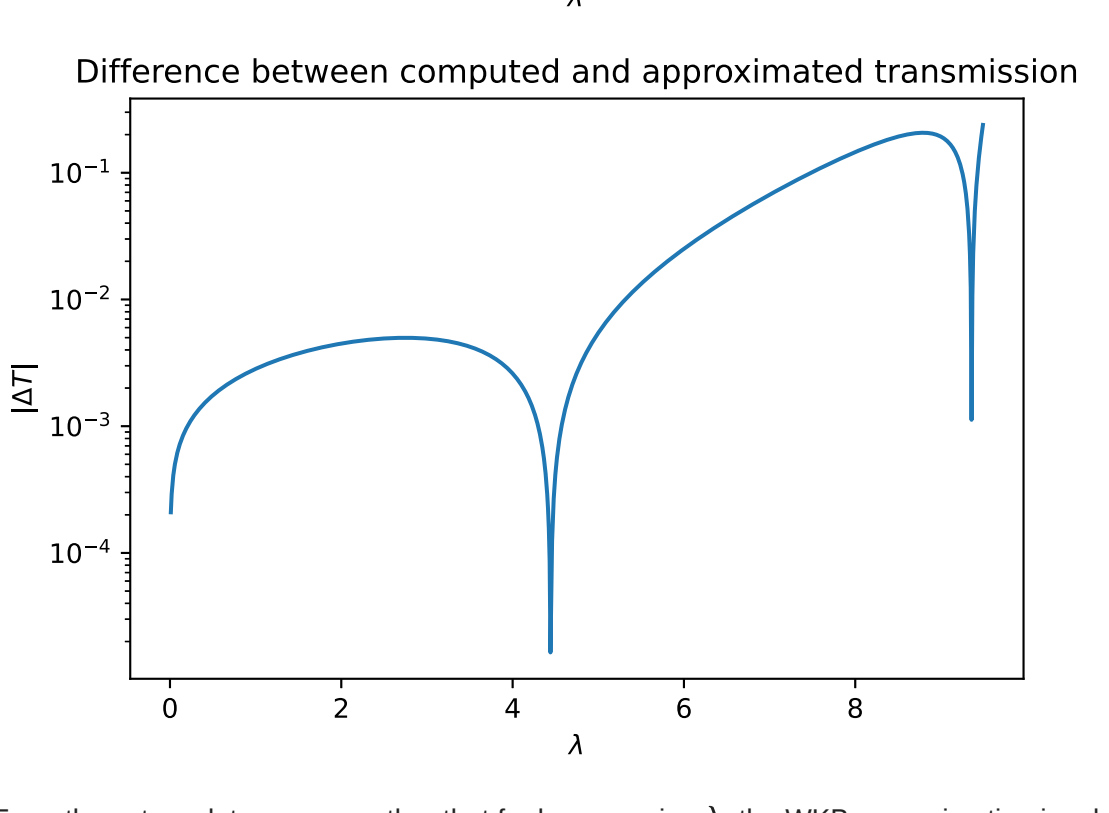
Section 3: The WKB approximation

a)

Both a plot of $T(\lambda)$ and $T_{WKB}(\lambda)$ and a plot of the difference between the two can be found below.

```
In [10]: V1 = 10
         a = 1
         dV = 0.5
         epsilon = dV / a

         lambda_ = np.linspace(0.01, V1-dV, 1000, endpoint=False)

         T = []
         Twkb = []

         for l in lambda_:
             pot = Potential(-0.5*a, 0.5*a, 100, \
                             expr = lambda y: V1 - epsilon * (y + 0.5*a), \
                             outside_range = (lambda _: 0, lambda _: -epsilon * a))
             sm = ScatterMatrix(pot, l)
             sm.midpoint = True

             T.append(sm.transmission())
             Twkb.append(sm.T_wkb(lambda lam: (16*lam*(V1-dV-lam))/V1**2))

         plt.semilogy(lambda_, T, label="Computed transmission")
         plt.semilogy(lambda_, Twkb, label="WKB approximation")
         plt.xlabel(r"$\lambda$")
         plt.ylabel("Transmission Coefficient")
         plt.title("Computed and approximated transmission for distorted barrier")
         plt.legend()

         plt.figure()
         plt.semilogy(lambda_, np.abs(np.array(T)-np.array(Twkb)))
         plt.xlabel(r"$\lambda$")
         plt.ylabel(r"$|\Delta T|$")
         plt.title("Difference between computed and approximated transmission");
```



