# BVP-and-Eigenvalue-problems-v2

February 20, 2024

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
```

#### 0.0.1 Shooting method

Consider the equation

$$\frac{dy}{dx} = f(x, y, k), \tag{1}$$

where, $p$ is an unknown parameter. We are also given values of $y(x_1)$ and $y(x_2)$. This problem cannot be directly solved as an initial value problem, as we do not know $y'(x_1)$ or $y'(x_2)$. For specific values of $p$, the solution $y(x)$ satisfies the boundary condition.

The shooting method involves the following steps: * shoot from $y(x_1)$ with a guessed $p$ and integrate upto $x_2$ * compare the end point values $y(x_2)$ (between the computed and the given) * minimize the difference by changing $p$

The shooting method is commonly applied to boundary value problems, e.g. oscillating strings, Schroedinger equation, etc.

Let us apply the shooting method to find the normal modes of a vibrating string of length $L$ which is fixed at both ends. So the boundary conditions are $y(0) = y(L) = 0$. The corresponding equation is:

$$\frac{\partial^2 y}{\partial^2 t} = v^2 \frac{\partial^2 y}{\partial^2 x} \tag{2}$$

For oscillating solutions, one can use a trial solution $y(x,t) = f(x)Cos(\omega t)$. The equation reduces to:

$$\frac{d^2 f}{dx^2} = -\frac{\omega^2}{v^2} f(x) = -k^2 f(x) \tag{3}$$

We choose $L = 1$ and $v = 1$, and we need to find $y(x)$ for various $\omega$ values (or $k$ values).

We reduce the second-order differential equation to the first-order form:

$$\frac{df}{dx} = p \tag{4}$$

$$\frac{dp}{dx} = -k^2 f \tag{5}$$

Using a vector $y$ to store $f$ and $p$, we get,

$$\frac{dy[1]}{dx} = y[2] \tag{6}$$

$$\frac{dy[2]}{dx} = -k^2 y[1] \tag{7}$$

or

$$\frac{d}{dt}\begin{pmatrix} y[1] \\ y[2] \end{pmatrix} = \begin{pmatrix} y[2] \\ -k^2 y[1] \end{pmatrix} \tag{8}$$

[2]:

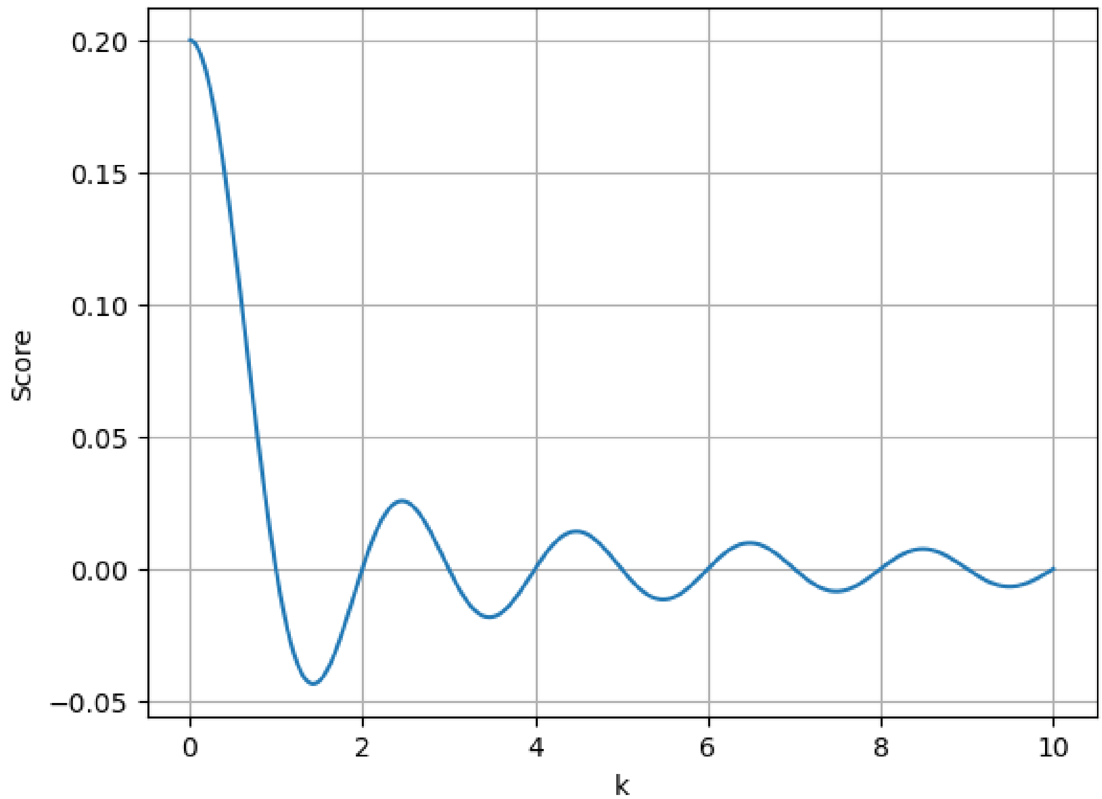
```
# The following is standard RK4. It calls a function f(x,y,k) where k is a␣
↪paramter
def rk4(f,x,y,k,h):
    k1 = h*f(x,y,k)
    k2 = h*f(x + h/2, y + k1/2, k)
    k3 = h*f(x + h/2, y + k2/2, k)
    k4 = h*f(x + h, y + k3, k)
    return y + (k1+2*k2+2*k3+k4)/6

