griffiths_introduction_to_qm

July 6, 2024

[2]:
```
"""
Solutions of Selected Problems Related to Quantum Mechanics

References:
===========


"""
import copy
import os
import sys
# Import path for library functions.
lstPaths = ["../../../libphysics/src", "../../../libpython/src"]
lstPaths = ["../../src", "../../../libpython/src"]
for ipath in lstPaths:
    if ipath not in sys.path:
        sys.path.append(ipath)
# The following is not compatible with jupyter-notebook.
# for ipath in lstPaths:
#    if os.path.join(os.path.dirname(__file__), ipath) not in sys.path:
#        sys.path.append(os.path.join(os.path.dirname(__file__), ipath))
from libsympy import *
from sympy.abc import*
from quantum_mechanics import *
import libquantum
import scienceplots
plt.style.use(['science', 'notebook'])
```

```
libquantum was loaded.
```

0.0.1 Settings

[3]:
```
class sets:
    """
    Setttings class.

    Instead of settings class, settings nametuble might be used.
    Settings = namedtuple("Settings", "type dropinf delta")
```

```
    sets = Settings(type="symbolic", dropinf=True, delta=0.1)
    """
    global dictflow, test_all

    def __init__(self):
        pass

    # File settings
    input_dir  = "input/quantum_mechanics"
    output_dir = "output/quantum_mechanics"

    # Plotting settings
    plot_time_scale = {1:"xy", 2:"xz", 3:"yz"}[3]

    # Execution settings.
    use_libphysics = {0:False, 1:True}[1]
    test_all = {0:False, 1:True}[1]
    dictflow = dict(
        ch1 = {13:"p1.3",15:"p1.5",19:"p1.9",17:"p1.17"},
        ch2 = {24:"p2.4",27:"p2.7",29:"p2.9",25:"e2.5",211:"p2.11",212:"p2.12",
               232:"ch2.3.2",260:"e2.6",222:"p2.22",260:"ch2.6",233:"p2.33",241:
↪"p2.41"},
        ch3 = {322:"p3.22", 330:"p3.30"},
        ch4 = {401:"p4.1",402:"e4.1",403:"e4.3",420:"ch4.2",421:"ch4.2.1",411:
↪"p4.11",
               412:"p4.12",404:"fig4.4",
               413:"p4.13",414:"p4.14",415:"p4.15",9:"ch4.3.1",
               430:"e4.3",
               440:"ch4.4",441:"ch4.4.1",
               11:"e4.2",427:"p4.27",449:"p4.49",16:"p4.55 todo"},
        ch5 = {1:"p5.1 todo"},
        ch6 = {61:"p6.1",62:"p6.2",6310:"ch6.3.1"},
        ch7 = {701:"e7.1",702:"e7.2"})
    flow = [dictflow["ch4"][i] for i in [430]]
    if test_all: flow = flatten([list(dictflow[i].values()) for i in dictflow.
↪keys()])
```

[34]:
```
print("Test of the {0}.".format(sets.flow))
```

```
Test of the ['p1.3', 'p1.5', 'p1.9', 'p1.17', 'p2.4', 'p2.7', 'p2.9', 'e2.5',
'p2.11', 'p2.12', 'ch2.3.2', 'ch2.6', 'p2.22', 'p2.33', 'p2.41', 'p3.22',
'p3.30', 'p4.1', 'e4.1', 'e4.3', 'ch4.2', 'ch4.2.1', 'p4.11', 'p4.12', 'fig4.4',
'p4.13', 'p4.14', 'p4.15', 'ch4.3.1', 'e4.3', 'ch4.4', 'ch4.4.1', 'e4.2',
'p4.27', 'p4.49', 'p4.55 todo', 'p5.1 todo', 'p6.1', 'p6.2', 'ch6.3.1', 'e7.1',
'e7.2'].
```

0.0.2 get_formulary

[ ]:
```
if "get_formulary" in sets.flow:
    omec.__init__()
    omec.get_formulary()
    omec.get_formulary(style="eq")
```

0.0.3 get_subformulary

[ ]:
```
if "get_subformulary" in sets.flow:
    omech.__init__()
    omech.get_subformulary()
```

0.1 Chapter 1 The Wave Function

0.1.1 1.1 The Schrodinger Equation

0.1.2 1.2 The Statistical Interpretation

0.1.3 1.3 Probability

0.1.4 —-> p1.3

[4]:
```
#----> p1.3
if "p1.3" in sets.flow:
    oqmec.__init__("position_space")
    oqmec.verbose = True

    [A,a,l] = symbols('A a lambda', real=True, positive=True)
    psi = Wavefunction(sqrt(A*exp(-l*(x-a)**2)), x)
    npsi = psi.normalize()
    normconst = psi.norm

    substitutions = {oqmec.Psi:npsi.expr, xmin:-Inf, xmax:Inf}
    expX_1 = oqmec.exp_x.evalf(subs=substitutions)
    expX_2 = oqmec.exp_x.evalf(subs=substitutions).doit()
    expX2_1 = oqmec.exp_x2.evalf(subs=substitutions)
    expX2_2 = oqmec.exp_x2.evalf(subs=substitutions).doit()
    deltaX2 = oqmec.delta_x2.evalf(subs=substitutions).doit()

    pprints("~p1.3:",
            "Probability distribution",
            "a)",
            "psi=", psi,
            "normalised psi=", npsi.expr,
            "normalization constant=", simplify(normconst),
            "normconst**2=", normconst**2,
            "|A|^2=", solve(normconst**2-1, A**2),
```

```
            "b)",
            "<x>",   oqmec.exp_x, expX_1,  expX_2,
            "<x^2>", oqmec.exp_x2, expX2_1, expX2_2,

            "c)",
            "sigmaX2 = DeltaX2=", oqmec.delta_x2, deltaX2,
            "sigmaX=", sqrt(deltaX2.rhs),

            output_style="display")
```

'~p1.3:'

'Probability distribution'

'a)'

'psi='

Wavefunction $\left(\sqrt{A}e^{-\frac{\lambda(-a+x)^2}{2}}, x\right)$

'normalised psi='

$$\frac{\sqrt[4]{\lambda}e^{-\frac{\lambda(-a+x)^2}{2}}}{\sqrt[4]{\pi}}$$

'normalization constant='

$$\frac{\sqrt[4]{\pi}\sqrt{A}}{\sqrt[4]{\lambda}}$$

'normconst**2='

$$\frac{\sqrt{\pi}A}{\sqrt{\lambda}}$$

'|A|^2='

$$\left[\frac{\lambda}{\pi}\right]$$

'b)'

'<x>'

$$\langle x\rangle = \int_{x_{min}}^{x_{max}} x\Psi(x,y,z,t)\overline{\Psi(x,y,z,t)}\,dx$$

$$\langle x\rangle = \int_{-\infty}^{\infty} \frac{\sqrt{\lambda}xe^{-\lambda(-a+x)^2}}{\sqrt{\pi}}\,dx$$

$$\langle x\rangle = a$$

'<x^2>'

$$\langle x^2 \rangle = \int_{x_{min}}^{x_{max}} x^2 \Psi(x,y,z,t) \overline{\Psi(x,y,z,t)}\, dx$$

$$\langle x^2 \rangle = \int_{-\infty}^{\infty} \frac{\sqrt{\lambda} x^2 e^{-\lambda(-a+x)^2}}{\sqrt{\pi}}\, dx$$

$$\langle x^2 \rangle = a^2 + \frac{1}{2\lambda}$$

'c)'

'sigmaX2 = DeltaX2='

$$(\Delta x)^2 = \langle x^2 \rangle - \langle x \rangle^2 = -\left(\int_{x_{min}}^{x_{max}} x \Psi(x,y,z,t) \overline{\Psi(x,y,z,t)}\, dx \right)^2 + \int_{x_{min}}^{x_{max}} x^2 \Psi(x,y,z,t) \overline{\Psi(x,y,z,t)}\, dx$$

$$(\Delta x)^2 = \langle x^2 \rangle - \langle x \rangle^2 = \frac{1}{2\lambda}$$

'sigmaX='

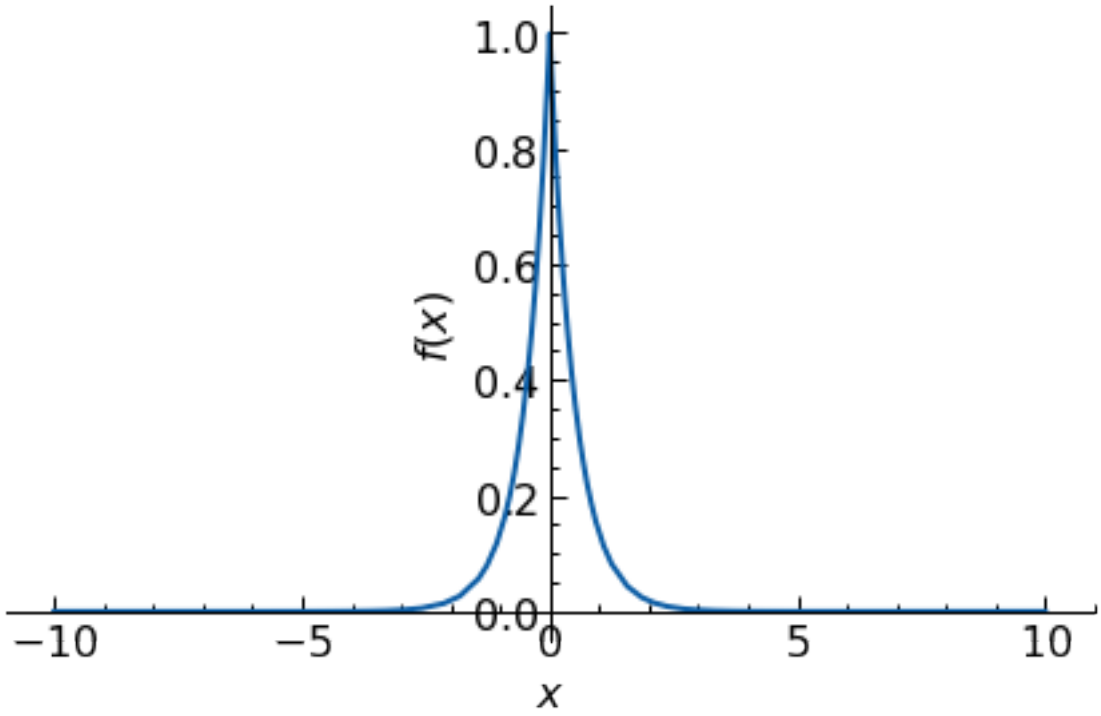
$$\frac{\sqrt{2}}{2\sqrt{\lambda}}$$

0.1.5 1.4 Normalization

0.1.6 —-> p1.5

[9]:
```
#----> p1.5
if "p1.5" in sets.flow:
    if sets.use_libphysics:
        oqmec.__init__("position_space")
        oqmec.verbose = True

    else:
        [A,l,w] = symbols('A lambda w', real=True, positive=True)
        psi = A*exp(-l*abs(x))*exp(-I*w*t)
        Ipsi = integrate(psi*conjugate(psi),(x,-oo, oo))
        solA = solve(Ipsi-1, A)
        npsi = psi.subs({A:solA[0]})
        expX = integrate(conjugate(npsi)*x*npsi,(x,-oo, oo))
        expX2 = integrate(conjugate(npsi)*x**2*npsi,(x,-oo, oo))
        sigmaX = sqrt(expX2-expX**2)

        pprints("p1.5: 1. Way:",
                "a)",
                "psi=", psi,
                "psi*=", conjugate(psi),
```

```
                "Ipsi=", Ipsi,
                "A=", solA,
                "Normalized psi=", npsi,

                "b)",
                "<x>=<psi|x|psi>=", expX,
                "<x^2>=<psi|x^2|psi>", expX2,

                "c)",
                "sigma=", sigmaX,
                "|psi(sigma)|^2=", (conjugate(npsi)*npsi).subs({x:sigmaX}),
                "plot of |psi|^2=", plot((conjugate(npsi)*npsi).subs({w:1, l:
↪1})),
                output_style="display"
                )

        print("\r\n","p1.5: 2. Way:")
        psi = Wavefunction(A*exp(-l*abs(x))*exp(-I*w*t), x)
        npsi = psi.normalize()
        expX = integrate(conjugate(npsi.expr)*x*npsi.expr, (x,-oo, oo))
        expX2 = integrate(conjugate(npsi.expr)*x**2*npsi.expr, (x,-oo, oo))
        pprints("a)",
                "psi=",psi,
                "psi*=",conjugate(psi),
                "psi.variables=",npsi.variables,
                "Normalized psi=",npsi,
                "Normalized psi.expr=",npsi.expr,

                "b)",
                "<x>=",expX,
                "<x^2>=",expX2,

                "c)",
                "sigma=",sigmaX,
                "|psi(sigma)|^2=",(npsi.prob().subs({x:sigmaX})).expr,
                output_style="display")
```



'p1.5: 1. Way:'

'a)'

'psi='

$Ae^{-\lambda|x|}e^{-itw}$

'psi*='

$Ae^{-\lambda|x|}e^{itw}$

'Ipsi='

$\frac{A^2}{\lambda}$

'A='

$\left[\sqrt{\lambda}\right]$

'Normalized psi='

$\sqrt{\lambda}e^{-\lambda|x|}e^{-itw}$

'b)'

'<x>=<psi|x|psi>='

0

'<x^2>=<psi|x^2|psi>'

$$\frac{1}{2\lambda^2}$$

'c)'

'sigma='

$$\frac{\sqrt{2}}{2\lambda}$$

'|psi(sigma)|^2='

$$\frac{\lambda}{e^{\sqrt{2}}}$$

'plot of |psi|^2='

<sympy.plotting.plot.Plot at 0x7ff49ad02880>

p1.5: 2. Way:

'a)'

'psi='

Wavefunction $\left(Ae^{-\lambda|x|}e^{-itw}, x\right)$

'psi*='

Wavefunction $\left(Ae^{-\lambda|x|}e^{itw}, x\right)$

'psi.variables='

$(x,)$

'Normalized psi='

Wavefunction $\left(\sqrt{\lambda}e^{-\lambda|x|}e^{-itw}, x\right)$

'Normalized psi.expr='

$\sqrt{\lambda}e^{-\lambda|x|}e^{-itw}$

'b)'

'<x>='

0

'<x^2>='

$$\frac{1}{2\lambda^2}$$

'c)'

'sigma='

$$\frac{\sqrt{2}}{2\lambda}$$

'|psi(sigma)|^2='

$$\frac{\lambda}{e^{\sqrt{2}}}$$

0.1.7 1.5 Momentum

0.1.8 1.6 The Uncertainty Principle

0.1.9 —-> p1.9

[12]:
```
#----> p1.9
if "p1.9" in sets.flow:
    if sets.use_libphysics:
        oqmec.__init__("position_space")
        [A,a,m] = symbols('A a m', real=True, positive=True)
        psi = Wavefunction(A*exp(-a*((m*x**2/hbar)+I*t)), x)
        npsi = psi.normalize()
        solA = solve(psi.norm-1, A)
        substitutions = {oqmec.Psi:npsi.expr, xmin:-Inf, xmax:Inf}

        schrodingerEq = oqmec.SchrodingerEq.subs(substitutions).doit()
        solV = solve(schrodingerEq, oqmec.V)
        expX_1 = oqmec.exp_x.xreplace(substitutions)
        expX_2 = oqmec.exp_x.xreplace(substitutions).doit()
        expX2_1 = oqmec.exp_x2.xreplace(substitutions)
        expX2_2 = oqmec.exp_x2.xreplace(substitutions).doit()
        expP_1  = oqmec.exp_px.xreplace(substitutions)
        expP_2  = oqmec.exp_px.xreplace(substitutions).doit()
        expP2_2  = oqmec.exp_px2.xreplace(substitutions).doit()
        sigmaX = oqmec.delta_x.xreplace(substitutions).doit().rhs
        sigmaP = oqmec.delta_px.xreplace(substitutions).doit().rhs

        expX  = expX_2
        expX2 = expX2_2
        expP  = expP_2
        expP2 = expP2_2

    else:
        [A,a,m] = symbols('A a m', real=True, positive=True)
        psi = A*exp(-a*((m*x**2/hbar)+I*t))
        Ipsi = integrate(psi*conjugate(psi),(x,-oo, oo))
        solA = solve(Ipsi-1, A)

        print("p1.9: 1. Way:")
        pprints("a)",
                "psi=", psi,
```

```
                "psi*=", conjugate(psi),
                "Ipsi=", Ipsi,
                "A=", solA,
                output_style="display")

        print("p1.9: 2. Way:")
        psi = A*exp(-a*(m*x**2/hbar+I*t))
        psi = Wavefunction(psi, x)
        npsi = psi.normalize()
        solA = solve(psi.expr-npsi.expr, A)

        #V=symbols('V', cls=Function)
        V = Function('V')
        schrodingerEq = Eq(-(hbar**2)/(2*m)*diff(npsi.expr, x, 2) + V(x)*npsi.
↪expr, I*hbar*diff(npsi.expr, t, 1))
        solV = solve(schrodingerEq, V(x))

        expX   = integrate(conjugate(npsi.expr)*x*npsi.expr,(x,-oo, oo))
        expX2  = integrate(conjugate(npsi.expr)*x**2*npsi.expr,(x,-oo, oo))
        expP   = integrate(conjugate(npsi.expr)*(hbar/I)*diff(npsi.expr, x,␣
↪1),(x,-oo, oo))
        expP2  = integrate(conjugate(npsi.expr)*(hbar/I)*diff((hbar/I*diff(npsi.
↪expr, x, 1)),x, 1),(x,-oo, oo))
        sigmaX = sqrt(expX2-expX**2)
        sigmaP = sqrt(expP2-expP**2)

    pprints("a)",
            "psi=", psi,
            "psi*=", conjugate(psi),
            "psi.variables=", npsi.variables,
            "Normalized psi=", npsi,
            "Normalized psi.expr=", npsi.expr.simplify(),
            "Normalization=", npsi.norm,
            "A=", solA,

            "b)",
            "Schrödinger Equation=", schrodingerEq,
            "V(x)=", solV,

            "c)",
            "<x>=", expX,
            "<x^2>=", expX2,
            "<p>=",
            "<psi|-I*hb d/dx()|psi>=",
            "<psi|hb/I d/dx()|psi>=", expP,
            "<p^2>=", expP2,
```

```
            "d)",
            "sigmaX=", sigmaX,
            "sigmaP=", sigmaP,
            "sigmaX*sigmaP=", sigmaX*sigmaP,
            output_style="display")
```

'a)'

'psi='

Wavefunction $\left(Ae^{-a\left(\frac{mx^2}{\hbar}+it\right)}, x\right)$

'psi*='

Wavefunction $\left(Ae^{-a\left(\frac{mx^2}{\hbar}-it\right)}, x\right)$

'psi.variables='

$(x,)$

'Normalized psi='

Wavefunction $\left(\frac{\sqrt[4]{2}\sqrt[4]{a}\sqrt[4]{m}e^{-a\left(\frac{mx^2}{\hbar}+it\right)}}{\sqrt[4]{\hbar}\sqrt[4]{\pi}}, x\right)$

'Normalized psi.expr='

$$\frac{\sqrt[4]{2}\sqrt[4]{a}\sqrt[4]{m}e^{-\frac{a\left(mx^2+\hbar it\right)}{\hbar}}}{\sqrt[4]{\hbar}\sqrt[4]{\pi}}$$

'Normalization='

1

'A='

$$\left[\frac{\sqrt[4]{2}\sqrt[4]{a}\sqrt[4]{m}}{\sqrt[4]{\hbar}\sqrt[4]{\pi}}\right]$$

'b)'

'Schrödinger Equation='

$$-\frac{\sqrt[4]{2}\hbar^{\frac{3}{4}}a^{\frac{5}{4}}m^{\frac{5}{4}}\cdot\left(\frac{2amx^2}{\hbar}-1\right)e^{-a\left(\frac{mx^2}{\hbar}+it\right)}}{\sqrt[4]{\pi}m}+\frac{\sqrt[4]{2}\sqrt[4]{a}\sqrt[4]{m}V(x)e^{-a\left(\frac{mx^2}{\hbar}+it\right)}}{\sqrt[4]{\hbar}\sqrt[4]{\pi}}=\frac{\sqrt[4]{2}\hbar^{\frac{3}{4}}a^{\frac{5}{4}}\sqrt[4]{m}e^{-a\left(\frac{mx^2}{\hbar}+it\right)}}{\sqrt[4]{\pi}}$$

'V(x)='

$$\left[\frac{a\left(2am^2x^2+\hbar m-\hbar m\right)}{m}\right]$$

'c)'

'<x>='

$\langle x \rangle = 0$

'<x^2>='

$\langle x^2 \rangle = \frac{\hbar}{4am}$

'<p>='

'<psi|-I*hb d/dx()|psi>='

'<psi|hb/I d/dx()|psi>='

$\langle p_x \rangle = 0$

'<p^2>='

$\langle p_x^2 \rangle = \hbar am$

'd)'

'sigmaX='

$\frac{\sqrt{\hbar}}{2\sqrt{a}\sqrt{m}}$

'sigmaP='

$\sqrt{\hbar}\sqrt{a}\sqrt{m}$

'sigmaX*sigmaP='

$\frac{\hbar}{2}$

0.1.10 —-> p1.17

[13]:
```
#----> p1.17
if "p1.17" in sets.flow:

    if sets.use_libphysics:
        oqmec.__init__("position_space")
        oqmec.verbose = True

        [A,a,m] = symbols('A a m', real=True, positive=True)
        f = Piecewise((0, x < -a), (0, x > a), (A*(a**2-x**2), True))
        psi = Wavefunction(f, x)
        npsi = psi.normalize()
        solA = solve(psi.norm-1, A)[0]

        substitutions = {oqmec.Psi:npsi.expr, xmin:-a, xmax:a}
        expX_1 = oqmec.exp_x.evalf(subs=substitutions)
```

```
        expX_2 = oqmec.exp_x.evalf(subs=substitutions).doit()
        expX2_1 = oqmec.exp_x2.evalf(subs=substitutions)
        expX2_2 = oqmec.exp_x2.evalf(subs=substitutions).doit()

        exp_px_1 = oqmec.exp_px.evalf(subs=substitutions)
        exp_px_2 = oqmec.exp_px.evalf(subs=substitutions).doit()
        exp_px2_1 = oqmec.exp_px2.evalf(subs=substitutions)
        exp_px2_2 = oqmec.exp_px2.evalf(subs=substitutions).doit()
        exp_px2_3= oqmec.exp_px2.xreplace(substitutions).doit()

        commands = ["xreplace", "oqmec.exp_px2", substitutions]
        exp_px2_4 = oqmec.process(commands).doit()

        sigmaX = sqrt(expX2_2.rhs-expX_2.rhs**2)
        sigmaP = sqrt(exp_px2_2.rhs-exp_px_2.rhs**2)

        pprints("p1.17",
                "a)",
                "psi=", psi,
                "npsi=", npsi,
                "Normalization constant, A=", solA,
                "b)",
                "<x>=<psi|x|psi>=", oqmec.exp_x, expX_1, expX_2,
                "c)",
                "<p>=", oqmec.exp_px, exp_px_1, exp_px_2,
                "d)",
                "<x^2>=<psi|x^2|psi>=", oqmec.exp_x2, expX2_1, expX2_2,
                "e)",
                "<p^2>=", oqmec.exp_px2, exp_px2_1, exp_px2_3,
                "f)",
                "sigma_X=<x^2>-<x>^2=", sigmaX,
                "g)",
                "sigma_p_x=", sigmaP,
                "h)",
                "sigma_X*sigma_P=", sigmaX*sigmaP,
                output_style="display")

    else:
        [A,a,m] = symbols('A a m', real=True, positive=True)
        f = Piecewise((0, x < -a), (0, x > a), (A*(a**2-x**2), True))
        psi = Wavefunction(f, x)
        npsi = psi.normalize()
        solA = solve(psi.norm-1, A)[0]
        expX = integrate(conjugate(npsi.expr)*x*npsi.expr,(x,-a, a))
        expX2 = integrate(conjugate(npsi.expr)*x**2*npsi.expr,(x,-a, a))
        expP = integrate(conjugate(npsi.expr)*(hbar/I)*diff(npsi.expr, x,␣
↪1),(x,-a, a))
```

```
        expP2 = integrate(conjugate(npsi.expr)*(hbar/I)*diff((hbar/I*diff(npsi.
↪expr, x, 1)),x, 1),(x,-a, a))
        sigmaX = sqrt(expX2-expX**2)
        sigmaP = sqrt(expP2-expP**2)

        pprints("p1.17",
                "a)",
                "psi=", psi,
                "npsi=", npsi,
                "Normalization constant, A=", solA,

                "b)",
                "<x>=", expX,
                "<x>=libquantum.expX(npsi,(-a, a))=", libquantum.expX(npsi,(-a,␣
↪a)),

                "c)",
                "<p>=", expP,

                "d)",
                "<x^2>=", expX2,
                "<x^2>=libquantum.expX2(npsi,(-a, a))=", libquantum.
↪expX2(npsi,(-a, a)),

                "e)",
                "<p^2>=", expP2,
                "<p^2>=libquantum.expP2(npsi,(-a, a))=",libquantum.
↪expP2(npsi,(-a, a)),

                "f)",
                "\sigma_X=<x^2>-<x>^2=", sigmaX,
                "\sigma_X=<x^2>-<x>^2=", libquantum.sigmaX(npsi,(-a, a)),

                "g)",
                "\sigma_p=", sigmaP,
                "\sigma_p=", libquantum.sigmaP(npsi,(-a, a)),

                "h)",
                "\sigma_X \sigma_P=", sigmaX*sigmaP,
                "\sigmaX \sigma_P=", libquantum.sigmaXsigmaP(npsi,(-a, a)),
                output_style="display")
```

```
'xreplace oqmec.exp_px2 {Psi(x, y, z, t): sqrt(15)*Piecewise((0, (a < x) | (a <␣
 ↪-x)), (A*(a**2 - x**2), True))/(4*A*a**(5/2)), x_{min}: -a, x_{max}: a}'
```

```
Eq(\langle{p_x^2}\rangle, Integral(-hbar**2*conjugate(Psi(x, y, z,
t))*Derivative(Psi(x, y, z, t), (x, 2)), (x, x_{min}, x_{max})))(xreplace,
```

```
{Psi(x, y, z, t): sqrt(15)*Piecewise((0, (a < x) | (a < -x)), (A*(a**2 - x**2),
True))/(4*A*a**(5/2)), x_{min}: -a, x_{max}: a}
```

$$\langle p_x^2 \rangle = \int_{-a}^{a} \begin{cases} 0 & \text{for } a < -x \vee a < x \\ -\frac{\sqrt{15}\hbar^2\left(a^2-x^2\right)\frac{\partial^2}{\partial x^2}\frac{\sqrt{15}\left(a^2-x^2\right)}{4a^{\frac{5}{2}}}}{4a^{\frac{5}{2}}} & \text{otherwise} \end{cases} dx$$

```
'p1.17'
```

```
'a)'
```

```
'psi='
```

$$\text{Wavefunction}\left(\begin{cases} 0 & \text{for } a < -x \vee a < x \\ A\left(a^2 - x^2\right) & \text{otherwise} \end{cases}, x\right)$$

```
'npsi='
```

$$\text{Wavefunction}\left(\frac{\sqrt{15}\left(\begin{cases} 0 & \text{for } a < -x \vee a < x \\ A\left(a^2 - x^2\right) & \text{otherwise} \end{cases}\right)}{4Aa^{\frac{5}{2}}}, x\right)$$

```
'Normalization constant, A='
```

$$\frac{\sqrt{15}}{4a^{\frac{5}{2}}}$$

```
'b)'
```

```
'<x>=<psi|x|psi>='
```

$$\langle x \rangle = \int_{x_{min}}^{x_{max}} x\Psi(x,y,z,t)\overline{\Psi(x,y,z,t)}\,dx$$

$$\langle x \rangle = \int_{-a}^{a} \begin{cases} 0 & \text{for } a < -x \vee a < x \\ \frac{15x\left(a^2-x^2\right)^2}{16a^5} & \text{otherwise} \end{cases} dx$$

$$\langle x \rangle = 0$$

```
'c)'
```

```
'<p>='
```

$$\langle p_x \rangle = \int_{x_{min}}^{x_{max}} \left(-\hbar i\overline{\Psi(x,y,z,t)}\frac{\partial}{\partial x}\Psi(x,y,z,t)\right) dx$$

$$\langle p_x \rangle = \int_{-a}^{a} \begin{cases} 0 & \text{for } a < -x \vee a < x \\ -\frac{\sqrt{15}\hbar i\left(a^2-x^2\right)\frac{\partial}{\partial x}\frac{\sqrt{15}\left(a^2-x^2\right)}{4a^{\frac{5}{2}}}}{4a^{\frac{5}{2}}} & \text{otherwise} \end{cases} dx$$

$$\langle p_x \rangle = 0$$

```
'd)'
```

```
'<x^2>=<psi|x^2|psi>='
```

$$\langle x^2 \rangle = \int_{x_{min}}^{x_{max}} x^2 \Psi(x,y,z,t) \overline{\Psi(x,y,z,t)}\, dx$$

$$\langle x^2 \rangle = \int_{-a}^{a} \begin{cases} 0 & \text{for } a < -x \vee a < x \\ \frac{15x^2\left(a^2-x^2\right)^2}{16a^5} & \text{otherwise} \end{cases} dx$$

$$\langle x^2 \rangle = \frac{a^2}{7}$$

```
'e)'
```

```
'<p^2>='
```

$$\langle p_x^2 \rangle = \int_{x_{min}}^{x_{max}} \left(-\hbar^2 \overline{\Psi(x,y,z,t)} \frac{\partial^2}{\partial x^2} \Psi(x,y,z,t) \right) dx$$

$$\langle p_x^2 \rangle = \int_{-a}^{a} \begin{cases} 0 & \text{for } a < -x \vee a < x \\ -\frac{\sqrt{15}\hbar^2\left(a^2-x^2\right)\frac{\partial^2}{\partial x^2}\frac{\sqrt{15}\left(a^2-x^2\right)}{4a^{\frac{5}{2}}}}{4a^{\frac{5}{2}}} & \text{otherwise} \end{cases} dx$$

$$\langle p_x^2 \rangle = \frac{5\hbar^2}{2a^2}$$

```
'f)'
```

```
'sigma_X=<x^2>-<x>^2='
```

$$\frac{\sqrt{7}a}{7}$$

```
'g)'
```

```
'sigma_p_x='
```

$$\frac{\sqrt{10}\hbar}{2a}$$

```
'h)'
```

```
'sigma_X*sigma_P='
```

$$\frac{\sqrt{70}\hbar}{14}$$

0.2 Chapter 2 Time-Independent Schrodinger Equation

0.2.1 2.1 Stationary States

0.2.2 2.2 The Infinite Square Well

0.2.3 2.3 The Harmonic Oscillator

0.2.4 2.3.1 Algebraic Method

0.2.5 2.3.2 Analytic Method

0.2.6 2.4 The Free Particle

0.2.7 2.5 The Delta-Function Potential

0.2.8 2.5.1 Bound States and Scattering States

0.2.9 2.5.2 The Delta-Function Well

0.2.10 2.6 The Finite Square Well

0.2.11 2.2 The Infinite Square Well

0.2.12 —-> p2.4

[14]:
```
#----> p2.4 todo
if "p2.4" in sets.flow:

    if sets.use_libphysics:
        oqmec.__init__("position_space")
        oqmec.verbose = True

        psi = {1:Wavefunction(oqmec.iqw.psix().rhs, x).expr,
               2:oqmec.iqw.psix().rhs}[2]
        substitutions = {Psi:psi, xmin:0, xmax:a}

        exp_x  = oqmec.exp_x.xreplace(substitutions)
        exp_x2 = oqmec.exp_x2.xreplace(substitutions)
        exp_px = oqmec.exp_px.xreplace(substitutions)
        exp_px2= oqmec.exp_px2.xreplace(substitutions)
        delta_x = oqmec.delta_x.xreplace(substitutions)
        delta_px = oqmec.delta_px.xreplace(substitutions)
        delta_XP = oqmec.delta_XP.xreplace(substitutions)
        min_deltaXP = simplify(delta_XP.doit()).subs({n:1})

        pprints(
            "p2.4",
            "1. Way: SymPy derivative function used in operator definitions.",
            "psi=", psi,
            "psi*=", conjugate(psi),
            "<x>=",     exp_x,  exp_x.doit(),  simplify(exp_x.doit()),
            "<x^2>=",   exp_x2, exp_x2.doit(), simplify(exp_x2.doit()),
```

```
            "<p_x>=",   exp_px, exp_px.doit(), simplify(exp_px.doit()),
            "<p_x^2>=", exp_px2, exp_px2.doit(), simplify(exp_px2.doit()),
            "delta x=",  delta_x, delta_x.doit(), simplify(delta_x.doit()),
            "delta p_x=",delta_px, delta_px.doit(), simplify(delta_px.doit()),
            "deltaX*deltaP=", delta_XP, delta_XP.doit(), simplify(delta_XP.
↪doit()),
            "At n=1, uncertaninty becomes minimum=", min_deltaXP,␣
↪simplify(min_deltaXP),
            output_style="display")

        exp_x  = oqmec.exp_xop.xreplace(substitutions)
        exp_x2 = oqmec.exp_x2op.xreplace(substitutions)
        exp_fx = oqmec.exp_fx(x**2).xreplace(substitutions) # Calculate by␣
↪function call.
        exp_px = oqmec.exp_pxop.xreplace(substitutions)
        exp_px2= oqmec.exp_px2op.xreplace(substitutions)
        delta_x = oqmec.delta_xop.xreplace(substitutions)
        delta_px = oqmec.delta_pxop.xreplace(substitutions)
        delta_XP = oqmec.delta_xop_pxop.xreplace(substitutions)
        min_deltaXP = delta_XP.doit().subs({n:1})

        # todo check operator formalism
       ␣
↪pprints("================================================================================
                "2. Way: DifferentialOperator used from sympy.physics.quantum.
↪operator in operator definitions.",
              ␣
↪"================================================================================
                "psi=", psi,
                "psi*=", conjugate(psi),
                "<x>=",     exp_x,  exp_x.doit(),  simplify(exp_x.doit()),
                "<x^2>=",   exp_x2, exp_x2.doit(), simplify(exp_x2.doit()),
                "<x^2>=",   exp_fx, exp_fx.doit(), simplify(exp_fx.doit()),
                "<p_x>=",   exp_px, exp_px.doit(), simplify(exp_px.doit()),
                "<p_x^2>=", exp_px2, exp_px2.doit(), simplify(exp_px2.doit()),
                "delta x=",  delta_x, delta_x.doit(), simplify(delta_x.doit()),
                "delta p_x=",delta_px, delta_px.doit(), simplify(delta_px.
↪doit()),
                "deltaX*deltaP=", delta_XP, delta_XP.doit(),
                "At n=1, uncertaninty becomes minimum=", min_deltaXP,␣
↪simplify(min_deltaXP),
                output_style="display")
    else:
        [A,a,m] = symbols('A a m', real=True, positive=True)
        n = symbols('n', positive = True, integer = True)
        #global_assumptions.add(Q.is_true(n>0))
```

```
        psi = sqrt(2/a)*sin(n*pi*x/a)
        bounds=(0, a)
        expX = libquantum.expX(psi, bounds)
        expX2 = libquantum.expX2(psi, bounds)
        expP = libquantum.expP(psi, bounds)
        expP2 = libquantum.expP2(psi, bounds)
        (sigmaX, sigmaP)=(libquantum.sigmaX(psi, bounds),libquantum.sigmaP(psi,␣
↪bounds))
        uncert = libquantum.sigmaXsigmaP(psi, bounds)
        minUncert = simplify(uncert).subs({n:1})
```

'p2.4'

'1. Way: SymPy derivative function used in operator definitions.'

'psi='

$$\frac{\sqrt{2}\sin\left(\frac{\pi n x}{a}\right)}{\sqrt{a}}$$

'psi*='

$$\frac{\sqrt{2}\sin\left(\frac{\pi n x}{a}\right)}{\sqrt{a}}$$

'<x>='

$$\langle x\rangle = \int_0^a \frac{2x\sin^2\left(\frac{\pi n x}{a}\right)}{a}\,dx$$

$$\langle x\rangle = \frac{a}{2}$$

$$a = 2\langle x\rangle$$

'<x^2>='

$$\langle x^2\rangle = \int_0^a \frac{2x^2\sin^2\left(\frac{\pi n x}{a}\right)}{a}\,dx$$

$$\langle x^2\rangle = \frac{2\left(\frac{a^3}{6} - \frac{a^3}{4\pi^2 n^2}\right)}{a}$$

$$\langle x^2\rangle = \frac{a^2}{3} - \frac{a^2}{2\pi^2 n^2}$$

'<p_x>='

$$\langle p_x\rangle = \int_0^a \left(-\frac{\sqrt{2}\hbar i\sin\left(\frac{\pi n x}{a}\right)\frac{\partial}{\partial x}\frac{\sqrt{2}\sin\left(\frac{\pi n x}{a}\right)}{\sqrt{a}}}{\sqrt{a}}\right)dx$$

$$\langle p_x\rangle = 0$$

$\langle p_x \rangle = 0$

```
'<p_x^2>='
```

$$\langle p_x^2 \rangle = \int_0^a \left(-\frac{\sqrt{2}\hbar^2 \sin\left(\frac{\pi n x}{a}\right) \frac{\partial^2}{\partial x^2} \frac{\sqrt{2}\sin\left(\frac{\pi n x}{a}\right)}{\sqrt{a}}}{\sqrt{a}} \right) dx$$

$$\langle p_x^2 \rangle = \frac{\hbar^2 \pi^2 n^2}{a^2}$$

$$\langle p_x^2 \rangle = \frac{\hbar^2 \pi^2 n^2}{a^2}$$

```
'delta x='
```

$$\Delta x = x - \langle x \rangle = \sqrt{-\left(\int_0^a \frac{2x \sin^2\left(\frac{\pi n x}{a}\right)}{a} dx \right)^2 + \int_0^a \frac{2x^2 \sin^2\left(\frac{\pi n x}{a}\right)}{a} dx}$$

$$\Delta x = x - \langle x \rangle = \sqrt{-\frac{a^2}{4} + \frac{2\left(\frac{a^3}{6} - \frac{a^3}{4\pi^2 n^2}\right)}{a}}$$

$$\Delta x = x - \langle x \rangle = \frac{a\sqrt{3\pi^2 n^2 - 18}}{6\pi n}$$

```
'delta p_x='
```

$$\Delta p_x = \sqrt{\int_0^a \left(-\frac{\sqrt{2}\hbar^2 \sin\left(\frac{\pi n x}{a}\right) \frac{\partial^2}{\partial x^2} \frac{\sqrt{2}\sin\left(\frac{\pi n x}{a}\right)}{\sqrt{a}}}{\sqrt{a}} \right) dx - \left(\int_0^a \left(-\frac{\sqrt{2}\hbar i \sin\left(\frac{\pi n x}{a}\right) \frac{\partial}{\partial x} \frac{\sqrt{2}\sin\left(\frac{\pi n x}{a}\right)}{\sqrt{a}}}{\sqrt{a}} \right) dx \right)^2}$$

$$\Delta p_x = \frac{\hbar \pi n}{a}$$

$$\Delta p_x = \frac{\hbar \pi n}{a}$$

```
'deltaX*deltaP='
```

$$\Delta x \Delta p_x = \left(\sqrt{-\left(\int_0^a \frac{2x \sin^2\left(\frac{\pi n x}{a}\right)}{a} dx \right)^2 + \int_0^a \frac{2x^2 \sin^2\left(\frac{\pi n x}{a}\right)}{a} dx} \right) \sqrt{\int_0^a \left(-\frac{\sqrt{2}\hbar^2 \sin\left(\frac{\pi n x}{a}\right) \frac{\partial^2}{\partial x^2} \frac{\sqrt{2}\sin\left(\frac{\pi n x}{a}\right)}{\sqrt{a}}}{\sqrt{a}} \right) dx - \left(\right.}$$

$$\Delta x \Delta p_x = \frac{\hbar \pi n \sqrt{-\frac{a^2}{4} + \frac{2\left(\frac{a^3}{6} - \frac{a^3}{4\pi^2 n^2}\right)}{a}}}{a}$$

$$\Delta x \Delta p_x = \frac{\hbar \sqrt{3\pi^2 n^2 - 18}}{6}$$

```
'At n=1, uncertaninty becomes minimum='
```

$$\Delta x \Delta p_x = \frac{\hbar\sqrt{\pi^2 n^2 - 18}}{6}$$

$$\Delta x \Delta p_x = \frac{\hbar\sqrt{\pi^2 n^2 - 18}}{6}$$

```
---------------------------------------------------------------------------
PolynomialError                           Traceback (most recent call last)
File /usr/local/lib/python3.8/dist-packages/sympy/polys/polytools.py:6766, in␣
 ↪cancel(f, _signsimp, *gens, **args)
   6765     raise PolynomialError()
-> 6766 R, (F, G) = sring((p, q), *gens, **args)
   6767 if not R.ngens:

File /usr/local/lib/python3.8/dist-packages/sympy/polys/rings.py:163, in␣
 ↪sring(exprs, *symbols, **options)
    162 # TODO: rewrite this so that it doesn't use expand() (see poly()).
--> 163 reps, opt = _parallel_dict_from_expr(exprs, opt)
    165 if opt.domain is None:

File /usr/local/lib/python3.8/dist-packages/sympy/polys/polyutils.py:329, in␣
 ↪_parallel_dict_from_expr(exprs, opt)
    328 if any(expr.is_commutative is False for expr in exprs):
--> 329     raise PolynomialError('non-commutative expressions are not supported')
    331 if opt.gens:

PolynomialError: non-commutative expressions are not supported

During handling of the above exception, another exception occurred:

KeyboardInterrupt                         Traceback (most recent call last)
Cell In[14], line 60
     44     min_deltaXP = delta_XP.doit().subs({n:1})
     46     # todo check operator formalism
     47     ␣
 ↪pprints("================================================================================
     48             "2. Way: DifferentialOperator used from sympy.physics.quantum
 ↪operator in operator definitions.",
     49             ␣
 ↪"================================================================================
     50             "psi=", psi,
     51             "psi*=", conjugate(psi),
     52             "<x>=",     exp_x,  exp_x.doit(),  simplify(exp_x.doit()),
     53             "<x^2>=",   exp_x2, exp_x2.doit(), simplify(exp_x2.doit()),
     54             "<x^2>=",   exp_fx, exp_fx.doit(), simplify(exp_fx.doit()),
     55             "<p_x>=",   exp_px, exp_px.doit(), simplify(exp_px.doit()),
     56             "<p_x^2>=", exp_px2, exp_px2.doit(), simplify(exp_px2.doit()),
     57             "delta x=",  delta_x, delta_x.doit(), simplify(delta_x.doit()),
```

```
     58             "delta p_x=",delta_px, delta_px.doit(), simplify(delta_px.
 ↪doit()),
     59             "deltaX*deltaP=", delta_XP, delta_XP.doit(),
---> 60             "At n=1, uncertaninty becomes minimum=", min_deltaXP,␣
 ↪simplify(min_deltaXP),
     61             output_style="display")
     62 else:
     63     [A,a,m] = symbols('A a m', real=True, positive=True)

File /usr/local/lib/python3.8/dist-packages/sympy/simplify/simplify.py:601, in␣
 ↪simplify(expr, ratio, measure, rational, inverse, doit, **kwargs)
    599 _eval_simplify = getattr(expr, '_eval_simplify', None)
    600 if _eval_simplify is not None:
--> 601     return _eval_simplify(**kwargs)
    603 original_expr = expr = collect_abs(signsimp(expr))
    605 if not isinstance(expr, Basic) or not expr.args:  # XXX: temporary hack

File /usr/local/lib/python3.8/dist-packages/sympy/core/relational.py:691, in␣
 ↪Equality._eval_simplify(self, **kwargs)
    689 def _eval_simplify(self, **kwargs):
    690     # standard simplify
--> 691     e = super()._eval_simplify(**kwargs)
    692     if not isinstance(e, Equality):
    693         return e

File /usr/local/lib/python3.8/dist-packages/sympy/core/relational.py:429, in␣
 ↪Relational._eval_simplify(self, **kwargs)
    427 if dif.is_comparable:
    428     v = dif.n(2)
--> 429 elif dif.equals(0):  # XXX this is expensive
    430     v = S.Zero
    431 if v is not None:

File /usr/local/lib/python3.8/dist-packages/sympy/core/expr.py:741, in Expr.
 ↪equals(self, other, failing_expression)
    735     return True
    737 # they aren't the same so see if we can make the difference 0;
    738 # don't worry about doing simplification steps one at a time
    739 # because if the expression ever goes to 0 then the subsequent
    740 # simplification steps that are done will be very fast.
--> 741 diff = factor_terms(simplify(self - other), radical=True)
    743 if not diff:
    744     return True

File /usr/local/lib/python3.8/dist-packages/sympy/simplify/simplify.py:644, in␣
 ↪simplify(expr, ratio, measure, rational, inverse, doit, **kwargs)
    642 expr = _bottom_up(expr, lambda w: getattr(w, 'normal', lambda: w)())
    643 expr = Mul(*powsimp(expr).as_content_primitive())
```

```
--> 644 _e = cancel(expr)
    645 expr1 = shorter(_e, _mexpand(_e).cancel())  # issue 6829
    646 expr2 = shorter(together(expr, deep=True), together(expr1, deep=True))

File /usr/local/lib/python3.8/dist-packages/sympy/polys/polytools.py:6780, in␣
↪cancel(f, _signsimp, *gens, **args)
   6776 if f.is_Add or f.is_Mul:
   6777     c, nc = sift(f.args, lambda x:
   6778         x.is_commutative is True and not x.has(Piecewise),
   6779         binary=True)
-> 6780     nc = [cancel(i) for i in nc]
   6781     return f.func(cancel(f.func(*c)), *nc)
   6782 else:

File /usr/local/lib/python3.8/dist-packages/sympy/polys/polytools.py:6780, in␣
↪<listcomp>(.0)
   6776 if f.is_Add or f.is_Mul:
   6777     c, nc = sift(f.args, lambda x:
   6778         x.is_commutative is True and not x.has(Piecewise),
   6779         binary=True)
-> 6780     nc = [cancel(i) for i in nc]
   6781     return f.func(cancel(f.func(*c)), *nc)
   6782 else:

File /usr/local/lib/python3.8/dist-packages/sympy/polys/polytools.py:6739, in␣
↪cancel(f, _signsimp, *gens, **args)
   6737 f = sympify(f)
   6738 if _signsimp:
-> 6739     f = signsimp(f)
   6740 opt = {}
   6741 if 'polys' in args:

File /usr/local/lib/python3.8/dist-packages/sympy/simplify/simplify.py:406, in␣
↪signsimp(expr, evaluate)
    404 # get rid of an pre-existing unevaluation regarding sign
    405 e = expr.replace(lambda x: x.is_Mul and -(-x) != x, lambda x: -(-x))
--> 406 e = sub_post(sub_pre(e))
    407 if not isinstance(e, (Expr, Relational)) or e.is_Atom:
    408     return e

File /usr/local/lib/python3.8/dist-packages/sympy/simplify/cse_opts.py:46, in␣
↪sub_post(e)
     44 replacements = []
     45 for node in preorder_traversal(e):
---> 46     if isinstance(node, Mul) and \
     47         node.args[0] is S.One and node.args[1] is S.NegativeOne:
     48         replacements.append((node, -Mul._from_args(node.args[2:])))
     49 for node, replacement in replacements:
```

```
KeyboardInterrupt:
```

0.2.13 —-> p2.7

[21]:
```
#----> p2.7
if "p2.7" in sets.flow:
    [A,a,m] = symbols('A a m', real=True, positive=True)
    n = symbols('n', positive = True, integer = True)
    # Wavefuncition at t=0.
    f = Piecewise( (A*x, ((x >= 0) & (x<=a/2)) ),
                   (A*(a-x), ((x >= a/2) & (x<=a)) ) )
    psi = Wavefunction(f, (x, 0, a))
    npsi0 = simplify(psi.normalize())
    # solve(Eq(psi.norm, 1), dic=True)
    solA = solve(psi.norm-1, A)[0]

    # Time dependent normalized wavefunction.
    # todo add c_n to libquantum
    cn = simplify(integrate( conjugate(sqrt(2/a)*sin(n*pi*x/a)) * npsi0.expr,␣
↪(x,0,a) ))
    Psi = Sum(cn*sqrt(2/a)*sin(n*pi*x/a)*exp(-I*n**2*pi**2*hbar*t/(2*m*a**2)),␣
↪(n, 1, oo))
    pn = conjugate(cn)*cn
    En = pi**2*hbar**2*n**2/(2*m*a**2)
    # 1. way <H>
    expH1 = summation(conjugate(cn)*cn*En, (n,1,oo)).doit()

    # 2. way <H>
    expP2 = libquantum.expP2(npsi0, (0,a))
    expH2 = expP2/(2*m)

    pprints("p2.7",
            "a)",
            "solve(Eq(psi.norm, 1), dic=True)",
            "solA = solve(psi.norm-1, A)[0]",
            "A=", solA,
            "\psi(x,0)=", npsi0,
            "Plot of \psi(x,0)(a=1)=", "todo",

            "b)",
            "c_n=", "c_n=integrate(psi_n(x)*f(x)dx=", cn,
            "\Psi(x,t)=", Psi,

            "c) cannot find exact result"
            "P_1=|c_1|^2=", pn.subs({n:1}),
```

```
            "d)",
            "1. way <H>",
            "<H>=sum(|c_n|^2 E_n, (n,1,oo))=", expH1,
            "2. way <H> todo check",
            "<H>=<psi|H|psi>=", "<npsi|p^2>/(2m)|npsi>=", expH2,
            output_style="display")
```

'p2.7'

'a)'

'solve(Eq(psi.norm, 1), dic=True)'

'solA = solve(psi.norm-1, A)[0]'

'A='

$$\frac{2\sqrt{3}}{a^{\frac{3}{2}}}$$

'\\psi(x,0)='

$$\text{Wavefunction}\left(\begin{cases}\frac{2\sqrt{3}x}{a^{\frac{3}{2}}} & \text{for } a \geq 2x \wedge x \geq 0 \\ \frac{2\sqrt{3}(a-x)}{a^{\frac{3}{2}}} & \text{for } a \geq x \wedge a \leq 2x\end{cases}, (x,\ 0,\ a)\right)$$

'Plot of \\psi(x,0)(a=1)='

'todo'

'b)'

'c_n='

'c_n=integrate(psi_n(x)*f(x)dx='

$$\frac{4\sqrt{6}\sin\left(\frac{\pi n}{2}\right)}{\pi^2 n^2}$$

'\\Psi(x,t)='

$$\sum_{n=1}^{\infty} \frac{8\sqrt{3}e^{-\frac{\hbar i\pi^2 n^2 t}{2a^2 m}}\sin\left(\frac{\pi n}{2}\right)\sin\left(\frac{\pi n x}{a}\right)}{\pi^2\sqrt{a}n^2}$$

'c) cannot find exact resultP_1=|c_1|^2='

$$\frac{96}{\pi^4}$$

'd)'

'1. way <H>'

'<H>=sum(|c_n|^2 E_n, (n,1,oo))='

$$\sum_{n=1}^{\infty} \frac{48\hbar^2 \sin^2\left(\frac{\pi n}{2}\right)}{\pi^2 a^2 m n^2}$$

'2. way <H> todo check'

'<H>=<psi|H|psi>='

'<npsi|p^2>/(2m)|npsi>='

0

0.2.14 —-> p2.9

[4]:
```
#----> p2.9
if "p2.9" in sets.flow:
    if sets.use_libphysics:
        oqmec.__init__("position_space")
        oqmec.verbose = True
        [A,a,m] = symbols('A a m', real=True, positive=True)
        psi = Wavefunction(A*x*(a-x), (x, 0, a))
        npsi= psi.normalize()
        substitutions = {oqmec.Psi:npsi.expr, xmin:0, xmax:a, oqmec.V:0}

        H = oqmec.H.xreplace(substitutions)
        exp_H = oqmec.exp_H.xreplace(substitutions)
        pprints("p2.9",
                "psi=", psi,
                "npsi=", npsi,
                "H=", H, H.doit(),
                "<npsi|H|npsi>=<npsi|p^2>/(2m)|npsi>=", exp_H, exp_H.doit())

    else:
        [A,a,m] = symbols('A a m', real=True, positive=True)
        print("p2.9: 1. Way: (old fashion)")
        psi = Wavefunction(A*x*(a-x), (x, 0, a))
        npsi= psi.normalize()
        bounds = (0, a)
        expP2 = libquantum.expP2(npsi, bounds)
        expH = expP2/(2*m)
        pprints("psi=", psi,
                "npsi=", npsi,
                "<H>=<p^2>/(2m)=", simplify(expH))

        print("p2.9: 2. Way:")
        H = DifferentialOperator(-hbar**2/(2*m)*Derivative(f(x),x, 2), f(x))
        cnpsi_H_npsi = conjugate(npsi.expr)*(qapply(H*npsi)).expr
        expH = integrate(cnpsi_H_npsi,(x, 0, a))
        pprints("p2.9",
                "psi=", psi,
```

```
                "npsi=", npsi,
                "H=", H,
                "cnpi*H*npsi=", cnpsi_H_npsi,
                "<npsi|H|npsi>=<npsi|p^2>/(2m)|npsi>=", expH)
```

```
'p2.9'
```

```
'psi='
```

Wavefunction $\left(Ax\left(a-x\right),\left(x,\ 0,\ a\right)\right)$

```
'npsi='
```

Wavefunction $\left(\frac{\sqrt{30}x\left(a-x\right)}{a^{\frac{5}{2}}},\left(x,\ 0,\ a\right)\right)$

```
'H='
```

$$H=-\frac{\hbar^2\frac{\partial^2}{\partial x^2}\frac{\sqrt{30}x(a-x)}{a^{\frac{5}{2}}}}{2m}$$

$$H=\frac{\sqrt{30}\hbar^2}{a^{\frac{5}{2}}m}$$

```
'<npsi|H|npsi>=<npsi|p^2>/(2m)|npsi>='
```

$$\langle H\rangle=\int_0^a\left(-\frac{\sqrt{30}\hbar^2x\left(a-x\right)\frac{\partial^2}{\partial x^2}\frac{\sqrt{30}x(a-x)}{a^{\frac{5}{2}}}}{2a^{\frac{5}{2}}m}\right)dx$$

$$\langle H\rangle=\frac{5\hbar^2}{a^2m}$$

0.2.15 2.3 The Harmonic Oscillator

0.2.16 2.3.1 Algebraic Method

0.2.17 —-> e2.5

[7]:
```
#----> e2.5
if "e2.5" in sets.flow:
    # --- e2.5
    ad = RaisingOp('a')
    a = LoweringOp('a')
    nk = SHOKet('n')
    nb = SHOBra('n')
    xop = sqrt(hbar/(2*m*w))*(ad+a)
    pop = I*sqrt(hbar*m*w/2)*(ad-a)
    x2op = xop*xop
    p2op = pop*pop
    V = 1/2*m*w**2*x2op
    H = p2op/(2*m) + V
```

```
#=============================================================================
# # todo use in Fourier Transform
# x = XKet()
# pprints("|x>=",x,
#         "|x> in x-space=",rep_innerproduct(XKet(), basis = XOp()),
#         "|x> in p-space=",rep_innerproduct(XKet(), basis = PxOp()),
#         )
#=============================================================================

pprints("e2.5",
        "ad=", ad,
        "a=", a,
        "|n>=", nk,
        "<n|=", nb,
        "<n|=", nk.dual,
        "x=", xop,
        "x2=", x2op,
        "p=", pop,
        "a+|n>=qapply(ad*n)=", qapply(ad*nk),
        "a-|n>=qapply(a*m)=", qapply(a*nk),
        "a+a-|n>=", qapply(ad*qapply(a*nk)),
        "a-a+|n>=", qapply(a*qapply(ad*nk)),
        "<x>=<n|x|n>=", qapply(nb*qapply(xop*nk)),
        "<p>=<n|p|n>=", qapply(nb*pop*nk),
        "<V>=<n|V|n>=", qapply(nb*qapply(V*nk)),
        "<H>=<n|H|n>=", qapply(nb*qapply(H*nk)),
        output_style="display")
```

'e2.5'

'ad='

$a^\dagger$

'a='

a

'|n>='

$|n\rangle$

'<n|='

$\langle n|$

'<n|='

$\langle n|$

'x='

$$\frac{\sqrt{2}\sqrt{\hbar}\sqrt{\frac{1}{w}}\left(a+a^{\dagger}\right)}{2\sqrt{m}}$$

'x2='

$$\frac{\hbar\left(a+a^{\dagger}\right)^{2}}{2mw}$$

'p='

$$\frac{\sqrt{2}\sqrt{\hbar}i\sqrt{m}\sqrt{w}\left(-a+a^{\dagger}\right)}{2}$$

'a+|n>=qapply(ad*n)='

$\sqrt{n+1}|n+1\rangle$

'a-|n>=qapply(a*m)='

$\sqrt{n}|n-1\rangle$

'a+a-|n>='

$n|n\rangle$

'a-a+|n>='

$(n+1)\,|n\rangle$

'<x>=<n|x|n>='

0

'<p>=<n|p|n>='

0

'<V>=<n|V|n>='

$0.5\hbar nw+0.25\hbar w$

'<H>=<n|H|n>='

$1.0\hbar nw+0.5\hbar w$

0.2.18 2.6 The Finite Square Well

0.2.19 —-> ch2.6

[16]:
```
#----> ch2.6
if "ch2.6" in sets.flow:
    # --- ch2.6 todo not working 2.6 The Finite Square Well ---
    way_no = 1
    a = symbols('a', real=True)
    pprint("""
          ch2.6: The Finite Square Well
          E>0 case; through 2.158 to 2.171 was coded
```

```
        E<0 case; through 2.145 to 2.157 was OMITTED !!!
        """)
print("{}.way".format(way_no))

# Symbols
En = Symbol('En', real = True, positive = True)
l  = Symbol('l', real = True, positive = True)
#k  = Symbol('k', real = True, positive = True) will produce error
[A, B, C, D, F] = symbols('A B C D F', real = True)

# Potential & energy expressions
fV = Piecewise((0, x > a), (0, x < -a), (-V0, True))
fE = lambda x:En

# -- Solution of Schrödinger equations --
# Schrodinger equations.
eq_bar1 = libquantum.schrodingerEq(psix, fV.args[0][0], fE(x))
eq_well = libquantum.schrodingerEq(psix, fV.args[1][0], fE(x))
eq_bar2 = libquantum.schrodingerEq(psix, fV.args[0][0], fE(x))

# E>V0 case:
# Substitutions.
sub_l = {2*m*(En+V0)/hbar**2:l**2}
sub_k = {2*m*En/hbar**2:k**2}
sub_l_rev = {l:sqrt(2*m*(En+V0))/hbar}
sub_k_rev = {k:sqrt(2*m*En)/hbar}
sub_num = [(a,4), (hbar, 1), (m,1), (V0, 1), (En,x)]

eq_bar1_sub = eq_bar1.subs(sub_k)
eq_well_sub = eq_well.subs(sub_l)
eq_bar2_sub = eq_bar2.subs(sub_k)

# Solutions of Schrodinger equations.
sol_bar1 = dsolve(eq_bar1_sub, psix).subs({C1:B, C2:A}).rhs
sol_well = dsolve(eq_well_sub, psix).subs({C1:C, C2:D}).rhs
sol_bar2 = dsolve(eq_bar2_sub, psix).subs({C1:0, C2:F}).rhs

# Derivatives of solutions.
dsol_bar1 = diff(sol_bar1, x)
dsol_well = diff(sol_well, x)
dsol_bar2 = diff(sol_bar2, x)

# Boundary conditions.
#eq_bc1 = Eq(sol_well.subs(x, 0), sol_bar1.subs(x, 0)).reversed
eq_bc1 = Eq(sol_bar1.subs(x,-a), sol_well.subs(x,-a))
eq_bc2 = Eq(sol_well.subs(x, a), sol_bar2.subs(x, a))
eq_bc_diff1 = Eq(dsol_bar1.subs(x,-a), dsol_well.subs(x,-a))
```

```
    eq_bc_diff2 = Eq(dsol_well.subs(x, a), dsol_bar2.subs(x, a))

    # -- Matrix Methods --
    # M[:,1] gives 2nd column of the matrix M.

    # 1. Way, Matrix Method.
    if way_no==1:
        M1= linear_eq_to_matrix([eq_bc1.lhs, eq_bc_diff1.lhs], [A, B])[0]
        M2= linear_eq_to_matrix([eq_bc1.rhs, eq_bc_diff1.rhs], [C, D])[0]
        M3= linear_eq_to_matrix([eq_bc2.lhs, eq_bc_diff2.lhs], [C, D])[0]
        M4= linear_eq_to_matrix([eq_bc2.rhs, eq_bc_diff2.rhs], [F])[0]

        (mA, mB, mC)=(Matrix([[A],[B]]),
                      Matrix([[C],[D]]),
                      Matrix([F]))

    # 2. Way, Matrix Method.
    elif way_no==2:
        M1 = linear_eq_to_matrix([eq_bc1.lhs, eq_bc_diff1.lhs], [A*exp(-I*k*a),␣
↪B*exp(I*k*a)])[0]
        M2 = linear_eq_to_matrix([eq_bc1.rhs, eq_bc_diff1.rhs], [C, D])[0]
        M3 = linear_eq_to_matrix([eq_bc2.lhs, eq_bc_diff2.lhs], [C, D])[0]
        M4 = linear_eq_to_matrix([eq_bc2.rhs, eq_bc_diff2.rhs],␣
↪[F*exp(I*k*a)])[0]

        (mA, mB, mC) = (Matrix([[A*exp(-I*k*a)],[B*exp(I*k*a)]]),
                        Matrix([[C],[D]]),
                        Matrix([F*exp( I*k*a)]))

    # 3. Way, Nonlinear solution method.
    elif way_no==3:
        sol = nonlinsolve([eq_bc1.lhs-eq_bc1.rhs,
                           eq_bc_diff1.lhs-eq_bc_diff1.rhs,
                           eq_bc2.lhs-eq_bc2.rhs,
                           eq_bc_diff2.lhs-eq_bc_diff2.rhs],[A, B, C, D, F])
        A = simplify(sol.args[0][0])
        B = simplify(sol.args[0][1])

    if way_no in [1, 2]:
        solA = simplify(simplify((M1**-1))*M2*simplify((M3**-1))*M4*mC)
        A = solA[0]
        B = solA[1]
        #divF_A = (simplify(solA[0]/F))**-1

    # -- Reflection and Transmission --
    divB_A = simplify(B/A)
    divB_A2 = rcollect(divB_A, exp(2*I*a*alpha))
```

```
divF_A = simplify(F/A)
R = simplify(divB_A*conjugate(divB_A).subs(conjugate(k),k)) # R=|B/A|^2
T = simplify(divF_A*conjugate(divF_A).subs(conjugate(k),k)) # T=|F/A|^2
R = collect(R, cos(4*a*l))
T = collect(T, cos(4*a*l))

R_vs_En = simplify(R.subs(sub_k_rev).subs(sub_l_rev))
T_vs_En = simplify(T.subs(sub_k_rev).subs(sub_l_rev))
NR_vs_En = simplify(R_vs_En.subs(sub_num))
NT_vs_En = simplify(T_vs_En.subs(sub_num))

# Maximum transmission values
maxE = solve(T_vs_En-1, En)

"""
T = Abs(divF_A)**2 does not work due to improper assumption module.
facts = Q.nonzero(A), Q.nonzero(a)
with assuming(*facts):
Does not work.
"""

pprints("The Finite Square Well",
        "--- Solution of Schrödinger equations ---",
        "Left barrier", eq_bar1,
        "Quantum well", eq_well,
        "Right barrier", eq_bar2,
        "Substitutions", sub_l, sub_k,
        "After substitutions", eq_bar1_sub, eq_well_sub, eq_bar2_sub,

        "Solutions of differential equations",
        "sol_bar1", sol_bar1,
        "sol_well", sol_well,
        "sol_bar2", sol_bar2,

        "Continuity Conditions",
        "psi(x) @ x=-a (2.163, 2.164)", eq_bc1, eq_bc_diff1,
        "psi(x) @ x= a (2.165, 2.166)", eq_bc2, eq_bc_diff2,
        output_style="display")

if way_no in [1, 2]:
    pprints(
        "--- Matrix Methods ---",
        "Continuity Conditions; eq_bc1, eq_bc_diff1 -> M1*A = M2*B",
        "{0}*{1}={2}*{3}", (M1, mA, M2, mB),
        "Continuity Conditions; eq_bc2, eq_bc_diff2 -> M3*B = M4*F",
        "{0}*{1}={2}*{3}", (M3, mB, M4, mC),
```

```
            "A=",mA, "B=",mB, "C=",mC,
            "M1=",M1, "M2=",M2, "M3=",M3, "M4=",M4,
            "A = M1^-1*M2*B",
            "B = M3^-1*M4*F",
            "=> A = M1^-1*M2*M3^-1*M4*F",
            "B= (2.167)", B,
            "F= (2.168)", F,
            output_style="display")

    pprints("--- Reflection and Transmission ---",
             "B/A", divB_A, "or", divB_A2,
             "F/A", divF_A,
             "R=|B/A|^2=", R,
             "T=|F/A|^2=", T,
             "R(E)=", R_vs_En,
             "T(E)=", T_vs_En,
             "hbar=1, V0=1 case:",
             "R(E)=", NR_vs_En,
             "T(E)=", NT_vs_En,

             "Maximum transmission values:",
             "E=", maxE,
              output_style="display")

    print("Fig. 2.19, Transmissin, reflection coefficients as a function of␣
↪energy.")
    plot_sympfunc([NT_vs_En, NR_vs_En], (0,5,301), plabels=["T","R"],␣
↪xlabel="$E$", ylabel="$T,R$")
```

ch2.6: The Finite Square Well
E>0 case; through 2.158 to 2.171 was coded
E<0 case; through 2.145 to 2.157 was OMITTED !!!

1.way

'The Finite Square Well'

'--- Solution of Schrödinger equations ---'

'Left barrier'

$$\frac{2Enm\psi(x)}{\hbar^2} + \frac{d^2}{dx^2}\psi(x) = 0$$

'Quantum well'

$$\frac{2m\left(En + V_0\right)\psi(x)}{\hbar^2} + \frac{d^2}{dx^2}\psi(x) = 0$$

'Right barrier'

$$\frac{2Enm\psi(x)}{\hbar^2} + \frac{d^2}{dx^2}\psi(x) = 0$$

'Substitutions'

$$\left\{\frac{2m\left(En + V_0\right)}{\hbar^2} : l^2\right\}$$

$$\left\{\frac{2Enm}{\hbar^2} : k^2\right\}$$

'After substitutions'

$$\frac{2Enm\psi(x)}{\hbar^2} + \frac{d^2}{dx^2}\psi(x) = 0$$

$$\frac{2m\left(En + V_0\right)\psi(x)}{\hbar^2} + \frac{d^2}{dx^2}\psi(x) = 0$$

$$\frac{2Enm\psi(x)}{\hbar^2} + \frac{d^2}{dx^2}\psi(x) = 0$$

'Solutions of differential equations'

'sol_bar1'

$$C_1 \sin\left(\frac{\sqrt{2}\sqrt{En}\sqrt{m}x}{\hbar}\right) + C_2 \cos\left(\frac{\sqrt{2}\sqrt{En}\sqrt{m}x}{\hbar}\right)$$

'sol_well'

$$C_1 e^{-\frac{\sqrt{2}\sqrt{m}x\sqrt{-En-V_0}}{\hbar}} + C_2 e^{\frac{\sqrt{2}\sqrt{m}x\sqrt{-En-V_0}}{\hbar}}$$

'sol_bar2'

$$C_1 \sin\left(\frac{\sqrt{2}\sqrt{En}\sqrt{m}x}{\hbar}\right) + C_2 \cos\left(\frac{\sqrt{2}\sqrt{En}\sqrt{m}x}{\hbar}\right)$$

'Continuity Conditions'

'psi(x) @ x=-a (2.163, 2.164)'

$$-C_1 \sin\left(\frac{\sqrt{2}\sqrt{En}a\sqrt{m}}{\hbar}\right) + C_2 \cos\left(\frac{\sqrt{2}\sqrt{En}a\sqrt{m}}{\hbar}\right) = C_1 e^{\frac{\sqrt{2}a\sqrt{m}\sqrt{-En-V_0}}{\hbar}} + C_2 e^{-\frac{\sqrt{2}a\sqrt{m}\sqrt{-En-V_0}}{\hbar}}$$

$$\frac{\sqrt{2}C_1\sqrt{En}\sqrt{m}\cos\left(\frac{\sqrt{2}\sqrt{En}a\sqrt{m}}{\hbar}\right)}{\hbar} + \frac{\sqrt{2}C_2\sqrt{En}\sqrt{m}\sin\left(\frac{\sqrt{2}\sqrt{En}a\sqrt{m}}{\hbar}\right)}{\hbar} =$$
$$-\frac{\sqrt{2}C_1\sqrt{m}\sqrt{-En-V_0}e^{\frac{\sqrt{2}a\sqrt{m}\sqrt{-En-V_0}}{\hbar}}}{\hbar} + \frac{\sqrt{2}C_2\sqrt{m}\sqrt{-En-V_0}e^{-\frac{\sqrt{2}a\sqrt{m}\sqrt{-En-V_0}}{\hbar}}}{\hbar}$$

'psi(x) @ x= a (2.165, 2.166)'

$$C_1 e^{-\frac{\sqrt{2}a\sqrt{m}\sqrt{-En-V_0}}{\hbar}} + C_2 e^{\frac{\sqrt{2}a\sqrt{m}\sqrt{-En-V_0}}{\hbar}} = C_1 \sin\left(\frac{\sqrt{2}\sqrt{En}a\sqrt{m}}{\hbar}\right) + C_2 \cos\left(\frac{\sqrt{2}\sqrt{En}a\sqrt{m}}{\hbar}\right)$$

$$-\frac{\sqrt{2}C_1\sqrt{m}\sqrt{-En-V_0}e^{-\frac{\sqrt{2}a\sqrt{m}\sqrt{-En-V_0}}{\hbar}}}{\hbar} + \frac{\sqrt{2}C_2\sqrt{m}\sqrt{-En-V_0}e^{\frac{\sqrt{2}a\sqrt{m}\sqrt{-En-V_0}}{\hbar}}}{\hbar} =$$
$$\frac{\sqrt{2}C_1\sqrt{En}\sqrt{m}\cos\left(\frac{\sqrt{2}\sqrt{En}a\sqrt{m}}{\hbar}\right)}{\hbar} - \frac{\sqrt{2}C_2\sqrt{En}\sqrt{m}\sin\left(\frac{\sqrt{2}\sqrt{En}a\sqrt{m}}{\hbar}\right)}{\hbar}$$

'--- Matrix Methods ---'

'Continuity Conditions; eq_bc1, eq_bc_diff1 -> M1*A = M2*B'

'{0}*{1}={2}*{3}'

$$\left(\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}, \begin{bmatrix}A\\ B\end{bmatrix}, \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}, \begin{bmatrix}C\\ D\end{bmatrix}\right)$$

'Continuity Conditions; eq_bc2, eq_bc_diff2 -> M3*B = M4*F'

'{0}*{1}={2}*{3}'

$$\left(\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}, \begin{bmatrix}C\\ D\end{bmatrix}, \begin{bmatrix}0\\ 0\end{bmatrix}, \begin{bmatrix}F\end{bmatrix}\right)$$

'A='

$$\begin{bmatrix}A\\ B\end{bmatrix}$$

'B='

$$\begin{bmatrix}C\\ D\end{bmatrix}$$

'C='

$$\begin{bmatrix}F\end{bmatrix}$$

'M1='

$$\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}$$

'M2='

$$\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}$$

'M3='

$$\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}$$

'M4='

$$\begin{bmatrix}0\\ 0\end{bmatrix}$$

'A = M1^-1*M2*B'

'B = M3^-1*M4*F'

'=> A = M1^-1*M2*M3^-1*M4*F'

'B= (2.167)'

NaN

'F= (2.168)'

F

'--- Reflection and Transmission ---'

'B/A'

NaN

'or'

NaN

'F/A'

NaN

'R=|B/A|^2='

NaN

'T=|F/A|^2='

NaN

'R(E)='

NaN

'T(E)='

NaN

'hbar=1, V0=1 case:'

'R(E)='

NaN

'T(E)='

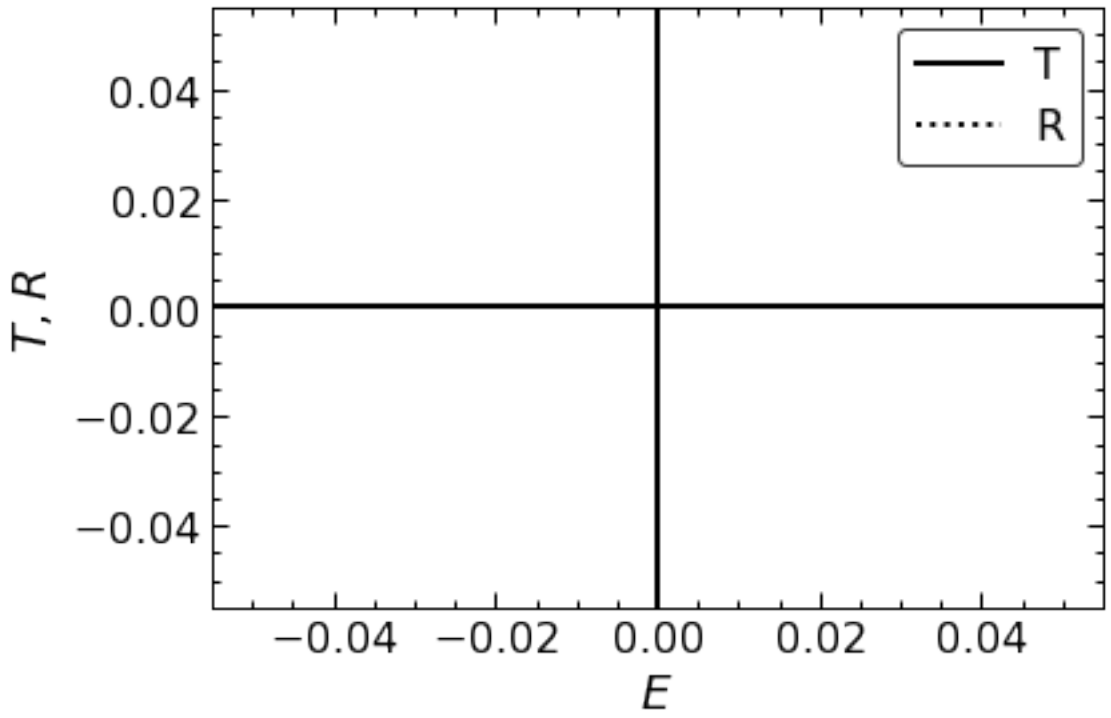
NaN

'Maximum transmission values:'

'E='

[]

Fig. 2.19, Transmissin, reflection coefficients as a function of energy.



0.2.20 —-> p2.11

[ ]:
```
#----> p2.11 todo
if "p2.11" in sets.flow:
    # kaldik solve with oqmec

    def psi(n):
        ksi = sqrt(m*w/hbar)*x
        res = Wavefunction((m*w/(pi*hbar))**(1/4)*(1/
 ↪sqrt((2**n)*factorial(n)))*hermite(n, ksi)*exp(-ksi**2/2), (x,-oo, oo))
        return res

    pprints("p2.11: (Harmonic Oscillator)",
            "psi(0)=", psi(0).expr if type(f) is Wavefunction else psi(0),
            "psi(1)=", psi(1).expr if type(f) is Wavefunction else psi(1),
            "a)",

            "For n = 0, psi(0):",
            "<x>=", libquantum.expX(psi(0)),
            "<x^2>=", libquantum.expX2(psi(0)),
            "<p>=", libquantum.expP(psi(0)),
            "<p^2>=", libquantum.expP2(psi(0)),
            "For n = 1, psi(1):",
```

```
            "<x>=", libquantum.expX(psi(1)),
            "<x^2>=", libquantum.expX2(psi(1)),
            "<p>=", libquantum.expP(psi(1)),
            "<p^2>=", libquantum.expP2(psi(1)),

            "b)",

            "For n = 0, psi(0):\n",
            "\sigma_x \sigma_p=", "todo",
            "For n = 1, psi(1):",
            "\sigma_x \sigma_p=", "todo",

            "c)",

            "For n = 0, psi(0):\n",
            "<T>=1/2m<p^2>=", libquantum.expT(psi(0)),
            "<V>=1/2mw^2<x^2>=", (1/2*m*w**2)*libquantum.expX2(psi(0)),
            "<H>=<T>+<V>=", simplify(libquantum.expT(psi(0))+(1/
↪2*m*w**2)*libquantum.expX2(psi(0))),

            "For n = 1, psi(1):\n",
            "<T>=1/2m<p^2>=", libquantum.expT(psi(1)),
            "<V>=1/2mw^2<x^2>=", (1/2*m*w**2)*libquantum.expX2(psi(1)),
            "<H>=<T>+<V>=", simplify(libquantum.expT(psi(1))+(1/
↪2*m*w**2)*libquantum.expX2(psi(1))),
            output_style="display")
```

0.2.21 —-> p2.12

[17]:
```
#----> p2.12
if "p2.12" in sets.flow:
    ad = RaisingOp('a')
    a = LoweringOp('a')
    nk = SHOKet('n')
    nb = SHOBra('n')
    xop = sqrt(hbar/(2*m*w))*(ad+a)
    pop = I*sqrt(hbar*m*w/2)*(ad-a)
    x2op = xop*xop
    p2op = pop*pop
    V = S(1)/2*m*w**2*x2op
    H = p2op/(2*m) + V

    expX = qapply(nb*xop*nk)
    expX2 = qapply(nb*qapply(x2op*nk))
    expP = qapply(nb*qapply(pop*nk))
    expP2 = qapply(nb*qapply(p2op*nk))
    sigmaX = sqrt(expX2-expX**2)
```

```
    sigmaP = sqrt(expP2-expP**2)

    pprints("p2.12: (Harmonic Oscillator)",
            "Algebraic Method:",
            "<x>=<n|x|n>=", expX,
            "<p>=<n|p|n>=", expP,
            "<x^2>=<n|x^2|n>=", expX2,
            "<p^2>=<n|x^2|n>=", expP2,
            "<V>=<n|V|n>=qapply(nb*V*nk)=", qapply(nb*V*nk),
            "<H>=<n|H|n>=qapply(nb*H*nk)=", qapply(nb*H*nk),

            "\sigma_x=", sigmaX,
            "sigma_p=", sigmaP,
            "\sigma_x \sigma_p=", simplify(sigmaX*sigmaP),
            output_style="display")
```

'p2.12: (Harmonic Oscillator)'

'Algebraic Method:'

'<x>=<n|x|n>='

0

'<p>=<n|p|n>='

0

'<x^2>=<n|x^2|n>='

$$\frac{\hbar n}{mw} + \frac{\hbar}{2mw}$$

'<p^2>=<n|x^2|n>='

$$\hbar mnw + \frac{\hbar mw}{2}$$

'<V>=<n|V|n>=qapply(nb*V*nk)='

$$\frac{\hbar nw}{2} + \frac{\hbar w}{4}$$

'<H>=<n|H|n>=qapply(nb*H*nk)='

$$\hbar nw + \frac{\hbar w}{2}$$

'\\sigma_x='

$$\sqrt{\frac{\hbar n}{mw} + \frac{\hbar}{2mw}}$$

'sigma_p='

$$\sqrt{\hbar mnw + \frac{\hbar mw}{2}}$$

'\\sigma_x \\sigma_p='

$$\frac{\hbar\sqrt{\frac{2n+1}{mw}}\sqrt{mw\,(2n+1)}}{2}$$

0.2.22 2.4 The Free Particle

0.2.23 —-> e2.6

[ ]:
```
#----> e2.6 todo
if "e2.6" in sets.flow:
    # --- e2.6 todo check
    print("""
          ex2.6: Free Particle, Wavefunction as a piecewise step function
                 Fourier transform, Inverse Fourier transform
          """)

    facts = Q.nonzero(A), Q.nonzero(a)
    [A,a] = symbols('A a', real=True, positive=True)
    with assuming(*facts):
        f = Piecewise((0, x < -a), (0, x > a), (A, True))
        psi = Wavefunction(f, x)
        npsi = psi.normalize()
        # Eq. 2.103
        psi_k1 = fourier_transform(1/sqrt(2*pi)*npsi.expr, x, k).subs({k:k/
↪(2*pi)})
        psi_k = psi_k1.args[0][0]

        pprints("psi=",psi,
            "a)",
            "npsi=",npsi,
            "Norm of npsi = npsi.norm=", npsi.norm,
            "Use npsi->2.103->2.100",
            "Fourier transform of the normalized wavefunction=", psi_k1,
            "Fourier transform of the normalized wavefunction=", psi_k,
            "Time evolution of psi=","There is no exact symbolic solution!. Eq.2.
↪104",
            output_style="display"
            )

    # Eq. 2.100 There is no exact symbolic solution!
    psi_t = inverse_fourier_transform(1/sqrt(2*pi)*psi_k*exp(-I*hbar*(k**2)*t/
↪(2*m)), k, x).subs({x:x/(2*pi)})
```

0.2.24 —-> p2.22

[18]:
```
#----> p2.22 todo
if "p2.22" in sets.flow:
    # --- p2.22 todo The Gausssian Wave Packet todo
    [A,a,k] = symbols('A a k', real=True, positive=True)    # --- p2.11
    # a)
    psi = A*exp(-a*x**2)
    psi = Wavefunction(psi, x)
    #psi = Wavefunction(psi,(x,-oo, oo))
    npsi = psi.normalize()

    # b) todo check inverse fourier
    # Eq. 2.103
    """
    .subs({k:k/(2*pi)} is necessary because sympy's Fourier transformation
    is formally different than the Fourier transformation given in the book.
    """
    psi_k = fourier_transform(1/sqrt(2*pi)*npsi.expr, x, k).subs({k:k/(2*pi)})
    # Eq. 2.100
    psi_xt = inverse_fourier_transform(1/sqrt(2*pi)*psi_k*exp(-I*hbar*(k**2)*t/
↪(2*m)), k, x).subs({x:x/(2*pi)})
    psi_xt = simplify(psi_xt)

    psi_k2 = simplify(1/sqrt(2*pi)*integrate(npsi.expr*exp(-I*k*x),(x,-oo, oo)))
    psi_xt2 = simplify(1/sqrt(2*pi)*integrate(psi_k2*exp(I*(k*x-hbar*(k**2)*t/
↪(2*m))),(k,-oo, oo)))

    #c)
    #psi_xt_sub1 = psi_xt.subs({2*hbar*a*t:theta*m})
    norm2 = simplify(Wavefunction(psi_xt, x).prob())

    #d) Complicated. todo
    #e) Complicated. todo
    pprints("p2.22: The Gausssian Wave Packet",
            "psi=",psi,
            "a)",
            "npsi=",npsi,

            "b) todo solve problems",
            "Use npsi->2.103->2.100",
            "npsi->2.103",
            "psi_k = psi_k2",
            "psi_k=",psi_k,
            "psi_k2=",psi_k2,
            "npsi->2.103->2.100",
            "psi_xt2=",psi_xt2,
```

```
            "psi_xt=",psi_xt,

            "c)",
            #"psi_xt_sub1=",psi_xt_sub1,
            "|psi_xt|^2=",norm2.expr,
            "Plotting graphics: todo"

            "d) todo",
            "<x>=", libquantum.expX(psi_xt),
            "<p>=", libquantum.expP(psi_xt),
            "<x^2>=", libquantum.expX2(psi_xt),
            "<p^2>=", "libquantum.expP2(psi_xt) difficult",
            "sigmaX=", "simplify(sigmaX(psi_xt))," # difficult
            "sigmaP=", "sigmaP(psi_xt)," # difficult

            "e)","todo",
            output_style="display")
```

```
KeyboardInterrupt
```

0.2.25 2.5 The Delta-Function Potential

0.2.26 2.5.1 Bound States and Scattering States

0.2.27 2.5.2 The Delta-Function Well

0.2.28 2.6 The Finite Square Well

0.2.29 —-> p2.33

[ ]:
```
#----> p2.23
if "p2.33" in sets.flow:
    # --- p2.33 todo not working The Finite Barrier ---
    way_no = 1
    print(
    """
    todo not working refer to solution manual go step by step !!!
    Refer to ch2.6
    The Finite Barrier
    A general solution to such sets.flow needs some other methods like in
    Transmission and Reflection of Electrons from an Arbitrary Potential
    - Shun Lien Chuang-Physics of Photonic Devices-Wiley (2009) p144.
    - Giuseppe Grosso, Giuseppe Pastori Parravicini Solid State Physics, Second␣
 ↪Edition (2013) p13.
    """)
    print("{}.way".format(way_no))
```

```
# Symbols
a  = Symbol('a', real = True, positive = True)
En = Symbol('En', real = True, positive = True)
l  = Symbol('l', real = True, positive = True)
k  = Symbol('k', real = True, positive = True)
[A, B, C, D, F] = symbols('A B C D F', real = True)

# Potential & energy expressions
fV = Piecewise((0, x > a), (0, x < -a), (+V0, True))
fE = lambda x:En

# -- Solution of Schrödinger equations --
# Schrodinger equations.
eq_bar1 = libquantum.schrodingerEq(psi, fV.args[0][0], fE(x), ptype="minus")
eq_well = libquantum.schrodingerEq(psi, fV.args[1][0], fE(x), ptype="minus")
eq_bar2 = libquantum.schrodingerEq(psi, fV.args[0][0], fE(x), ptype="minus")

# E>V0 case:
# Substitutions.
sub_l = {2*m*(-En+V0)/hbar**2:l**2}
sub_k = {2*m*En/hbar**2:k**2}
sub_l_rev = {l:sqrt(2*m*(-En+V0))/hbar}
sub_k_rev = {k:sqrt(2*m*En)/hbar}
sub_num = [(a,1), (hbar,1), (m,1), (V0,1), (En,x)]

eq_bar1_sub = eq_bar1.subs(sub_k)
eq_well_sub = eq_well.subs(sub_l)
eq_bar2_sub = eq_bar2.subs(sub_k)

# Solutions of Schrodinger equations.
sol_bar1 = dsolve(eq_bar1_sub, psi).subs({C1:B, C2:A}).rhs
sol_well = dsolve(eq_well_sub, psi).subs({C1:D, C2:C}).rhs
sol_bar2 = dsolve(eq_bar2_sub, psi).subs({C1:0, C2:F}).rhs

# Derivatives of solutions.
dsol_bar1 = diff(sol_bar1, x)
dsol_well = diff(sol_well, x)
dsol_bar2 = diff(sol_bar2, x)

# Boundary conditions.
#eq_bc1 = Eq(sol_well.subs(x, 0), sol_bar1.subs(x, 0)).reversed
eq_bc1 = Eq(sol_bar1.subs(x,-a), sol_well.subs(x,-a))
eq_bc2 = Eq(sol_well.subs(x, a), sol_bar2.subs(x, a))
eq_bc_diff1 = Eq(dsol_bar1.subs(x,-a), dsol_well.subs(x,-a))
eq_bc_diff2 = Eq(dsol_well.subs(x, a), dsol_bar2.subs(x, a))

# -- Matrix Methods --
```

```
    # M[:,1] gives 2nd column of the matrix M.

    # 1. Way, Matrix Method.
    if way_no == 1:
        M1 = linear_eq_to_matrix([eq_bc1.lhs, eq_bc_diff1.lhs], [A, B])[0]
        M2 = linear_eq_to_matrix([eq_bc1.rhs, eq_bc_diff1.rhs], [C, D])[0]
        M3 = linear_eq_to_matrix([eq_bc2.lhs, eq_bc_diff2.lhs], [C, D])[0]
        M4 = linear_eq_to_matrix([eq_bc2.rhs, eq_bc_diff2.rhs], [F])[0]

        (mA, mB, mC) = (Matrix([[A],[B]]),
                        Matrix([[C],[D]]),
                        Matrix([F]))

    # 2. Way, Matrix Method.
    elif way_no == 2:
        M1 = linear_eq_to_matrix([eq_bc1.lhs, eq_bc_diff1.lhs], [A*exp(-I*k*a),␣
↪B*exp(I*k*a)])[0]
        M2 = linear_eq_to_matrix([eq_bc1.rhs, eq_bc_diff1.rhs], [C, D])[0]
        M3 = linear_eq_to_matrix([eq_bc2.lhs, eq_bc_diff2.lhs], [C, D])[0]
        M4 = linear_eq_to_matrix([eq_bc2.rhs, eq_bc_diff2.rhs],␣
↪[F*exp(I*k*a)])[0]

        (mA, mB, mC) = (Matrix([[A*exp(-I*k*a)],[B*exp(I*k*a)]]),
                        Matrix([[C],[D]]),
                        Matrix([F*exp( I*k*a)]))

    # 3. Way, Nonlinear solution method.
    elif way_no == 3:
        sol = nonlinsolve([eq_bc1.lhs-eq_bc1.rhs,
                           eq_bc_diff1.lhs-eq_bc_diff1.rhs,
                           eq_bc2.lhs-eq_bc2.rhs,
                           eq_bc_diff2.lhs-eq_bc_diff2.rhs], [A, B, C, D, F])
        A = simplify(sol.args[0][0])
        B = simplify(sol.args[0][1])

    if way_no in [1, 2]:
        solA = simplify(simplify((M1**-1))*M2*simplify((M3**-1))*M4*mC)
        A = solA[0]
        B = solA[1]
        #divF_A = (simplify(solA[0]/F))**-1

    # -- Reflection and Transmission --
    divB_A = simplify(B/A)
    divB_A2 = rcollect(divB_A, exp(-2*I*a*k))
    divF_A = simplify(F/A)
    R = simplify(divB_A*conjugate(divB_A).subs(conjugate(k),k)) # R=|B/A|^2
    T = simplify(divF_A*conjugate(divF_A).subs(conjugate(k),k)) # T=|F/A|^2
```

```
R = collect(R, cos(4*a*l))
T = collect(T, cos(4*a*l))

R_vs_En = simplify(R.subs(sub_k_rev).subs(sub_l_rev))
T_vs_En = simplify(T.subs(sub_k_rev).subs(sub_l_rev))
NR_vs_En = simplify(R_vs_En.subs(sub_num))
NT_vs_En = simplify(T_vs_En.subs(sub_num))

# Maximum transmission values
# maxE = solve(T_vs_En-1, En)

"""
T = Abs(divF_A)**2 does not work due to improper assumption module.
facts = Q.nonzero(A), Q.nonzero(a)
with assuming(*facts):
Does not work.
"""

pprints("The Finite Square Well",
        "--- Solution of Schrödinger equations ---",
        "Left barrier", eq_bar1,
        "Quantum well", eq_well,
        "Right barrier", eq_bar2,
        "Substitutions", sub_l, sub_k,
        "After substitutions", eq_bar1_sub, eq_well_sub, eq_bar2_sub,

        "Solutions of differential equations",
        "sol_bar1", sol_bar1,
        "sol_well", sol_well,
        "sol_bar2", sol_bar2,

        "Continuity Conditions",
        "psi(x) @ x=-a", eq_bc1, eq_bc_diff1,
        "psi(x) @ x= a", eq_bc2, eq_bc_diff2,
        output_style="display")

if way_no in [1, 2]:
    pprints(
        "--- Matrix Methods ---",
        "Continuity Conditions; eq_bc1, eq_bc_diff1 -> M1*A = M2*B",
        "{0}*{1}={2}*{3}".format(M1, mA, M2, mB),
        "Continuity Conditions; eq_bc2, eq_bc_diff2 -> M3*B = M4*F",
        "{0}*{1}={2}*{3}".format(M3, mB, M4, mC),

        "A=",mA, "B=",mB, "C=",mC,
        "M1=",M1, "M2=",M2, "M3=",M3, "M4=",M4,
        "A = M1^-1*M2*B",
```

```
            "B = M3^-1*M4*F",
            "=> A = M1^-1*M2*M3^-1*M4*F",
            output_style="display")

    pprints("--- Reflection and Transmission ---",
             "B/A",divB_A, "or", divB_A2,
             "F/A",divF_A,
             "R=|B/A|^2=",R,
             "T=|F/A|^2=",T,
             "R(E)=", R_vs_En,
             "T(E)=", T_vs_En,
             "hbar=1, V0=1 case:",
             "R(E)=", NR_vs_En,
             "T(E)=", NT_vs_En,

             "Maximum transmission values:",
             "E=", "maxE",
              output_style="display")

    print("Fig. 2.19, Transmission, reflection coefficients as a function of␣
↪energy.")
    plot_sympfunc([NT_vs_En, NR_vs_En], (0.01,0.99,301), plabels=["T","R"],
                  xlabel="$E$", ylabel="$T,R$")
```

0.2.30 —-> p2.41

[ ]:
```
#----> p2.41
if "p2.41" in sets.flow:
    # --- 2.41 todo check
    # a)
    A = Symbol('A', real=True, positive=True)
    w = Symbol('w', real=True, positive=True)
    psi = A*(1-2*sqrt(m*w/hbar)*x)**2*exp(-m*w*x**2/(2*hbar))
    psi = Wavefunction(psi, x)
    normconst = solve(psi.norm-1, A)
    npsi = psi.subs({A:normconst})
    H = DifferentialOperator(-hbar**2/(2*m)*Derivative(f(x),x,2) + 1/2*m*w*x**2,␣
↪f(x))
    H_npsi = qapply(H*npsi)
    cnpsi_H_npsi = conjugate(npsi.expr)*H_npsi.expr
    expH = integrate(cnpsi_H_npsi, (x,-oo,oo))

    """
        b)
        # Eq. 2.103
        .subs({k:k/(2*pi)} is necessary because sympy's Fourier transformation
        is formally different than the Fourier transformation given in the book.
```

```
        psi_k = fourier_transform(1/sqrt(2*pi)*npsi.expr, x, k).subs({k:k/
↪(2*pi)})
        # Eq. 2.100
        psi_xt = inverse_fourier_transform(1/
↪sqrt(2*pi)*psi_k*exp(-I*hbar*(k**2)*t/(2*m)), k, x).subs({x:x/(2*pi)})
        psi_xt = simplify(psi_xt)

        psi_k2 = simplify(1/sqrt(2*pi)*integrate(npsi.expr*exp(-I*k*x),(x,-oo,␣
↪oo)))
        psi_xt2 = simplify(1/
↪sqrt(2*pi)*integrate(psi_k2*exp(I*(k*x-hbar*(k**2)*t/(2*m))),(k,-oo, oo)))

        c)
        #psi_xt_sub1 = psi_xt.subs({2*hbar*a*t:theta*m})
        norm2 = simplify(Wavefunction(psi_xt, x).prob())
    """

    pprints("\n p2.41: Time evolution of harmonic oscillator.",
            "psi=", psi,
            "a)",
            "npsi=", npsi,
            "npsi.expr=", npsi.expr,
            "normalization constant=", normconst,
            "normalization check=", npsi.norm,

            "H=", H,
            "H.function=", H.function,
            "H.variables=", H.variables,

            "qapply(H*npsi)=", H_npsi,
            "<psi|H|psi>=",
            "cnpsi_H_npsi=conjugate(npsi.expr)*H_npsi.expr=", cnpsi_H_npsi,
            "expH=integrate(cnpsi_H_npsi,(x,0,L))=", expH,
            "simplify(expH)=", simplify(expH),
            "todo gives different result than 73/50*hbar*w"
            "expH.evalf()=", simplify(expH).evalf(),

            "b) todo",
#            "Use npsi->2.103->2.100",
#            "npsi->2.103",
#            "psi_k = psi_k2",
#            "psi_k=",psi_k,
#            "psi_k2=",psi_k2,
#            "npsi->2.103->2.100",
#            "psi_xt2=",psi_xt2,
#            "psi_xt=",psi_xt,
```

```
#
#             "c)",
#             #"psi_xt_sub1=",psi_xt_sub1,
#             "|psi_xt|^2=",norm2.expr,
#             "Plotting graphics: todo"
#
#             "d)",
#             "<x>=",libquantum.expX(psi_xt),
#             "<p>=",libquantum.expP(psi_xt),
#             "<x^2>=",libquantum.expX2(psi_xt),
#             "<p^2>=","libquantum.expP2(psi_xt) difficult",
#             "sigmaX=",simplify(sigmaX(psi_xt)),
#             "sigmaP=",simplify(sigmaP(psi_xt)),

            output_style="display"
            )
```

0.2.31 —-> ch2.3.2

[ ]:
```
#----> ch2.3.2
if "ch2.3.2" in sets.flow:
    # --- ch2.3.2 todo solve diff eq.
    [En,m,w] = symbols('En m w', real=True, positive=True)
    fV = S(1)/2*m*w**2*x**2 # or    # --- p2.11
    fV = Rational(1,2)*m*w**2*x**2
    schEq = libquantum.schrodingerEq(psi, fV, En, ptype="plus")
    solschEq = dsolve(schEq, psi)
    pprints(
        "p2.3.2: Analytic Method (Harmonic Oscillator)",
        "Schrödinger Equation=", schEq,
        "Solution=", solschEq,
        output_style="display")
```

0.3 Chapter 3 Formalism

0.3.1 3.1 Hilbert Space

0.3.2 3.2 Observables

0.3.3 3.2.1 Hermitian Operators

0.3.4 3.2.2 Determinate States

0.3.5 3.3 Eigenfunctions of A Hermitian Operator

0.3.6 3.3.1 Discrete Spectra

0.3.7 3.3.2 Continuous Spectra

0.3.8 3.4 Generalized Statistical Interpretation

0.3.9 3.5 The Uncertainty Principle

0.3.10 3.5.1 Proof of the Generalized Uncertainty Principle

0.3.11 3.5.2 The Minimum-Uncertainty Wave Packet

0.3.12 3.5.3 The Energy-Time Uncertainty Principle

0.3.13 3.6 Dirac Notation

0.3.14 —-> p3.22

[19]:
```
#----> p3.22
if "p3.22" in sets.flow:
    (one, two, three) = (Ket(1), Ket(2), Ket(3))
    a = I*one - 2*two - I*three
    b = I*one + 2*three
    # A = Operator(qapply(a*Dagger(b))) alternative
    # A = |a><b|
    A = OuterProduct(a, Dagger(b))

    MA =[]
    strMA=[]
    for i in range(1,4): # Iteration over rows.
        row = []
        strrow = []
        for j in range(1,4): # Iteration over columns
            strrow.append("<{0}|A|{1}>".format(i,j))
            row.append(qapply(Bra(i)*A*Ket(j))) # <i|A|j>
        MA.append(row)
        strMA.append(strrow)

    mA = Matrix(MA)

    pprints("p3.22:","",
            "[i  for i in range(1,4)]=", [i  for i in range(1,4)],
```

```
            "|a>=", a,
            "<b|=", b,
            "a)",
            "<a|=", Dagger(a),
            "<b|=", Dagger(b),

            "b)",
            "<a|b>=", qapply(Dagger(a)*b),
            "<b|a>=", qapply(Dagger(b)*a),
            "<b|a>=<a|b>*", Dagger(qapply(Dagger(a)*b)),

            "b)",
            "A=|a><b|=", A,

            "c)",
            "MA", MA,
            "Qmn=Amn=<m|A|n>=", strMA,
            "MA=Matrix(MA)=", Matrix(MA),

            "Is MA Hermitian?",
            "mA == simplify(mA.conjugate().T)", mA == simplify(mA.conjugate().T),

            output_style="display"
            )
```

'p3.22:'

''

'[i for i in range(1,4)]='

$[1,\ 2,\ 3]$

'|a>='

$i|1\rangle - 2|2\rangle - i|3\rangle$

'<b|='

$i|1\rangle + 2|3\rangle$

'a)'

'<a|='

$-i\langle 1| - 2\langle 2| + i\langle 3|$

'<b|='

$-i\langle 1| + 2\langle 3|$

'b)'

'<a|b>='

$\langle 1|1\rangle - 2i\langle 1|3\rangle - 2i\langle 2|1\rangle - 4\langle 2|3\rangle - \langle 3|1\rangle + 2i\langle 3|3\rangle$

```
'<b|a>='
```

$\langle 1|1\rangle + 2i\langle 1|2\rangle - \langle 1|3\rangle + 2i\langle 3|1\rangle - 4\langle 3|2\rangle - 2i\langle 3|3\rangle$

```
'<b|a>=<a|b>*'
```

$\langle 1|1\rangle + 2i\langle 1|2\rangle - \langle 1|3\rangle + 2i\langle 3|1\rangle - 4\langle 3|2\rangle - 2i\langle 3|3\rangle$

```
'b)'
```

```
'A=|a><b|='
```

$|1\rangle\langle 1| + 2i|1\rangle\langle 3| + 2i|2\rangle\langle 1| - 4|2\rangle\langle 3| - |3\rangle\langle 1| - 2i|3\rangle\langle 3|$

```
'c)'
```

```
'MA'
```

$\left[\left[\langle 1|1\rangle^2 + 2i\langle 1|1\rangle\langle 1|2\rangle - \langle 1|1\rangle\langle 1|3\rangle + 2i\langle 1|1\rangle\langle 3|1\rangle - 4\langle 1|2\rangle\langle 3|1\rangle - 2i\langle 1|3\rangle\langle 3|1\rangle,\ \langle 1|1\rangle\langle 1|2\rangle + 2i\langle 1|1\rangle\langle 3|\right.\right.$

```
'Qmn=Amn=<m|A|n>='
```

```
[['<1|A|1>', '<1|A|2>', '<1|A|3>'],
 ['<2|A|1>', '<2|A|2>', '<2|A|3>'],
 ['<3|A|1>', '<3|A|2>', '<3|A|3>']]
```

```
'MA=Matrix(MA)='
```

$$\begin{bmatrix} \langle 1|1\rangle^2 + 2i\langle 1|1\rangle\langle 1|2\rangle - \langle 1|1\rangle\langle 1|3\rangle + 2i\langle 1|1\rangle\langle 3|1\rangle - 4\langle 1|2\rangle\langle 3|1\rangle - 2i\langle 1|3\rangle\langle 3|1\rangle & \langle 1|1\rangle\langle 1|2\rangle + 2i\langle 1 \\ \langle 1|1\rangle\langle 2|1\rangle + 2i\langle 1|1\rangle\langle 2|2\rangle - \langle 1|1\rangle\langle 2|3\rangle + 2i\langle 2|1\rangle\langle 3|1\rangle - 4\langle 2|2\rangle\langle 3|1\rangle - 2i\langle 2|3\rangle\langle 3|1\rangle & \langle 1|2\rangle\langle 2|1\rangle + 2i\langle 1|2\rangle \\ \langle 1|1\rangle\langle 3|1\rangle + 2i\langle 1|1\rangle\langle 3|2\rangle - \langle 1|1\rangle\langle 3|3\rangle + 2i\langle 3|1\rangle^2 - 4\langle 3|1\rangle\langle 3|2\rangle - 2i\langle 3|1\rangle\langle 3|3\rangle & \langle 1|2\rangle\langle 3|1\rangle + 2i\langle 1 \end{bmatrix}$$

```
'Is MA Hermitian?'
```

```
'mA == simplify(mA.conjugate().T)'
```

```
False
```

0.3.15 —-> p3.30

[20]:
```
#----> p3.30
if "p3.30" in sets.flow:
    [A,a] = symbols('A a', real=True, positive=True)
    p = symbols('p', real=True)

    # Position space calculations.
    f = A/(x**2+a**2)
    psi = Wavefunction(f, (x, -oo, oo))
    npsi = psi.normalize()
    solA = solve(psi.norm**2-1, A)
    expX = libquantum.expX(npsi)
    expX2 = libquantum.expX2(npsi)
    sigmaX = libquantum.sigmaX(npsi)
```

```
    # Momentum space calculations.
    phi = libquantum.xp_transform(npsi)
    wfphi = Wavefunction(phi, (p, -oo, oo))
    expP = libquantum.expP(wfphi)
    expP2 = libquantum.expP2(wfphi)
    sigmaX = libquantum.sigmaX(wfphi)
    sigmaP = libquantum.sigmaP(wfphi)
    sigmaXsigmaP = libquantum.sigmaXsigmaP(wfphi)

    pprints("p3.30:","",
            "Fourier Transforms, Change of Basis from Position Space to Momentum␣
↪Space",
            "a)",
            "Wavefunction in position space=",
            "\psi(x,0)=", psi,
            "A=", solA,
            "normalized \psi(x,0)=", npsi,

            "b)",
            "<x>=", expX,
            "<x^2>=", expX2,
            "sigmaX=\Delta x=", sigmaX,

            "c)",
            "Wavefunction in momentum space",
            "\phi(p,0)=", phi,
            "Norm of \phi(p,0)=", wfphi.norm,

            "d)",
            "at t=0,",
            "<p>=", expP,
            "<p^2>=", expP2,
            "<sigmaP>=\Delta p=", sigmaP,

            "e)",
            "Checking the Heisenberg uncertainty principle for this state.",
            "\Delta x \Delta p=",

            output_style="display"
            )
```

```
'p3.30:'
```

```
''
```

```
'Fourier Transforms, Change of Basis from Position Space to Momentum Space'
```

```
'a)'
```

'Wavefunction in position space='

'\\psi(x,0)='

$$\text{Wavefunction}\left(\frac{A}{a^2+x^2},(x,\ -\infty,\ \infty)\right)$$

'A='

$$\left[\frac{\sqrt{2}a^{\frac{3}{2}}}{\sqrt{\pi}}\right]$$

'normalized \\psi(x,0)='

$$\text{Wavefunction}\left(\frac{\sqrt{2}a^{\frac{3}{2}}}{\sqrt{\pi}\,(a^2+x^2)},(x,\ -\infty,\ \infty)\right)$$

'b)'

'<x>='

0

'<x^2>='

a^2

'sigmaX=\\Delta x='

ia

'c)'

'Wavefunction in momentum space'

'\\phi(p,0)='

$$\frac{\sqrt{a}e^{-\frac{a|p|}{\hbar}}}{\sqrt{\hbar}}$$

'Norm of \\phi(p,0)='

1

'd)'

'at t=0,'

'<p>='

0

'<p^2>='

$$\frac{\hbar^2}{2a^2}$$

'<sigmaP>=\\Delta p='

$$\frac{\sqrt{2}\hbar}{2a}$$

'e)'

'Checking the Heisenberg uncertainty principle for this state.'

'\\Delta x \\Delta p='

0.4 Chapter 4 Quantum Mechanics in Three Dimensions

0.4.1 4.1 Schrodinger Equation in Spherical Coordinates

0.4.2 —-> p4.1

[25]:
```
#----> p4.1
if "p4.1" in sets.flow:
    [xop, yop, zop] = [XOp(), YOp(), ZOp()]
    pxop = PxOp()
    pyop = libquantum.PyOp()

    com_xy = Commutator(xop, yop)
    com_xop_pxop = Commutator(xop, pxop)
    com_xop_pyop = Commutator(xop, pyop)
    com_yop_pxop = Commutator(yop, pxop)
    com_pxop_xop = Commutator(pxop, xop)
    com_pxop_pyop = Commutator(pxop, pyop)

    pprints("p4.1 Operators",
            "Overview of operators, basis and representations in QM."

            "a)",
            "represent(XOp(), basis=X)", represent(XOp(), basis=X),
            "represent(Px, basis=X)", represent(Px, basis=X),
            "[x, y]=", com_xy, com_xy.doit(),
            "[x, px]=", com_xop_pxop, com_xop_pxop.doit(),
            ".doit != .expand(commutator=True)",
            "[y, px]=", com_yop_pxop, com_yop_pxop.doit(),
            "[px, x]=", com_pxop_xop, com_pxop_xop.doit(),

            "[x, py]=", com_xop_pyop, com_xop_pyop.doit(),

            "[px, py]=", com_pxop_pyop, com_pxop_pyop.doit(),

            "b) todo",

            "c) todo",
            output_style="display")
```

'p4.1 Operators'

```
'Overview of operators, basis and representations in QM.a)'
'represent(XOp(), basis=X)'
```
$x_1 \delta(x_1 - x_2)$
```
'represent(Px, basis=X)'
```
$-\hbar i \delta(x_1 - x_2) x_1$
```
'[x, y]='
```
$[X, Y]$

$XY - YX$
```
'[x, px]='
```
$-[Px, X]$

$\hbar i$
```
'.doit != .expand(commutator=True)'
'[y, px]='
```
$-[Px, Y]$

$-(PxY - YPx)$
```
'[px, x]='
```
$[Px, X]$

$-\hbar i$
```
'[x, py]='
```
$-[Py, X]$

$-(PyX - XPy)$
```
'[px, py]='
```
$[Px, Py]$

$PxPy - PyPx$
```
'b) todo'
'c) todo'
```

0.4.3 4.1.1 Sepatation of Variables

0.4.4 4.1.2 The Angular Equation

0.4.5 4.1.3 The Radial Equation

0.4.6 —-> e4.1

[26]:
```
#----> e4.1 Infinite Spherical Well
if "e4.1" in sets.flow:
    [l,k,r,m,C1] = symbols('l k r m C1', real=True, positive=True)
    En = Symbol('E_n', real = True, positive = True)
    u = Function('u')(r)

    sub_k = {k:sqrt(2*m*En)/hbar}
    diffEq = Eq(u.diff(r,2), (l*(l+1)/r**2 - k**2)*u)
    sol_u = dsolve(diffEq, u, check=True)
    sol_u2 = sol_u.rewrite(jn)

    u3 = lambdify((r, l), sol_u2.rhs, modules='sympy')
    u4 = sqrt(r)*(sqrt(2)*C1*sqrt(k)*sqrt(r)*jn(sqrt(l*(l + 1) + 0.25) - 0.5,␣
 ↪k*r)/sqrt(pi))
    u4 = lambdify((r, l), u4, modules='sympy')

    def fEn(l,n):
        """
        l = l number,
        n = # of zeros
        """
        res = []
        zeros = jn_zeros(l,n)
        for i in range(n):
            # print(hbar**2*zeros[i]/(2*m*a**2))
            res.append(hbar**2*zeros[i]/(2*m*a**2))
        return(res)

    print("Spherical Bessel Function of order l, jn_l(l,x)")
    for i in range(4):
        pprints("j_l={0}(x)=".format(i),
                expand_func(jn(i,x)),
                output_style="display")

    print("Spherical Neumann Function of order l, yn_l(l,x)")
    for i in range(4):
        pprints("y_l={0}(x)=".format(i),
                expand_func(yn(i,x)),
                output_style="display")

    pprints("e4.1 Infinite Spherical Well",
```

```
            diffEq,
            "u(r,l)=", sol_u,
            "u(r,l)=", sol_u2,
            "u(r,l)=", u3,
            "After appying r=a => u(a,l)=0 boundary condition, in the solution␣
↪Neumann function part must vanish",  u4(r,l),
            "Energy eigenvalues= E(0,3)=", fEn(0,3),
            output_style="display")

   plot(jn(0,x), jn(1,x), jn(2,x), (x,0,14))
```

```
Spherical Bessel Function of order l, jn_l(l,x)
```

```
'j_l=0(x)='
```

$$\frac{\sin(x)}{x}$$

```
'j_l=1(x)='
```

$$-\frac{\cos(x)}{x}+\frac{\sin(x)}{x^2}$$

```
'j_l=2(x)='
```

$$\left(-\frac{1}{x}+\frac{3}{x^3}\right)\sin(x)-\frac{3\cos(x)}{x^2}$$

```
'j_l=3(x)='
```

$$\left(-\frac{6}{x^2}+\frac{15}{x^4}\right)\sin(x)+\left(\frac{1}{x}-\frac{15}{x^3}\right)\cos(x)$$

```
Spherical Neumann Function of order l, yn_l(l,x)
```

```
'y_l=0(x)='
```

$$-\frac{\cos(x)}{x}$$

```
'y_l=1(x)='
```

$$-\frac{\sin(x)}{x}-\frac{\cos(x)}{x^2}$$

```
'y_l=2(x)='
```

$$-\left(-\frac{1}{x}+\frac{3}{x^3}\right)\cos(x)-\frac{3\sin(x)}{x^2}$$

```
'y_l=3(x)='
```

$$-\left(-\frac{6}{x^2}+\frac{15}{x^4}\right)\cos(x)+\left(\frac{1}{x}-\frac{15}{x^3}\right)\sin(x)$$

```
'e4.1 Infinite Spherical Well'
```

$$\frac{d^2}{dr^2}u(r) = \left(-k^2 + \frac{l\,(l+1)}{r^2}\right)u(r)$$

```
'u(r,l)='
```

$$u(r) = \sqrt{r}\left(C_1 J_{\sqrt{l(l+1)+\frac{1}{4}}}\,(kr) + C_2 Y_{\sqrt{l(l+1)+\frac{1}{4}}}\,(kr)\right)$$

```
'u(r,l)='
```

$$u(r) = \sqrt{r}\left(\frac{\sqrt{2}C_1\sqrt{k}\sqrt{r}j_{\sqrt{l(l+1)+\frac{1}{4}}-\frac{1}{2}}\,(kr)}{\sqrt{\pi}} + C_2 Y_{\sqrt{l(l+1)+\frac{1}{4}}}\,(kr)\right)$$

```
'u(r,l)='
```

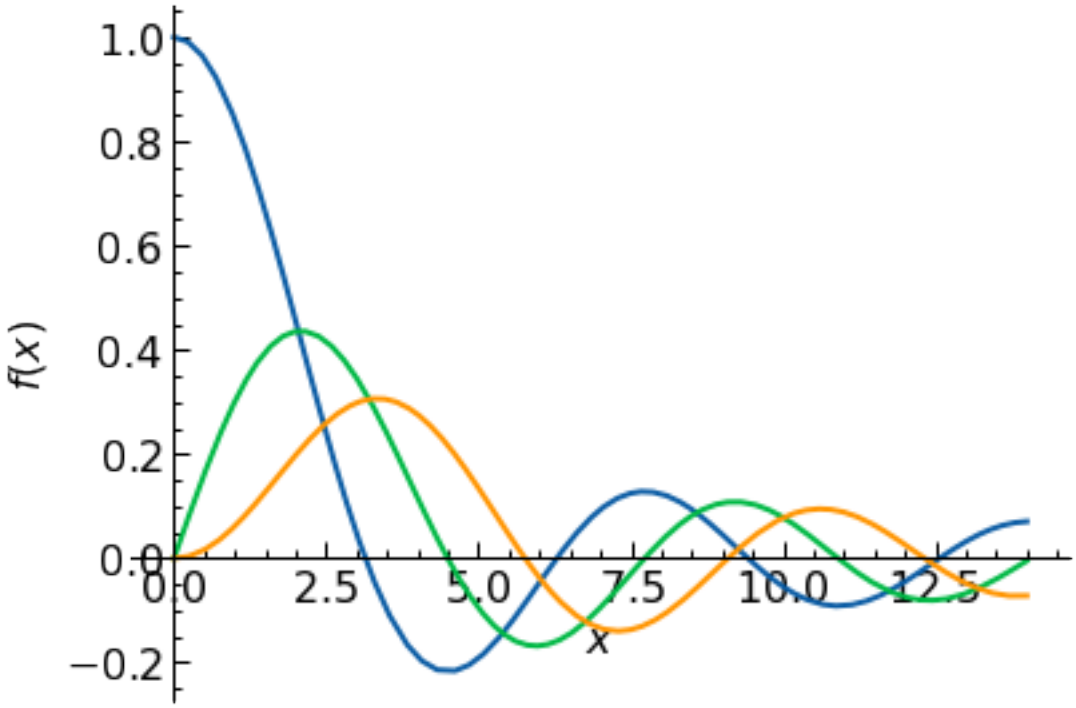
```
<function _lambdifygenerated(r, l)>
```

```
'After appying r=a => u(a,l)=0 boundary condition, in the solution Neumann␣
↪function part must vanish'
```

$$0.564189583547756\sqrt{2}C_1\sqrt{kr}j_{\sqrt{l(l+1)+0.25}-0.5}\,(kr)$$

```
'Energy eigenvalues= E(0,3)='
```

$$\left[\frac{1.5707963267949\hbar^2}{a^2m},\ \frac{3.14159265358979\hbar^2}{a^2m},\ \frac{4.71238898038469\hbar^2}{a^2m}\right]$$



0.4.7 4.2 The Hydrogen Atom

0.4.8 —-> ch4.2

[31]:
```
#----> p4.2
if "ch4.2" in sets.flow:
    oqmec.Psi = oqmec.hydrogen.psi_sy(3,0,0).rhs
    exp_invr = oqmec.exp_fxSph(1/r)
    exp_invr2 = oqmec.exp_fxSph(1/r**2)

    pprints("ch4.2 The Hydrogen Atom",
            "oqmec.exp_fxSph(1/r)=", exp_invr,
            "oqmec.exp_fxSph(1/r).doit()", exp_invr.doit(),
            "oqmec.exp_fxSph(1/r^2)=", exp_invr2,
            "oqmec.exp_fxSph(1/r^2).doit()", exp_invr2.doit())

    # Mathematica Client
    from wolframclient.evaluation import WolframLanguageSession
    from wolframclient.language import wl, wlexpr
    from sympy.parsing.mathematica import parse_mathematica

    oqmec.Psi = oqmec.hydrogen.psi_sy(n,0,0).rhs
    expinvr = oqmec.exp_fxSphR(1/r).rhs
    session = WolframLanguageSession()
    math_expr = wlexpr(mathematica_code(expinvr))
    pprints("<1/r>", math_expr)
```

'ch4.2 The Hydrogen Atom'

'oqmec.exp_fxSph(1/r)='

$$\langle 1/r \rangle = \int_0^{\pi}\int_0^{2\pi}\int_0^{\infty} \frac{r^2 \cdot \left(\frac{2r^2}{9} - 2r + 3\right)\left(\frac{2\overline{r}^2}{9} - 2\overline{r} + 3\right) e^{-\frac{r}{3}} e^{-\frac{\overline{r}}{3}} \sin(\theta)}{243\pi r}\, dr\, d\phi\, d\theta$$

'oqmec.exp_fxSph(1/r).doit()'

$$\langle 1/r \rangle = \frac{1}{r}$$

'oqmec.exp_fxSph(1/r^2)='

$$\langle r**(-2) \rangle = \int_0^{\pi}\int_0^{2\pi}\int_0^{\infty} \frac{r^2 \cdot \left(\frac{2r^2}{9} - 2r + 3\right)\left(\frac{2\overline{r}^2}{9} - 2\overline{r} + 3\right) e^{-\frac{r}{3}} e^{-\frac{\overline{r}}{3}} \sin(\theta)}{243\pi r^2}\, dr\, d\phi\, d\theta$$

'oqmec.exp_fxSph(1/r^2).doit()'

$$\langle r**(-2) \rangle = \frac{1}{r^2}$$

'<1/r>'

```
(Hold[Integrate[4*r^2*Exp[-r/n]*Exp[-Conjugate[r]/n]*LaguerreL[n - 1, 1, 2*r/
↪n]*LaguerreL[n - 1, 1, 2*Conjugate[r]/n]*Factorial[n - 1]/(n^4*r*Factorial[n]),␣
↪{r, 0, Infinity}]])
```

0.4.9 4.2.1 The Radial Wave Function

0.4.10 —-> ch4.2.1

[32]:
```
#----> chp4.2.1
if "ch4.2.1" in sets.flow:
    [a0, c0, epsilon_0, m_e, e] = symbols("a0 c0 epsilon_0 m_e e")
    n, l, m, r, phi, theta, Z = S(n), S(l), S(m), S(r), S(phi), S(theta), S(Z)
    a0 = 4*pi*epsilon_0*hbar**2/(m_e*e**2)     # Bohr radius

    for i in range(5):
        for j in range(i):
            pprints("R_{}{}=".format(i,j),
                    R_nl(i, j, r, Z=1/a),
                    output_style="display")

    pprints("ch4.2.1",
            "R_nl=", R_nl(n, l, r, Z=1/a),
            "R_10=", R_nl(1, 0, r, Z=1/a),
            "R_n0=", [R_nl(i, 0, r, Z=1/a) for i in range(1,4)],
            "R_nl=",
            [[R_nl(i, j, r, Z=1/a) for j in range(i)] for i in range(5)],

            "(4.89)",
            "psi_nlm=", Psi_nlm(n, l, m, r, phi, theta, Z=1/a),
            "psi_200=", Psi_nlm(2, 0, 0, r, phi, theta, Z=1/a),
            "psi_200=", simplify(Psi_nlm(2, 0, 0, r, phi, theta, Z=1/a)),
            output_style="display")
```

'R_10='

$$\frac{2e^{-\frac{r}{a}}}{a^{\frac{3}{2}}}$$

'R_20='

$$\frac{\sqrt{2}\cdot\left(2-\frac{r}{a}\right)e^{-\frac{r}{2a}}}{4a^{\frac{3}{2}}}$$

'R_21='

$$\frac{\sqrt{6}re^{-\frac{r}{2a}}}{12a^{\frac{5}{2}}}$$

'R_30='

$$\frac{2\sqrt{3}\cdot\left(3-\frac{2r}{a}+\frac{2r^2}{9a^2}\right)e^{-\frac{r}{3a}}}{27a^{\frac{3}{2}}}$$

'R_31='

$$\frac{\sqrt{6}r\left(4-\frac{2r}{3a}\right)e^{-\frac{r}{3a}}}{81a^{\frac{5}{2}}}$$

'R_32='

$$\frac{2\sqrt{30}r^2e^{-\frac{r}{3a}}}{1215a^{\frac{7}{2}}}$$

'R_40='

$$\frac{\left(4-\frac{3r}{a}+\frac{r^2}{2a^2}-\frac{r^3}{48a^3}\right)e^{-\frac{r}{4a}}}{16a^{\frac{3}{2}}}$$

'R_41='

$$\frac{\sqrt{15}r\left(10-\frac{5r}{2a}+\frac{r^2}{8a^2}\right)e^{-\frac{r}{4a}}}{480a^{\frac{5}{2}}}$$

'R_42='

$$\frac{\sqrt{5}r^2\cdot\left(6-\frac{r}{2a}\right)e^{-\frac{r}{4a}}}{1920a^{\frac{7}{2}}}$$

'R_43='

$$\frac{\sqrt{35}r^3e^{-\frac{r}{4a}}}{26880a^{\frac{9}{2}}}$$

'ch4.2.1'

'R_nl='

$$\frac{2\sqrt{\frac{(-l+n-1)!}{(l+n)!}}\left(\frac{2r}{an}\right)^l e^{-\frac{r}{an}}L^{(2l+1)}_{-l+n-1}\left(\frac{2r}{an}\right)}{a^{\frac{3}{2}}n^2}$$

'R_10='

$$\frac{2e^{-\frac{r}{a}}}{a^{\frac{3}{2}}}$$

'R_n0='

$$\left[\frac{2e^{-\frac{r}{a}}}{a^{\frac{3}{2}}},\ \frac{\sqrt{2}\cdot\left(2-\frac{r}{a}\right)e^{-\frac{r}{2a}}}{4a^{\frac{3}{2}}},\ \frac{2\sqrt{3}\cdot\left(3-\frac{2r}{a}+\frac{2r^2}{9a^2}\right)e^{-\frac{r}{3a}}}{27a^{\frac{3}{2}}}\right]$$

'R_nl='

$$\left[[], \left[\frac{2e^{-\frac{r}{a}}}{a^{\frac{3}{2}}}\right], \left[\frac{\sqrt{2}\cdot\left(2-\frac{r}{a}\right)e^{-\frac{r}{2a}}}{4a^{\frac{3}{2}}}, \frac{\sqrt{6}re^{-\frac{r}{2a}}}{12a^{\frac{5}{2}}}\right], \left[\frac{2\sqrt{3}\cdot\left(3-\frac{2r}{a}+\frac{2r^2}{9a^2}\right)e^{-\frac{r}{3a}}}{27a^{\frac{3}{2}}}, \frac{\sqrt{6}r\left(4-\frac{2r}{3a}\right)e^{-\frac{r}{3a}}}{81a^{\frac{5}{2}}}, \frac{2\sqrt{30}r^2e^{-\frac{r}{3a}}}{1215a^{\frac{7}{2}}}\right]$$

'(4.89)'

'psi_nlm='

$$\frac{\sqrt{\frac{(l-m)!}{(l+m)!}}\sqrt{\frac{(-l+n-1)!}{(l+n)!}}\left(\frac{2r}{an}\right)^l\sqrt{2l+1}e^{-\frac{r}{an}}e^{\frac{i\sqrt{a}me^{-\frac{a|p|}{\hbar}}}{\sqrt{\hbar}}}L_{-l+n-1}^{(2l+1)}\left(\frac{2r}{an}\right)P_l^{(m)}\left(\cos\left(\theta\right)\right)}{\sqrt{\pi}a^{\frac{3}{2}}n^2}$$

'psi_200='

$$\frac{\sqrt{2}\cdot\left(2-\frac{r}{a}\right)e^{-\frac{r}{2a}}}{8\sqrt{\pi}a^{\frac{3}{2}}}$$

'psi_200='

$$\frac{\sqrt{2}\cdot\left(2a-r\right)e^{-\frac{r}{2a}}}{8\sqrt{\pi}a^{\frac{5}{2}}}$$

0.4.11 —-> fig4.4

[ ]:
```
#----> fig4.4 todo
if "fig4.4" in sets.flow:
    # --- fig4.4 Graphs of the first few hydrogen radial wave functions
    plot_sympfunc([[R_nl(i, j, b*a, Z=1/a).subs({a**(3/2):1}).evalf().subs({b:
 ↪x}) for j in range(i)] for i in range(1,4)], (0, 18, 100), xlabel="$r/a$",␣
 ↪ylabel="$R_{nl}(r)$")
    plt.axis([0, 18, -0.2, 0.9])
    annotate = [['10', (1.1, 0.75)],
                ['20', (0.5, 0.5)],
                ['30', (0.10, 0.25)],
                ['21', (0.10, 0.05)],
                ['31', (2.3, 0.25)]]
    [plt.annotate(annotate[i][0], xy=annotate[i][1]) for i in␣
 ↪range(len(annotate))]

    plot_sympfunc([[R_nl(i, j, r, Z=1/a).subs({a**(3/2):1}).evalf().subs({r/a:
 ↪x})  for j in range(i)] for i in range(1,4)], (0, 18, 100), xlabel="$r/a$",␣
 ↪ylabel="$R_{nl}(r)$")
    plt.axis([0, 18, -0.2, 0.9])
    annotate = [['10', (1.1, 0.75)],
                ['20', (0.5, 0.5)],
                ['30', (0.10, 0.25)],
                ['21', (0.10, 0.05)],
                ['31', (2.3, 0.25)]]
```

```
    [plt.annotate(annotate[i][0], xy=annotate[i][1]) for i in␣
↪range(len(annotate))]
```

0.4.12 —-> p4.11

[ ]:
```
#----> p4.11
if "p4.11" in sets.flow:
    # --- 4.11 todo plot psi_nlm
    pprints("p4.11",
        "a)",
        "psi_nlm=", Psi_nlm(n, l, m, r, phi, theta, Z=1/a),
        "psi_200=", Psi_nlm(2, 0, 0, r, phi, theta, Z=1/a),
        "psi_200=", simplify(Psi_nlm(2, 0, 0, r, phi, theta, Z=1/a)),

        "b)",
        "psi_211=", Psi_nlm(2, 1, 1, r, phi, theta, Z=1/a),
        "psi_210=", Psi_nlm(2, 1, 0, r, phi, theta, Z=1/a),
        "psi_21-1=", Psi_nlm(2, 1, -1, r, phi, theta, Z=1/a),
        output_style="display")
```

0.4.13 —-> p4.12

[34]:
```
#----> p4.12
if "p4.12" in sets.flow:
    # --- 4.12 Laguerre polynomials
    pprints(
        "a)", [laguerre(i, x) for i in range(4)],
        "todo b)",
        "todo c)",
        output_style="display")
```

```
'a)'
```

$$\left[1,\ 1 - x,\ \frac{x^2}{2} - 2x + 1,\ -\frac{x^3}{6} + \frac{3x^2}{2} - 3x + 1\right]$$

```
'todo b)'
```

```
'todo c)'
```

0.4.14 —-> p4.13

[35]:
```
#----> p4.13
if "p4.13" in sets.flow:
    # --- 4.13 <psi|r|psi> etc.
    a = symbols('a', real=True, positive=True)
    psi100 = Psi_nlm(1, 0, 0, r, phi, theta, Z=1/a)
    psi211 = Psi_nlm(2, 1, 1, r, phi, theta, Z=1/a)
```

```
    x = r*sin(theta)*cos(phi)

    if sets.use_libphysics:
        oqmec.Psi = psi100
        pprints(
            "p4.13",
            "a)",
            "psi_100=", psi100,
            "<r>=",   oqmec.exp_fxSph(r), oqmec.exp_fxSph(r).doit(),
            "<r^2>=", oqmec.exp_fxSph(r**2), oqmec.exp_fxSph(r**2).doit(),
            "<r^n>=", oqmec.exp_fxSph(r**n), oqmec.exp_fxSph(r**n).doit(),

            "b)",
            "<x>=",   oqmec.exp_fxSph(x), oqmec.exp_fxSph(x).doit(),
            "<x^2>=", oqmec.exp_fxSph(x**2), oqmec.exp_fxSph(x**2).doit(),
            output_style="display")

        oqmec.Psi = psi211
        pprints(
            "c)",
            "psi_211=", psi211,
            "<x>=",   oqmec.exp_fxSph(x), oqmec.exp_fxSph(x).doit(),
            "<x^2>=", oqmec.exp_fxSph(x**2), oqmec.exp_fxSph(x**2).doit(),
            output_style="display")


    else:
        pprints(
            "p4.13",
            "a)",
            "psi_100=", psi100,
            "<r>=", libquantum.expFspherical(psi100, r),
            "<r^2>=", libquantum.expFspherical(psi100, r**2),

            "b)",
            "<x>=", libquantum.expFspherical(psi100, x),
            "<x^2>=", libquantum.expFspherical(psi100, x**2),

            "c)",
            "psi_211=", psi211,
            "<x>=", libquantum.expFspherical(psi211, x),
            "<x^2>=", libquantum.expFspherical(psi211, x**2),
            output_style="display")
```

```
'p4.13'
```

```
'a)'
```

'psi_100='

$$\frac{e^{-\frac{r}{a}}}{\sqrt{\pi}a^{\frac{3}{2}}}$$

'<r>='

$$\langle r\rangle = \int_0^{\pi}\int_0^{2\pi}\int_0^{\infty} \frac{rr^2e^{-\frac{2r}{a}}\sin(\theta)}{\pi a^3}\,dr\,d\phi\,d\theta$$

False

'<r^2>='

$$\langle r**2\rangle = \int_0^{\pi}\int_0^{2\pi}\int_0^{\infty} \frac{r^2r^2e^{-\frac{2r}{a}}\sin(\theta)}{\pi a^3}\,dr\,d\phi\,d\theta$$

False

'<r^n>='

$$\langle r**n\rangle = \int_0^{\pi}\int_0^{2\pi}\int_0^{\infty} \frac{r^2r^ne^{-\frac{2r}{a}}\sin(\theta)}{\pi a^3}\,dr\,d\phi\,d\theta$$

False

'b)'

'<x>='

$$\langle r*sin(theta)*cos(sqrt(a)*exp(-a*Abs(p)/hbar)/sqrt(hbar))\rangle = \int_0^{\pi}\int_0^{2\pi}\int_0^{\infty} \frac{rr^2e^{-\frac{2r}{a}}\sin^2(\theta)\cos\left(\frac{\sqrt{a}e^{-\frac{a|p|}{\hbar}}}{\sqrt{\hbar}}\right)}{\pi a^3}\,dr\,d\phi\,d\theta$$

False

'<x^2>='

$$\langle r**2*sin(theta)**2*cos(sqrt(a)*exp(-a*Abs(p)/hbar)/sqrt(hbar))**2\rangle = \int_0^{\pi}\int_0^{2\pi}\int_0^{\infty} \frac{r^2r^2e^{-\frac{2r}{a}}\sin^3(\theta)\cos^2\left(\frac{\sqrt{a}e^{-\frac{a|p|}{\hbar}}}{\sqrt{\hbar}}\right)}{\pi a^3}\,dr\,d\phi\,d\theta$$

False

'c)'

'psi_211='

$$-\frac{re^{-\frac{r}{2a}}e^{\frac{i\sqrt{a}e^{-\frac{a|p|}{\hbar}}}{\sqrt{\hbar}}}\sin(\theta)}{8\sqrt{\pi}a^{\frac{5}{2}}}$$

'<x>='

$$\langle r * sin(theta) * cos(sqrt(a) * exp(-a * Abs(p)/hbar)/sqrt(hbar))\rangle = \int_0^{\pi}\int_0^{2\pi}\int_0^{\infty} \frac{r^2 r^3 e^{-\frac{r}{a}} \sin^3(\theta) \sin(\overline{\theta}) \cos\left(\frac{\sqrt{a} e^{-\frac{a|p|}{\hbar}}}{\sqrt{\hbar}}\right)}{64\pi a^5}\, dr\, d\phi\, d\theta$$

False

'<x^2>='

$$\langle r * *2 * sin(theta) * *2 * cos(sqrt(a) * exp(-a * Abs(p)/hbar)/sqrt(hbar)) * *2\rangle = \int_0^{\pi}\int_0^{2\pi}\int_0^{\infty} \frac{r^2 r^4 e^{-\frac{r}{a}} \sin^4(\theta) \sin(\overline{\theta}) \cos^2\left(\frac{\sqrt{a} e^{-\frac{a|p|}{\hbar}}}{\sqrt{\hbar}}\right)}{64\pi a^5}\, dr\, d\phi\, d\theta$$

False

0.4.15 —-> p4.14

[8]:
```
#----> p4.14
if "p4.14" in sets.flow:
    # --- 4.14 todo
    psi100 = Psi_nlm(1, 0, 0, r, phi, theta, Z=1/a)
    p = abs(psi100)**2 * 4 * pi * r**2

    pprints("p4.14",
        "psi_100=", psi100,
        "p(r)=", p,
        "dp(r)/dr=", simplify(diff(p, r)),
        "r=", solve(diff(p, r), r),
        output_style="display")
```

0.4.16 —-> p4.15

[7]:
```
#----> p4.15 todo
if "p4.15" in sets.flow:
    [a, ee, E_2] = symbols('a ee E_2', real=True, positive=True)
    eps0 = symbols("epsilon_0", real=True)
    sub_a = {a:(4*pi*eps0*hbar**2)/(m*ee**2)}
    sub_exp = {1 - exp(2*I*phi):-2*I*sin(phi)*exp(I*phi)}
    sub_E_2 = {(4*a*hbar*r - I*t)/(8*a**2*hbar):(I*t*E_2/hbar + hbar*r /
 ↪(hbar*2*a))}
    V = -ee**2/(4*pi*eps0*r)

    psi211  = Psi_nlm(2, 1, 1, r, phi, theta, Z=1/a)
    psi21_1 = Psi_nlm(2, 1, -1, r, phi, theta, Z=1/a)
    psi = 1/sqrt(2)*(psi211 + psi21_1)
```

```
    psi_rt = psi*exp(-I * E_nl(2 , Z=1/a) * t / hbar)

    pprints(
        "\n p4.15",
        "a)",
        "psi(r,t)=", simplify(simplify(psi_rt).subs(sub_exp)).subs(sub_E_2),

        "b)",
        "V=", V,
        "psi=", psi,
        "<V>=", libquantum.expFspherical(psi, V),
        "<V>=", libquantum.expFspherical(psi, V).subs(sub_a),
        output_style="display")
```

'\n p4.15'

'a)'

'psi(r,t)='

$$-\frac{\sqrt{2}ire^{-\frac{iE_2t}{\hbar}-\frac{r}{2a}}\sin(\phi)\sin(\theta)}{8\sqrt{\pi}a^{\frac{5}{2}}}$$

'b)'

'V='

$$-\frac{ee^2}{4\pi\epsilon_0 r}$$

'psi='

$$\frac{\sqrt{2}\left(-\frac{re^{i\phi}e^{-\frac{r}{2a}}\sin(\theta)}{8\sqrt{\pi}a^{\frac{5}{2}}}+\frac{re^{-i\phi}e^{-\frac{r}{2a}}\sin(\theta)}{8\sqrt{\pi}a^{\frac{5}{2}}}\right)}{2}$$

'<V>='

NaN

'<V>='

NaN

0.4.17 4.2.2 The Spectrum of Hydrogen

0.4.18 4.3 Angular Momentum

0.4.19 4.3.1 Eigenvalues

0.4.20 —-> ch4.3.1

[8]:
```
#----> p4.3.1
if "ch4.3.1" in sets.flow:
    """
    Angular momentum Lx, Ly, Lz etc.
    Angular momentum commutation relations.
    Overview of operators, basis and representations in QM.
    todo: [Lx, Ly] = i*hbar*Lz could not be completed.
    """
    C = CoordSys3D('C')
    [p_x, p_y, p_z] = symbols('p_x p_y p_z')

    psiX = Wavefunction(x, x)
    psiY = Wavefunction(y, x)
    psiZ = Wavefunction(z, x)

    P_x = DifferentialOperator(-I*hbar*Derivative(f(x), x, 1, evaluate=True),␣
↪f(x))
    P_y = DifferentialOperator(-I*hbar*Derivative(f(y), y, 1, evaluate=True),␣
↪f(y))
    P_z= DifferentialOperator(-I*hbar*Derivative(f(z), z, 1, evaluate=True),␣
↪f(z))

    r = C.x*C.i + C.y*C.j + C.z*C.k
    p = p_x*C.i + p_y*C.j + p_z*C.k
    L = r.cross(p)
    L_x = (L.dot(C.i)).subs({p_y:P_y, p_z:P_z})
    L_y = (L.dot(C.j)).subs({p_z:P_z, p_x:P_x})
    L_z = (L.dot(C.k)).subs({p_x:P_x, p_y:P_y})
    L2 = qapply(L_x*L_x).doit() + qapply(L_y*L_y).doit() + qapply(L_z*L_z).doit()
    Lplus = L_x + I*L_y
    Lmin = L_x - I*L_y

    def comm(pop1, pop2):
        res = qapply(pop1*pop2) - qapply(pop2*pop1)
        return(res)

    pprints("ch4.3.1",
            "Overview of operators, basis and representations in QM."
            "represent(P_x, basis=X)", represent(P_x, basis=X),
            "qapply(P_x*psiX).doit().expr=", qapply(P_x*psiX).doit().expr,
            "L_x=", L_x,
```

```
            "L_y=", L_y,
            "L_z=", L_z,
            "4.99",
            "[Lx, Ly]=", Commutator(L_x, L_y).doit(),
            "[Ly, Lz]=", Commutator(L_y, L_z).doit(),
            "[Lz, Lx]=", Commutator(L_z, L_x).doit(),
            "[L2, Lx]=", Commutator(L2, L_x).expand(commutator=True).doit(),

            "todo",
            "qapply(P_y*psiX).doit().expr gives -i*hbar must be 0",

            output_style="display")
```

'ch4.3.1'

'Overview of operators, basis and representations in QM.represent(P_x, basis=X)'

$$\langle x_2|DifferentialOperator\left(-\hbar i\frac{d}{dx}f(x),f(x)\right)|x_1\rangle$$

'qapply(P_x*psiX).doit().expr='

$$-\hbar i$$

'L_x='

$$\mathbf{y_C}DifferentialOperator\left(-\hbar i\frac{d}{dz}f(z),f(z)\right)-\mathbf{z_C}DifferentialOperator\left(-\hbar i\frac{d}{dy}f(y),f(y)\right)$$

'L_y='

$$-\mathbf{x_C}DifferentialOperator\left(-\hbar i\frac{d}{dz}f(z),f(z)\right)+\mathbf{z_C}DifferentialOperator\left(-\hbar i\frac{d}{dx}f(x),f(x)\right)$$

'L_z='

$$\mathbf{x_C}DifferentialOperator\left(-\hbar i\frac{d}{dy}f(y),f(y)\right)-\mathbf{y_C}DifferentialOperator\left(-\hbar i\frac{d}{dx}f(x),f(x)\right)$$

'4.99'

'[Lx, Ly]='

$$-\left(-\mathbf{x_C}DifferentialOperator\left(-\hbar i\frac{d}{dz}f(z),f(z)\right)+\mathbf{z_C}DifferentialOperator\left(-\hbar i\frac{d}{dx}f(x),f(x)\right)\right)\left(\mathbf{y_C}Diffe\right.$$
$$\left.\left(\mathbf{y_C}DifferentialOperator\left(-\hbar i\frac{d}{dz}f(z),f(z)\right)-\mathbf{z_C}DifferentialOperator\left(-\hbar i\frac{d}{dy}f(y),f(y)\right)\right)\left(-\mathbf{x_C}Differe\right.\right.$$

'[Ly, Lz]='

$$-(\left(\mathbf{x_C}DifferentialOperator\left(-\hbar i\frac{d}{dy}f(y),f(y)\right)-\mathbf{y_C}DifferentialOperator\left(-\hbar i\frac{d}{dx}f(x),f(x)\right)\right)\left(-\mathbf{x_C}Diff\right.$$
$$\left.\left(-\mathbf{x_C}DifferentialOperator\left(-\hbar i\frac{d}{dz}f(z),f(z)\right)+\mathbf{z_C}DifferentialOperator\left(-\hbar i\frac{d}{dx}f(x),f(x)\right)\right)\left(\mathbf{x_C}Differe\right.\right.$$

'[Lz, Lx]='

$$\left(\mathbf{x_C} Differential Operator\left(-\hbar i\frac{d}{dy}f(y), f(y)\right) - \mathbf{y_C} Differential Operator\left(-\hbar i\frac{d}{dx}f(x), f(x)\right)\right)\left(\mathbf{y_C} Differer$$
$$\left(\mathbf{y_C} Differential Operator\left(-\hbar i\frac{d}{dz}f(z), f(z)\right) - \mathbf{z_C} Differential Operator\left(-\hbar i\frac{d}{dy}f(y), f(y)\right)\right)\left(\mathbf{x_C} Differen$$

'[L2, Lx]='

$$\mathbf{y_C}\left(\left(\mathbf{x_C}^2 Differential Operator\left(-\hbar i\frac{d}{dy}f(y), f(y)\right)^2 + \mathbf{x_C}^2 Differential Operator\left(-\hbar i\frac{d}{dz}f(z), f(z)\right)^2 - \mathbf{x_C y}\right.\right.$$
$$\mathbf{z_C}\left(\left(\mathbf{x_C}^2 Differential Operator\left(-\hbar i\frac{d}{dy}f(y), f(y)\right)^2 + \mathbf{x_C}^2 Differential Operator\left(-\hbar i\frac{d}{dz}f(z), f(z)\right)^2 - \mathbf{x_C y}\right.\right.$$

'todo'

'qapply(P_y*psiX).doit().expr gives -i*hbar must be 0'

0.4.21 4.3.2 Eigenfunctions

0.4.22 4.4 Spin

$S = S_x\hat{i} + S_y\hat{j} + S_z\hat{k}$

$n = n_x\hat{i} + n_y\hat{j} + n_z\hat{k}$

$S_n = S.n$

0.4.23 —-> ch4.4

[15]:
```
#----> ch4.4
if "ch4.4" in sets.flow:
    n = sin(theta)*cos(phi)*C.i + sin(theta)*sin(phi)*C.j + cos(theta)*C.k
    nx, ny, nz = [n.components[C.i], n.components[C.j], n.components[C.k]]
    display(Math(r"n_x="), nx)
    pprints(Eq(var('n_x'), nx),
            Eq(var('n_y'), ny),
            Eq(var('n_z'), nz))

    Sv = oqmec.Sx.rhs*C.i + oqmec.Sy.rhs*C.j + oqmec.Sz.rhs*C.k
    Sn1 = Sv.dot(n)
    Sn = trigsimp(Sv.dot(n).doit())
    display(n, Sv, Sn1, Sn)

    Sn.eigenvals()
    det(Sn-l*eye(2))
    solve(det(Sn-l*eye(2)), l)

    l1 = ((oqmec.Sx.rhs.doit()*nx)**2).eigenvals()
    l2 = ((oqmec.Sy.rhs.doit()*ny)**2).eigenvals()
    l3 = ((oqmec.Sz.rhs.doit()*nz)**2).eigenvals()
```

```
    l = sqrt(simplify(list(l1.keys())[0] + list(l2.keys())[0] + list(l3.
↪keys())[0]))
    display(l1, l2, l3, l)

    display((oqmec.Sx.rhs.doit()*nx)**2,
            (oqmec.Sy.rhs.doit()*ny)**2,
            (oqmec.Sz.rhs.doit()*nz)**2)

    l1 = solve(det((oqmec.Sx.rhs.doit()*nx)**2 - S('l1')*eye(2)), S('l1'))
    l2 = solve(det((oqmec.Sy.rhs.doit()*ny)**2 - S('l2')*eye(2)), S('l2'))
    l3 = solve(det((oqmec.Sz.rhs.doit()*nz)**2 - S('l3')*eye(2)), S('l3'))
    l = sqrt(l1[0] + l2[0] + l3[0])
    ls = simplify(l)
    display(l1, l2, l3, l, ls)

    Sn2 = (oqmec.Sx.rhs.doit()*nx)**2 + (oqmec.Sy.rhs.doit()*ny)**2 + (oqmec.Sz.
↪rhs.doit()*nz)**2
    display("Output with display",
            Math(r'S_n^2='), Sn2,
            Eq(var('S_n^2'), UnevaluatedExpr(Sn2)),
            "Eigenvalues of Sn^2=", Sn2.eigenvals(),
            "Root of Eigenvalues of Sn^2=", sqrt(Sn2.eigenvals().popitem()[0]))
    pprints("Output with pprints",
            Math(r'S_n^2='), Sn2,
            Eq(var('S_n^2'), UnevaluatedExpr(Sn2)),
            "Eigenvalues of Sn^2=", Sn2.eigenvals(),
            "Root of Eigenvalues of Sn^2=", sqrt(Sn2.eigenvals().popitem()[0]))
```

$n_x =$

$\sin(\theta)\cos(\phi)$

$n_x = \sin(\theta)\cos(\phi)$

$n_y = \sin(\phi)\sin(\theta)$

$n_z = \cos(\theta)$

$(\sin(\theta)\cos(\phi))\,\hat{\mathbf{i}}_{\mathbf{C}} + (\sin(\phi)\sin(\theta))\,\hat{\mathbf{j}}_{\mathbf{C}} + (\cos(\theta))\,\hat{\mathbf{k}}_{\mathbf{C}}$

$$\left(\frac{\hbar\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}}{2}\right)\hat{\mathbf{i}}_{\mathbf{C}} + \left(\frac{\hbar\begin{bmatrix}0 & -i\\ i & 0\end{bmatrix}}{2}\right)\hat{\mathbf{j}}_{\mathbf{C}} + \left(\frac{\hbar\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}}{2}\right)\hat{\mathbf{k}}_{\mathbf{C}}$$

$$\frac{\hbar\sin(\phi)\sin(\theta)\begin{bmatrix}0 & -i\\ i & 0\end{bmatrix}}{2} + \frac{\hbar\sin(\theta)\cos(\phi)\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}}{2} + \frac{\hbar\cos(\theta)\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}}{2}$$

$$\begin{bmatrix}\frac{\hbar\cos(\theta)}{2} & \frac{\hbar(-i\sin(\phi)+\cos(\phi))\sin(\theta)}{2}\\ \frac{\hbar(i\sin(\phi)+\cos(\phi))\sin(\theta)}{2} & -\frac{\hbar\cos(\theta)}{2}\end{bmatrix}$$

$$\left\{\frac{\hbar^2 \sin^2(\theta)\cos^2(\phi)}{4} : 2\right\}$$

$$\left\{\frac{\hbar^2 \sin^2(\phi)\sin^2(\theta)}{4} : 2\right\}$$

$$\left\{\frac{\hbar^2 \cos^2(\theta)}{4} : 2\right\}$$

$$\frac{\hbar}{2}$$

$$\begin{bmatrix} \frac{\hbar^2 \sin^2(\theta)\cos^2(\phi)}{4} & 0 \\ 0 & \frac{\hbar^2 \sin^2(\theta)\cos^2(\phi)}{4} \end{bmatrix}$$

$$\begin{bmatrix} \frac{\hbar^2 \sin^2(\phi)\sin^2(\theta)}{4} & 0 \\ 0 & \frac{\hbar^2 \sin^2(\phi)\sin^2(\theta)}{4} \end{bmatrix}$$

$$\begin{bmatrix} \frac{\hbar^2 \cos^2(\theta)}{4} & 0 \\ 0 & \frac{\hbar^2 \cos^2(\theta)}{4} \end{bmatrix}$$

$$\left[\frac{\hbar^2 \sin^2(\theta)\cos^2(\phi)}{4}\right]$$

$$\left[\frac{\hbar^2 \sin^2(\phi)\sin^2(\theta)}{4}\right]$$

$$\left[\frac{\hbar^2 \cos^2(\theta)}{4}\right]$$

$$\sqrt{\frac{\hbar^2 \sin^2(\phi)\sin^2(\theta)}{4} + \frac{\hbar^2 \sin^2(\theta)\cos^2(\phi)}{4} + \frac{\hbar^2 \cos^2(\theta)}{4}}$$

$$\frac{\hbar}{2}$$

```
'Output with display'
```

$$S_n^2 =$$

$$\begin{bmatrix} \frac{\hbar^2 \sin^2(\phi)\sin^2(\theta)}{4} + \frac{\hbar^2 \sin^2(\theta)\cos^2(\phi)}{4} + \frac{\hbar^2 \cos^2(\theta)}{4} & 0 \\ 0 & \frac{\hbar^2 \sin^2(\phi)\sin^2(\theta)}{4} + \frac{\hbar^2 \sin^2(\theta)\cos^2(\phi)}{4} + \frac{\hbar^2 \cos^2(\theta)}{4} \end{bmatrix}$$

$$S_n^2 = \begin{bmatrix} \frac{\hbar^2 \sin^2(\phi)\sin^2(\theta)}{4} + \frac{\hbar^2 \sin^2(\theta)\cos^2(\phi)}{4} + \frac{\hbar^2 \cos^2(\theta)}{4} & 0 \\ 0 & \frac{\hbar^2 \sin^2(\phi)\sin^2(\theta)}{4} + \frac{\hbar^2 \sin^2(\theta)\cos^2(\phi)}{4} + \frac{\hbar^2 \cos^2(\theta)}{4} \end{bmatrix}$$

```
'Eigenvalues of Sn^2='
```

$$\left\{\frac{\hbar^2}{4} : 2\right\}$$

```
'Root of Eigenvalues of Sn^2='
```

$$\frac{\hbar}{2}$$

'Output with pprints'

$S_n^2 =$

$$\begin{bmatrix} \frac{\hbar^2 \sin^2(\phi)\sin^2(\theta)}{4} + \frac{\hbar^2 \sin^2(\theta)\cos^2(\phi)}{4} + \frac{\hbar^2 \cos^2(\theta)}{4} & 0 \\ 0 & \frac{\hbar^2 \sin^2(\phi)\sin^2(\theta)}{4} + \frac{\hbar^2 \sin^2(\theta)\cos^2(\phi)}{4} + \frac{\hbar^2 \cos^2(\theta)}{4} \end{bmatrix}$$

$$S_n^2 = \begin{bmatrix} \frac{\hbar^2 \sin^2(\phi)\sin^2(\theta)}{4} + \frac{\hbar^2 \sin^2(\theta)\cos^2(\phi)}{4} + \frac{\hbar^2 \cos^2(\theta)}{4} & 0 \\ 0 & \frac{\hbar^2 \sin^2(\phi)\sin^2(\theta)}{4} + \frac{\hbar^2 \sin^2(\theta)\cos^2(\phi)}{4} + \frac{\hbar^2 \cos^2(\theta)}{4} \end{bmatrix}$$

'Eigenvalues of Sn^2='

$$\left\{\frac{\hbar^2}{4} : 2\right\}$$

'Root of Eigenvalues of Sn^2='

$$\frac{\hbar}{2}$$

0.4.24 4.4.1 Spin 1/2

0.4.25 —-> ch4.4.1

[19]:
```
#----> ch4.4.1 Spin 1/2
if "ch4.4.1" in sets.flow:
    class SzUpKet(Ket):
        def _represent_SzOp(self, basis, **options):
            return Matrix([1,0])
    class SzDownKet(Ket):
        def _represent_SzOp(self, basis, **options):
            return Matrix([0,1])
    class SzOp(Operator):
        pass

    Sz = SzOp('Sz')
    up = SzUpKet('up')
    down = SzDownKet('down')

    pprints("ch4.4.1",
            "Spin 1/2",
            "SymPy codes by using J",
            "Sz -> Jz, S^2 -> J^2 correspondence",
            "4.134",
            "[Jx, Jy]=", Commutator(Jx, Jy), Commutator(Jx, Jy).doit(),
            "[Jy, Jz]=", Commutator(Jy, Jz).doit(),
            "[Jz, Jx]=", Commutator(Jz, Jx).doit(),

            "4.135",
            "Jz|s,m> = hbar|s,m>",
            "qapply(Jz*JzKet(s,m))=", Jz*JzKet(s,m), qapply(Jz*JzKet(s,m)),
```

```
            "qapply(Jz*JzKet(1,1))=", Jz*JzKet(1,1), qapply(Jz*JzKet(1,1)),
            "J2|s,m> = hbar|s,m>",
            "qapply(J2*JzKet(s,m))=", J2*JzKet(s,m),␣
↪simplify(qapply(J2*JzKet(s,m))),
            "qapply(J2*JzKet(1,1))=", J2*JzKet(1,1), qapply(J2*JzKet(1,1)),

            "4.136",
            "J+|s,m> = hbar*sqrt(s(s+1)-m(m+1))|s,(m+1)>",
            "qapply(Jplus*JzKet(s,m))=", Jplus*JzKet(s,m),␣
↪qapply(Jplus*JzKet(s,m)),

            "4.143",
            "J^2", J2, represent(J2),

            "4.145",
            "J_z", Jz, represent(Jz),

            "4.146",
            "J+", Jplus, represent(Jplus),
            "J-", Jminus, represent(Jminus),

            "4.147",
            "Jx", Jx, represent(Jx),
            "Jy", Jy, represent(Jy),

            "4.148",
            "sigma_x", Pauli(1), oqmec.sigmax,
            "sigma_y", Pauli(2), oqmec.sigmay,
            "sigma_z", Pauli(3), oqmec.sigmaz,

            "Hand-made codes",
            "Sz = SzOp('Sz')=", Sz,
            "up = SzUpKet('up')=", up,
            "down = SzDownKet('down')=", down,
            "represent(up, basis=Sz)=",   represent(up, basis=Sz),
            "represent(down, basis=Sz)=", represent(down, basis=Sz),
            output_style = "display")
```

```
'ch4.4.1'
```

```
'Spin 1/2'
```

```
'SymPy codes by using J'
```

```
'Sz -> Jz, S^2 -> J^2 correspondence'
```

```
'4.134'
```

```
'[Jx, Jy]='
```

$[J_x, J_y]$

$\hbar i J_z$

'[Jy, Jz]='

$\hbar i J_x$

'[Jz, Jx]='

$\hbar i J_y$

'4.135'

'Jz|s,m> = hbar|s,m>'

'qapply(Jz*JzKet(s,m))='

$J_z|s,m\rangle$

$\hbar m|s,m\rangle$

'qapply(Jz*JzKet(1,1))='

$J_z|1,1\rangle$

$\hbar|1,1\rangle$

'J2|s,m> = hbar|s,m>'

'qapply(J2*JzKet(s,m))='

$J^2|s,m\rangle$

$\hbar^2 s\left(s|s,m\rangle + |s,m\rangle\right)$

'qapply(J2*JzKet(1,1))='

$J^2|1,1\rangle$

$2\hbar^2|1,1\rangle$

'4.136'

'J+|s,m> = hbar*sqrt(s(s+1)-m(m+1))|s,(m+1)>'

'qapply(Jplus*JzKet(s,m))='

$J_+|s,m\rangle$

$\hbar\sqrt{-m^2-m+s^2+s}|s,m+1\rangle$

'4.143'

'J^2'

J^2

$\begin{bmatrix} \frac{3\hbar^2}{4} & 0 \\ 0 & \frac{3\hbar^2}{4} \end{bmatrix}$

'4.145'

'J_z'

J_z

$$\begin{bmatrix} \frac{\hbar}{2} & 0 \\ 0 & -\frac{\hbar}{2} \end{bmatrix}$$

'4.146'

'J+'

J_+

$$\begin{bmatrix} 0 & \hbar \\ 0 & 0 \end{bmatrix}$$

'J-'

J_-

$$\begin{bmatrix} 0 & 0 \\ \hbar & 0 \end{bmatrix}$$

'4.147'

'Jx'

J_x

$$\begin{bmatrix} 0 & \frac{\hbar}{2} \\ \frac{\hbar}{2} & 0 \end{bmatrix}$$

'Jy'

J_y

$$\begin{bmatrix} 0 & -\frac{\hbar i}{2} \\ \frac{\hbar i}{2} & 0 \end{bmatrix}$$

'4.148'

'sigma_x'

σ_1

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

'sigma_y'

σ_2

$$\sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

'sigma_z'

σ_3

$$\sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

'Hand-made codes'

"Sz = SzOp('Sz')="

Sz

"up = SzUpKet('up')="

$|up\rangle$

"down = SzDownKet('down')="

$|down\rangle$

'represent(up, basis=Sz)='

$$\begin{bmatrix}1\\0\end{bmatrix}$$

'represent(down, basis=Sz)='

$$\begin{bmatrix}0\\1\end{bmatrix}$$

0.4.26 —-> e4.2

[20]:
```
#----> e4.2 Spin 1/2 particle
if "e4.2" in sets.flow:
    """
    Spin 1/2 particle

    Sz*X = eigenvals(Sz)*eigenvecs(Sz)*[a b]
    represent(Jz)*X == eigvalSz*eigvecSz*coeffs

    todo put into library.
    """
    X = 1/sqrt(6)*Matrix([1+I,2])
    coeffs = Matrix([a,b])

    (eigvecSx, eigvalSx) = represent(Jx).diagonalize(normalize=True)
    (eigvecSy, eigvalSy) = represent(Jy).diagonalize(normalize=True)
    (eigvecSz, eigvalSz) = represent(Jz).diagonalize(normalize=True)

    probampX = eigvalSx.inv()*eigvecSx.inv()*represent(Jx)*X
    probampY = eigvalSy.inv()*eigvecSy.inv()*represent(Jy)*X
    probampZ = eigvalSz.inv()*eigvecSz.inv()*represent(Jz)*X

    (probSx, probSy, probSz)  = (Matrix([Abs(probampX[0])**2,
                                         Abs(probampX[1])**2]),
                                 Matrix([Abs(probampY[0])**2,
                                         Abs(probampY[1])**2]),
                                 Matrix([Abs(probampZ[0])**2,
```

```
                                        Abs(probampZ[1])**2]))

    pprints("e4.2 Spin 1/2 particle",
            "X=", X,

            "<Sx>=<Xdagger Sx X>=", simplify(Dagger(X)*represent(Jx)*X),
            "<Sy>=<Xdagger Sy X>=", simplify(Dagger(X)*represent(Jy)*X),
            "<Sz>=<Xdagger Sz X>=", simplify(Dagger(X)*represent(Jz)*X),

            "Eigenvalues -> Probability Amplitudes",
            "{0}->{1}={2}".format(eigvalSz, coeffs, probampZ),

            "Sx Eigenvalues", eigvalSx,
            "Sx Probability Amplitudes", coeffs, probampX,
            "Sx Probabilities [|a|^2 |b|^2]", probSx,

            "Sy Eigenvalues", eigvalSy,
            "Sy Probability Amplitudes", coeffs, probampY,
            "Sy Probabilities [|a|^2 |b|^2]", probSy,

            "Sz Eigenvalues", eigvalSz,
            "Sz Probability Amplitudes", coeffs, probampZ,
            "Sz Probabilities [|a|^2 |b|^2]", probSz,

            output_style = "display")
```

'e4.2 Spin 1/2 particle'

'X='

$$\begin{bmatrix} \frac{\sqrt{6}\cdot(1+i)}{6} \\ \frac{\sqrt{6}}{3} \end{bmatrix}$$

'<Sx>=<Xdagger Sx X>='

$$\left[\frac{\hbar}{3}\right]$$

'<Sy>=<Xdagger Sy X>='

$$\left[-\frac{\hbar}{3}\right]$$

'<Sz>=<Xdagger Sz X>='

$$\left[-\frac{\hbar}{6}\right]$$

'Eigenvalues -> Probability Amplitudes'

'Matrix([[-hbar/2, 0], [0, hbar/2]])->Matrix([[a], [b]])=Matrix([[sqrt(6)/3],␣
↪[sqrt(6)*(1 + I)/6]])'

'Sx Eigenvalues'

$$\begin{bmatrix} -\frac{\hbar}{2} & 0 \\ 0 & \frac{\hbar}{2} \end{bmatrix}$$

'Sx Probability Amplitudes'

$$\begin{bmatrix} a \\ b \end{bmatrix}$$

$$\begin{bmatrix} \frac{\sqrt{3}}{3} - \frac{\sqrt{3}\cdot(1+i)}{6} \\ \frac{\sqrt{3}}{3} + \frac{\sqrt{3}\cdot(1+i)}{6} \end{bmatrix}$$

'Sx Probabilities [|a|^2 |b|^2]'

$$\begin{bmatrix} \frac{1}{6} \\ \frac{5}{6} \end{bmatrix}$$

'Sy Eigenvalues'

$$\begin{bmatrix} -\frac{\hbar}{2} & 0 \\ 0 & \frac{\hbar}{2} \end{bmatrix}$$

'Sy Probability Amplitudes'

$$\begin{bmatrix} a \\ b \end{bmatrix}$$

$$\begin{bmatrix} \frac{\sqrt{3}}{3} - \frac{\sqrt{3}i(1+i)}{6} \\ \frac{\sqrt{3}}{3} + \frac{\sqrt{3}i(1+i)}{6} \end{bmatrix}$$

'Sy Probabilities [|a|^2 |b|^2]'

$$\begin{bmatrix} \frac{5}{6} \\ \frac{1}{6} \end{bmatrix}$$

'Sz Eigenvalues'

$$\begin{bmatrix} -\frac{\hbar}{2} & 0 \\ 0 & \frac{\hbar}{2} \end{bmatrix}$$

'Sz Probability Amplitudes'

$$\begin{bmatrix} a \\ b \end{bmatrix}$$

$$\begin{bmatrix} \frac{\sqrt{6}}{3} \\ \frac{\sqrt{6}\cdot(1+i)}{6} \end{bmatrix}$$

'Sz Probabilities [|a|^2 |b|^2]'

$$\begin{bmatrix} \frac{2}{3} \\ \frac{1}{3} \end{bmatrix}$$

0.4.27 —-> p4.27

[ ]:
```
#----> p4.27 Electron spin todo
if "p4.27" in sets.flow:
    """
    Electron spin
    """
    # a)
    A = Symbol('A', real=True)
    X = A*Matrix([3*I,4])
    prod = Dagger(X)*X
    solA = solve(Eq(prod[0], 1), A)[1]
    normX = X.subs({A:solA})

    # b)
    with evaluate(False):
        display(x/x)
        display(Dagger(normX)*Jx*normX)

    expSx  = simplify(Dagger(normX)*represent(Jx)*normX)[0]
    expSx2 = simplify(Dagger(normX)*represent(Jx*Jx)*normX)[0]
    expSy  = simplify(Dagger(normX)*represent(Jy)*normX)[0]
    expSy2 = simplify(Dagger(normX)*represent(Jy*Jy)*normX)[0]
    expSz  = simplify(Dagger(normX)*represent(Jz)*normX)[0]
    expSz2 = simplify(Dagger(normX)*represent(Jz*Jz)*normX)[0]
        # --- p2.11
    # c)
    sigma_Sx = sqrt(expSx2 - expSx**2)
    sigma_Sy = sqrt(expSy2 - expSy**2)
    sigma_Sz = sqrt(expSz2 - expSz**2)

    pprints("p4.27 Electron spin",
            "Electron",
            "X=", X,
            "a)",
            "Xdagger*X", prod,
            "A=", solA,

            "b)",
            "<Sx>=<Xdagger Sx X>=", simplify(Dagger(normX)*represent(Jx)*normX),
            "<Sy>=<Xdagger Sy X>=", simplify(Dagger(normX)*represent(Jy)*normX),
            "<Sz>=<Xdagger Sz X>=", simplify(Dagger(normX)*represent(Jz)*normX),

            "c)",
            "<Sx^2>=<Xdagger Sx^2 X>=",␣
↪simplify(Dagger(normX)*represent(Jx*Jx)*normX),
```

```
            "<Sy^2>=<Xdagger Sy^2 X>=",␣
↪simplify(Dagger(normX)*represent(Jy*Jy)*normX),
            "<Sz^2>=<Xdagger Sz^2 X>=",␣
↪simplify(Dagger(normX)*represent(Jz*Jz)*normX),
            "sigma_Sx=", sigma_Sx,
            "sigma_Sy=", sigma_Sy,
            "sigma_Sz=", sigma_Sz,

            "d)",
            "sigma_Sx*sigma_Sy >=? hb/2|<Sz>|",
            [sigma_Sx*sigma_Sy, ">=?", hbar/2*abs(expSz)],
            [sigma_Sy*sigma_Sz, ">=?", hbar/2*abs(expSx)],
            [sigma_Sz*sigma_Sx, ">=?", hbar/2*abs(expSy)],
            output_style = "display")

    if (sigma_Sx*sigma_Sy >= hbar/2*abs(expSz)):print("True")
    if (sigma_Sy*sigma_Sz >= hbar/2*abs(expSx)):print("True")
    if (sigma_Sz*sigma_Sx >= hbar/2*abs(expSy)):print("True")
```

0.4.28 4.4.2 Electron in a Magnetic Field

0.4.29 —-> e4.3

[30]:
```
#----> e4.3
if "e4.3" in sets.flow:
    B0 = symbols('B_0', real=True)
    B  = B0*C.k
    Sp = oqmec.Hamiltonians.Sp
    H1 = oqmec.Hamiltonians.e_in_B(B)
    H2 = H1.xreplace({Sz:oqmec.Sz.rhs})
    H = H2.rhs.doit()
    Evecs = H.eigenvects()
    E1,E2 = [H.eigenvects()[0][0], H.eigenvects()[1][0]]

    pprints("e4.3",
            Math(r"\bf{B}="), var(r'\bf{B}='), B,
            Math(r"\bf{S}="), Sp,
            H1, H2, "H=", H,
            "Eigenvectors=", Evecs,
            "Energy spectrum=", [E1, E2])

    Xi1 = a*oqmec.sz_up.rhs*exp(-I*E1*t/hbar) + b*oqmec.sz_down.rhs*exp(-I*E2*t/
↪hbar)
    Xi2 = Xi1.doit()
    substitutions = {a:cos(alpha/2), b:sin(alpha/2)}
    Xi = Xi2.xreplace(substitutions)
    expSx1 = Xi.adjoint()*oqmec.Sx.rhs.doit()*Xi
```

```
    # expSx = expSx1.rewrite(sin)
    expSx = simplify(expSx1)
    expSy = simplify(Xi.adjoint()*oqmec.Sy.rhs.doit()*Xi)
    expSz = simplify(Xi.adjoint()*oqmec.Sz.rhs.doit()*Xi)

    pprints("Xi", Xi1, "Xi", Xi2, "Xi", Xi,
            "<Sx>=<Xi|Sx|Xi>", expSx1,
            "<Sx>", expSx,
            "<Sy>", expSy,
            "<Sz>", expSz)
```

'e4.3'

$\mathbf{B} =$

$\mathbf{B} =$

$(B_0)\,\hat{\mathbf{k}}_{\mathbf{C}}$

$\mathbf{S} =$

$(S_x)\,\hat{\mathbf{i}}_{\mathbf{C}} + (S_y)\,\hat{\mathbf{j}}_{\mathbf{C}} + (S_z)\,\hat{\mathbf{k}}_{\mathbf{C}}$

$H = -B_0 S_z \gamma$

$$H = -\frac{\hbar B_0 \gamma \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}}{2}$$

'H='

$$\begin{bmatrix} -\frac{\hbar B_0 \gamma}{2} & 0 \\ 0 & \frac{\hbar B_0 \gamma}{2} \end{bmatrix}$$

'Eigenvectors='

$$\left[\left(-\frac{\hbar B_0 \gamma}{2},\ 1,\ \left[\begin{bmatrix} 1 \\ 0 \end{bmatrix}\right]\right),\ \left(\frac{\hbar B_0 \gamma}{2},\ 1,\ \left[\begin{bmatrix} 0 \\ 1 \end{bmatrix}\right]\right)\right]$$

'Energy spectrum='

$$\left[-\frac{\hbar B_0 \gamma}{2},\ \frac{\hbar B_0 \gamma}{2}\right]$$

'Xi'

$$ae^{\frac{iB_0\gamma t}{2}} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + be^{-\frac{iB_0\gamma t}{2}} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

'Xi'

$$\begin{bmatrix} ae^{\frac{iB_0\gamma t}{2}} \\ be^{-\frac{iB_0\gamma t}{2}} \end{bmatrix}$$

'Xi'

$$\begin{bmatrix} e^{\frac{iB_0\gamma t}{2}}\cos\left(\frac{\alpha}{2}\right) \\ e^{-\frac{iB_0\gamma t}{2}}\sin\left(\frac{\alpha}{2}\right) \end{bmatrix}$$

'<Sx>=<Xi|Sx|Xi>'

$$\left[\frac{\hbar e^{iB_0\gamma t}\sin\left(\frac{\overline{\alpha}}{2}\right)\cos\left(\frac{\alpha}{2}\right)}{2} + \frac{\hbar e^{-iB_0\gamma t}\sin\left(\frac{\alpha}{2}\right)\cos\left(\frac{\overline{\alpha}}{2}\right)}{2}\right]$$

'<Sx>'

$$\left[\frac{\hbar\left(e^{2iB_0\gamma t}\sin\left(\frac{\overline{\alpha}}{2}\right)\cos\left(\frac{\alpha}{2}\right)+\sin\left(\frac{\alpha}{2}\right)\cos\left(\frac{\overline{\alpha}}{2}\right)\right)e^{-iB_0\gamma t}}{2}\right]$$

'<Sy>'

$$\left[\frac{\hbar i\left(e^{2iB_0\gamma t}\sin\left(\frac{\overline{\alpha}}{2}\right)\cos\left(\frac{\alpha}{2}\right)-\sin\left(\frac{\alpha}{2}\right)\cos\left(\frac{\overline{\alpha}}{2}\right)\right)e^{-iB_0\gamma t}}{2}\right]$$

'<Sz>'

$$\left[\frac{\hbar\cos\left(\frac{\alpha}{2}+\frac{\overline{\alpha}}{2}\right)}{2}\right]$$

0.4.30 4.4.3 Addition of Angular Momenta

0.4.31 —-> p4.49

[37]:
```
#----> p4.49 Electron spin
if "p4.49" in sets.flow:
    A = Symbol('A', real=True)
    X = A*Matrix([1-2*I,2])
    prod = Dagger(X)*X
    solA = solve(Eq(prod[0], 1), A)[1]
    normX = X.subs({A:solA})

    pprints("p4.49",
            "Electron spin",
            "X=", X,

            "a)",
            "Xdagger*X", prod,
            "A=", solA,
            "normX=", normX,
            output_style = "display")

    X = normX
    coeffs = Matrix([a,b])

    (eigvecSx, eigvalSx) = represent(Jx).diagonalize(normalize=True)
    (eigvecSy, eigvalSy) = represent(Jy).diagonalize(normalize=True)
    (eigvecSz, eigvalSz) = represent(Jz).diagonalize(normalize=True)

    probampX = eigvalSx.inv()*eigvecSx.inv()*represent(Jx)*X
```

```
    probampY = eigvalSy.inv()*eigvecSy.inv()*represent(Jy)*X
    probampZ = eigvalSz.inv()*eigvecSz.inv()*represent(Jz)*X

    (probSx, probSy, probSz)  = (Matrix([Abs(probampX[0])**2,
                                         Abs(probampX[1])**2]),
                                 Matrix([Abs(probampY[0])**2,
                                         Abs(probampY[1])**2]),
                                 Matrix([Abs(probampZ[0])**2,
                                         Abs(probampZ[1])**2]))

    pprints("b)",
            "Eigenvalues -> Probability Amplitudes",
            "{0}->{1}={2}".format(eigvalSz, coeffs, probampZ),
            "Sz Eigenvalues", eigvalSz,
            "Sz Probability Amplitudes", coeffs, probampZ,
            "Sz Probabilities [|a|^2 |b|^2]", probSz,
            "<Sz>=<Xdagger Sz X>=", simplify(Dagger(X)*represent(Jz)*X),

            "c)",
            "Sx Eigenvalues", eigvalSx,
            "Sx Probability Amplitudes", coeffs, probampX,
            "Sx Probabilities [|a|^2 |b|^2]", probSx,
            "<Sx>=<Xdagger Sx X>=", simplify(Dagger(X)*represent(Jx)*X),

            "d)",
            "Sy Eigenvalues", eigvalSy,
            "Sy Probability Amplitudes", coeffs, probampY,
            "Sy Probabilities [|a|^2 |b|^2]", probSy,
            "<Sy>=<Xdagger Sy X>=", simplify(Dagger(X)*represent(Jy)*X),
            output_style = "display")
```

'p4.49'

'Electron spin'

'X='

$$\begin{bmatrix} A\,(1-2i) \\ 2A \end{bmatrix}$$

'a)'

'Xdagger*X'

$$\left[4A^2 + A^2 \cdot (1-2i)\,(1+2i)\right]$$

'A='

$$\frac{1}{3}$$

'normX='

$$\begin{bmatrix} \frac{1}{3} - \frac{2i}{3} \\ \frac{2}{3} \end{bmatrix}$$

'b)'

'Eigenvalues -> Probability Amplitudes'

'Matrix([[-hbar/2, 0], [0, hbar/2]])->Matrix([[a], [b]])=Matrix([[2/3], [1/3 - 2*I/3]])'

'Sz Eigenvalues'

$$\begin{bmatrix} -\frac{\hbar}{2} & 0 \\ 0 & \frac{\hbar}{2} \end{bmatrix}$$

'Sz Probability Amplitudes'

$$\begin{bmatrix} a \\ b \end{bmatrix}$$

$$\begin{bmatrix} \frac{2}{3} \\ \frac{1}{3} - \frac{2i}{3} \end{bmatrix}$$

'Sz Probabilities [|a|^2 |b|^2]'

$$\begin{bmatrix} \frac{4}{9} \\ \frac{5}{9} \end{bmatrix}$$

'<Sz>=<Xdagger Sz X>='

$$\begin{bmatrix} \frac{\hbar}{18} \end{bmatrix}$$

'c)'

'Sx Eigenvalues'

$$\begin{bmatrix} -\frac{\hbar}{2} & 0 \\ 0 & \frac{\hbar}{2} \end{bmatrix}$$

'Sx Probability Amplitudes'

$$\begin{bmatrix} a \\ b \end{bmatrix}$$

$$\begin{bmatrix} \frac{\sqrt{2}}{3} - \frac{\sqrt{2} \cdot \left(\frac{1}{3} - \frac{2i}{3}\right)}{2} \\ \frac{\sqrt{2}}{3} + \frac{\sqrt{2} \cdot \left(\frac{1}{3} - \frac{2i}{3}\right)}{2} \end{bmatrix}$$

'Sx Probabilities [|a|^2 |b|^2]'

$$\begin{bmatrix} \frac{5}{18} \\ \frac{13}{18} \end{bmatrix}$$

'<Sx>=<Xdagger Sx X>='

$$\begin{bmatrix} \frac{2\hbar}{9} \end{bmatrix}$$

'd)'

'Sy Eigenvalues'

$$\begin{bmatrix} -\frac{\hbar}{2} & 0 \\ 0 & \frac{\hbar}{2} \end{bmatrix}$$

'Sy Probability Amplitudes'

$$\begin{bmatrix} a \\ b \end{bmatrix}$$

$$\begin{bmatrix} \frac{\sqrt{2}}{3} - \frac{\sqrt{2}i\left(\frac{1}{3} - \frac{2i}{3}\right)}{2} \\ \frac{\sqrt{2}}{3} + \frac{\sqrt{2}i\left(\frac{1}{3} - \frac{2i}{3}\right)}{2} \end{bmatrix}$$

'Sy Probabilities [|a|^2 |b|^2]'

$$\begin{bmatrix} \frac{1}{18} \\ \frac{17}{18} \end{bmatrix}$$

'<Sy>=<Xdagger Sy X>='

$$\left[\frac{4\hbar}{9}\right]$$

0.5 Chapter 5 Identical Particles

0.5.1 5.1 Two-Particle Systems

0.5.2 5.1.1 Bosons and Fermions

0.5.3 5.1.2 Exchange Forces

0.5.4 5.2 Atoms

0.5.5 5.2.1 Helium

0.5.6 5.2.2 The Periodic Table

0.5.7 5.3 Solids

0.5.8 5.3.1 The Free Electron Gas

0.5.9 5.3.2 Band Structure

0.5.10 5.4 Quantum Statistical Mechanics

0.5.11 5.4.1 An Example

0.5.12 5.4.2 The General Case

0.5.13 5.4.3 The Most Probable Configuration

0.5.14 5.4.4 Physical Significance of α and β

0.5.15 5.4.5 The Blackbody Spectrum

0.6 Chapter 6 Time-Independent Perturbation Theory

0.6.1 6.1 Nondegenerate Perturbation Theory

0.6.2 6.1.1 General Formulation

0.6.3 6.1.2 First-Order Theory

0.6.4 —-> p6.2

[ ]:
```
#----> 6.2
if "p6.2" in sets.flow:

    if sets.use_libphysics:
        print("p6.2 Deformed Harmonic Oscillator")
        oqmec.__init__("position_space")
        oqmec.verbose = True
        A,m,w,k,eps = symbols('A m w k epsilon', real=True)
        psi0c = oqmec.qho.nb
        psi0  = oqmec.qho.nk
        En0   = oqmec.qho.En
        Hp = S(1)/2*k*oqmec.qho.xop.rhs**2*eps
        En1 = oqmec.En_ND_PT(1, psi0c, psi0, Hp, En0)
        En2 = oqmec.En_ND_PT(2, psi0c, psi0, Hp, En0, k2min=n-2, k2max=n+2)
        En3full = oqmec.En_ND_PT(3, psi0c, psi0, Hp, En0)
```

```
        En3 = oqmec.En_ND_PT(3, psi0c, psi0, Hp, En0, k2min=n-3, k2max=n+3)

        pprints("Hp", Hp,
                "En1=", En1, simplify(En1),
                "En2=", En2, simplify(En2),
                "En3=", En3, simplify(En3full),
                "En3=", En3, simplify(En3),
                output_style = "display")

    else:
        def npsi(n):
            """
            Normalized wavefunction for harmonic oscillator
            """
            ksi = sqrt(m*w/hbar)*x
            res = Wavefunction((m*w/(pi*hbar))**(S(1)/4)*(1/
↪sqrt((2**n)*factorial(n)))*hermite(n, ksi)*exp(-ksi**2/2), (x,-oo, oo))
            return res

        n = 3
        psi = npsi(n)
        T = libquantum.expT(psi)
        V = (S(1)/2*m*w**2)*libquantum.expX2(psi)
        H0 = T + V
        H = H0 + S(1)/2*epsilon*m*w**2*libquantum.expX2(psi)
        pertH = H - H0

        E0n = simplify(libquantum.expT(npsi(0))+(S(1)/2*m*w**2)*libquantum.
↪expX2(npsi(0)))
        E1n = integrate(conjugate(psi.expr)*pertH*psi.expr,(x,-oo,oo))
        En = simplify(E0n + E1n)

        # Perturbation results
        pprints("H0=", simplify(H0),
                "E0n=", E0n,
                "E1n=", E1n,
                "En=", En,
                output_style = "display")
```

0.6.5 6.1.3 Second-Order Energies

0.6.6 6.2 Degenerate Pertubation Theory

0.6.7 6.2.1 Two-Fold Degeneracy

0.6.8 6.2.2 Higher-Order Degeneracy

0.6.9 6.3 The Fine Structure of Hydrogen

0.6.10 6.3.1 The Relativistic Correction

0.6.11 —-> ch6.3.1

[21]:
```
#----> ch6.3.1
if "ch6.3.1" in sets.flow:
    print("exp_fxSph")
    psi100 = Psi_nlm(1, 0, 0, r, phi, theta, Z=1/a)
    oqmec.Psi = psi100

    pprints(
        "psi_100=", psi100,
        "<r>=",   oqmec.exp_fxSph(r), oqmec.exp_fxSph(r).doit(),
        "<1/r>=", oqmec.exp_fxSph(1/r), oqmec.exp_fxSph(1/r).doit(),
        output_style="display")
```

```
exp_fxSph
```

```
'psi_100='
```

$$\frac{e^{-\frac{r}{a}}}{\sqrt{\pi}a^{\frac{3}{2}}}$$

```
'<r>='
```

$$\langle r\rangle = \int_0^{\pi}\int_0^{2\pi}\int_0^{\infty} \frac{r^3 e^{-\frac{r}{a}} e^{-\frac{r}{a}} \sin(\theta)}{\pi a^3}\, dr\, d\phi\, d\theta$$

$$\langle r\rangle = \frac{3a}{2}$$

```
'<1/r>='
```

$$\langle 1/r\rangle = \int_0^{\pi}\int_0^{2\pi}\int_0^{\infty} \frac{r e^{-\frac{r}{a}} e^{-\frac{r}{a}} \sin(\theta)}{\pi a^3}\, dr\, d\phi\, d\theta$$

$$\langle 1/r\rangle = \frac{1}{a}$$

0.6.12 6.3.2 Spin-Orbit Coupling

0.6.13 6.4 The Zeeman Effect

0.6.14 6.4.1 Weak-Field Zeeman Effect

0.6.15 6.4.2 Strong-Field Zeeman Effect

0.6.16 6.4.3 Intermediate-Field Zeeman Effect

0.6.17 6.5 Hyperfine Splitting

0.7 Chapter 7 The Variational Principle

0.7.1 7.1 Theory

0.7.2 —-> e7.1

[35]:
```
#----> 7.1
if "e7.1" in sets.flow:
    # todo write a variationH function
    print("Griffiths2005 e7.1")
    oqmec.__init__("position_space")
    oqmec.verbose = True
    varfx = Wavefunction(A*exp(-b*x**2), x)
    nvarfx = varfx.normalize().simplify()
    Vx = S(1)/2*m*w**2*x**2
    xreplaces = {xmin:-oo, xmax:oo, Psi:nvarfx.expr, V:Vx}
    expH = oqmec.exp_H.xreplace(xreplaces)
    expHs = expH.doit()
    solb = solve(diff(expHs.rhs, b), b)[1]
    expHmin = expHs.subs({b:solb})

    pprints(
        "V(x)=", Vx,
        "<H>=", expH,
        "<H>=", expH.doit(),
        "b=", solb,
        "<H>min=", expHmin,
        output_style="display")
```

```
Griffiths2005 e7.1
```

```
'V(x)='
```

$$\frac{mw^2x^2}{2}$$

```
'<H>='
```

$$\langle H \rangle = \int_{-\infty}^{\infty} \frac{\sqrt[4]{2}\sqrt[4]{b}\left(\frac{\sqrt[4]{2}\sqrt[4]{b}mw^2x^2e^{-bx^2}}{2\sqrt[4]{\pi}} - \frac{\hbar^2\frac{\partial^2}{\partial x^2}\frac{\sqrt[4]{2}\sqrt[4]{b}e^{-bx^2}}{\sqrt[4]{\pi}}}{2m}\right)e^{-bx^2}}{\sqrt[4]{\pi}}\,dx$$

'<H>='

$$\langle H \rangle = \frac{\hbar^2 b}{2m} + \frac{mw^2}{8b}$$

'b='

$$\frac{mw}{2\hbar}$$

'<H>min='

$$\langle H \rangle = \frac{\hbar w}{2}$$

0.7.3 —-> e7.2

[36]:
```
#----> 7.2 todo write a variation function
if "e7.2" in sets.flow:
    print("Griffiths2005 e7.2")
    oqmec.__init__("position_space")
    oqmec.verbose = True
    varfx = Wavefunction(A*exp(-b*x**2), x)
    nvarfx = varfx.normalize().simplify()
    Vx = -alpha*DiracDelta(x)
    xreplaces = {xmin:-oo, xmax:oo, Psi:nvarfx.expr, V:Vx}
    expH = oqmec.exp_H.xreplace(xreplaces)

    pprints(
        "V(x)=", Vx,
        "<H>=", expH,
        "<H>=", expH.doit(),
        output_style="display")
```

Griffiths2005 e7.2

'V(x)='

$$-\alpha\delta\left(x\right)$$

'<H>='

$$\langle H \rangle = \int_{-\infty}^{\infty} \frac{\sqrt[4]{2}\sqrt[4]{b}\left(-\frac{\sqrt[4]{2}\alpha\sqrt[4]{b}e^{-bx^2}\delta(x)}{\sqrt[4]{\pi}} - \frac{\hbar^2\frac{\partial^2}{\partial x^2}\frac{\sqrt[4]{2}\sqrt[4]{b}e^{-bx^2}}{\sqrt[4]{\pi}}}{2m}\right)e^{-bx^2}}{\sqrt[4]{\pi}}\,dx$$

'<H>='

$$\langle H \rangle = \frac{\hbar^2 b}{4m} + \frac{-4\sqrt{2}\alpha\sqrt{b}m + \hbar^2\sqrt{\pi}b}{4\sqrt{\pi}m}$$

[ ]: