```
import numpy as np
from scipy.constants import hbar
from scipy.special import factorial, eval_hermite
import matplotlib.pyplot as plt
!export PATH=/Library/TeX/texbin:$PATH
```

Midterm exam question 1j

For the sake of visualizing the approximated quantum harmonic oscillator, I will use normalized units. To find these I start with x, as I want the turning points to be at ± 1. This gives the unitless $\hat{x} = \sqrt{\frac{k}{2E}}\, x$. The momentum p is now $p = \sqrt{2mE\left(1-\hat{x}^2\right)} = \sqrt{2m\hbar\omega\left(n-1/2\right)\left(1-\hat{x}^2\right)}$, where E is the constant total energy, and m is the mass. The next step is dividing p by $\sqrt{\hbar\omega m}$, and defining the unitless energy $\varepsilon = \frac{E}{\hbar\omega}$, which gives the unitless $\hat{p} = \sqrt{2\varepsilon\left(1-\hat{x}^2\right)}$.

The integral in the exp/sin fucntions change as the varibale x is substituted, and removes the factor of $\frac{1}{\hbar}$ to give the unitless

$$\psi\left(\frac{\hbar}{m\omega}\right)^{\frac{1}{4}} = \frac{A}{\sqrt{\hat{p}}}\sin\left(2\varepsilon\int_{-1}^{\hat{x}}\sqrt{1-x^2}\,dx + \frac{\pi}{4}\right).$$

The extra factor of $\sqrt{\hbar}$ is absorbed into the normalisation constant in order to make psi unitless. We see that this is necessary when doing the same with the actual harmonic oscillator eigenfunctions. When $\hat{x}$ is bigger/smaller than ± 1, the function is the same, except for sin $\rightarrow$ exp, and the factor of $\frac{\pi}{4}$ in the argument dissapears, and you use the absolute value of p.

```
def harmonic_oscillator(n, x):
    """return the normal quantum (unitless) quantum harmonic
oscillator"""
    E = n+1/2
    term1 = 1/np.sqrt(2**n*factorial(n))
    term2 = 1/np.pi**(1/4)
    term3 = np.exp(-(n+1/2)*x**2)
    term4 = eval_hermite(n, np.sqrt(2*(n+1/2))*x)
    return term1*term3*term4

def p_int(lim, E, N):
    x_arr = np.linspace(lim[0],lim[1],N)
    p_arr = 2*E*np.sqrt(abs(1-x_arr**2))
    return np.trapezoid(p_arr,x_arr)

def wkb_oscillator_singlex(n,x, N=1000):
    """Return the unitless approximated quantum harmonic
       oscillator for a single x-value"""
```