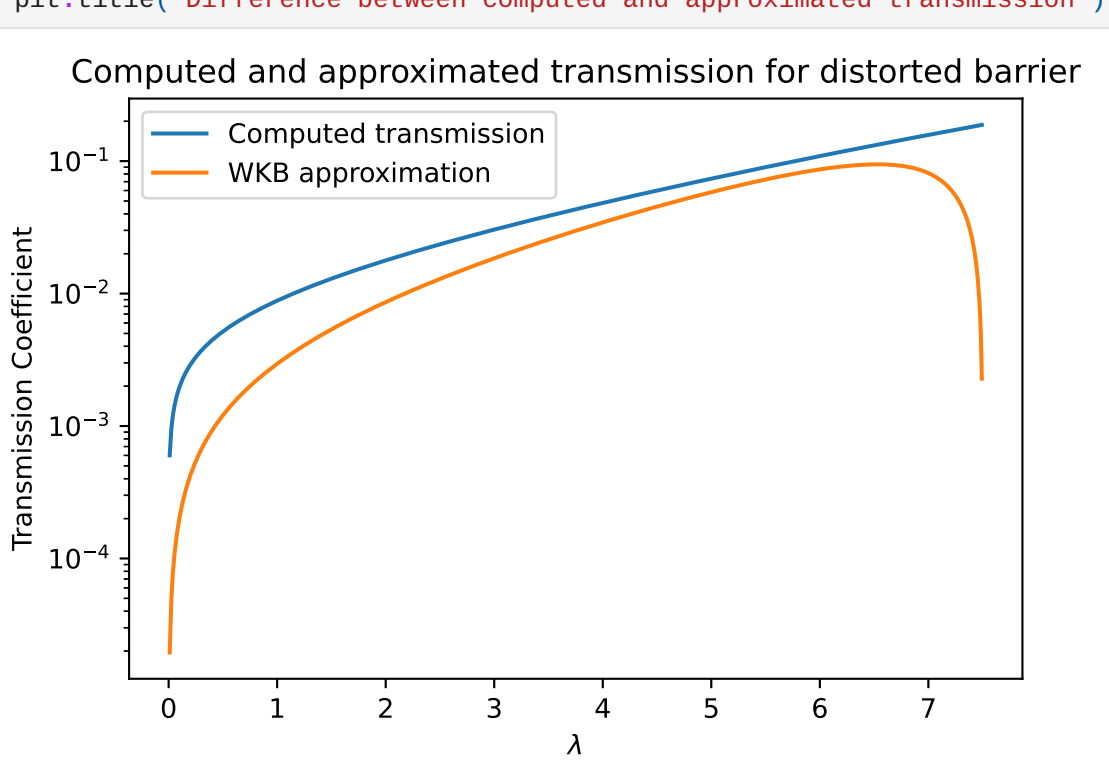


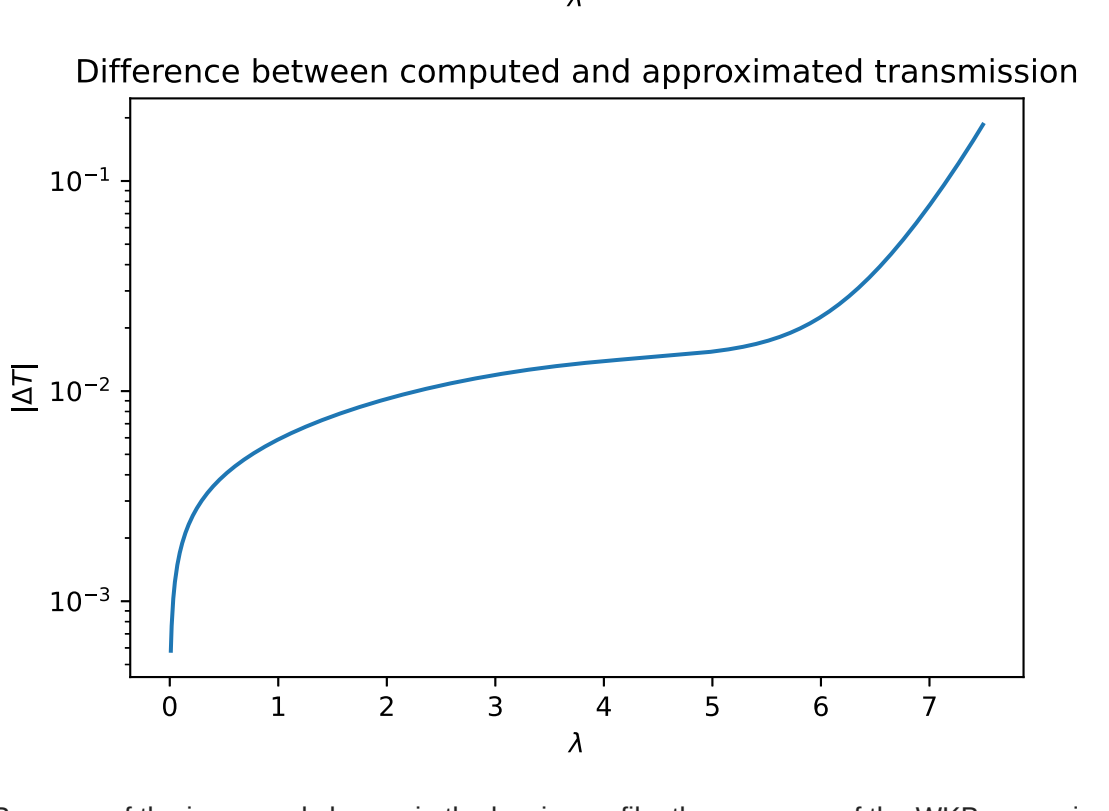
From these two plots we can gather that for low energies λ, the WKB approximation is relatively accurate. Up to around $\lambda \approx 6$, the accuracy in the approximated transmission coefficient is $\leq 10^{-3}$. For higher energies however, the WKB approximation starts to fail. We see that around $\lambda \approx 9$ the approximated transmission coefficient starts to decrease rapidly. We would however expect that the transmission reaches 1 as the energy goes to infinity. Thus, for higher energies the computed transmission coefficient seems to hold on.

b)

Modifying the code from 3a) slightly, we obtain the results as found below.

```
In [11]: V1 = 10
         a = 1
         dV = 2.5
         epsilon = dV / a

         lambda_ = np.linspace(0.01, V1-dV, 1000, endpoint=False)

         T = []
         Twkb = []

         for l in lambda_:
             pot = Potential(-0.5*a, 0.5*a, 100, \
                             expr = lambda y: V1 - epsilon * (y + 0.5*a), \
                             outside_range = (lambda _: 0, lambda _: -epsilon * a))
             sm = ScatterMatrix(pot, l)
             sm.midpoint = True

             T.append(sm.transmission())
             Twkb.append(sm.T_wkb(lambda lam: (16*lam*(V1-dV-lam))/V1**2))

         plt.semilogy(lambda_, T, label="Computed transmission")
         plt.semilogy(lambda_, Twkb, label="WKB approximation")
         plt.xlabel(r"$\lambda$")
         plt.ylabel("Transmission Coefficient")
         plt.title("Computed and approximated transmission for distorted barrier")
         plt.legend()

         plt.figure()
         plt.semilogy(lambda_, np.abs(np.array(T)-np.array(Twkb)))
         plt.xlabel(r"$\lambda$")
         plt.ylabel(r"$|\Delta T|$")
         plt.title("Difference between computed and approximated transmission");
```