# The following is a caller function. It integrates f from xlim[0] to xlim[1]␣
↪with initial condition yini.
# The parameter k is included in the argument along with the number of points N.
def caller_rk4(f,xlim,yini,k,N):
    x1, x2 = xlim
    xs = np.linspace(x1, x2, N)
    h = xs[1] - xs[0]
    y = yini
    ys = np.zeros((N,len(yini)))
    for i in range(N):
        ys[i] = np.array(y)
        y = rk4(f, xs[i], y, k, h)
    return xs, ys

# The following function calculates the departure from the boundary condition␣
↪at the other end.
def score(k,f,ybound):
    xs, ys = caller_rk4(f,xlim,yini,k,N)
    return ys[-1][0] - ybound[1]

# This is an implementation of the secant method used for finding the root of␣
↪the function score.
def secant(ks,f,method,ybound):
    k1, k2 = ks
    iter = 0
    while abs(method(k2,f,ybound))>tol and iter<maxiter:
        f1 = method(k1,f,ybound)
        f2 = method(k2,f,ybound)
```

```
        k1, k2 = k2, (f2*k1 - f1*k2)/(f2 - f1)
        iter += 1
        print(iter, k1, k2, method(k2,f,ybound))
    if iter == maxiter:
        return iter, None
    else:
        return iter, k2

# The following is an implementation of Simpson's 1/3 method (for normalization)
def simp13(y,h):
    store = y[0]**2 + y[-1]**2
    for i in range(2,len(y)):
        if i%2==0:
            store += 4*y[i]**2
        else:
            store += 2*y[i]**2
    return store*(h/3)
```

[3]:
```
# define the ODE function
def odefun1(x,y,k):
    return np.array([y[1], -(np.pi*k)**2 * y[0]]) # here k is wavenumber in the␣
 ↪unit of pi

# Initialization
xlim = (0.0, 1.0)
yini = (0.0, 0.2) # 0.2 is an arbitrary number finally adjusted by normalization

# Boundary condition
ybound = (0.0, 0.0)

# Number of points
N = 129

# Max iteration and tolerance for secant
maxiter = 50
tol = 1.0e-6
```

[4]:
```
ks = np.linspace(0.01,10,500);
scores = [score(k,odefun1,ybound) for k in ks];
```

[5]:
```
plt.plot(ks, scores);
plt.grid();
plt.xlabel("k");
plt.ylabel("Score");
```

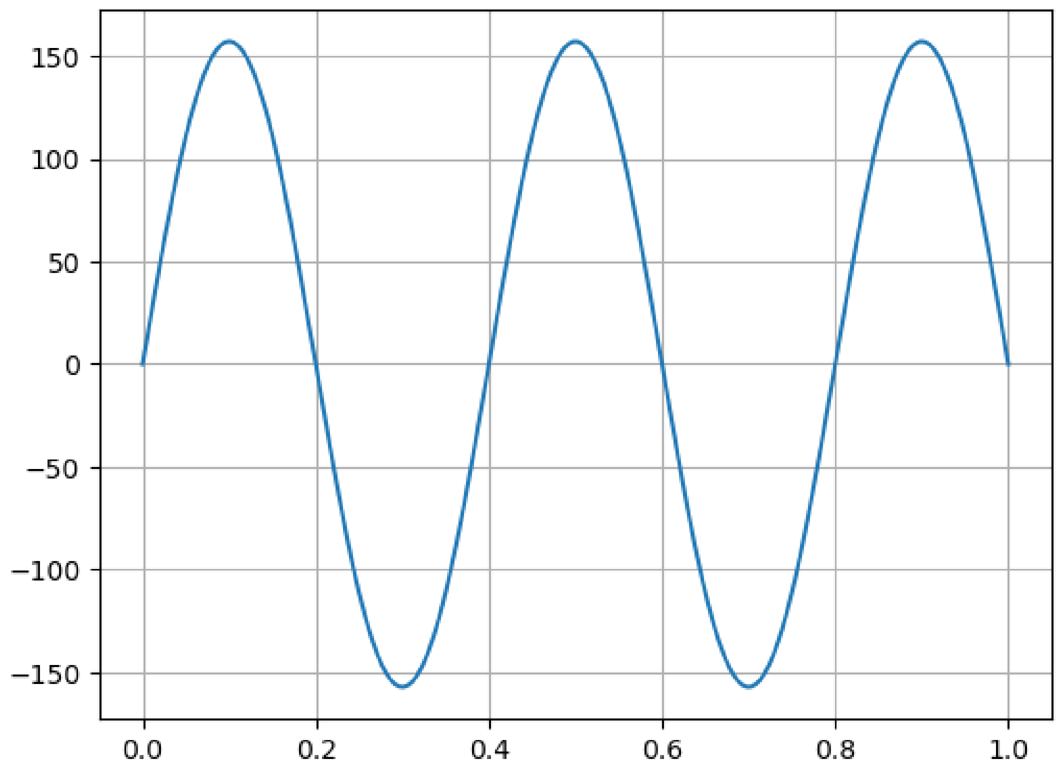



[6]:
```
kini=(4.8,4.9);
iter,keigen = secant(kini,odefun1,score,ybound);
```

```
1 4.9 5.006197039829386 -0.0002471805964420666
2 5.006197039829386 5.000038461670652 -1.1624739747362767e-06
3 5.000038461670652 5.000009361428933 1.5123026953986485e-09
```

[7]:
```
xs, ys = caller_rk4(odefun1,xlim,yini,keigen,200);
h = xs[1]-xs[0]
y1 = [y[0] for y in ys];
normfact = simp13(y1,h);
y1 = y1/normfact;
```

[8]:
```
plt.plot(xs,y1);
plt.grid()
```




## 0.1 Schroedinger Equation: bound state solutions

Let us try to find the solutions of Schoedinger equation for a negative potential well. So we try to solve an equation of the form:

$$\frac{d^2\psi}{dx^2} + k^2(x)\psi(x) = 0 \tag{9}$$

where,

$$k^2(x) = \frac{2m}{\hbar^2}\left(E - V(x)\right). \tag{10}$$

For simplicity let us assume $\frac{2m}{\hbar^2} = 1$. Also let us choose $V(x)$ as,

$$V(x) = \begin{cases} 0 & \text{for } |x| > L_\circ \\ -V_\circ & \text{for } |x| \leq L_\circ \end{cases} \tag{11}$$

with $\psi(\pm 3L_\circ) = 0$

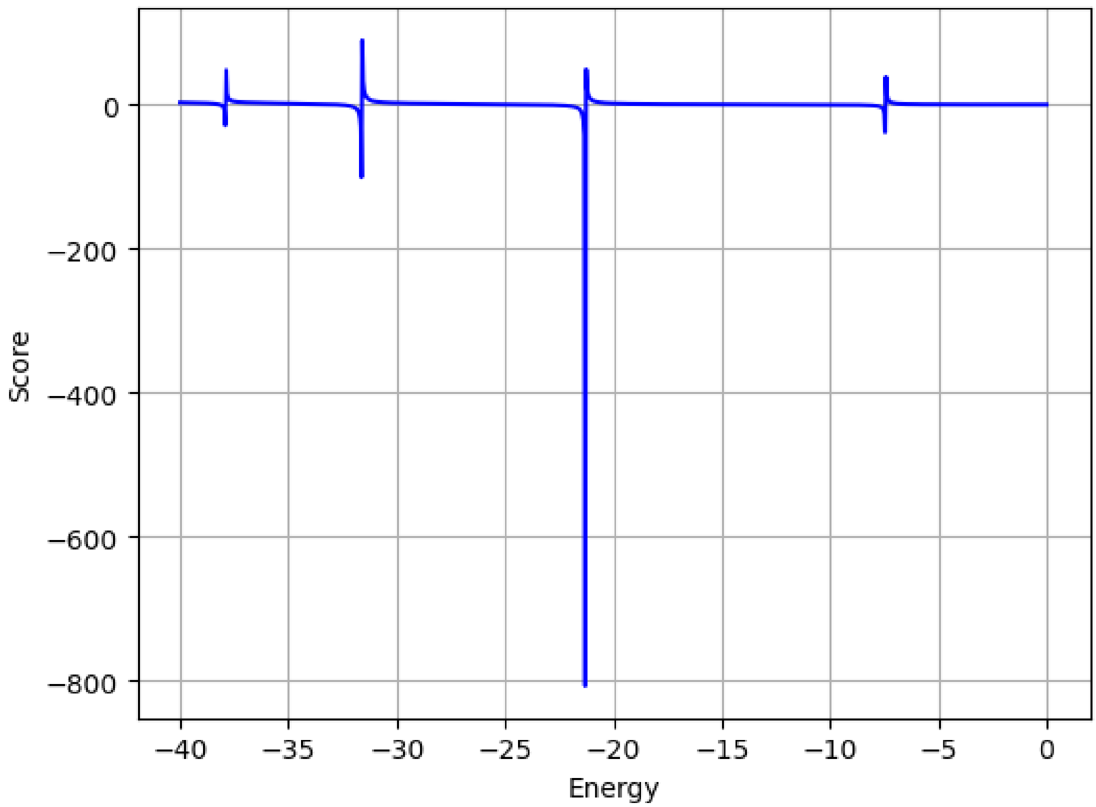
[9]:
```
# The rhs of Schoedinger equation (using vectorized form)
def odefun2(x,y,E):
    k2 = E - pot(x);
    return np.array([y[1], -k2 * y[0]])
```

```python
# The potential
def pot(x):
    if abs(x)>Lo:
        return 0.0
    else:
        return -Vo

# The score. Note: we are integrating from right and from left and checking the␣
↪difference at one of the turning points.
def score2(E,f,ybound):
    yini1 = (ybound[0], 1.0e-2);
    xs1, ys1 = caller_rk4(odefun2,xlim,yini1,E,N)
    a = abs(xs1 + Lo);
    n1=np.where(a == np.min(a))[0][0]
    yini2 = (ybound[1], 1.0e-2)
    xs2, ys2 = caller_rk4(odefun2,xlim[-1::-1],yini2,E,N)
    a = abs(xs2 + Lo);
    n2=np.where(a == np.min(a))[0][0]
    ys2 = ys1[n1][0]*ys2/ys2[n2][0]
    return ys1[n1][1] - ys2[n2][1]
```

[10]:
```python
Vo = 40.0
Lo = 1.0
xlim = (-2.0, 2.0)
ybound = (0.0, 0.0)
yini = (ybound[0], 0.1) # 1.0e-2 is an arbitrary number finally adjusted by␣
↪normalization
N = 129
maxiter = 50
tol = 1.0e-5
```

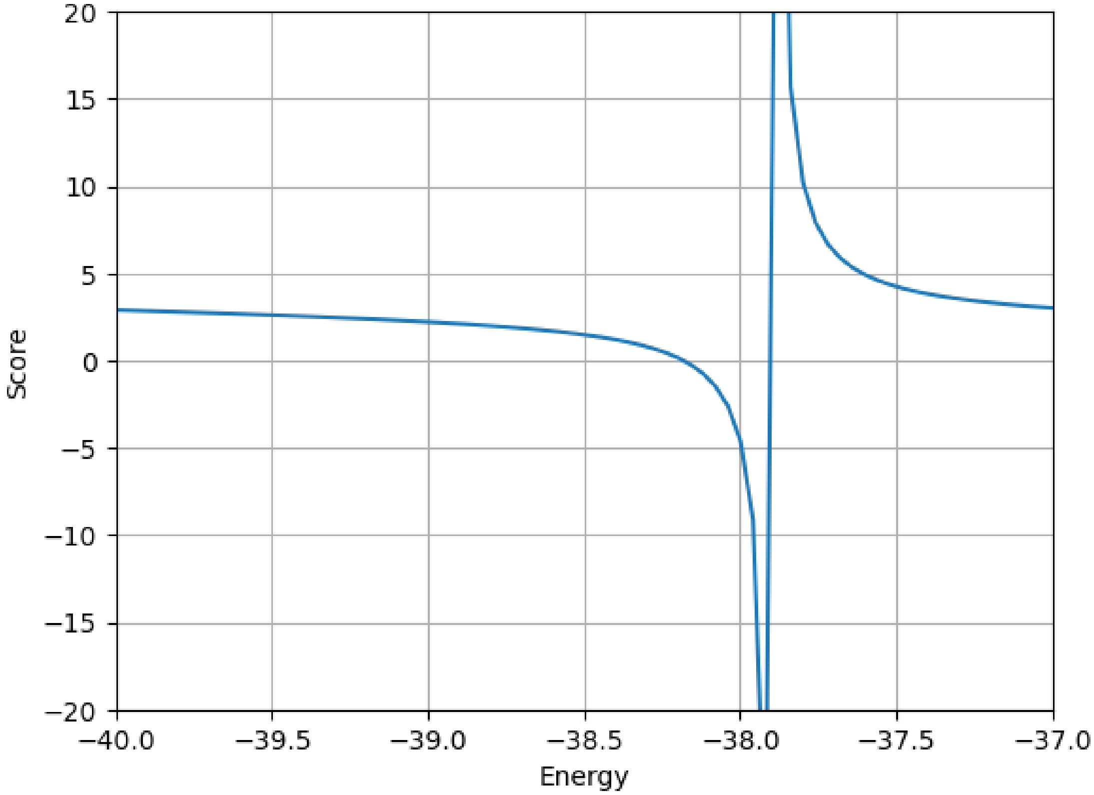
[11]:
```python
Es = np.linspace(-Vo, 0, 1000);
scores = [score2(E, odefun2, ybound) for E in Es];
```

[12]:
```python
plt.plot(Es,scores,'b-');
plt.grid()
plt.xlabel("Energy");
plt.ylabel("Score");
```




[16]:
```
plt.plot(Es,scores);
plt.grid()
plt.xlabel("Energy");
plt.ylabel("Score");
plt.ylim([-20.,20.])
plt.xlim([-40.0,-37.0])
```

[16]: (-40.0, -37.0)

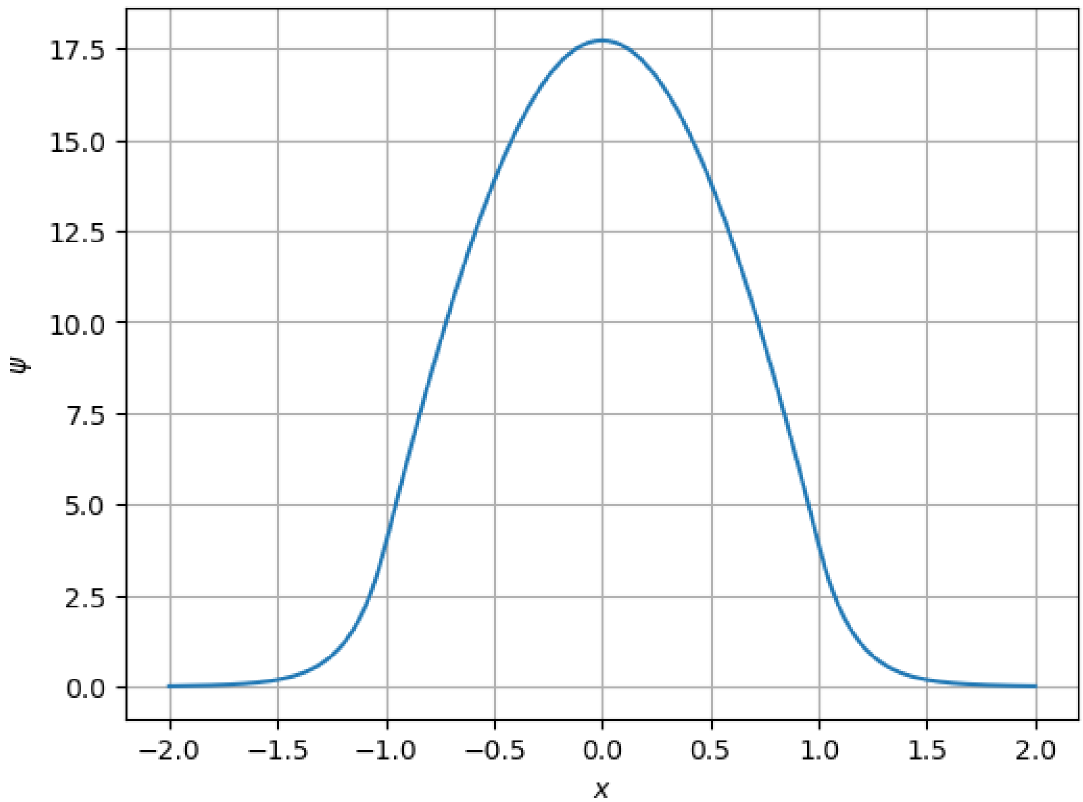



[17]:
```
Eini=(-38.35,-38.25)
iter,Eeigen = secant(Eini,odefun2,score2,ybound);
```

```
1 -38.25 -38.143554958403364 -0.4041526958569399
2 -38.143554958403364 -38.189804385618416 0.07895675172808847
3 -38.189804385618416 -38.18224563353762 0.010245476102339612
4 -38.18224563353762 -38.181118554857996 -0.00029805236087288023
5 -38.181118554857996 -38.18115041596241 1.0944452468741872e-06
```

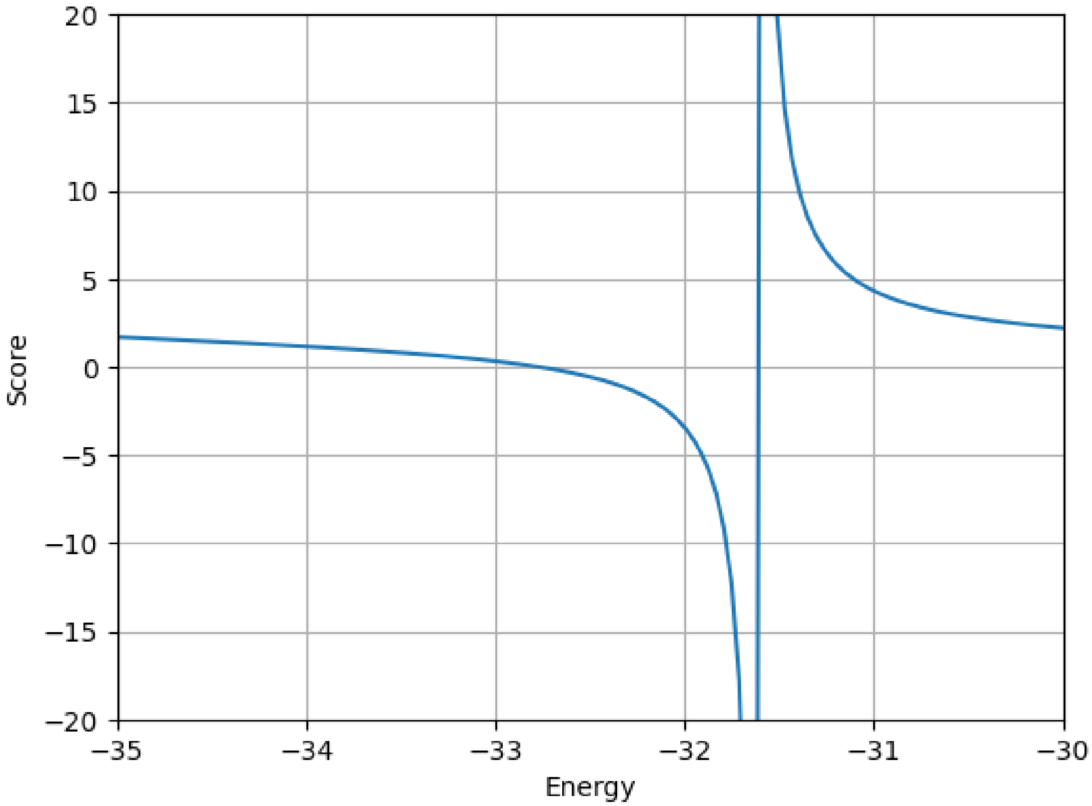
[18]:
```
xs, ys = caller_rk4(odefun2, xlim, yini, Eeigen, N);
y1 = [y[0] for y in ys];
plt.plot(xs, y1);
plt.grid();
plt.xlabel("$x$")
plt.ylabel("$\\psi$")
```

[18]: `Text(0, 0.5, '$\\psi$')`




[21]:
```
plt.plot(Es,scores);
plt.grid()
plt.xlabel("Energy");
plt.ylabel("Score");
plt.ylim([-20.,20.])
plt.xlim([-35.0,-30.0])
```

[21]: (-35.0, -30.0)

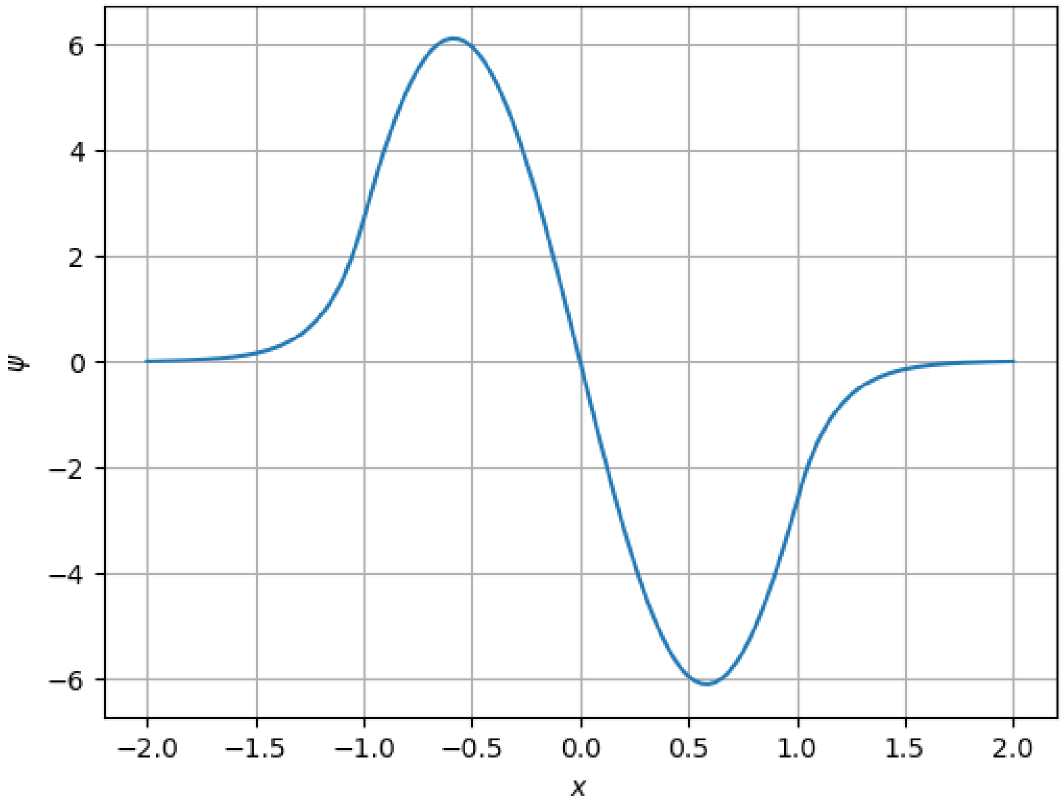



[22]:
```
Eini=(-33.0,-32.5)
iter,Eeigen = secant(Eini,odefun2,score2,ybound);
```

```
1 -32.5 -32.81943843303172 0.07274709694859349
2 -32.81943843303172 -32.78317792526891 0.015377159781380323
3 -32.78317792526891 -32.77345883458911 -0.0005100830594169103
4 -32.77345883458911 -32.77377088014405 3.484037833334561e-06
```

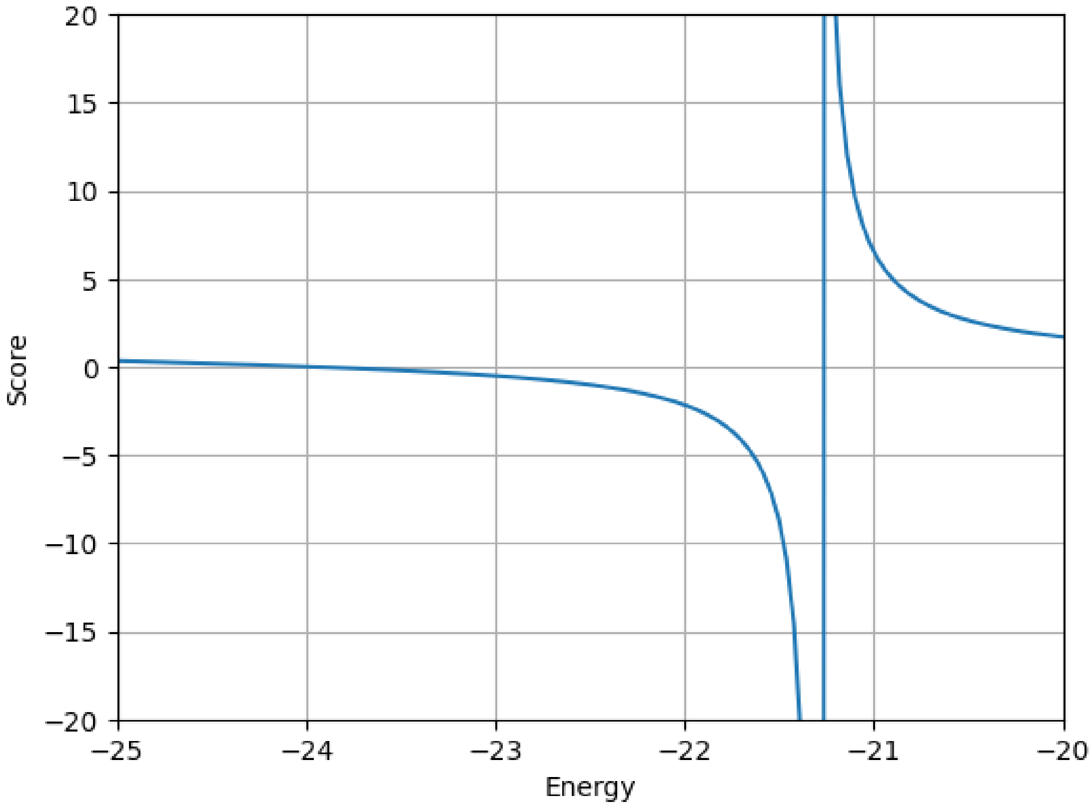
[23]:
```
xs, ys = caller_rk4(odefun2, xlim, yini, Eeigen, N);
y1 = [y[0] for y in ys];
plt.plot(xs, y1);
plt.grid();
plt.xlabel("$x$")
plt.ylabel("$\\psi$")
```

[23]:
```
Text(0, 0.5, '$\\psi$')
```




[24]:
```
plt.plot(Es,scores);
plt.grid()
plt.xlabel("Energy");
plt.ylabel("Score");
plt.ylim([-20.,20.])
plt.xlim([-25.0,-20.0])
```

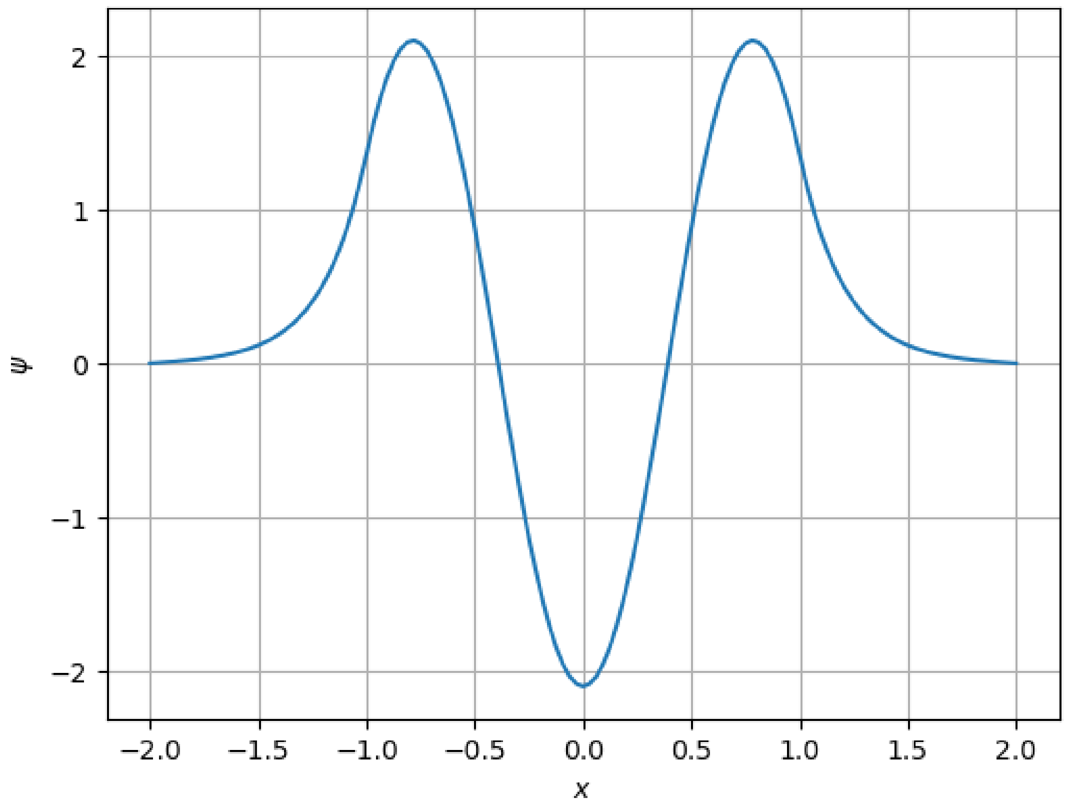
[24]: (-25.0, -20.0)




[25]:
```
Eini=(-25.0,-23.0)
iter,Eeigen = secant(Eini,odefun2,score2,ybound);
xs, ys = caller_rk4(odefun2, xlim, yini, Eeigen, N);
y1 = [y[0] for y in ys];
plt.plot(xs, y1);
plt.grid();
plt.xlabel("$x$")
plt.ylabel("$\\psi$")
```

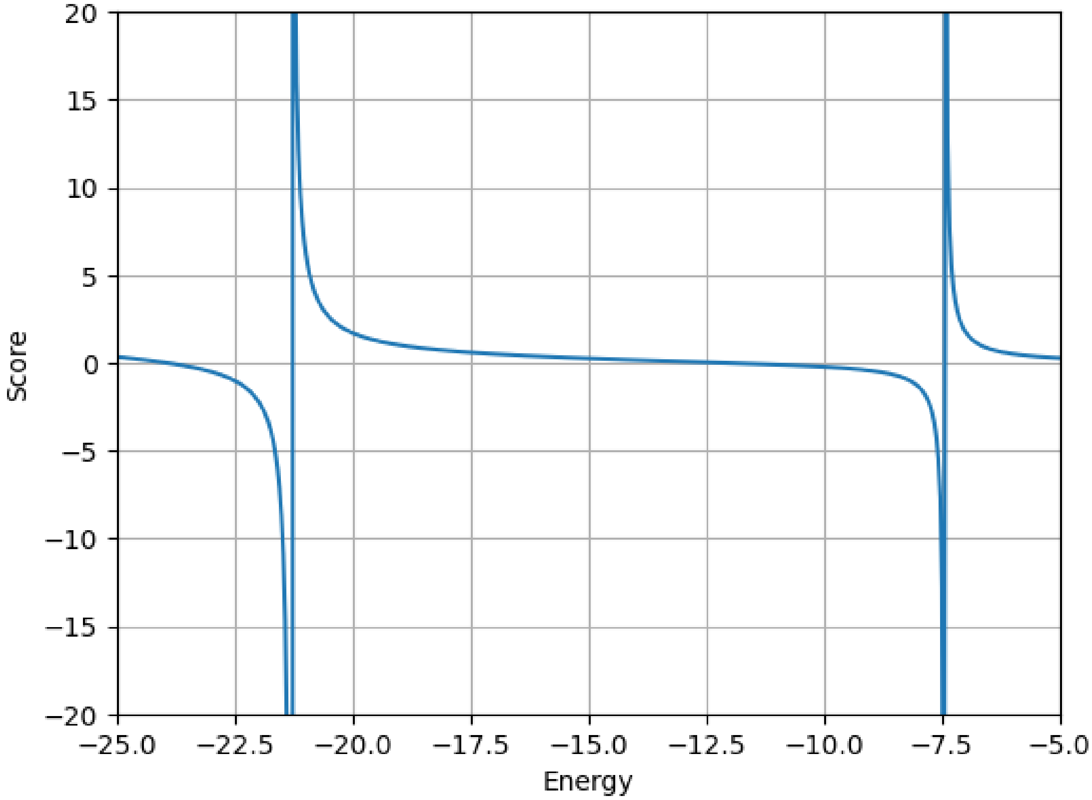
```
1 -23.0 -24.195783608388357 0.08841032702395601
2 -24.195783608388357 -24.0184016481794 0.023322259596669337
3 -24.0184016481794 -23.954842407795113 -0.0012108820415317467
4 -23.954842407795113 -23.957979500626095 1.6512139691893246e-05
5 -23.957979500626095 -23.957937297301488 1.1668482291504745e-08
```

[25]: `Text(0, 0.5, '$\\psi$')`




[28]:
```
plt.plot(Es,scores);
plt.grid()
plt.xlabel("Energy");
plt.ylabel("Score");
plt.ylim([-20.,20.])
plt.xlim([-25.0,-5.0])
```

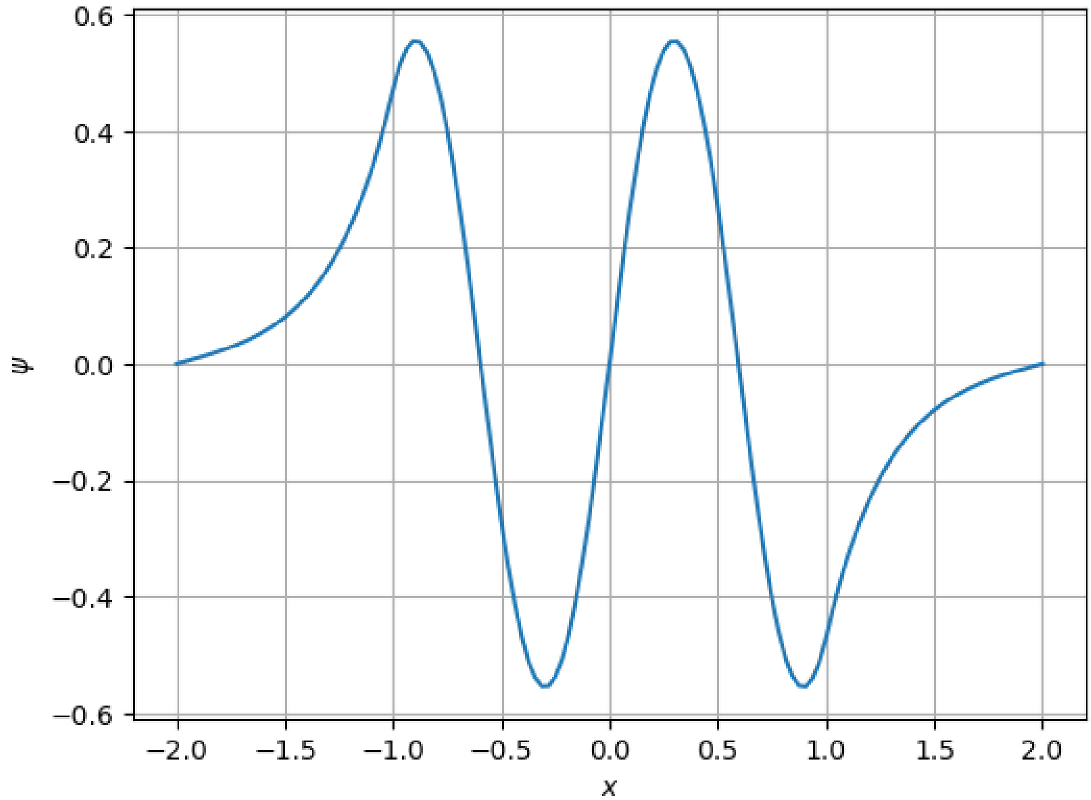
[28]: (-25.0, -5.0)




[29]:
```
Eini=(-15.0,-10.0)
iter,Eeigen = secant(Eini,odefun2,score2,ybound);
xs, ys = caller_rk4(odefun2, xlim, yini, Eeigen, N);
y1 = [y[0] for y in ys];
plt.plot(xs, y1);
plt.grid();
plt.xlabel("$x$")
plt.ylabel("$\\psi$")
```

```
1 -10.0 -12.429930864307522 0.017829175891996785
2 -12.429930864307522 -12.256621287514847 0.003291560244823216
3 -12.256621287514847 -12.217381089750448 -1.7044188481973865e-05
4 -12.217381089750448 -12.21758323459437 1.9033394582645968e-08
```

[29]:
```
Text(0, 0.5, '$\\psi$')
```


0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
ψ
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x

```
[ ]:
```