Code

In [25]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
```

In [26]:
```
import scipy.optimize
def V(x): return 2*x**4-8*x**2
max_x = scipy.optimize.fmin(lambda x: V(x), 0)
```

```
Optimization terminated successfully.
         Current function value: -8.000000
         Iterations: 26
         Function evaluations: 52
```

4th order Runge Kutta method

In [27]:

```
##################################################################################
######################## RK_4th ##################################################
def RK_4th(a, b, h, d2y, ya, d1ya, plot_enabled = True, normalized = False, label = None

    """
    General info:
        This function solves 2nd order differential equation using 4th order
        runge kutta method.
    Arguments:
        a    : lower limit
        b    : higher limit
        h    : interval length (dx)
        d2y  : function handle to 2nd derivative of y
        ya   : value of y   at starting point a
        d1ya : value of d1y at starting point a
    """

    import numpy as np
    import matplotlib.pyplot as plt
    plt.style.use('seaborn-whitegrid')

    def d1y(x, y, z):
        return z

    xpoints = np.arange(a,b,h)
    ypoints = []
    zpoints = []

    y = ya
    z = d1ya

    for x in xpoints:
        ypoints.append(y)
        zpoints.append(d1y)

        k1 = h * d1y(x, y, z)
        l1 = h * d2y(x, y, z)

        k2 = h * d1y(x+0.5*h, y+0.5*k1, z+0.5*l1)
        l2 = h * d2y(x+0.5*h, y+0.5*k1, z+0.5*l1)

        k3 = h * d1y(x+0.5*h, y+0.5*k2, z+0.5*l2)
        l3 = h * d2y(x+0.5*h, y+0.5*k2, z+0.5*l2)

        k4 = h * d1y(x+h, y+k3, z+l3)
        l4 = h * d2y(x+h, y+k3, z+l3)

        y = y + (k1 + 2*k2 + 2*k3 + k4) / 6
```

```python
        z = z + (l1 + 2*l2 + 2*l3 + l4) / 6

    if normalized:

        ypoints_sqr = np.square(ypoints)
        # Calculating integration >>
        f_a = ypoints_sqr[0]
        f_b = ypoints_sqr[-1]
        I_n = (f_a + f_b) / 2

        # Loop for adding n-1 terms >>
        for ypoint_sqr in ypoints_sqr[1:-1]:
            I_n = I_n + ypoint_sqr * abs(h)
        ypoints = ypoints / np.sqrt(I_n)
        y       = y       / np.sqrt(I_n)
        z       = z       / np.sqrt(I_n)

    if plot_enabled == True:

        # Plotting R_average vs N for many trials >>
        fig  = plt.figure(figsize = (8, 5))
        axes = plt.gca()
        if label: axes.plot(xpoints, ypoints, label = label); plt.legend()
        else: axes.plot(xpoints, ypoints)

        # Setting plot elements >>
        axes.set_title("Y (wave-function) vs x")
        axes.set_xlabel("X")
        axes.set_ylabel("Y (wave-function)")
        plt.show()
        return y, z, axes

    else:
        return y, z, None
######################## RK_4th ################################################
################################################################################
```

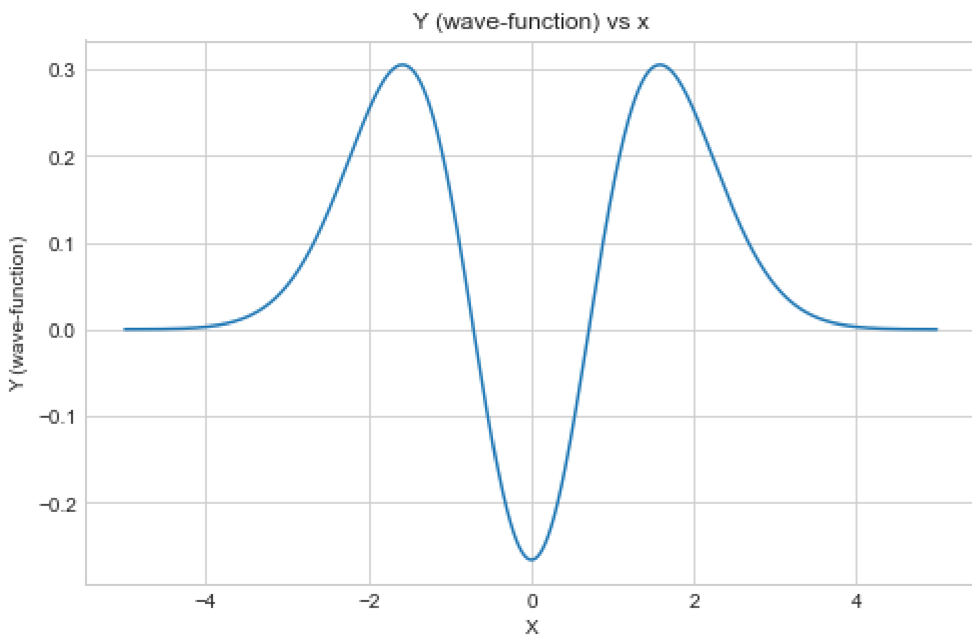
In [28]:
```python
##################################################################################
######################## plot_wavefunc_with_V ##################################
def plot_wavefunc_with_V( func_V,
                          axes_wavefunc,
                          figsize  = (8, 6),
                          rangex_V = (-2.5, 2.5),
                          xlim     = None,
                          ylim     = None
                        ):

    import numpy as np

    # Making new figure for V >>
    fig    = plt.figure(figsize = figsize)
    axes_V = plt.gca()
    xpoints_V = np.linspace(rangex_V[0], rangex_V[1], num = 200)
    ypoints_V = func_V(xpoints_V)
    axes_V.plot(xpoints_V, ypoints_V)

    if type(axes_wavefunc) == list:
        for ax_wavefunc in axes_wavefunc:
            # Adding wave-functions plot to V figure >>
            xydata_wavefunc  = ax_wavefunc.get_lines()[0].get_xydata()
            xpoints_wavefunc = xydata_wavefunc[:, 0]
            ypoints_wavefunc = xydata_wavefunc[:, 1]
            axes_V.plot(xpoints_wavefunc, ypoints_wavefunc)

    else:
        ax_wavefunc = axes_wavefunc
        xydata_wavefunc  = ax_wavefunc.get_lines()[0].get_xydata()
        xpoints_wavefunc = xydata_wavefunc[:, 0]
        ypoints_wavefunc = xydata_wavefunc[:, 1]
        axes_V.plot(xpoints_wavefunc, ypoints_wavefunc)

    # Setting plot elements >>
    axes_V.set_title("Y (wave-function) vs x")
    axes_V.set_xlabel("X")
    axes_V.set_ylabel("Y (wave-function)")
    if xlim: axes_V.set_xlim(xlim)
    if ylim: axes_V.set_ylim(ylim)
    plt.show()

    return None
######################## plot_wavefunc_with_V ##################################
##################################################################################
```

Test RK_4th Simple Harmonic Oscillator

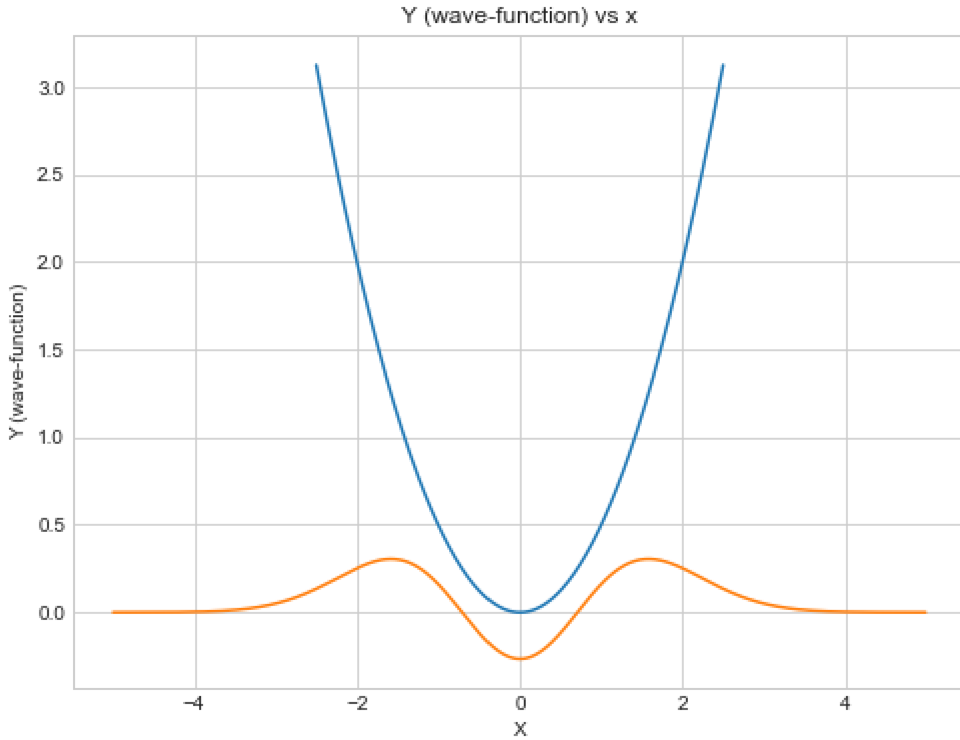
In [29]:
```
def V(x):
    return 1/2*x**2