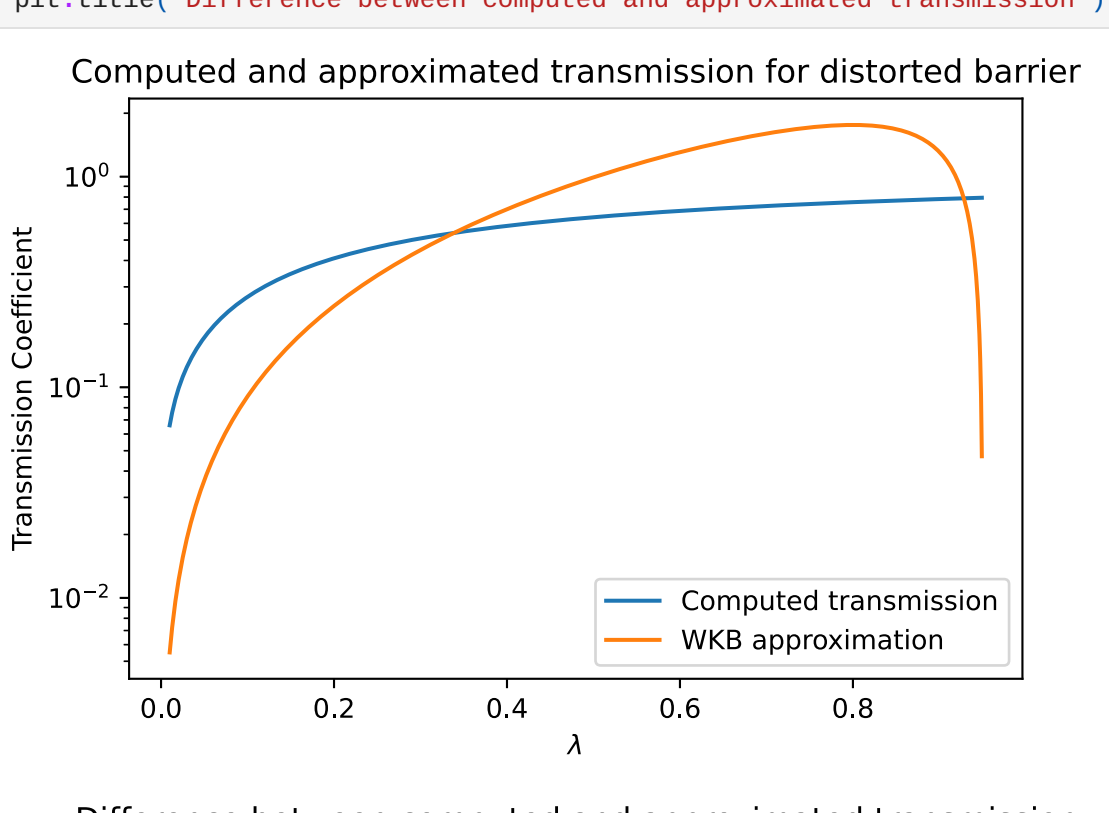


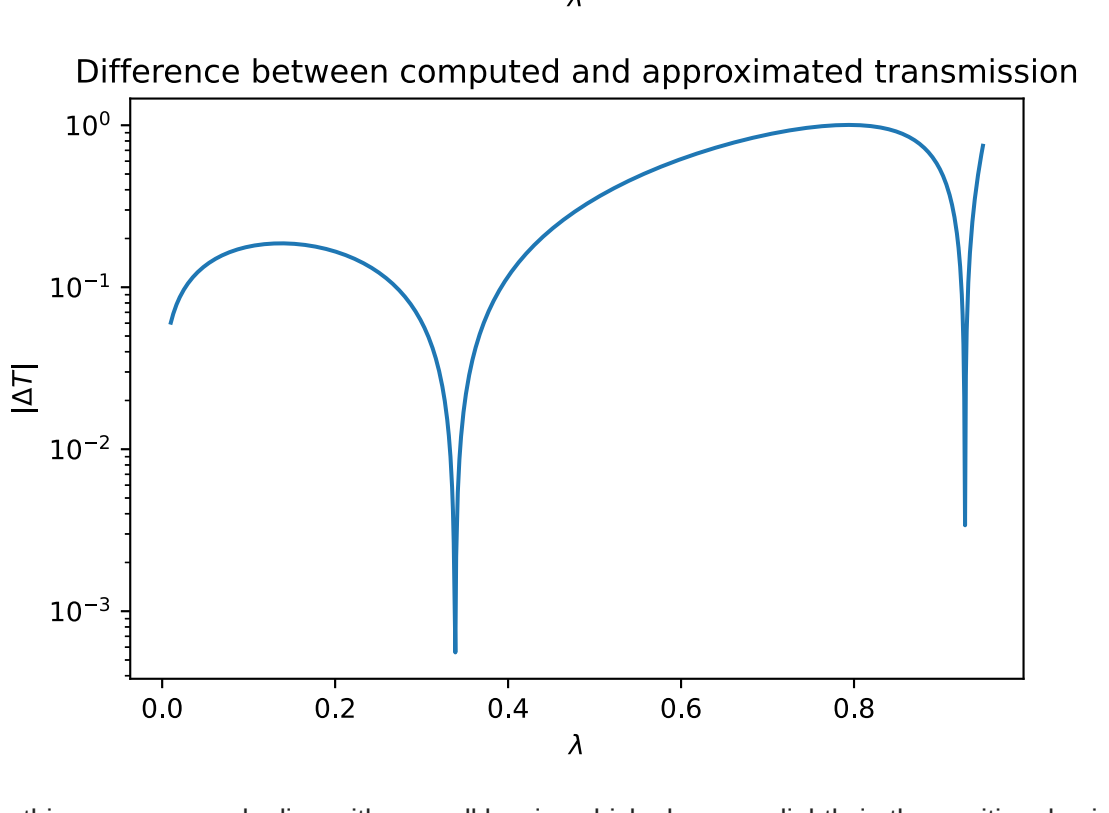
Because of the increased change in the barrier profile, the accuracy of the WKB approximation has taken an overall hit. For small energies $\lambda \leq 1$ the accuracy is still reasonable ($\leq 10^{-3}$). For larger energies however, the approximation starts deviating more and more from the expected value. The WKB approximation is functions if an ballpark estimate for the transmission coefficient needs to be determined. With this barrier profile then, it still can prove useful.

c)

We again modify the code to obtain the results found below.

```
In [12]: V1 = 1
         a = 1
         dV = 0.05
         epsilon = dV / a

         lambda_ = np.linspace(0.01, V1-dV, 1000, endpoint=False)

         T = []
         Twkb = []

         for l in lambda_:
             pot = Potential(-0.5*a, 0.5*a, 100, \
                             expr = lambda y: V1 - epsilon * (y + 0.5*a), \
                             outside_range = (lambda _: 0, lambda _: -epsilon * a))
             sm = ScatterMatrix(pot, l)
             sm.midpoint = True

             T.append(sm.transmission())
             Twkb.append(sm.T_wkb(lambda lam: (16*lam*(V1-dV-lam))/V1**2))

         plt.semilogy(lambda_, T, label="Computed transmission")
         plt.semilogy(lambda_, Twkb, label="WKB approximation")
         plt.xlabel(r"$\lambda$")
         plt.ylabel("Transmission Coefficient")
         plt.title("Computed and approximated transmission for distorted barrier")
         plt.legend()

         plt.figure()
         plt.semilogy(lambda_, np.abs(np.array(T)-np.array(Twkb)))
         plt.xlabel(r"$\lambda$")
         plt.ylabel(r"$|\Delta T|$")
         plt.title("Difference between computed and approximated transmission");
```





In this case we are dealing with a small barrier which changes slightly in the positional axis. For this scenario the WKB approximation can be considered invalid. The computed transmission curve does not resemble the WKB approximation curve in the slightest, and the two small difference peaks are merely a side effect of the usage of absolute values. The WKB approximation is also unable to provide a ballpark estimate for the transmission coefficient. At best, it would be able to at least give a correct order approximation, but given its extreme inaccuracy I'd recommend not using it at all in this case.

d)

From the three experiments found above, we can determine a region for which the WKB approximation is valid. The WKB approximation falls apart when the potential is relatively small or when the energy λ becomes too large. The accuracy of the WKB approximation also decreases when the potential (profile) change increases.

Thus, the WKB approximation is valid and most accurate when $\lambda \leq V_1 - \Delta V$, $\Delta V \ll V_1$ and(!) $V_1 \gg 1$.

Section 4: MIM Diode

a)

The build-in potential ΔV_{BI} is given by $\Delta V_{BI} = \Phi_2 - \Phi_1$. Given $\Phi_1 = 0, 4eV$ and $\Phi_2 = 1, 8eV$, we obtain $\Delta V_{BI} = 1, 4eV$.

In order to determine the scaling constant V_0, we will use the follwing expression:

$$x_0 = \frac{\hbar}{\sqrt{2mV_0}}. \tag{2}$$

Given that $x_0 = 1\text{nm}$ and m is the electron mass, we can solve for V_0. Continuing from the equation above,

$$x_0^2 = \frac{\hbar^2}{2mV_0},$$
$$V_0 = \frac{\hbar^2}{2mx_0^2}.$$

Filling in,

$$V_0 = 6,104... \cdot 10^{-21} J = 3,8010... \cdot 10^{-2} eV \tag{3}$$

b)

The current calculations have been implemented according to LN sec 6.3. This includes the changing of integration boundaries depending on the bias and Fermi level. The resulting I,U-diagram can be found below.

In [13]:
```
I = lambda T, E: -c.e / (c.pi * c.value("reduced Planck constant in eV s")) * np.trapz

x0 = 1E-9
V0 = 3.8010E-2

x = 3E-9
psi = (1.1, 1.1)

Ef = 1 / V0
a = x / x0

IT = []
biases = np.linspace(-2/V0, 2/V0, 100)

for bias in biases:
    T = []

    V = [Ef + i / V0 for i in psi]
    dV = V[1] - V[0] - bias
    epsilon = dV / a

    E = np.linspace(max(Ef-dV, 1E-9), -dV if -dV > Ef else Ef, 50)
    pot = Potential(-0.5*a, 0.5*a, 100, \
                    expr = lambda y: V[0] - epsilon * (y + 0.5*a), \
                    outside_range = (lambda _: 0, lambda _: -epsilon * a))
    for lambda_ in E:
        sm = ScatterMatrix(pot, lambda_)
        T.append(sm.transmission())

    IT.append(I(T, E))

plt.semilogy(biases*V0, np.abs(IT)/1E-9)
plt.xlabel("U (V)")
plt.ylabel("I (nA)");
```

The resulting plot looks similar to the IV-characteristic of a non-polar diode of a few atomic layers thick. This is to be expected, since these were the parameters given to the simulation. The behaviour is also consistent with the behaviour of a diode. At low voltages, the potential on either side of the barrier is nearly identical. This discourages electrons to tunnel through the barrier, resulting in a low current. When the bias increases, there is potential energy for the electrons to gain. Thus, the electrons are far more likely to tunnel, resulting in increased current. Of note are the small currents, but these are to be expected given the relatively thick barrier.

c)

The code from 4b) has been taken and slightly altered. The results can be found below.

In [14]:
```
I = lambda T, E: -c.e / (c.pi * c.value("reduced Planck constant in eV s")) * np.trapz

x0 = 1E-9
V0 = 3.8010E-2

x = 3E-9
psi = (0.4, 1.8)

Ef = 1 / V0
a = x / x0

IT = []
biases = np.linspace(-2/V0, 2/V0, 200)

for bias in biases:
    T = []

    V = [Ef + i / V0 for i in psi]
    dV = bias
    epsilon = (V[0] - V[1] + bias) / a

    E = np.linspace(max(Ef-dV, 0.001), -dV if -dV > Ef else Ef, 50)
    pot = Potential(-0.5*a, 0.5*a, 100, \
                    expr = lambda y: V[0] - epsilon * (y + 0.5*a), \
                    outside_range = (lambda _: 0, lambda _: -dV))
    for lambda_ in E:
        sm = ScatterMatrix(pot, lambda_)
        T.append(sm.transmission())

    IT.append(I(T, E))

plt.semilogy(biases*V0, np.abs(IT)/1E-9)
plt.xlabel("U (V)")
plt.ylabel("I (nA)");
```