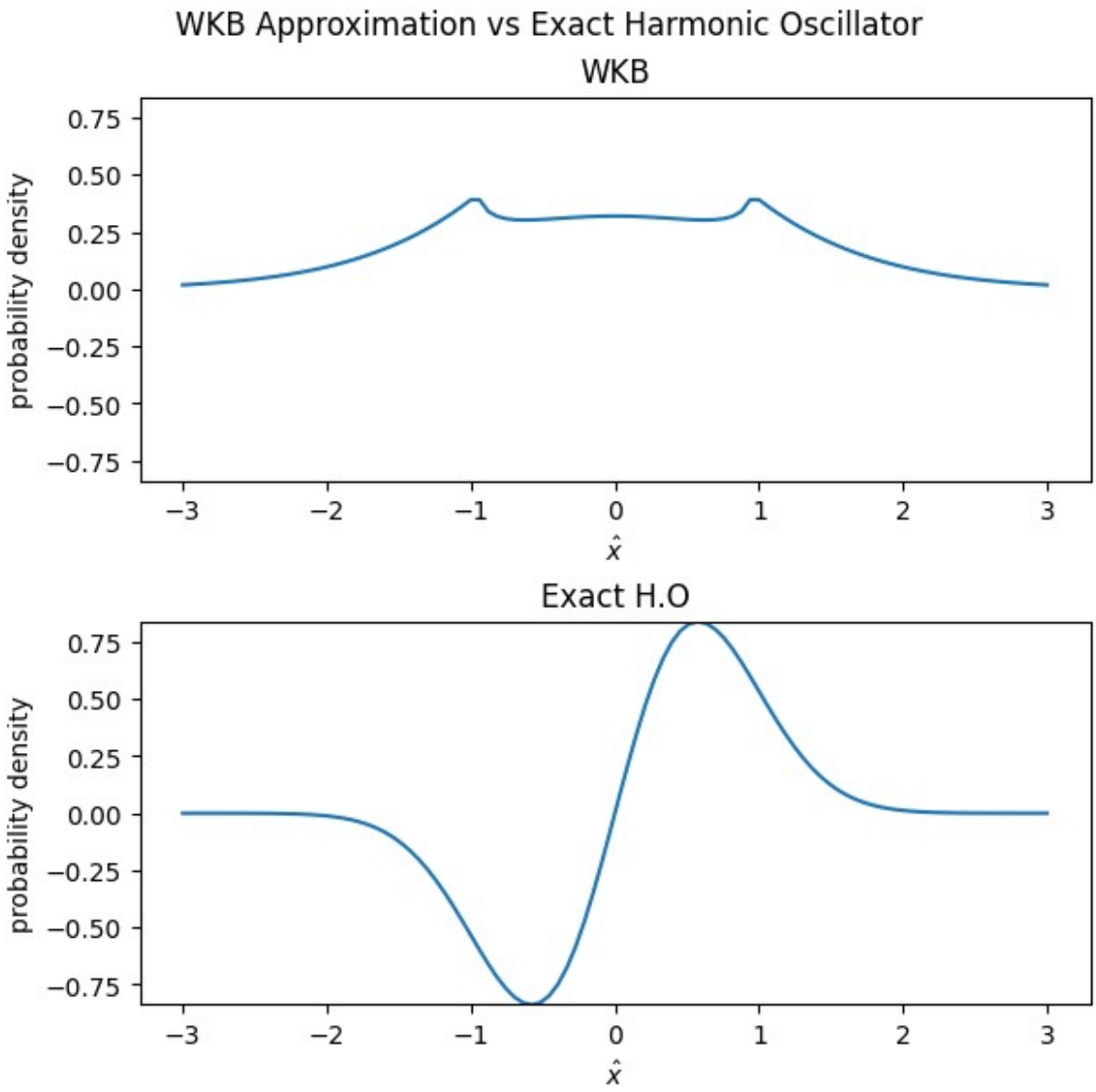
```
    eps = n-1/2
    p0 = np.sqrt(2*eps*(1-x**2))

    return 1/np.sqrt(p0)*np.sin(p_int([-1,x],eps,N) + np.pi/4)
    """
    elif x=<-0.95:
        eps = n-1/2
        p0 = np.sqrt(2*eps*abs((1-x**2)))

        return 1/np.sqrt(p0)*np.exp(-p_int([x,-1],eps,N))

    elif x>=0.95:
        eps = n-1/2
        p0 = np.sqrt(2*eps*abs((1-x**2)))
    """



        #return 1/np.sqrt(p0)*np.exp(-p_int([1,x],eps,N))
def Airy(z):
    """return the value of the airy function"""
    if z<0:
        return 1/2/np.sqrt(np.pi)/(-z)**(1/4)*np.exp(-2/3*(-z)**(3/2))
    elif z>0:
        return 1/2/np.sqrt(np.pi)/(z)**(1/4)*np.exp(-2/3*(z)**(3/2))

def wkb_oscillator(n,x, N=1000):
    """return an array of the unitless quantum harmonic oscillator"""
    psis = np.empty_like(x)

    #Make sure the airy and psi function overlapp at the regions of
interest
    checked1 = None
    overlapp1 = None
    checked2 = None
    overlapp2 = None

    for i, x_ in enumerate(x):
        if x_<-0.95:
            psis[i] = Airy(x_)
        elif x_>=-0.95 and x_<=0.95:
            psis[i] = wkb_oscillator_singlex(n,x_, N)
            if not checked1:
                checked1 = psis[i]
                overlapp1 = psis[i-1]
        elif x_>0.95:
            psis[i] = Airy(x_)
            if not checked2:
                checked2 = psis[i-1]
                overlapp2 = psis[i]
```

```
    #another for loop to fix the equal-condition on psi-and-airy
funciton
    for i, x_ in enumerate(x):
        if x_<-0.95:
            psis[i] = psis[i]*checked1/overlapp1
        elif x_>0.95:
            psis[i] = psis[i]*checked2/overlapp2


    return psis

N = 100
xs = np.linspace(-3,3,N)
n=1
psis = wkb_oscillator(n,xs, N)
print(np.trapezoid(psis**2,xs))
psis = psis/np.trapezoid(psis**2,xs)
harm = harmonic_oscillator(n,xs)
harm = harm/np.trapezoid(harm**2,xs)
```

```
3.1314237561967433
```

```
fig, ax = plt.subplots(2,1, figsize=(6,6), constrained_layout=True)
#fig.tight_layout(pad=3.0)
limy = [np.min(harm),np.max(harm)]
fig.suptitle('WKB Approximation vs Exact Harmonic Oscillator')
ax[0].plot(xs,psis)
ax[0].set_ylim(limy[0],limy[1])
ax[0].set_title('WKB')
ax[0].set_ylabel('probability density')
ax[0].set_xlabel(r'$\hat{x}$')
ax[1].plot(xs,harm)
ax[1].set_ylim(limy[0],limy[1])
ax[1].set_title('Exact H.O')
ax[1].set_ylabel('probability density')
ax[1].set_xlabel(r'$\hat{x}$')
```

```
Text(0.5, 0, '$\\hat{x}$')
```