def d2y(x, y, z):
    n = 2; h = 1; w = 1
    E = (n + 0.5) * h * w
    return 2*(V(x)-E) * y
```

Not normalized

In [30]:
```
_, _, axes_sh = RK_4th( a          = -5,
                        b          =  5,
                        h          = 0.01,
                        d2y        = d2y,
                        ya         = 0.0001,
                        d1ya       = 0,
                        normalized = False )
```

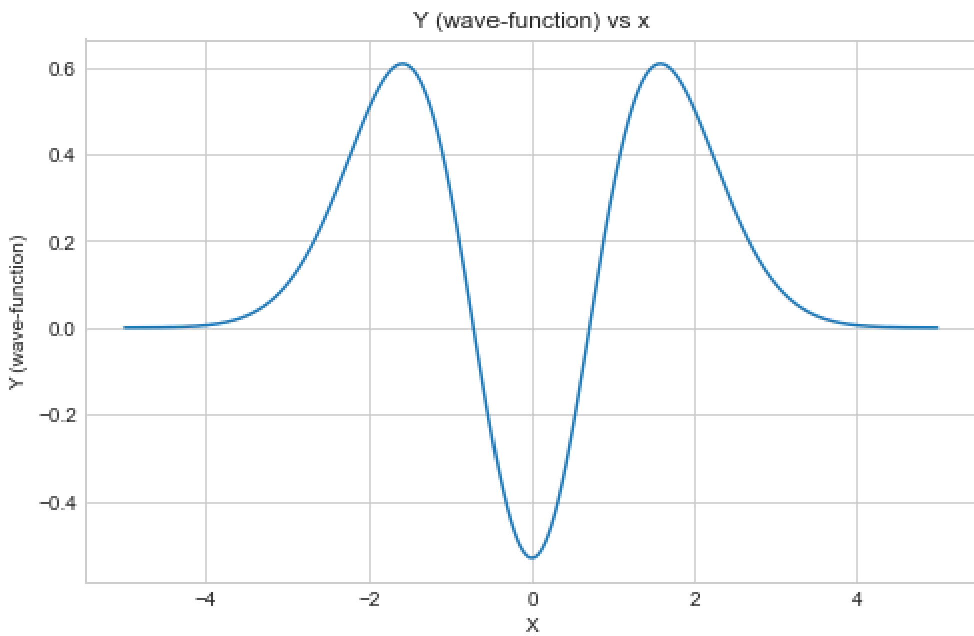


In [31]:
```
plot_wavefunc_with_V( func_V        = V,
                      axes_wavefunc = axes_sh,
                      figsize       = (8, 6),
                      rangex_V      = (-2.5, 2.5) )
```

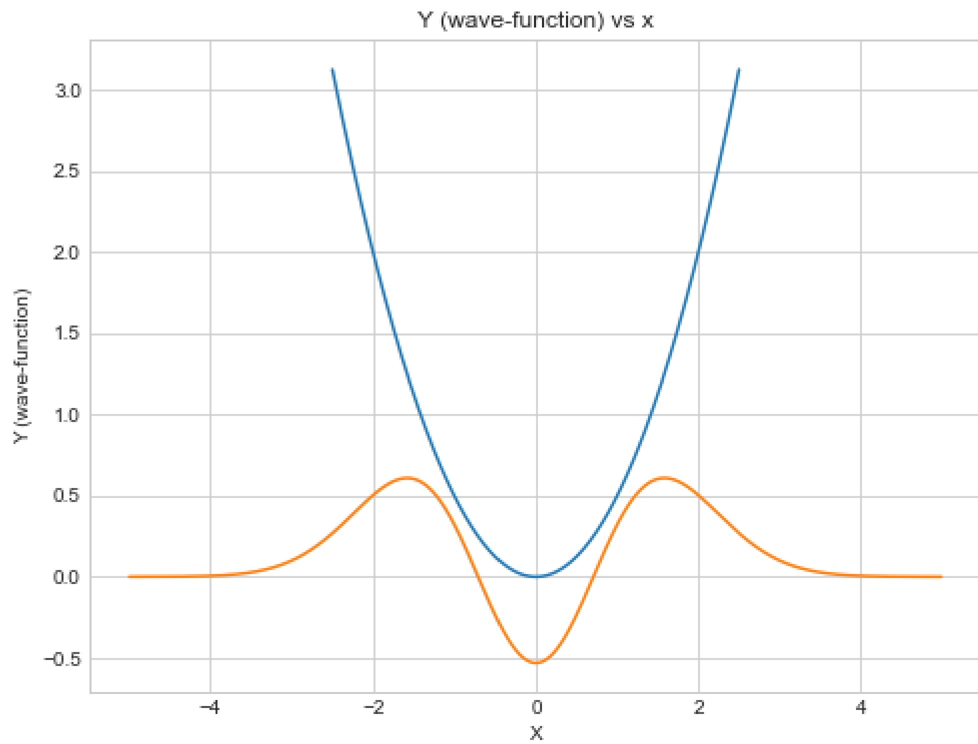


Normalized

In [32]:
```
_, _, axes_sh_norm = RK_4th( a          = -5,
                             b          =  5,
                             h          = 0.01,
                             d2y        = d2y,
                             ya         = 0.0001,
                             d1ya       = 0,
                             normalized = True )
```



In [33]:
```
plot_wavefunc_with_V( func_V        = V,
                      axes_wavefunc = axes_sh_norm,
                      figsize       = (8, 6),
                      rangex_V      = (-2.5, 2.5))
```



Double shooting method using RK_4th

In [34]:

```python
###################################################################################
######################## solve_schrod_RK_4th ####################################
def solve_schrod_RK_4th(a, b, h, d2y, ya, d1ya, El, Eh, dE, xlim = None, ylim = None, ma

    """
    General info:
        This function solves schrodinger's equation using 4th order
        runge kutta method.
    Arguments:
        a    : lower limit of x
        b    : upper limit of x
        h    : interval length (dx)
        d2y  : function handle to 2nd derivative of wave function y
        ya   : value of wave function y at starting point a
        d1ya : value of 1st derivative of wave function y at starting point a
        El   : Lower limit of energy
        Eh   : Upper limit of energy
    """

    import numpy as np
    from progressbar import ProgressBar
    import matplotlib.pyplot as plt
    plt.style.use('seaborn-whitegrid')

    global E
    energies = np.arange(El, Eh, dE)
    axes_sol = []

    diff_y1_y_ratios = []
    eigen_values     = []

    pbar = ProgressBar()
    i = 0
    for E in pbar(energies):
        # i = i + 1

        ya_left = (-1)**(i)*ya
        y_L, z_L, _ = RK_4th( a, (b - (a+b)/2) * 0.05,  h, d2y, ya_left, d1ya, plot_enab
        y_R, z_R, _ = RK_4th( b, (b - (a+b)/2) * 0.05, -h, d2y, ya,      d1ya, plot_enab

        y1_y_ratio = abs(z_L/y_L - z_R/y_R)

        if abs(z_L/y_L - z_R/y_R) < match_ratio:
            print("")
            print("i           =", i)
            print("E           =", E)
            y, _, ax_sol = RK_4th( a, b, h, d2y, ya_left, d1ya, plot_enabled = True, nor
            axes_sol.append(ax_sol)
```

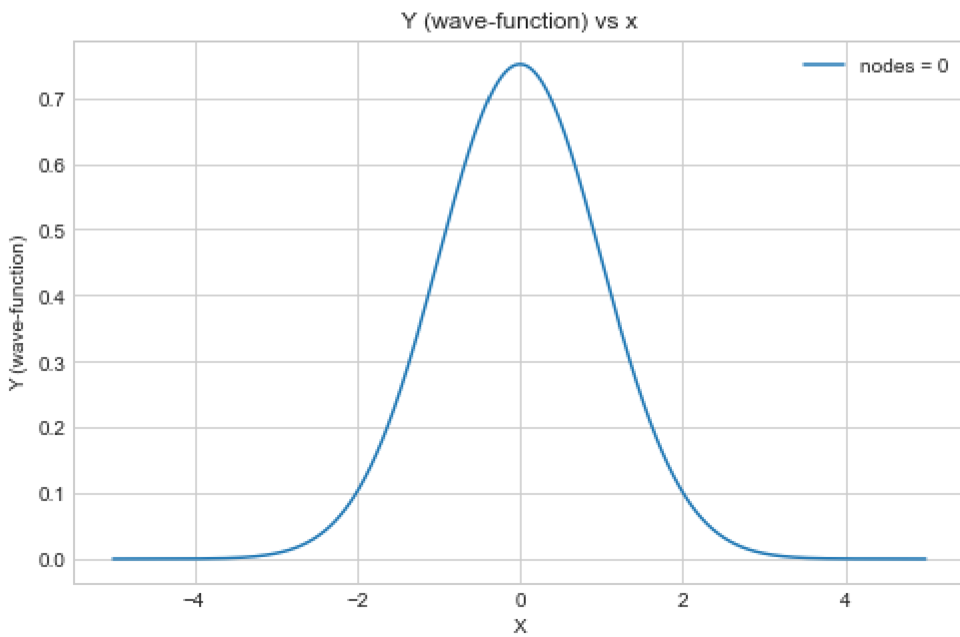
```
            eigen_values.append(E)
            i = i + 1

        diff_y1_y_ratios.append(y1_y_ratio)

    fig  = plt.figure(figsize = (8, 6))
    axes = plt.gca()
    if xlim: axes.set_xlim(xlim)
    if ylim: axes.set_ylim(ylim)
    axes.plot(energies, diff_y1_y_ratios)
    axes.set_title("(y1/y)L - (y1/y)R   vs   Eigen-energies")
    axes.set_xlabel("Eigen-energies")
    axes.set_ylabel("(y1/y)L - (y1/y)R")

    return eigen_values, axes_sol
######################## solve_schrod_RK_4th ###################################
################################################################################
```

Testing double shooting method on SHO

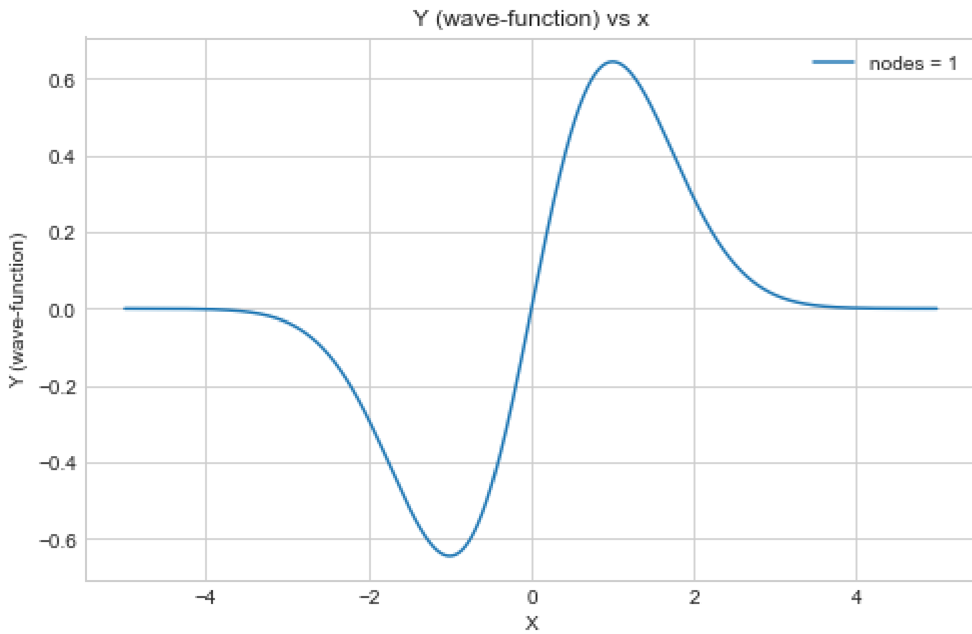
In [35]:
```
def V(x):
    return 1/2*x**2

def d2y(x, y, z):
    return 2*(V(x)-E) * y
```

In [36]:
```
eigen_values, axes_sh = solve_schrod_RK_4th( a    = -5,
                                             b    =  5,
                                             h    = 0.01,
                                             d2y  = d2y,
                                             ya   = 0.0001,
                                             d1ya = 0,
                                             El   = 0,
                                             Eh   = 5,
                                             dE   = 0.1,
                                             match_ratio = 0.01,
                                             normalized = True )
                                             # xlim = None,
                                             # ylim = (0, 10) )