Compared to the results of 4b), we can see that the two legs of this plot are not similar in shape. In the forward bias regime, the electrons can tunnel similarly to the simulation in 4b). The barrier decreases and the electrons can tunnel more easily. This continues until the barrier depth does not decrease much further as seen from an electron. At that point the increase in current slows down too. In the reverse bias regime however, the potential on the right hand side of the barrier will surpass the Fermi level of the left-hand side metal. This makes transmission from the left to the right incredibly difficult, reducing the the transmission coefficient. Electrons from the right hand side need to do most, if not all, of the heavy lifting.

The device seems to function like a diode, since the increase in current in the negative bias regime is slower than of the forward regime.

d)

In [15]:
```
J = lambda T1, T2, E1, E2, pm: -c.e**3 * c.m_e / (c.pi**2 * c.hbar**3) * (np.trapz(T1,

x0 = 1E-9
V0 = 3.8010E-2

x = 3E-9
psi = (0.4, 1.8)

Ef = 1 / V0
a = x / x0

JT = []
biases = np.linspace(-2/V0, 2/V0, 100)

for bias in biases:
    T1 = []
    T2 = []

    V = [Ef + i / V0 for i in psi]
    dV = bias
    epsilon = (V[0] - V[1] + bias) / a

    if dV >= 0:
        E1 = np.linspace(0.001, 0.001 if dV > Ef else Ef-dV, 50)
        E2 = np.linspace(0.001 if dV > Ef else Ef-dV, Ef, 50)
    else:
        E1 = np.linspace(-dV, -dV if -dV > Ef else Ef, 50)
        E2 = np.linspace(-dV if -dV > Ef else Ef, Ef-dV, 50)

    pot = Potential(-0.5*a, 0.5*a, 100, \
                    expr = lambda y: V[0] - epsilon * (y + 0.5*a), \
                    outside_range = (lambda _: 0, lambda _: -dV))

    for lambda1, lambda2 in zip(E1, E2):
        sm1 = ScatterMatrix(pot, lambda1)
        sm2 = ScatterMatrix(pot, lambda2)
        T1.append(V0*dV*sm1.transmission())
        T2.append((Ef - lambda2 if dV >= 0 else Ef - dV - lambda2) * V0 * sm2.transmis

    JT.append(J(T1, T2, E1, E2, np.sign(dV)))

plt.semilogy(biases*V0, np.abs(JT)*1E-12)
plt.xlabel("U (V)")
plt.ylabel("I (A)");
```

Apart from a higher current due to the larger surface area, the shape of the IV-characteristic remains identical to the 1D case. Hence, the same conclusions as for the 1D case apply here.

Section 5: Resonant-tunneling diode (RTD)

a)

We can use the same expressions for computing the scaling constants as we have done in 4a). So,

$$V_0 = \frac{\hbar^2}{2mx_0^2}.$$

We have $x_0 = 1\ \mathrm{nm}$, $m = 0.067m_0$. Filling this in,

$$\begin{aligned} V_0 &= \frac{(1.055 \cdot 10^{-34})^2}{2 \cdot 0.067 \cdot 9.109 \cdot 10^{-31} \cdot (10^{-9})^2} \\ &= 9.119 \cdot 10^{-20}\ \mathrm{J} \\ &= 0.567\ \mathrm{eV} \end{aligned}$$

b)

The code and the resulting transmission plot can be found below. The quasi-bound states emerge at the local transmission maxima. Here, the electrons are able to easily penetrate the first barrier, such that they are trapped in the square well. The energies at which this occurs are $0.105\mathrm{eV}$, $0.405\mathrm{eV}$ and $0.881\mathrm{eV}$.