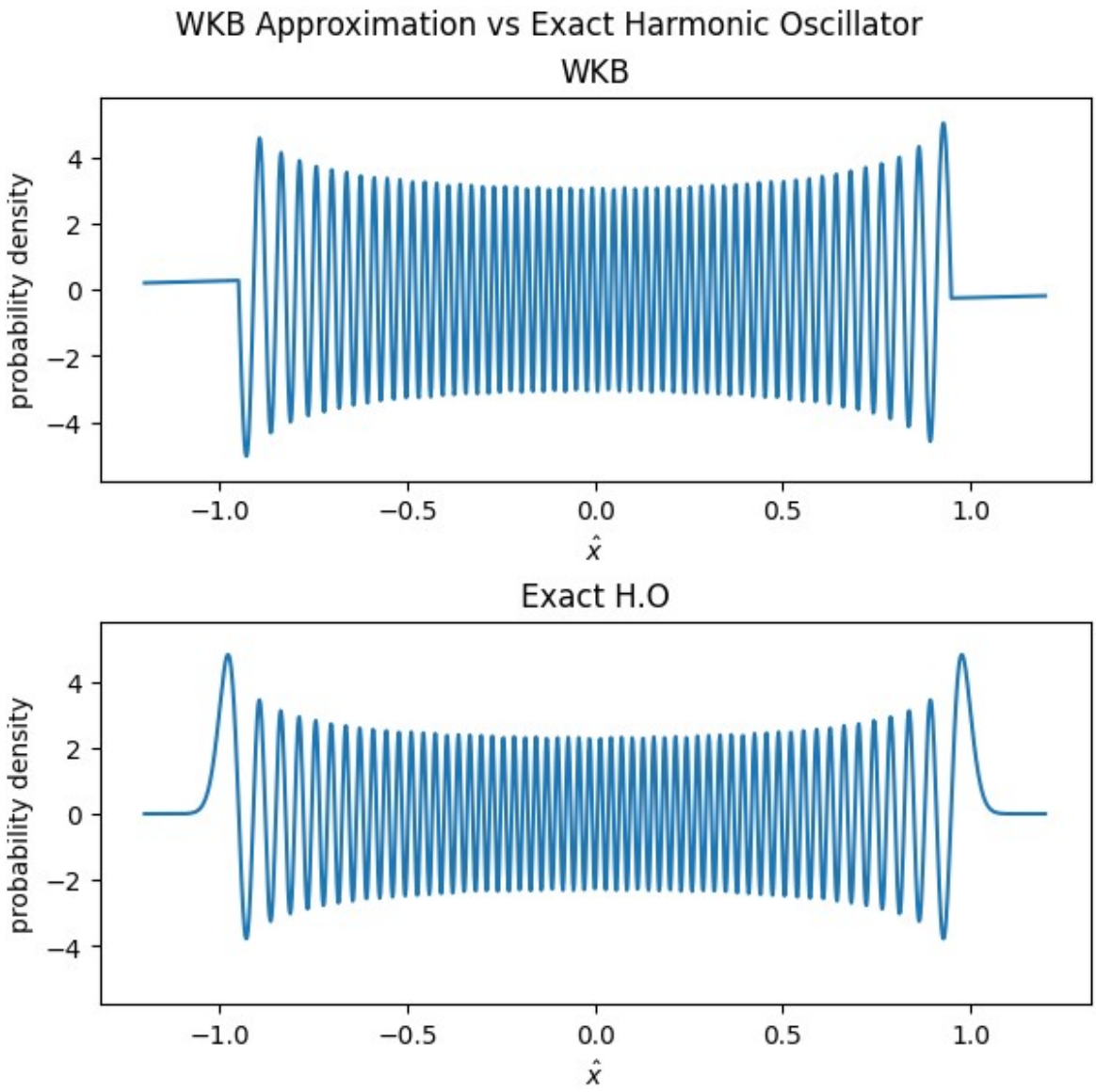
Wee see that the approximation is quite poor for low energy levels.

```
N = 1200
xs = np.linspace(-1.2,1.2,N)
n=110
psis = wkb_oscillator(n,xs, N)
psis = psis/np.trapezoid(psis**2,xs)
harm = harmonic_oscillator(n,xs)
print(np.trapezoid(harm,xs))
harm = harm/np.trapezoid(harm**2,xs)

fig, ax = plt.subplots(2,1, figsize=(6,6), constrained_layout=True)
#fig.tight_layout(pad=3.0)
limy = [np.min(harm),np.max(harm)]
```

```
fig.suptitle('WKB Approximation vs Exact Harmonic Oscillator')
ax[0].plot(xs,psis)
ax[0].set_ylim(limy[0]-2,limy[1]+1)
ax[0].set_title('WKB')
ax[0].set_ylabel('probability density')
ax[0].set_xlabel(r'$\hat{x}$')
ax[1].plot(xs,harm)
ax[1].set_ylim(limy[0]-2,limy[1]+1)
ax[1].set_title('Exact H.O')
ax[1].set_ylabel('probability density')
ax[1].set_xlabel(r'$\hat{x}$')
```

```
0.046453896398056437
```

```
Text(0.5, 0, '$\\hat{x}$')
```



For much larger n, wee see that the approximation is much better. It would be even more similar at the ends if I were to use the regular psi-function (not airy) beyond the turning point as well.