print("eigen_values: ", eigen_values)
```

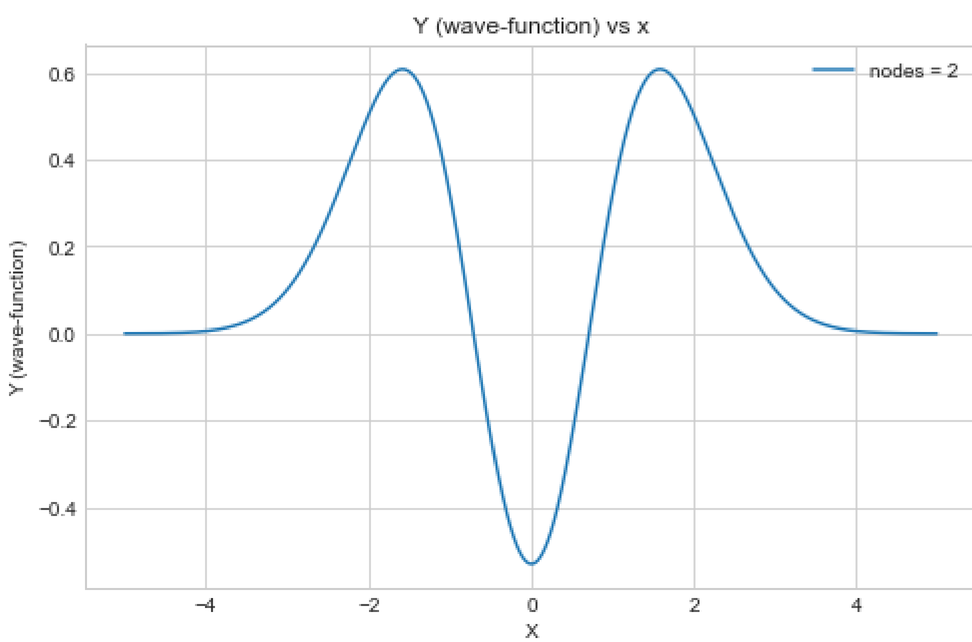
```
 10% |#######                                                                       |


i           = 0
E           = 0.5
```



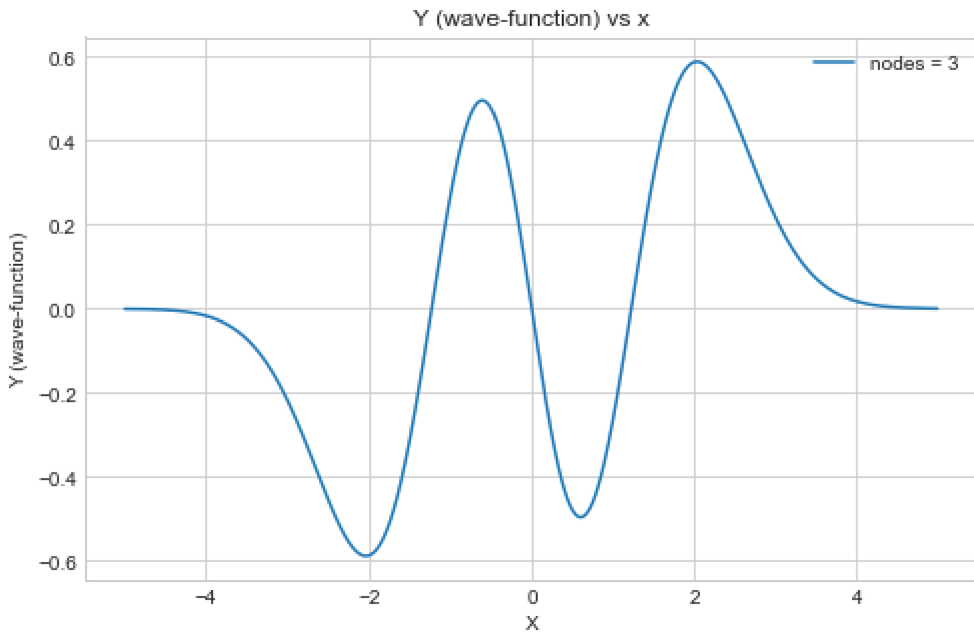

```
 30% |#####################                                                         |


i           = 1
E           = 1.5
```



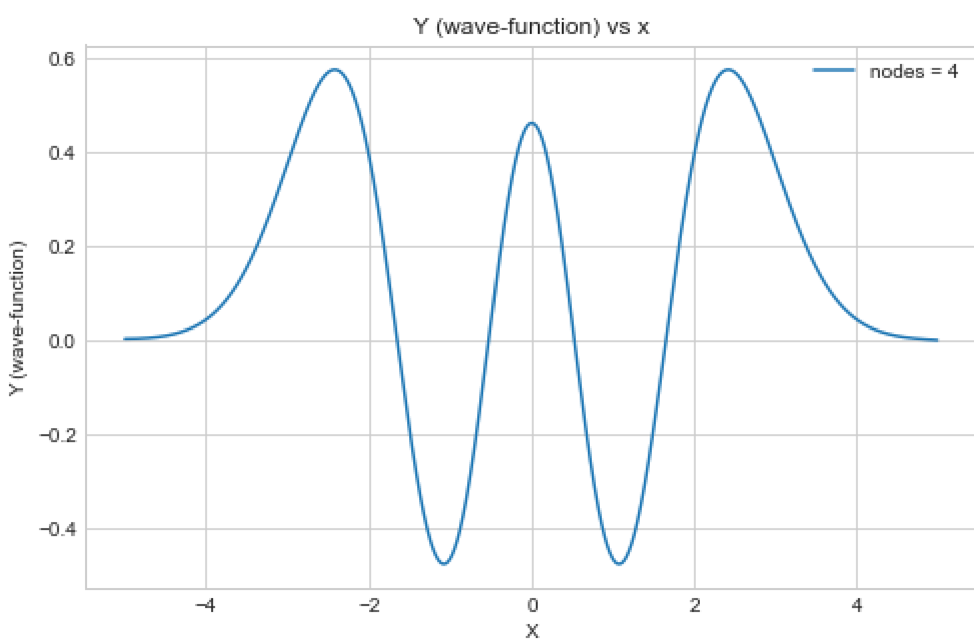

```
50% |##################################                    |

i             = 2
E             = 2.5
```



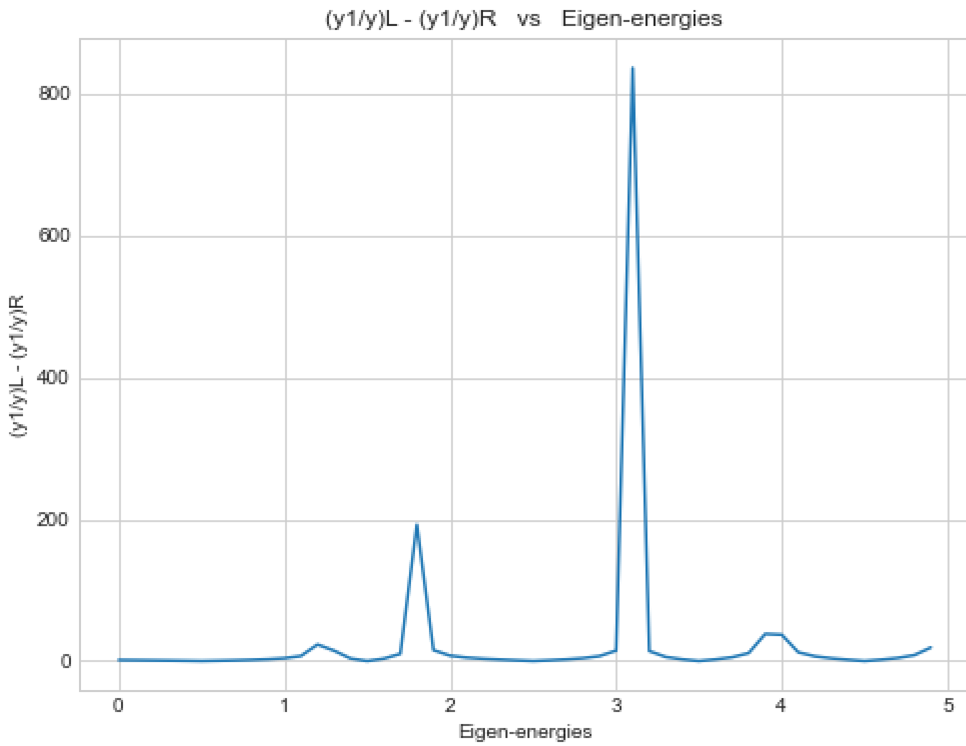

```
70% |###############################################       |

i             = 3
E             = 3.5
```



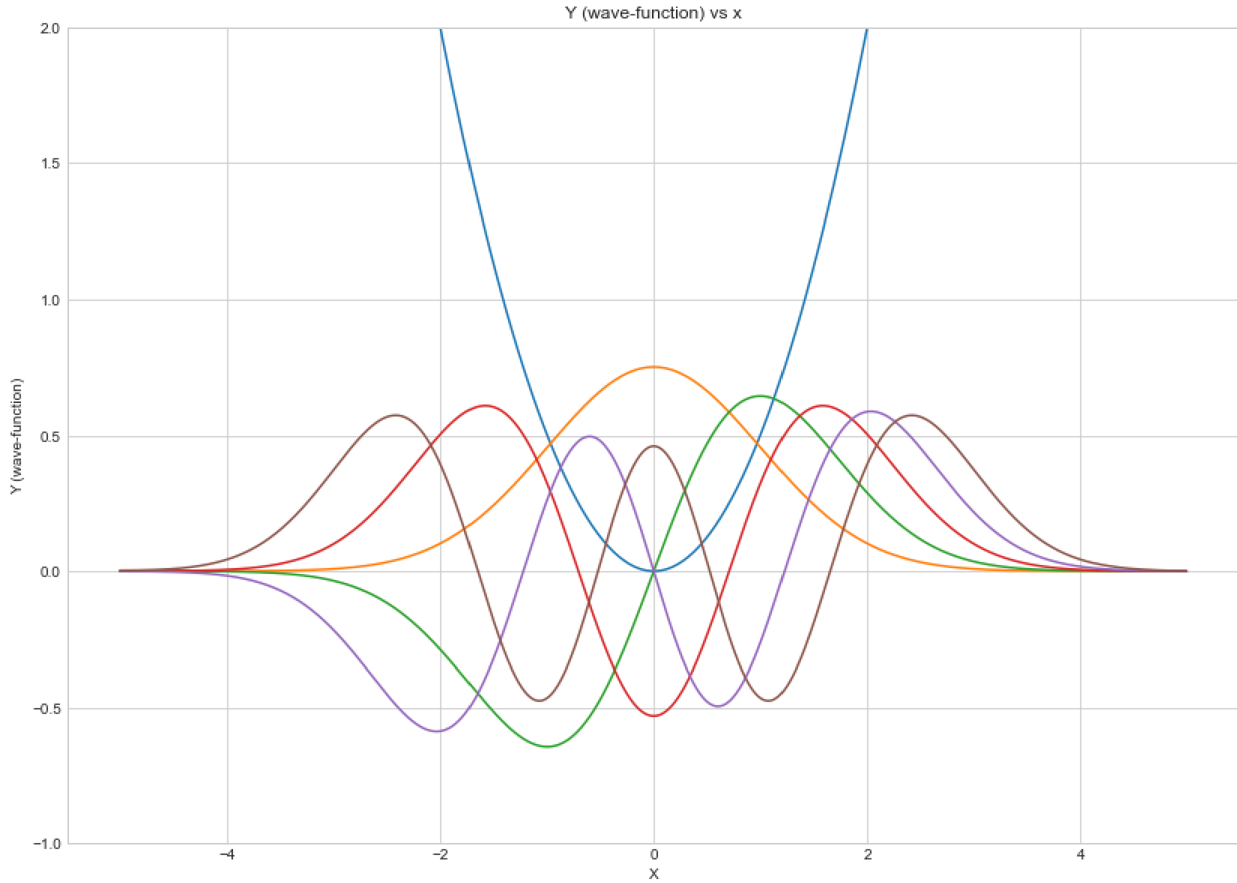