In [16]:
```
from scipy.signal import find_peaks

V0 = 0.567
V1 = 0.5/V0
Ef = 0.005/V0
E = np.linspace(0.01/V0, 1.5/V0, 1000)

T = np.zeros(E.shape)

pot = Potential(0, 9, 250, \
    expr = lambda y: 0 if y >= 2 and y <= 7 else V1, \
    outside_range = (lambda _: Ef, lambda _: Ef))

for i, lambda_ in enumerate(E):
    sm = ScatterMatrix(pot, lambda_)
    T[i] = sm.transmission()

peaks, _  = find_peaks(T, height=0.8)

plt.plot(E*V0, T)
plt.xlabel("Electron energy (eV)")
plt.ylabel("Transmission")

bound_energy = V0 * E[peaks]

str = np.array2string(bound_energy, precision=3, separator="eV, ")
print(f"Bound states at E = {str[1:-1]}eV")
```

```
Bound states at E = 0.105eV, 0.405eV, 0.881eV
```

In order to produce the wave functions corresponding to these energies, we approximate the inside well as an infinite sqaure well. This isn't a very accuracte approximation, as the barrier is not infinite, but it does provide some basic insight.

For an infinite square well, we have

$$E = \frac{\hbar^2 k^2}{2m},$$

so

$$k = \frac{\sqrt{2mE}}{\hbar}.$$

For the wave function, we have

$$\psi = \sqrt{\frac{2}{L}}\sin(kx).$$

The wave functions are plotted below.

In [17]:
```
L = 5
x = np.linspace(-L, L, 100)

plt.figure()
E = bound_energy
plt.plot(x, np.sqrt(2/L)*np.cos(np.sqrt(2*0.067*c.m_e*E[0]*c.e) / c.hbar * x * 1e-9),
plt.plot(x, np.sqrt(2/L)*np.sin(np.sqrt(2*0.067*c.m_e*E[1]*c.e) / c.hbar * x * 1e-9),
plt.plot(x, np.sqrt(2/L)*np.cos(np.sqrt(2*0.067*c.m_e*E[2]*c.e) / c.hbar * x * 1e-9),
plt.vlines([-L/2 - 2, -L/2, L/2, L/2 + 2], -0.7, 0.7, colors="black", label="Barrier"

plt.xlabel("x (nm)")
plt.ylabel("Wave function (a.u.)")
plt.legend();
```

We see from this plot that the wave functions penetrate the barrier, indicating that indeed the infinite square well approximation is not great.

c)

The code and the resulting I-V curve are found below. From this plot we clearly observe the negative differential resistance (NDR) effect. This happens at the locations where the current goes down for increasing bias. The reason this occurs, is because at specific biases we can have a resonant condition, such that tunneling becomes easy. If we move away from this bias, the tunneling current will decrease as the resonant condition fades.

In [18]:
```
iter = 100
V0 = 0.567
V1 = 0.5/V0
Ef = 0.005/V0

Icalc = lambda T, E: -c.e / (c.pi * c.value("reduced Planck constant in eV s")) * np.t

bias = np.linspace(0.01/V0, 1.5/V0, iter)
I = np.zeros(iter)

for i in range(iter):
    dV = bias[i]
    E = np.linspace(max(Ef-dV, 0.001), -dV if -dV > Ef else Ef, 50)

    def barrier(y):
        slope = 0.5*dV / 2
        if y >= 2 and y <= 7:
            return -0.5*dV
        elif y < 2:
            return V1 - y*slope
        else:
            return V1 - 0.5*dV - (y-7)*slope

    T = np.zeros(E.shape)

    pot = Potential(0, 9, 125, \
        expr = barrier, \
        outside_range = (lambda _: 0, lambda _: -dV))

    for j, lambda_ in enumerate(E):
        sm = ScatterMatrix(pot, lambda_)
        T[j] = sm.transmission()

    I[i] = Icalc(T, E)

plt.semilogy(bias*V0, np.abs(I))
plt.xlabel("Bias (V)")
plt.ylabel("Current (A)");
```