HW2

December 6, 2022

1 Quantum Dynamics

Alfaifi, Ammar

KFUPM, Department of Physics

1.1 Time-Dependent Schrödinger Equation

We start off with solving the time-dependent schrödinger equation by discretizing the dependent variable x, so we instead obtain a matrix equation as following

$$H^{N\times N}\Psi^{N\times 1} = i\hbar\frac{d}{dt}\Psi^{N\times 1}$$

where the superscripts are the matrix dimension.

Then the solution to this equation can be written as

$$\vec{\Psi}(t+\Delta t) = \mathsf{U}(\Delta t)\,\vec{\Psi}(t) \tag{1}$$

Now if the Hamiltonian matrix H is time-independent, the exact expression for the time-evolution operator is

$$\mathsf{U}(\Delta t) = e^{-i\mathsf{H}\Delta t/\hbar} \tag{2}$$

But for small Δt, the time-evolution operator can be approximated with Taylor expansion of e as

$$\mathsf{U}(\Delta t) \approx 1 - i\mathsf{H}\,\frac{\Delta t}{\hbar}$$

or more preferably with Cayley's form,

$$\mathsf{U}(\Delta t) \approx \frac{1 - \frac{1}{2}i\frac{\Delta t}{\hbar}\mathsf{H}}{1 + \frac{1}{2}i\frac{\Delta t}{\hbar}\mathsf{H}} \tag{3}$$

And substituting this into the solution of the the TDSE, we get

$$\left(1 + \frac{1}{2}i\frac{\Delta t}{\hbar}\mathsf{H}\right)\vec{\Psi}(t+\Delta t) = \left(1 - \frac{1}{2}i\frac{\Delta t}{\hbar}\mathsf{H}\right)\vec{\Psi}(t) \tag{4}$$

Now we need to check that the Eq. (3) is accurate to the second order. We expand Eqs. (1) & (3) as power series in Δt, for the first one we have

$$\mathsf{U}(\Delta t) = 1 - i\mathsf{H}\,\frac{\Delta t}{\hbar} + \frac{1}{2}\left(i\mathsf{H}\,\frac{\Delta t}{\hbar}\right)^2 - \ldots$$

and for the (3)

$$\mathsf{U}(\Delta t) \approx \frac{1-\frac{1}{2}i\frac{\Delta t}{\hbar}\mathsf{H}}{1+\frac{1}{2}i\frac{\Delta t}{\hbar}\mathsf{H}} = \left(1-\frac{i}{2}\frac{\Delta t\mathsf{H}}{\hbar}\right)\left(1-\frac{i}{2}\frac{\Delta t\mathsf{H}}{\hbar}-\frac{1}{2}\left(\frac{\Delta t\mathsf{H}}{\hbar}\right)^2+\dots\right) = 1-i\frac{\Delta t\mathsf{H}}{\hbar}-\frac{1}{2}\left(\frac{\Delta t\mathsf{H}}{\hbar}\right)^2+\dots$$

which agrees up to the Δt^2

Now to check that U is unitary matrix, by definition, we require that $\mathsf{U}\mathsf{U}^\dagger = 1$

$$\mathsf{U}\mathsf{U}^\dagger = \left(\frac{1-\frac{1}{2}i\frac{\Delta t}{\hbar}\mathsf{H}}{1+\frac{1}{2}i\frac{\Delta t}{\hbar}\mathsf{H}}\right)\left(\frac{1+\frac{1}{2}i\frac{\Delta t}{\hbar}\mathsf{H}}{1-\frac{1}{2}i\frac{\Delta t}{\hbar}\mathsf{H}}\right) = 1$$

1.2 Construct The Hamiltonian Matrix

We construct the the Hamiltonian matrix H for $N+1 = 100$ spatial grid points, and the spatial boundaries with dimensionless length of $\xi = \pm 10$. With setting, $m = 1$, $\omega = 1$, and $\hbar = 1$.

1.3 Lowest Two Eigenvalues of H

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from scipy.linalg import eigh_tridiagonal, solve
from scipy.special import hermite
from scipy.integrate import quad
from scipy.misc import derivative

from matplotlib_inline.backend_inline import set_matplotlib_formats

set_matplotlib_formats('pdf', 'svg')
plt.style.use('seaborn')
plt.rcParams |= {
    'text.usetex': True,
    'figure.figsize': (14, 5)
}
```

[ ]:
```
m = omega = hbar = 1
Nx = 100
xmin = -10
xmax = 10
dx = (xmax - xmin) / (Nx + 1)
Nt = 100
dt = 4 * np.pi / omega / Nt

# The second derivative operator
D2 = (
    np.diag([-2] * Nx)
```

```python
    + np.diag(np.ones(Nx - 1), 1)
    + np.diag(np.ones(Nx - 1), -1)
) / dx**2

# The position operator
X = np.diag(np.arange(xmin, xmax - dx, dx))

# The Hamiltonian
H = -0.5 / m * D2 + 0.5 * m * omega**2 * X**2

# diagonal elements of H
d = H.diagonal(0)
# above-diagonal elements of H
e = H.diagonal(1)
# Find the eigenvalues and eigenvectors of H
# This algorithm is much faster than using H entirly
eigenvalues, eigenvectors = eigh_tridiagonal(
    d, e, select="i", select_range=(0, 1)
)

# check if the first eigenvector is normalized, which is by default
np.round(np.dot(eigenvectors[:, 0], eigenvectors[:, 0]), 10)
```

[ ]: 1.0

The exact eigenvalues for the harmonic oscillator are,

$$E_n = \left(n + \frac{1}{2}\right) \hbar\omega$$

In the above units, we have $E_0 = 0.5$ and $E_1 = 1.5$

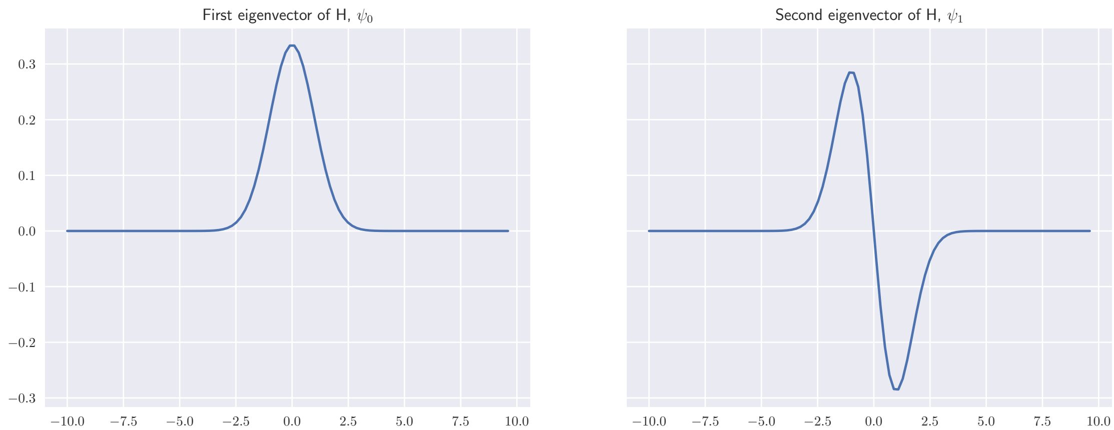
[ ]:
```python
print(f'The first eigenvalue is {eigenvalues[0]:.3f} with error␣
 ↪{abs(eigenvalues[0] - 0.5) / 0.5 *100 :.2f}%')
print(f'The second eigenvalue is {eigenvalues[1]:.3f} with error␣
 ↪{abs(eigenvalues[1] - 1.5) / 1.5 *100 :.2f}%')
```

```
The first eigenvalue is 0.499 with error 0.25%
The second eigenvalue is 1.494 with error 0.41%
```

1.4 Plot The Eigenvector

[ ]:
```python
fig, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
x = X.diagonal()

ax1.plot(x, eigenvectors[:, 0])
ax2.plot(x, eigenvectors[:, 1])
ax1.set_title('First eigenvector of $\mathsf{H}$, $\psi_0$')
ax2.set_title('Second eigenvector of $\mathsf{H}$, $\psi_1$')
```

```
plt.show()
```



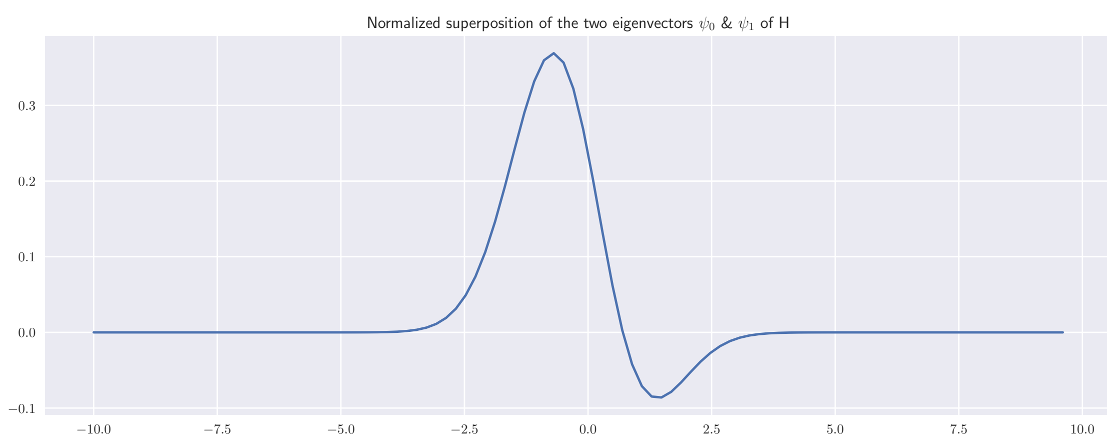
1.5 Evolve The Wave Function

Now we need to show an animation of the evolving wave function using (4) from time $t = 0$ to $t = 4\pi/\omega$. We will take $\Psi(0)$ to be

$$\Psi(0) = \frac{\psi_0 + \psi_1}{\sqrt{2}}$$

Note for the exp approximation to hold, the time steps should be at least $N_t = 100$.

[ ]:
```
# Taking this if time equals 0
Psi0 = (eigenvectors[:, 0] + eigenvectors[:, 1]) / np.sqrt(2)

plt.plot(x, Psi0)
plt.title('Normalized superposition of the two eigenvectors $\psi_0$ \& $\psi_1$ of $\mathsf{H}$')
plt.show()
```



1.5.1 Construct U

From (3) we build the following

[ ]:
```python
# The denominator in Cayley's form
U_plus = np.identity(Nx) + 0.5j * dt / hbar * H
# The numerator in Cayley's form
U_minus = np.identity(Nx) - 0.5j * dt / hbar * H
```

1.5.2 Solve TDSE with Exact Solution

Now we solve numerically the time-dependent Schrödinger equation, that is solving the matrix equation in (4):

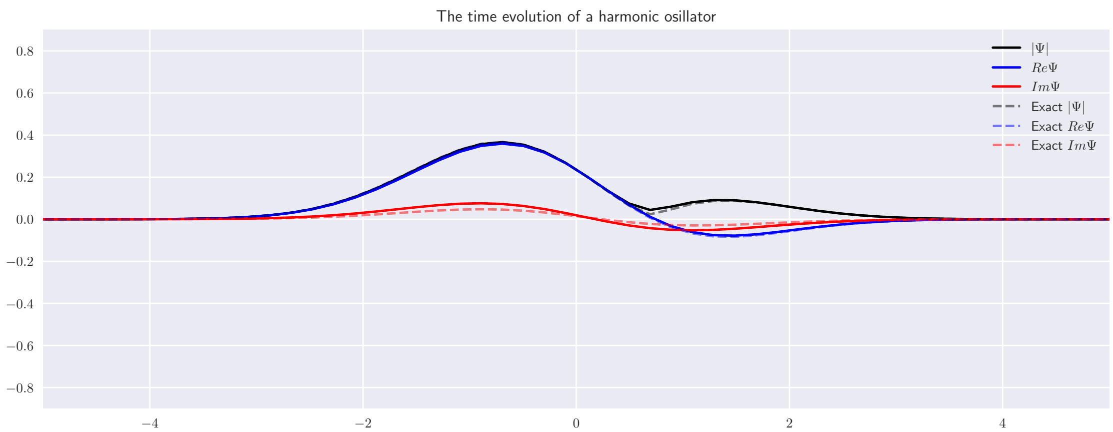
$$\left(1 + \frac{1}{2} i \frac{\Delta t}{\hbar} \mathsf{H}\right) \vec{\Psi}(t + \Delta t) = \left(1 - \frac{1}{2} i \frac{\Delta t}{\hbar} \mathsf{H}\right) \vec{\Psi}(t) \tag{4}$$

We imagine it as in the form of

$$\mathsf{M}\vec{x} = \vec{b}$$

[ ]:
```python
def normalize(vec):
    return vec / np.linalg.norm(vec)

def psi(n, x):
    # The harmonic osillator wavefunction
    return (
        (m * omega / np.pi / hbar) ** 0.25
        * (1 / np.sqrt(2**n * np.prod(np.arange(1, n + 1))))
        * hermite(n)(np.sqrt(m * omega / hbar) * x)
        * np.exp(-0.5 * ((np.sqrt(m * omega / hbar) * x) ** 2))
    )

def Psi(t):
    """
    the exact superposition.
    with our assumption we have to re-normalize psi.
    Minus sign is due to different sign convention in
        numeric/analytic states
    """
    return (
        normalize(psi(0, x)) * np.exp(-0.5j * omega * t)
        - normalize(psi(1, x)) * np.exp(-1.5j * omega * t)
    ) / np.sqrt(2)

fig, ax = plt.subplots()
# numerical ones
(graph1,) = ax.plot([], [], "k", label=r"$|\Psi|$")
```

```
(graph2,) = ax.plot([], [], "b", label=r"$Re{\Psi}$")
(graph3,) = ax.plot([], [], "r", label=r"$Im{\Psi}$")
# exact ones
(graph4,) = ax.plot([], [], "k--", label=r"Exact $|\Psi|$", alpha=0.5)
(graph5,) = ax.plot([], [], "b--", label=r"Exact $Re{\Psi}$", alpha=0.5)
(graph6,) = ax.plot([], [], "r--", label=r"Exact $Im{\Psi}$", alpha=0.5)
ax.set_title("The time evolution of a harmonic oscillator")
ax.set_ylim([-0.9, 0.9])
ax.set_xlim([-5, 5])
ax.legend()

psi_solution = Psi0

def update(i):
    global psi_solution
    new_Psi = Psi(i * dt)
    if i:
        # solve the linear matrix eq
        psi_solution = solve(U_plus, U_minus @ psi_solution)

    graph1.set_data(x, np.abs(psi_solution))
    graph2.set_data(x, np.real(psi_solution))
    graph3.set_data(x, np.imag(psi_solution))
    graph4.set_data(x, np.abs(new_Psi))
    graph5.set_data(x, np.real(new_Psi))
    graph6.set_data(x, np.imag(new_Psi))

    return graph1, graph2, graph3, graph4, graph5, graph6


animation = FuncAnimation(
    fig,
    update,
    frames=Nt,
    blit=True,
)
animation.save("evolve_exact.mp4", fps=50, dpi=150)
```



1.6 Time-Dependent Hamiltonian

Now we need to use the above techneuqes to observe the time evolving of time-dependent Schrödinger equation and where the Hamiltonian *does* depend on time. Instead of (4), we will use the following and evaluating H at the midpoint of each time step

$$\left[1+\frac{1}{2}i\frac{\Delta t}{\hbar}\mathsf{H}\left(t+\frac{\Delta t}{2}\right)\right]\vec{\Psi}(t+\Delta t)=\left[1-\frac{1}{2}i\frac{\Delta t}{\hbar}\mathsf{H}\left(t+\frac{\Delta t}{2}\right)\right]\vec{\Psi}(t)$$

Consider the driving harmonic oscillator

$$f(t)=A\sin\left(\Omega t\right)$$

where $A=1$ is a constant with units of length and Ω is the driving frequency.

[ ]:

```
# The driving frequency value
Omega = omega / 5
Nt = 1000
dt = 2 * np.pi / Omega / Nt
A = 1



def f(t):
    """driving harmonic function"""
    return A * np.sin(Omega * t)



def xc(t):
    return omega * quad(lambda tp: f(tp) * np.sin(omega * (t - tp)), 0, t)[0]



def Psi(n, x, t):
    """The time-dependent nth state."""
```

```python
    exp = np.exp(
        1j
        / hbar
        * (
            -hbar * omega * t * (n + 0.5)
            + m * derivative(xc, t, dx=1e-6) * (x - xc(t) / 2)
            + m * omega**2 / 2 * quad(lambda tp: f(tp) * xc(tp), 0, t)[0]
        )
    )
    return psi(n, x - xc(t)) * exp


def H(t):
    """The time-dep Hamiltonian"""
    return (
        -0.5 * hbar**2 / m * D2
        + 0.5 * m * omega**2 * X**2
        - m * omega**2 * f(t) * X
    )


def U_plus(t):
    """The numerator in Cayley's form"""
    return np.identity(Nx) + 1j * dt / 2 * H(t + dt / 2)


def U_minus(t):
    """The denominator in Cayley's form"""
    return np.identity(Nx) - 1j * dt / 2 * H(t + dt / 2)


# diagonal elements of H(0)
d = H(0).diagonal(0)
# above-diagonal elements of H(0)
e = H(0).diagonal(1)

# Find the eigenvalues and eigenvectors of H at t = 0
eigenvalues, eigenvectors = eigh_tridiagonal(
    d, e, select="i", select_range=(0, 1)
)
# The wavefunction of H at t = 0
Psi0 = eigenvectors[:, 0]
```

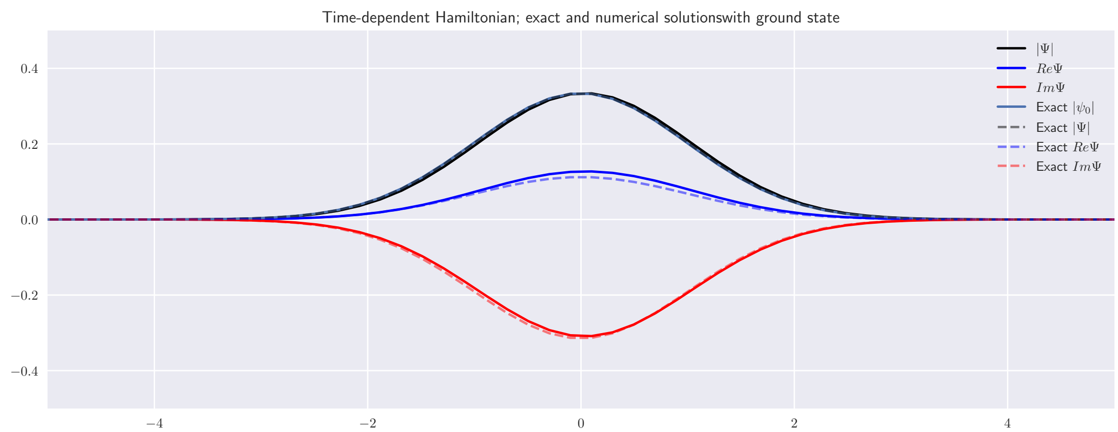
$\Omega = 5\omega$

[ ]:
```python
def animate_time_dep_H():
    fig, ax = plt.subplots()
    # numrical ones
```

```python
    (graph1,) = ax.plot([], [], "k", label=r"$|\Psi|$")
    (graph2,) = ax.plot([], [], "b", label=r"$Re{\Psi}$")
    (graph3,) = ax.plot([], [], "r", label=r"$Im{\Psi}$")
    # instantaneous ground state
    (graph4,) = ax.plot(
        x, np.abs(normalize(psi(0, x - f(0)))), label=r"Exact $|\psi_0|$"
    )
    # exact ones at t = 0
    (graph5,) = ax.plot([], [], "k--", label=r"Exact $|\Psi|$", alpha=0.5)
    (graph6,) = ax.plot([], [], "b--", label=r"Exact $Re{\Psi}$", alpha=0.5)
    (graph7,) = ax.plot([], [], "r--", label=r"Exact $Im{\Psi}$", alpha=0.5)
    ax.set_title(
        "Time-dependent Hamiltonian; exact and numerical solutions "
        "with ground state"
    )
    ax.set_ylim([-0.5, 0.5])
    ax.set_xlim([-5, 5])
    ax.legend()

    # initially it's Psi0
    psi_solution = Psi0

    def update(i):
        nonlocal psi_solution
        print(round(i / Nt * 100), end="\r", flush=True)
        if i:
            # to first plot Psi at t = 0
            # solve the linear matrix eq
            psi_solution = solve(
                U_plus((i - 1) * dt), U_minus((i - 1) * dt) @ psi_solution
            )

        new_Psi = normalize(Psi(0, x, i * dt))
        graph1.set_data(x, np.abs(psi_solution))
        graph2.set_data(x, np.real(psi_solution))
        graph3.set_data(x, np.imag(psi_solution))
        graph4.set_data(x, normalize(psi(0, x - f(i * dt))))
        graph5.set_data(x, np.abs(new_Psi))
        graph6.set_data(x, np.real(new_Psi))
        graph7.set_data(x, np.imag(new_Psi))

        return graph1, graph2, graph3, graph4, graph5, graph6, graph7

    return FuncAnimation(
        fig,
        update,
        frames=Nt,
```

```
        blit=True,
    )


animate_time_dep_H().save("driven_evolve_a.mp4", fps=300, dpi=150)
```

100

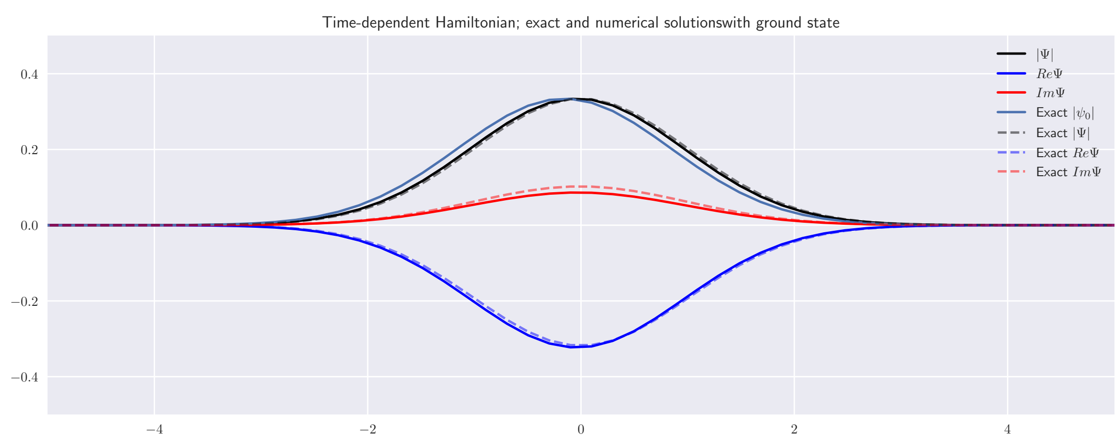


With driving frequency value of $\Omega = \omega/5$, the numerical solution is very close to the instantaneous ground state. Meaning that, this is an adiabatic process. The adiabatic theorem says that if the particle was initially in the nth eigenstate of $\hat{H}(0)$, it will be carried (under Schrödinger equation) into the nth eigenstate of $\hat{H}(T)$. That is what happening above, with small time-dependence in the Hamiltonian.

$\Omega = 5\omega$

[ ]:
```
Omega = 5 * omega
animate_time_dep_H().save("driven_evolve_b.mp4", fps=300, dpi=150)
```

100

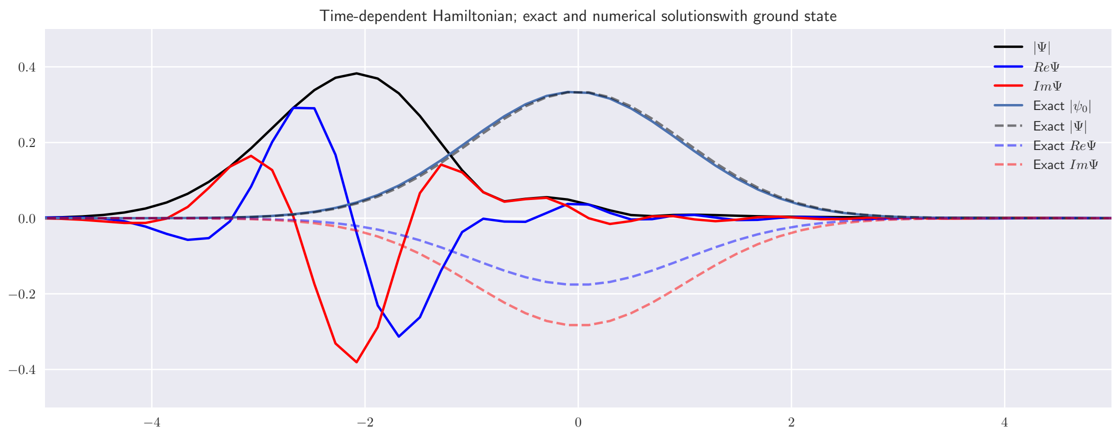


However, with large value of Ω, rapid change in the Hamiltonian, we see that the eigenstate is barely affected by this change, and it is almost in its initial state.

$\Omega = 6\omega/5$

[ ]:
```
Omega = 6/5 * omega
animate_time_dep_H().save("driven_evolve_c.mp4", fps=300, dpi=150)
```

```
100
```



In the above case it is much like an intermediate case of an adiabatic approximation and rapid change.