```
 90% |################################################################        |

i            = 4
E            = 4.5
```




```
100% |########################################################################|

eigen_values:  [0.5, 1.5, 2.5, 3.5, 4.5]
```


(y1/y)L - (y1/y)R   vs   Eigen-energies
800
600
400
200
0
(y1/y)L - (y1/y)R
0
1
2
3
4
5
Eigen-energies

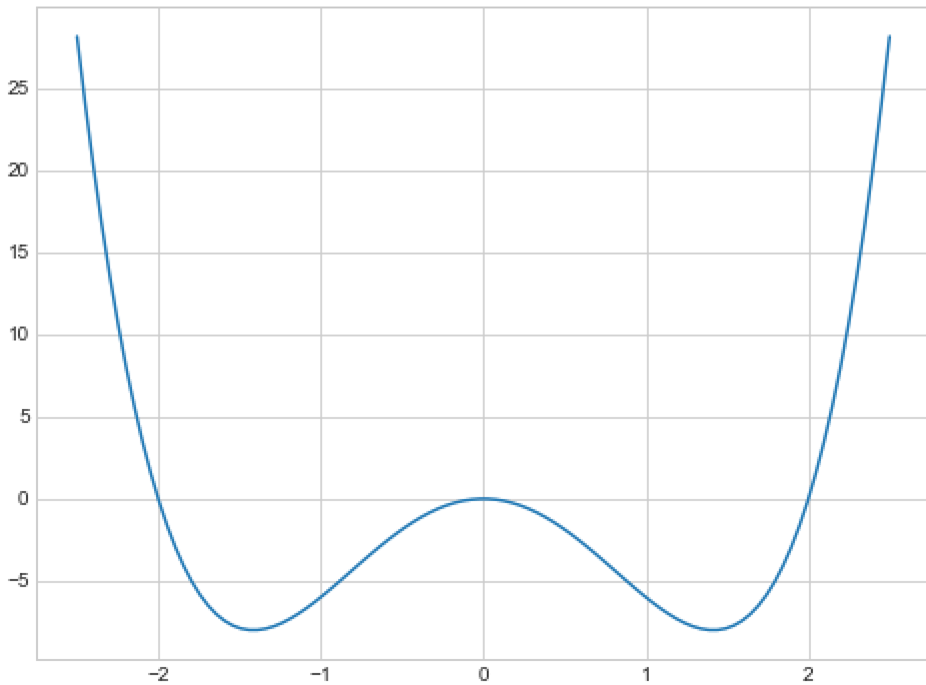
In [37]:
```
plot_wavefunc_with_V( func_V        = V,
                      axes_wavefunc = axes_sh,
                      figsize       = (14, 10),
                      rangex_V      = (-5, 5),
                      ylim          = (-1, 2,)
                    )
```



Double Well Potential

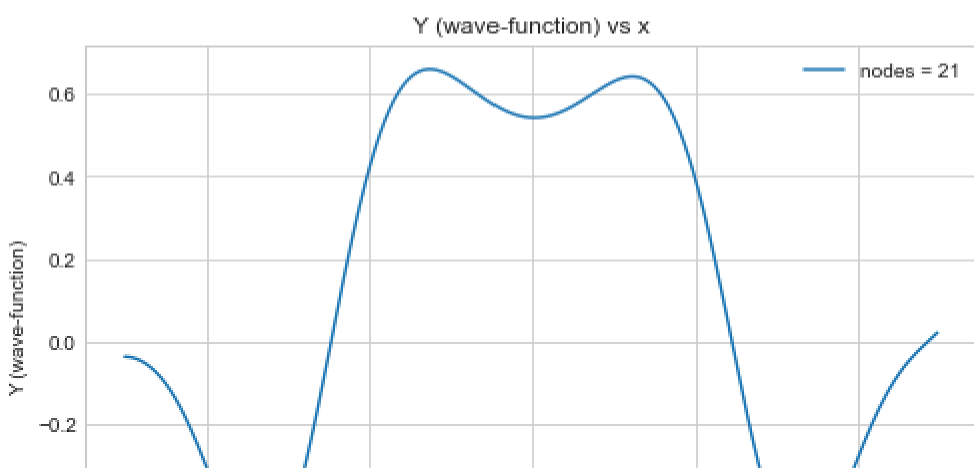
In [39]:
```
def V(x):
    return 2*x**4 - 8*x**2

def d2y(x, y, z):
    return 2*(V(x)-E) * y

fig  = plt.figure(figsize = (8, 6))
axes = plt.gca()
xpoints = np.linspace(-2.5, 2.5, num = 100)
ypoints = V(xpoints)
axes.plot(xpoints, ypoints)
```

Out[39]: [<matplotlib.lines.Line2D at 0x20006057d30>]



First attempt to get solution for double Well Potential

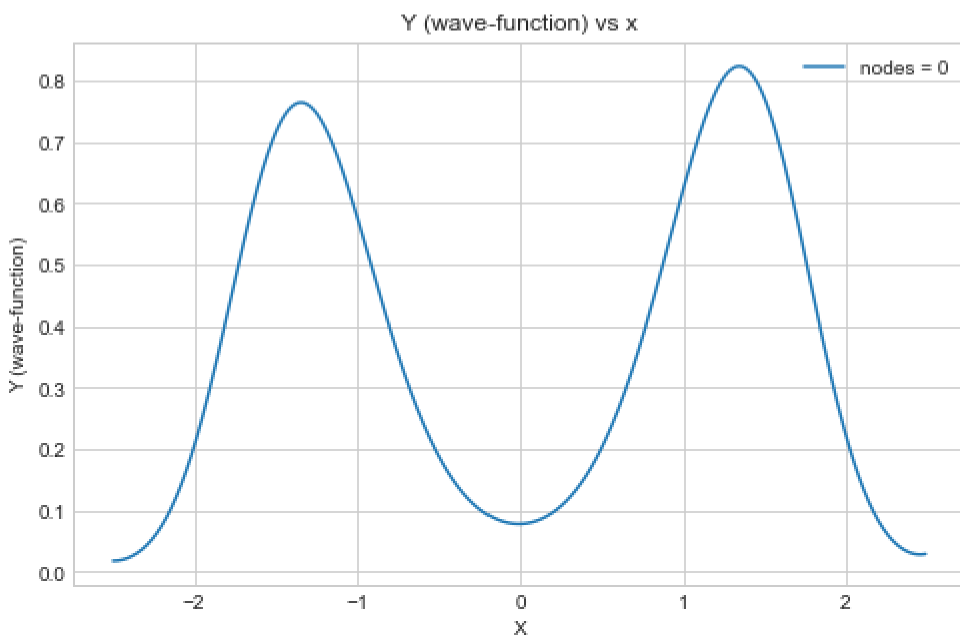
In [20]:
```
def V(x):
    return 2*x**4 - 8*x**2

def d2y(x, y, z):
    return 2*(V(x)-E) * y

eigen_values, axes_doublewell = solve_schrod_RK_4th( a     = -2.5,
                                                     b     =  2.5,
                                                     h     = 0.01,
                                                     d2y   = d2y,
                                                     ya    = 0.0001,
                                                     d1ya = 0,
                                                     El    = -8,
                                                     Eh    = 0,
                                                     dE    = 0.001,
                                                     match_ratio = 0.1,
                                                     # xlim = None,
                                                     ylim = (0, 10) )
```



1st two solutions in range (-5.5, -5.2)

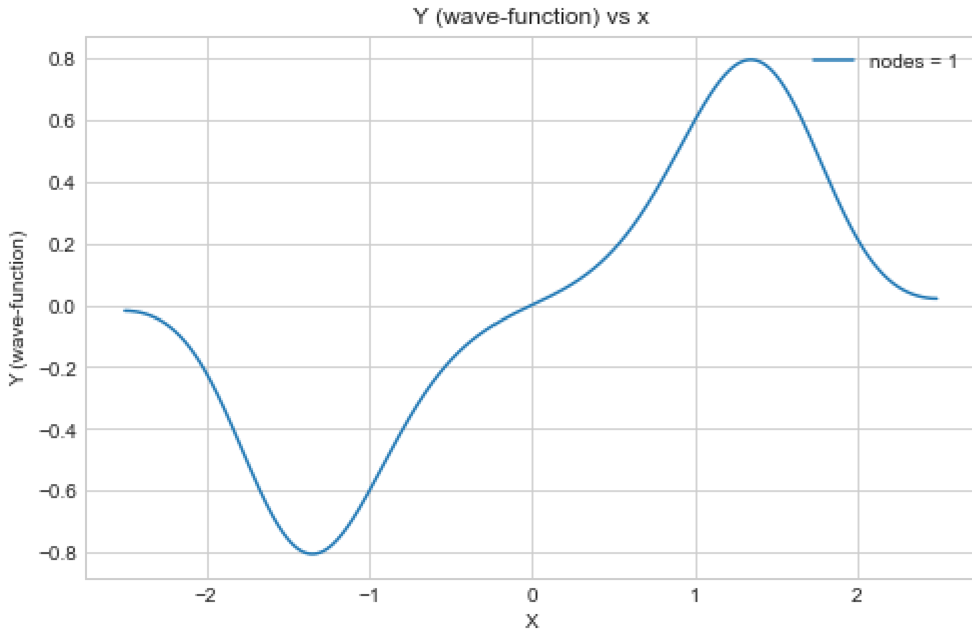
In [42]:
```
eigen_values, axes_doublewell_1_2 = solve_schrod_RK_4th( a    = -2.5,
                                                           b    =  2.5,
                                                           h    = 0.01,
                                                           d2y  = d2y,
                                                           ya   = 0.0001,
                                                           d1ya = 0,
                                                           El   = -5.5,
                                                           Eh   = -5.2,
                                                           dE   = 0.005,
                                                           match_ratio = 0.4,
                                                           ylim = (0, 10) )


eigen_values
```

```
 58% |##########################################                              |


i              = 0
E              = -5.325000000000004
```



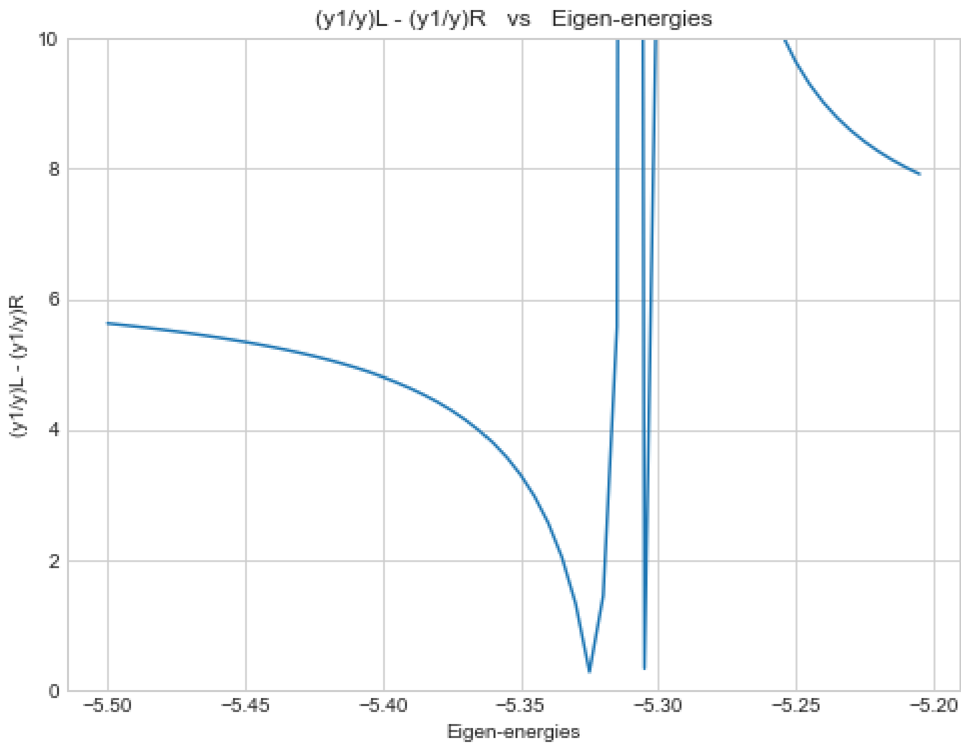

```
 65% |##############################################                          |
```

```
i            = 1
E            = -5.305000000000004
```



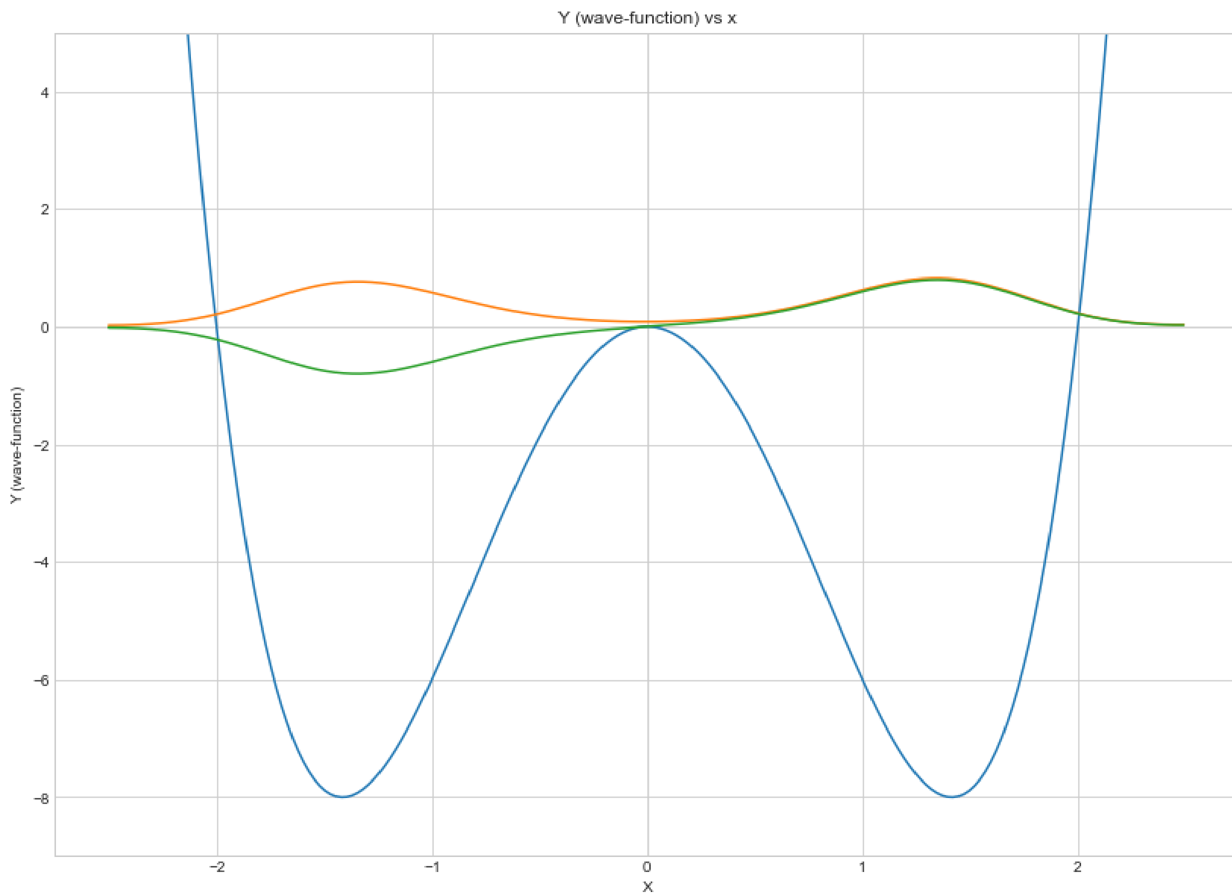

```
100% |##########################################################################|
```

Out[42]: `[-5.325000000000004, -5.305000000000004]`



(y1/y)L - (y1/y)R vs Eigen-energies
(y1/y)L - (y1/y)R
Eigen-energies
10
8
6
4
2
0
−5.50
−5.45
−5.40
−5.35
−5.30
−5.25
−5.20

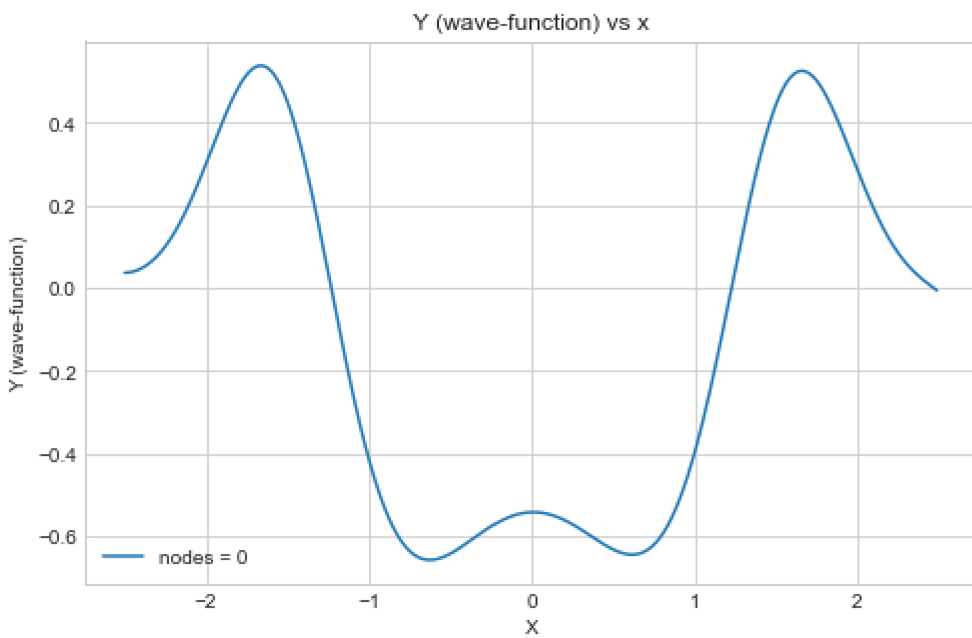
In [43]:
```
plot_wavefunc_with_V( func_V        = V,
                      axes_wavefunc = axes_doublewell_1_2,
                      figsize       = (14, 10),
                      rangex_V      = (-2.5, 2.5),
                      ylim          = (-9, 5)
                    )
```



3rd solutions in range (-1.5, -1)

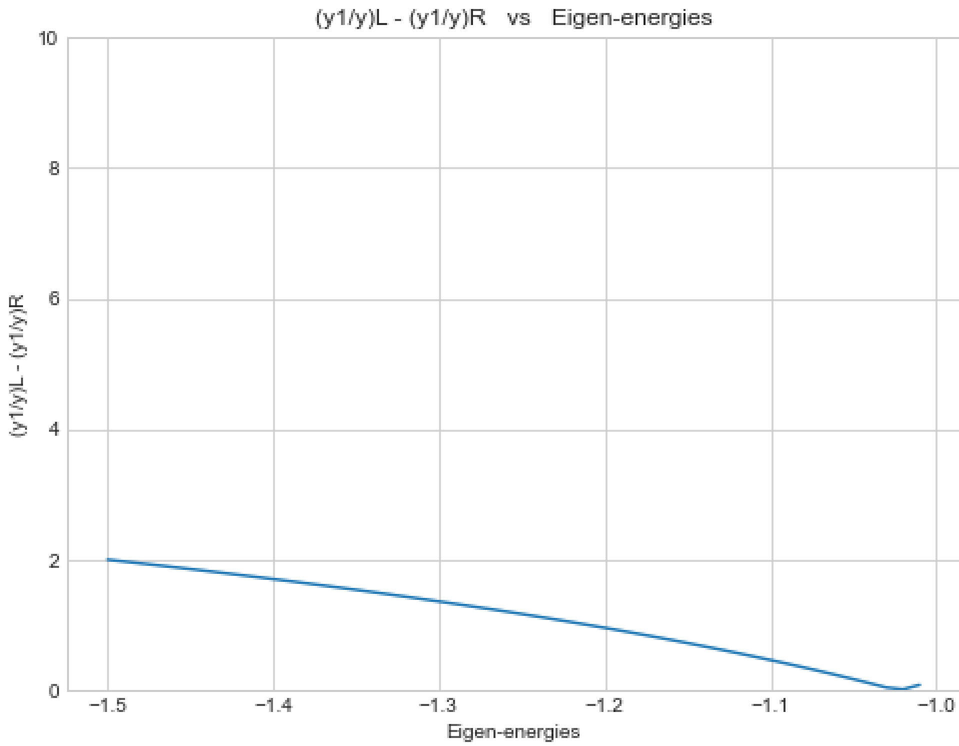
In [46]:
```
eigen_values, axes_doublewell_3 = solve_schrod_RK_4th( a    = -2.5,
                                                        b    =  2.5,
                                                        h    = 0.01,
                                                        d2y  = d2y,
                                                        ya   = 0.0001,
                                                        d1ya = 0,
                                                        El   = -1.5,
                                                        Eh   = -1,
                                                        dE   = 0.01,
                                                        match_ratio = 0.04,
                                                        ylim = (0, 10) )

eigen_values
```

```
 96% |######################################################################    |

i          = 0
E          = -1.0199999999999996
```

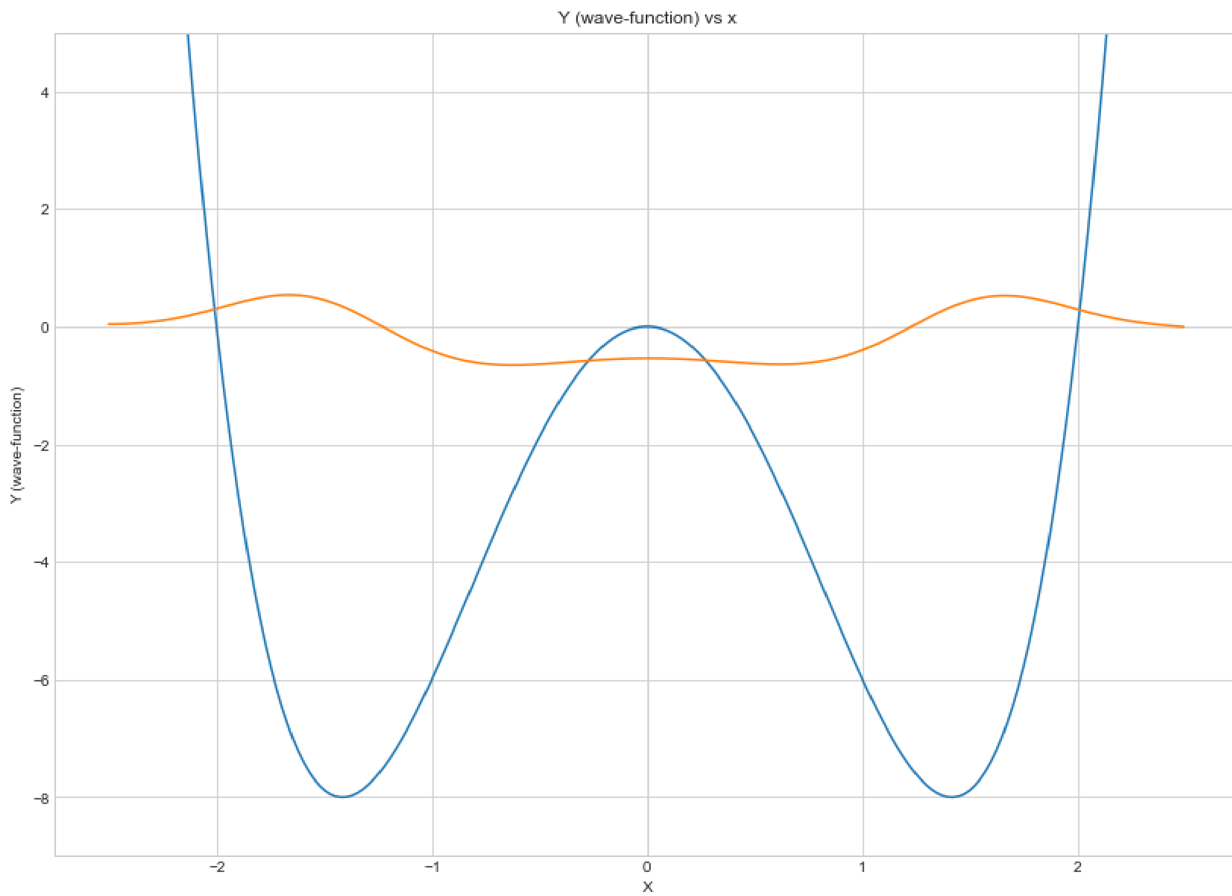


100% |##|

Out[46]: [-1.0199999999999996]



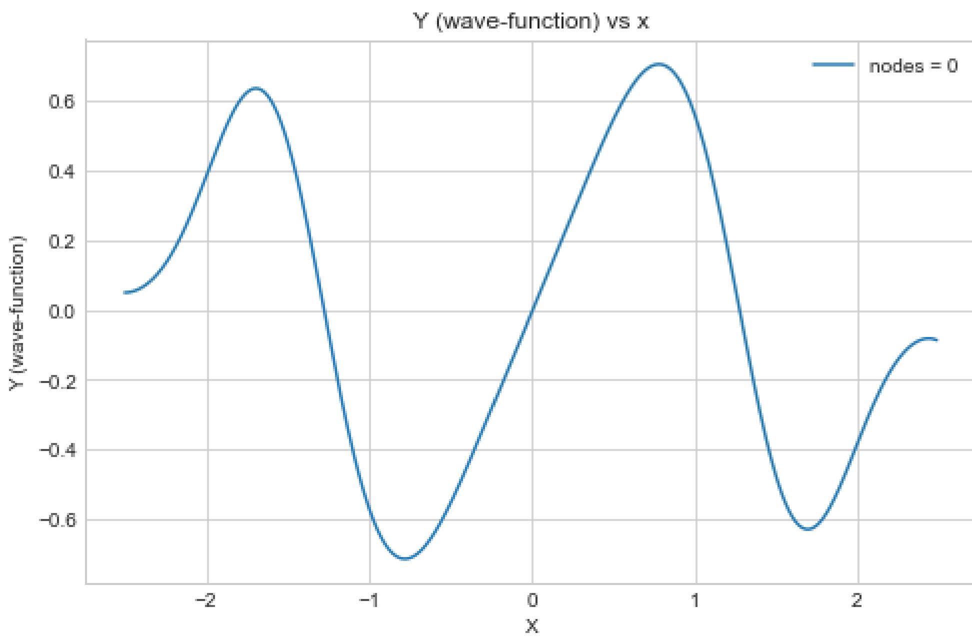
In [47]:
```
plot_wavefunc_with_V( func_V        = V,
                      axes_wavefunc = axes_doublewell_3,
                      figsize       = (14, 10),
                      rangex_V      = (-2.5, 2.5),
                      ylim          = (-9, 5)
                    )
```



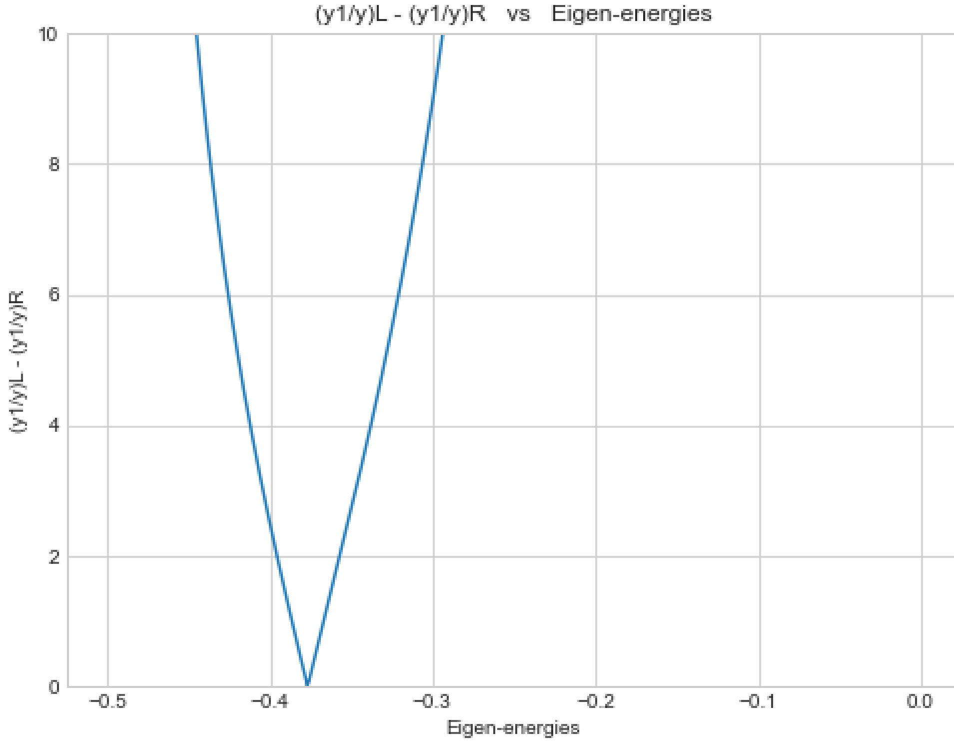
4th solutions in range (-0.5, 0)

In [48]:
```
eigen_values, axes_doublewell_4 = solve_schrod_RK_4th( a     = -2.5,
                                                        b     =  2.5,
                                                        h     = 0.01,
                                                        d2y   = d2y,
                                                        ya    = 0.0001,
                                                        d1ya  = 0,
                                                        El    = -0.5,
                                                        Eh    = 0,
                                                        dE    = 0.001,
                                                        match_ratio = 0.02,
                                                        # xlim = None,
                                                        ylim = (0, 10) )
```

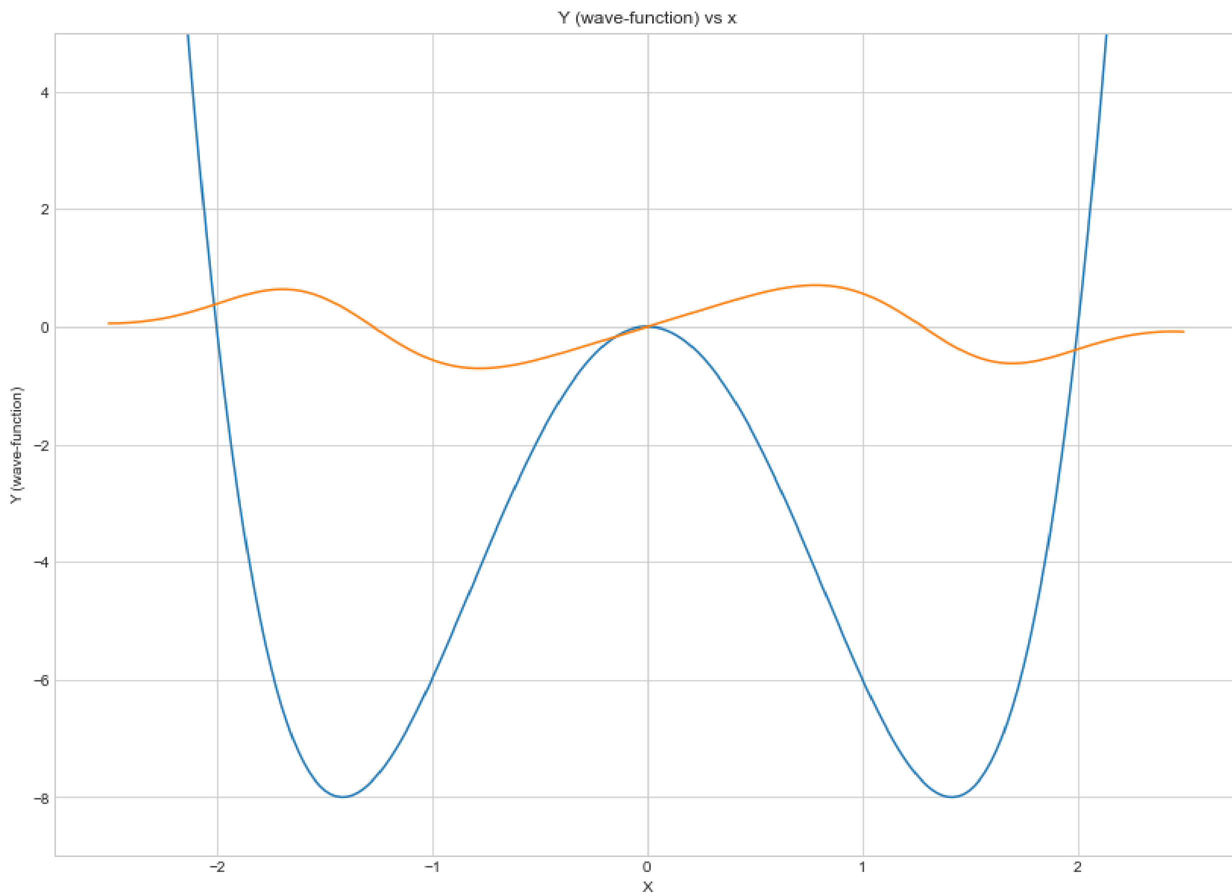
```
 24% |#################                                                          |

i            = 0
E            = -0.3769999999999999
```




```
100% |###########################################################################|
```


(y1/y)L - (y1/y)R vs Eigen-energies
(y1/y)L - (y1/y)R
Eigen-energies
10
8
6
4
2
0
−0.5
−0.4
−0.3
−0.2
−0.1
0.0

In [50]:
```
plot_wavefunc_with_V( func_V        = V,
                      axes_wavefunc = axes_doublewell_4,
                      figsize       = (14, 10),
                      rangex_V      = (-2.5, 2.5),
                      ylim          = (-9, 5)
                    )
```



Final Solution Problem 1

Eigen-Values for double wave potential

E_0 = -5.325000000000004

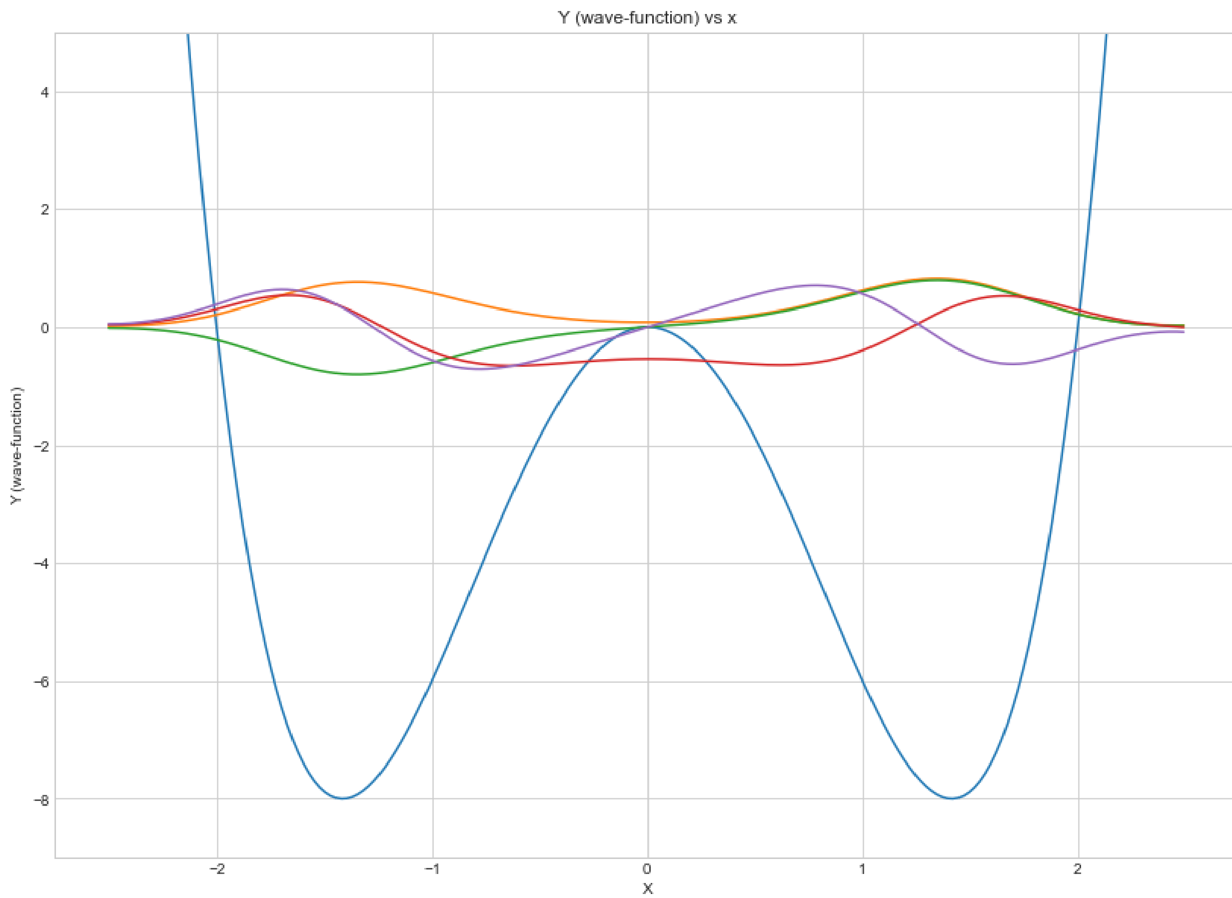
E_1 = -5.305000000000004

E_2 = -1.0199999999999996

E_3 = -0.3769999999999999

Eigen wave functions

In [51]:
```
plot_wavefunc_with_V( func_V        = V,
                      axes_wavefunc = axes_doublewell_1_2 + axes_doublewell_3 + axes_dou
                      figsize       = (14, 10),
                      rangex_V      = (-2.5, 2.5),
                      ylim          = (-9, 5)
                    )
```



In [ ]:
```

```

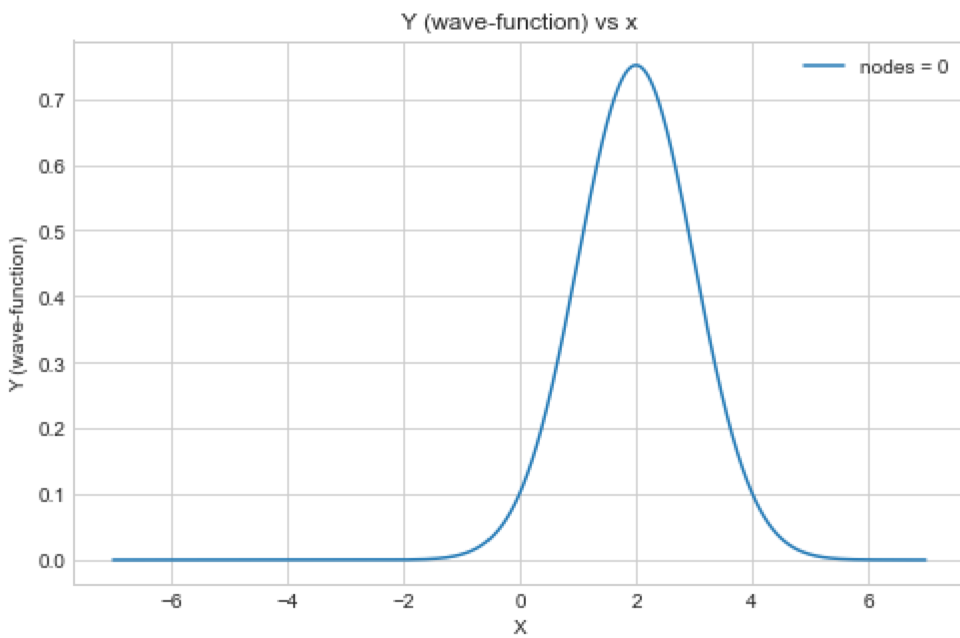
In [ ]:
```

```

Simple Harmonic Oscillator at x = 2

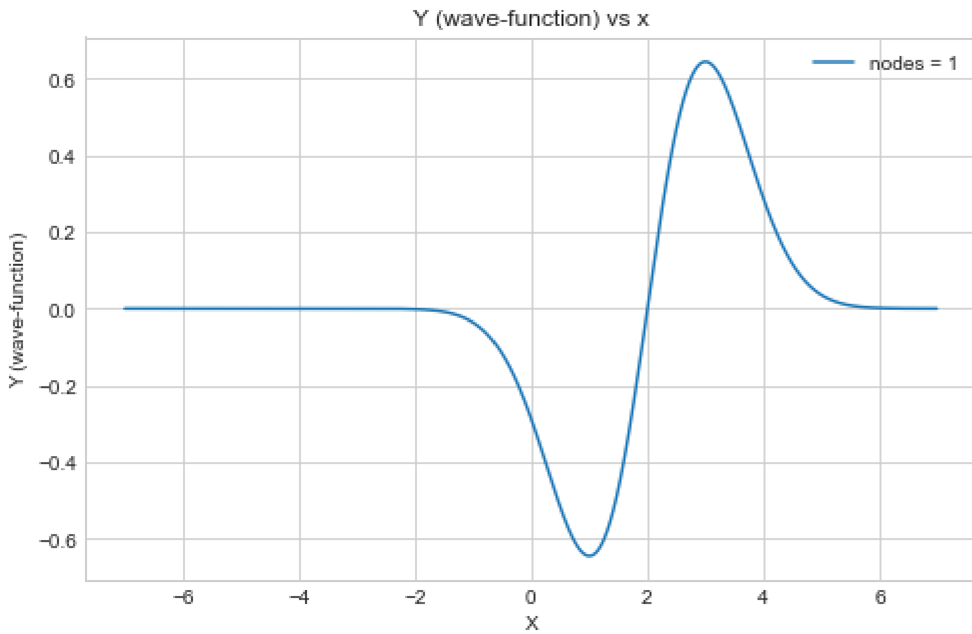
In [60]:
```
def V(x):
    return 1/2*(x-2)**2

def d2y(x, y, z):
    return 2*(V(x)-E) * y
```

In [61]:
```
eigen_values, axes_sh = solve_schrod_RK_4th( a     = -7,
                                             b     =  7,
                                             h     = 0.01,
                                             d2y   = d2y,
                                             ya    = 0.0001,
                                             d1ya  = 0,
                                             El    = 0,
                                             Eh    = 4,
                                             dE    = 0.1,
                                             match_ratio = 0.01,
                                             normalized = True )
                                             # xlim = None,
                                             # ylim = (0, 10) )


print("eigen_values: ", eigen_values)
```

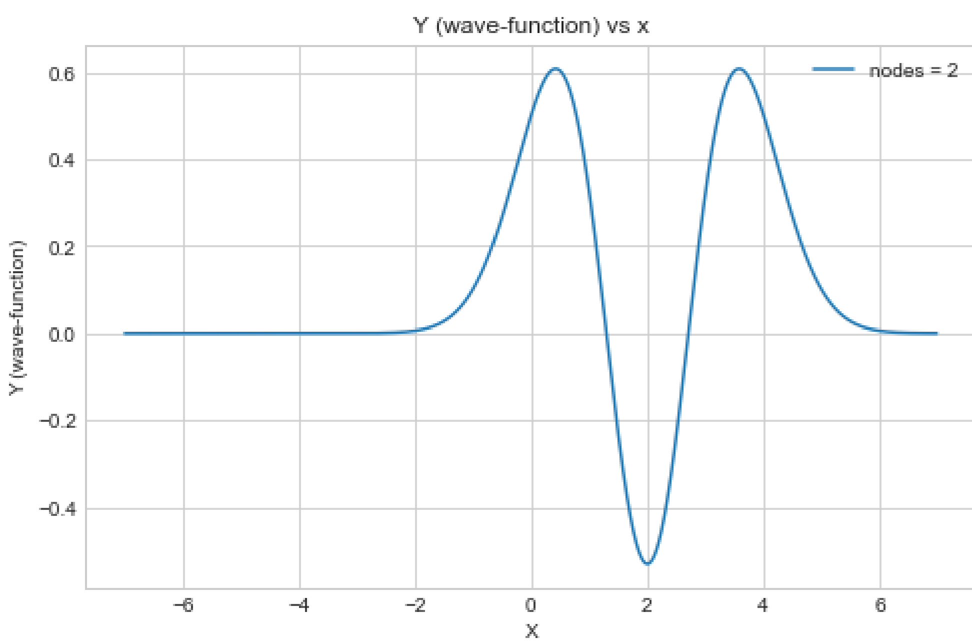
```
 12% |#########                                                                     |


i            = 0
E            = 0.5
```



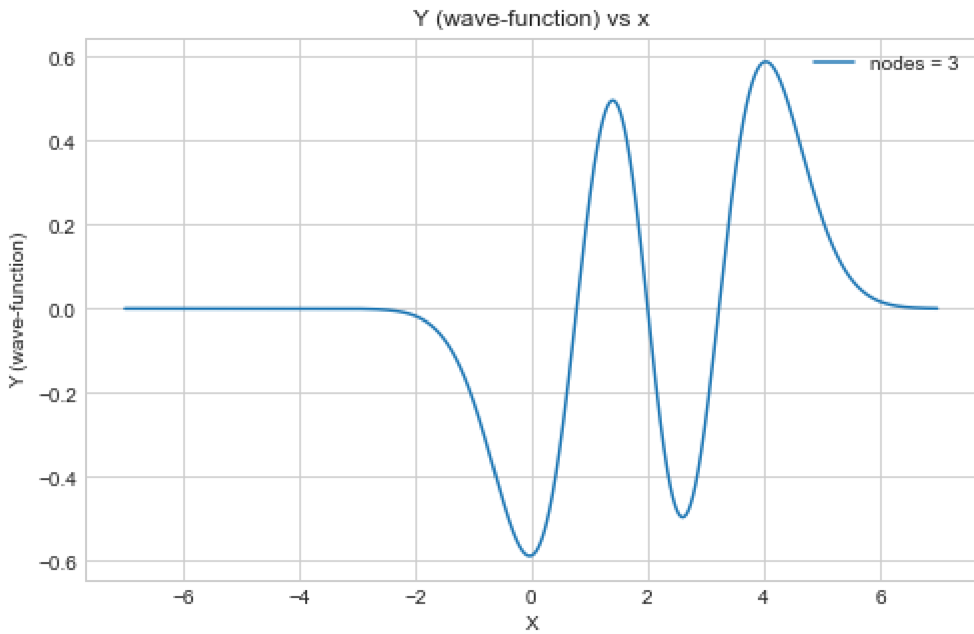

```
 37% |###########################                                                   |


i            = 1
E            = 1.5
```



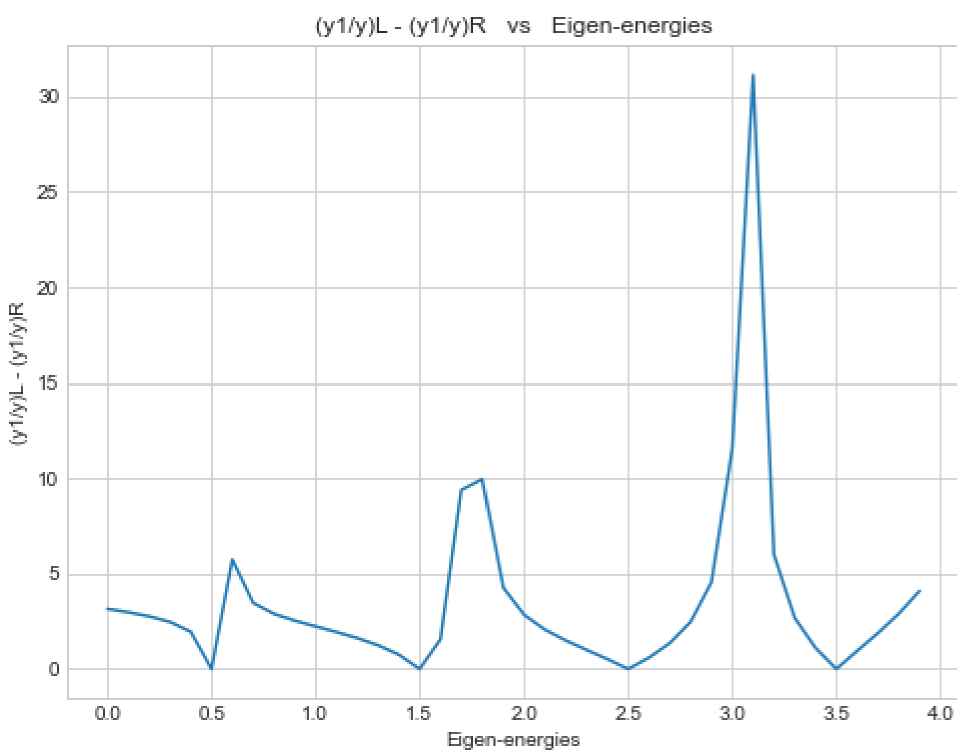

```
62% |##############################################                         |

i             = 2
E             = 2.5
```



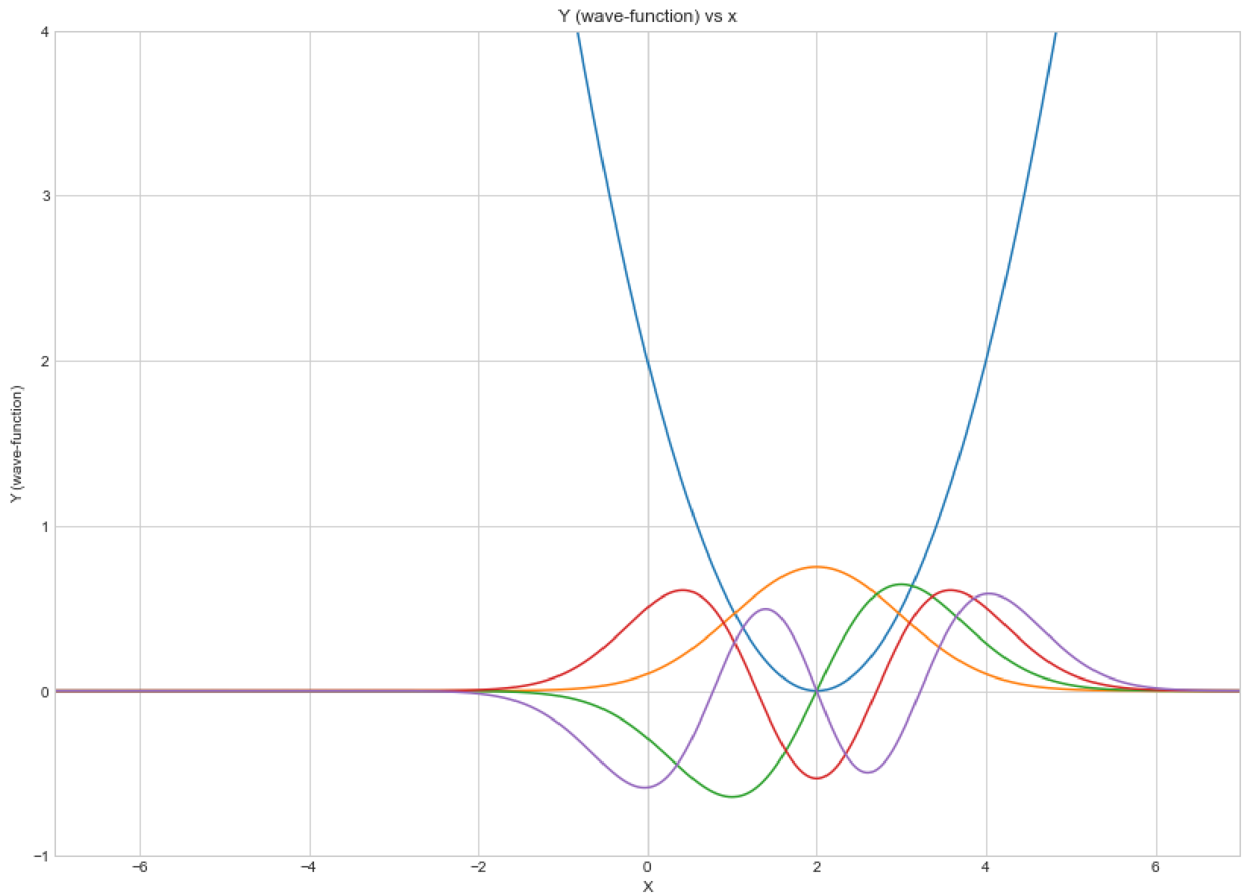

```
87% |#################################################################          |

i             = 3
E             = 3.5
```




```
100% |#########################################################################|

eigen_values:  [0.5, 1.5, 2.5, 3.5]
```



In [62]:
```
plot_wavefunc_with_V( func_V        = V,
                      axes_wavefunc = axes_sh,
                      figsize       = (14, 10),
                      rangex_V      = (-3, 7),
                      ylim          = (-1, 4),
                      xlim          = (-7, 7)
                    )
```



In [ ]: