Computational Physics: Homework 7: Approximation, Interpolation, and Least Squares

In [1]:
```
import scipy
import numpy as np
import matplotlib.pyplot as plt
from math import log, exp
from timeit import default_timer as timer
from scipy.optimize import curve_fit
from random import uniform, gauss
from scipy.constants import h, c, k
from scipy.interpolate import interp1d
from scipy.interpolate import CubicSpline
```

Problem 1:

In [2]:

```
def linear_interpolation(x_values, y_values, x):
    return y_values[0] + (y_values[1] - y_values[0]) * (x - x_values[0])

def f_bilinear(x, y):
    return np.cos(x) * np.sin(2 * y)

def bilinear_interpolation(x0, y0, x1, y1, x, y):
    f00 = f_bilinear(x0, y0)
    f01 = f_bilinear(x0, y1)
    f10 = f_bilinear(x1, y0)
    f11 = f_bilinear(x1, y1)

    return (
        f00 * (x1 - x) * (y1 - y) +
        f10 * (x - x0) * (y1 - y) +
        f01 * (x1 - x) * (y - y0) +
        f11 * (x - x0) * (y - y0)
    ) / ((x1 - x0) * (y1 - y0))

def f_trilinear(x, y, z):
    return np.sin(x) * np.sin(2 * y) * np.cos(5 * z)

def trilinear_interpolation(x0, y0, z0, x1, y1, z1, x, y, z):
    f000 = f_trilinear(x0, y0, z0)
    f001 = f_trilinear(x0, y0, z1)
    f010 = f_trilinear(x0, y1, z0)
    f011 = f_trilinear(x0, y1, z1)
    f100 = f_trilinear(x1, y0, z0)
    f101 = f_trilinear(x1, y0, z1)
    f110 = f_trilinear(x1, y1, z0)
    f111 = f_trilinear(x1, y1, z1)

    return (
        f000 * (x1 - x) * (y1 - y) * (z1 - z) +
        f100 * (x - x0) * (y1 - y) * (z1 - z) +
        f010 * (x1 - x) * (y - y0) * (z1 - z) +
        f110 * (x - x0) * (y - y0) * (z1 - z) +
        f001 * (x1 - x) * (y1 - y) * (z - z0) +
        f101 * (x - x0) * (y1 - y) * (z - z0) +
        f011 * (x1 - x) * (y - y0) * (z - z0) +
        f111 * (x - x0) * (y - y0) * (z - z0)
    ) / ((x1 - x0) * (y1 - y0) * (z1 - z0))

x_nodes = [0, 1]
y_nodes = [np.sin(x) for x in x_nodes]
x_interpolate = 2/3

print("Interpolation Bois:")
# part a: linear
result_a = linear_interpolation(x_nodes, y_nodes, x_interpolate)
print("Result from linear interpolation is = ",result_a)

# part b: bilinear
x0_bilinear, y0_bilinear = 0, 0
x1_bilinear, y1_bilinear = 1, 1
x_interp_bilinear, y_interp_bilinear = 0.5, 0.5
result_b = bilinear_interpolation(x0_bilinear, y0_bilinear, x1_bilinear,
```

```
print("Result from the bilinear interpolation is = ",result_b)

# part c: trilinear
x0_trilinear, y0_trilinear, z0_trilinear = 0, 0, 0
x1_trilinear, y1_trilinear, z1_trilinear = 1, 1, 1
x_interp_trilinear, y_interp_trilinear, z_interp_trilinear = 1/3, 1/2, 3
result_c = trilinear_interpolation(x0_trilinear, y0_trilinear, z0_trilin
print("Result from the trilinear interpolation is = ", result_c)
```

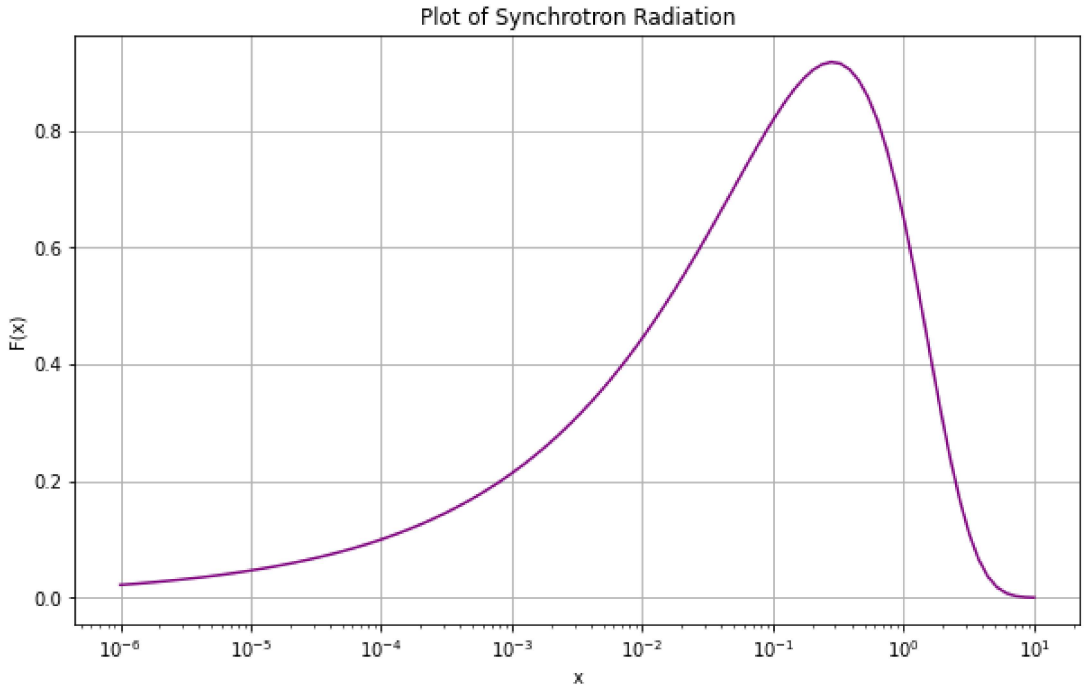
```
Interpolation Bois:
Result from linear interpolation is =  0.5609806565385976
Result from the bilinear interpolation is =  0.3501482308148909
Result from the trilinear interpolation is =  0.059011564722548954
```

Problem 2:

In [3]:

```
# fun functions
def synch_int_high(x, a):
    xeval = a + x / (1.0 - x)
    nu = 5.0/3.0
    I = scipy.special.kv(nu, xeval) / (1.0 - x)**2
    return I

def synch_int_low(x):
    xeval = exp(x)
    nu = 5.0/3.0
    I = scipy.special.kv(nu, xeval) * xeval
    return I

def inf_int(f, a, N):
    if N % 2 == 0:
        raise Exception("Simpson's rule requires an odd number of points
    elif N < 5:
        raise Exception("Simpson's rule requires at least five points")

    xs = np.linspace(0, 1, N)
    h = 1.0 / (N-1)

    fs = [2*(2-i%2)*f(x, a) for i, x in enumerate(xs[1:-1])]
    total = np.sum(fs) + f(xs[0], a) + 0.0
    fint = total * h / 3.0
    return fint

def simpsons_rule(f, points, h):
    n = len(points)
    if n % 2 == 0:
        raise Exception("Simpson's rule requires an odd number of points
    elif n < 5:
        raise Exception("Simpson's rule requires at least five points")

    fs = [2*(2-i%2)*f(x) for i, x in enumerate(points[1:-1])]
    total = np.sum(fs) + f(points[0]) + f(points[-1])
    fint = total * h / 3.0
    return fint

def synch_F_int(x):
    if x <= 0.0:
        raise Exception("Value passed must be positive")
    elif x > exp(-1.0):
        result = inf_int(synch_int_high, x, 501)
    else:
        xs = np.linspace(log(x), -1.0, 501)
        h = xs[2] - xs[1]
        result_lo = simpsons_rule(synch_int_low, xs, h)
        result_high = inf_int(synch_int_high, exp(-1.0), 501)
        result = result_lo + result_high
    return x * result

#part a
x_vals = np.logspace(-6, 1, 100)
F_values = [synch_F_int(x) for x in x_vals]

plt.figure(figsize=(10, 6))
```

```
plt.plot(x_vals, F_values, label="F(x)", color = 'purple')
plt.xscale('log')
plt.xlabel("x")
plt.ylabel("F(x)")
plt.title("Plot of Synchrotron Radiation")
plt.grid()
```



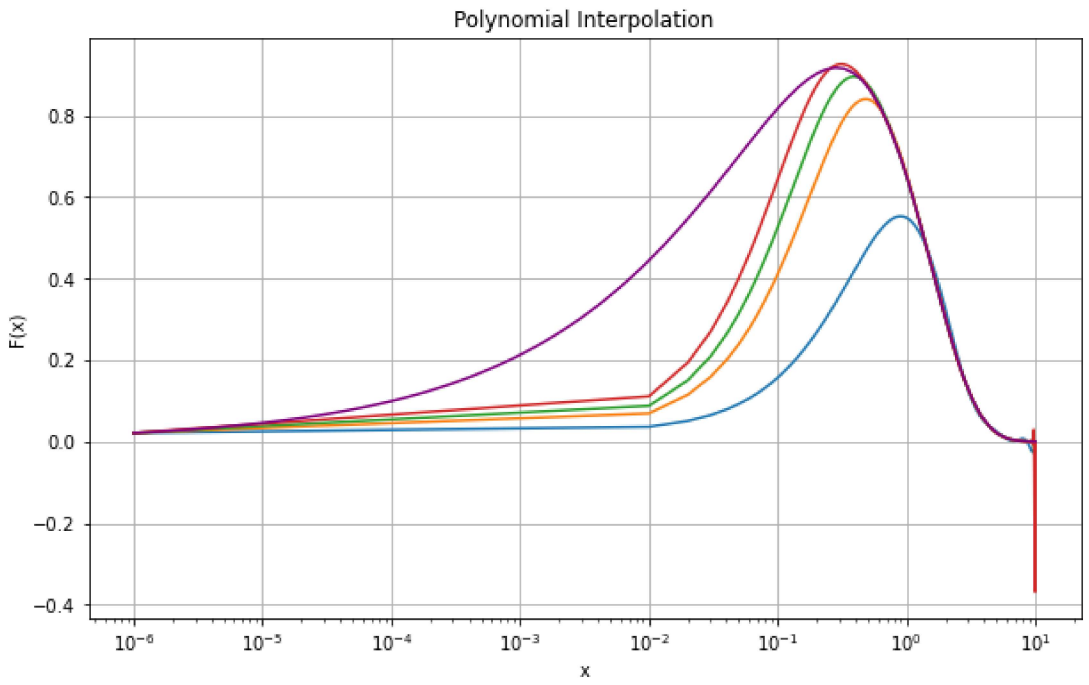
In [4]:

```
# part b
t_direct_start = timer()
direct_result = synch_F_int(1.0)
t_direct_end = timer()
time_direct = t_direct_end - t_direct_start
print("direct result = ", direct_result)
print("average evalution time = ", time_direct)
```

```
direct result =  0.6514228153556608
average evalution time =  0.0034218999999993116
```

In [5]:
```
# part c
N_vals = [8, 16, 24, 40]
x_nodes_eq = np.linspace(1e-6, 10, 1000)

plt.figure(figsize=(10, 6))
for N in N_vals:
    x_nodes = np.linspace(1e-6, 10, N)
    y_nodes = [synch_F_int(x) for x in x_nodes]
    poly = np.polyfit(x_nodes, y_nodes, N-1)
    poly_values = np.polyval(poly, x_nodes_eq)
    plt.plot(x_nodes_eq, poly_values, label=f"N={N}")

plt.plot(x_vals, F_values, label="F(x)", color = 'purple')
plt.xscale('log')
plt.xlabel("x")
plt.ylabel("F(x)")
plt.title("Polynomial Interpolation")
plt.grid()
```

```
C:\Users\vince\anaconda3\lib\site-packages\IPython\core\interactiveshel
l.py:3369: RankWarning: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
C:\Users\vince\anaconda3\lib\site-packages\IPython\core\interactiveshel
l.py:3369: RankWarning: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
```



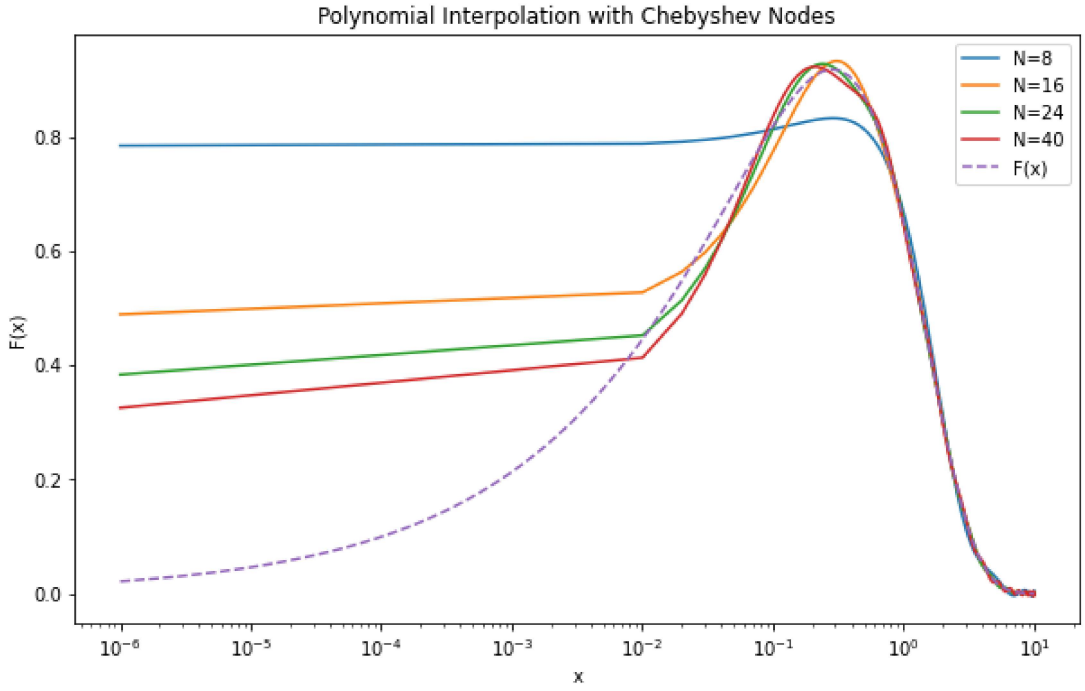
In [6]:
```
# part d
poly_16 = np.polyfit(x_nodes, y_nodes, 16)

t_poly_start = timer()
poly_result = np.polyval(poly_16, 1.0)
t_poly_end = timer()
time_poly = t_poly_end - t_poly_start
print("16th-degree = ", poly_result)
print("time to evaluate = ", time_poly)
```

```
16th-degree =  0.6505018852331318
time to evaluate =  0.00014459999999960615
```

In [8]:

```
# part e
def chebyshev_nodes(a, b, N):
    k = np.arange(1, N + 1)
    nodes = 0.5 * (a + b) + 0.5 * (b - a) * np.cos((2 * k - 1) * np.pi /
    return nodes

plt.figure(figsize=(10, 6))
for N in N_vals:
    x_nodes_cheb = chebyshev_nodes(1e-6, 10, N)
    y_nodes_cheb = [synch_F_int(x) for x in x_nodes_cheb]
    poly_cheb = np.polyfit(x_nodes_cheb, y_nodes_cheb, N-1)
    poly_values_cheb = np.polyval(poly_cheb, x_nodes_eq)
    plt.plot(x_nodes_eq, poly_values_cheb, label=f"N={N}")

plt.plot(x_vals, F_values, label="F(x)", linestyle='dashed')
plt.xscale('log')
plt.xlabel("x")
plt.ylabel("F(x)")
plt.title("Polynomial Interpolation with Chebyshev Nodes")
plt.legend()
plt.show()
```

```
C:\Users\vince\anaconda3\lib\site-packages\IPython\core\interactiveshel
l.py:3369: RankWarning: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
C:\Users\vince\anaconda3\lib\site-packages\IPython\core\interactiveshel
l.py:3369: RankWarning: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
```



In [11]:
```
# part f
max_errors = []

for N in N_vals:
    x_nodes_cheb = chebyshev_nodes(1e-6, 10, N)
    y_nodes_cheb = [synch_F_int(x) for x in x_nodes_cheb]
    poly_cheb = np.polyfit(x_nodes_cheb, y_nodes_cheb, N-1)
    poly_values_cheb = np.polyval(poly_cheb, x_nodes_cheb)
    relative_errors = np.abs(poly_values_cheb - y_nodes_cheb) / np.abs(y
    max_errors.append(np.max(relative_errors))

print("maximum error = ", max(max_errors))
```

```
maximum error =  17.83384677053889

C:\Users\vince\anaconda3\lib\site-packages\IPython\core\interactiveshel
l.py:3369: RankWarning: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
C:\Users\vince\anaconda3\lib\site-packages\IPython\core\interactiveshel
l.py:3369: RankWarning: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Problem 3:

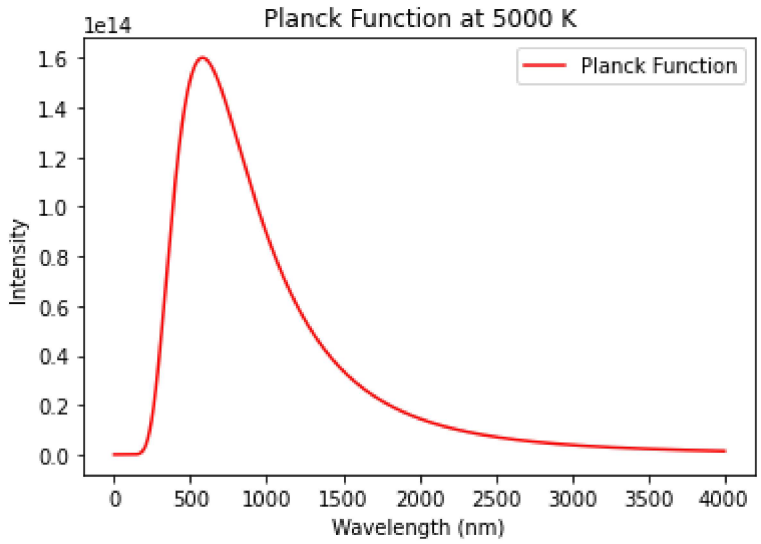
In [13]:
```
# part a
h = 6.626e-34  # Planck constant in J*s
c = 3.0e8      # Speed of light in m/s
k = 1.38e-23   # Boltzmann constant in J/K
T = 5000       # Temperature in K

# in nm
wavelengths = [0, 400e-9, 800e-9, 1200e-9, 2000e-9, 4000e-9]
B_lambda_values = []

for wavelength in wavelengths:
    # Avoid division by zero
    if wavelength == 0:
        B_lambda_values.append(np.inf)
    else:
        B_lambda = (2 * h * c**2) / (wavelength**5) / (np.exp((h * c) /
        B_lambda_values.append(B_lambda)

# Print results
for wavelength, B_lambda in zip(wavelengths, B_lambda_values):
    print(f"B_lambda({wavelength}) = {B_lambda:.2e}")
    # inf at 0 bc function does not as wavelength gets smaller
```

```
B_lambda(0) = inf
B_lambda(4e-07) = 8.68e+12
B_lambda(8e-07) = 1.02e+13
B_lambda(1.2e-06) = 4.78e+12
B_lambda(2e-06) = 1.16e+12
B_lambda(4e-06) = 1.10e+11
```

In [14]:

```
# part b
def planck(wavelength, temperature = 5000):
    wavelength_m = wavelength * 1e-9  # Convert wavelength to meters
    return (8 * np.pi * h * c ** 2) / (wavelength_m ** 5) * 1 / (np.exp(

# Wavelength range (in nanometers)
wavelength_range = np.linspace(1, 4000, 4000)

# Temperature in Kelvin
temperature = 5000

# Compute Planck function values
planck_values = planck(wavelength_range, temperature)

# Plot the Planck function
plt.plot(wavelength_range, planck_values, label='Planck Function', color
plt.xlabel('Wavelength (nm)')
plt.ylabel('Intensity')
plt.title(f'Planck Function at {temperature} K')
plt.legend()
plt.show()
```

```
C:\Users\vince\AppData\Local\Temp\ipykernel_31596\1654836780.py:4: Runt
imeWarning: overflow encountered in exp
  return (8 * np.pi * h * c ** 2) / (wavelength_m ** 5) * 1 / (np.exp(h
* c / (wavelength_m * k * temperature)) - 1)
```

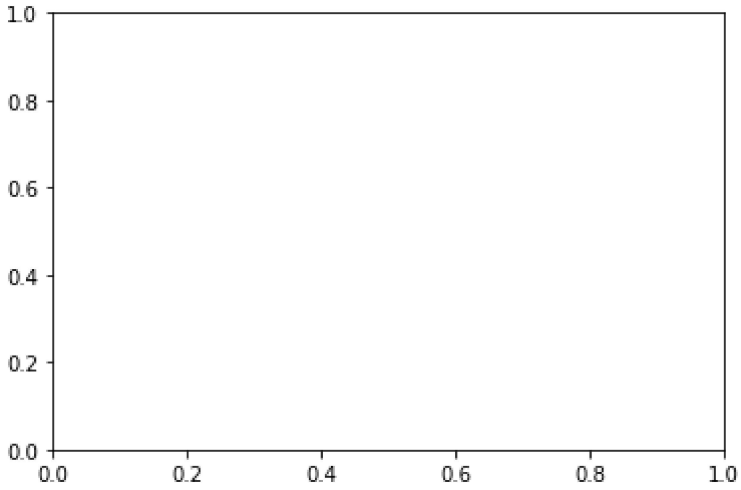


In [15]:
```
# part c
B_vals = np.isfinite(planck_values)
lambda_vals = wavelength_range[B_vals]
valid_B_vals = planck_values[B_vals]
spliney = CubicSpline(lambda_vals, valid_B_vals)\

dlambda = 2
err_lambda = np.arange(0, 4000 + dlambda, dlambda)

planck_vals_2 = planck(err_lambda * 1e-9)
index = np.isfinite(planck_vals_2) #& np.isfinite(CubicSpline(err_lambda

abs_err = CubicSpline((err_lambda[index]) , planck_vals_2[index])

plt.plot(err_lambda[index], abs_err, label="Cubic Spline Error")
plt.xlabel("Wavelength (nm)")
plt.ylabel("Error")
plt.legend()
plt.title("Error between Cubic Spline Fit and Planck Function")
plt.show()
# idk ):
```

```
C:\Users\vince\AppData\Local\Temp\ipykernel_31596\1654836780.py:4: Runt
imeWarning: divide by zero encountered in true_divide
  return (8 * np.pi * h * c ** 2) / (wavelength_m ** 5) * 1 / (np.exp(h
* c / (wavelength_m * k * temperature)) - 1)
C:\Users\vince\AppData\Local\Temp\ipykernel_31596\1654836780.py:4: Runt
imeWarning: overflow encountered in exp
  return (8 * np.pi * h * c ** 2) / (wavelength_m ** 5) * 1 / (np.exp(h
* c / (wavelength_m * k * temperature)) - 1)
C:\Users\vince\AppData\Local\Temp\ipykernel_31596\1654836780.py:4: Runt
imeWarning: invalid value encountered in true_divide
  return (8 * np.pi * h * c ** 2) / (wavelength_m ** 5) * 1 / (np.exp(h
* c / (wavelength_m * k * temperature)) - 1)
```

```
---------------------------------------------------------------------
----
ValueError                                Traceback (most recent call l
ast)
Input In [15], in <cell line: 15>()
     11 index = np.isfinite(planck_vals_2) #& np.isfinite(CubicSpline(e
rr_lambda))
     13 abs_err = CubicSpline((err_lambda[index]) , planck_vals_2[inde
x])
---> 15 plt.plot(err_lambda[index], abs_err, label="Cubic Spline Erro
r")
     16 plt.xlabel("Wavelength (nm)")
     17 plt.ylabel("Error")

File ~\anaconda3\lib\site-packages\matplotlib\pyplot.py:2757, in plot(s
calex, scaley, data, *args, **kwargs)
   2755 @_copy_docstring_and_deprecators(Axes.plot)
   2756 def plot(*args, scalex=True, scaley=True, data=None, **kwargs):
-> 2757     return gca().plot(
   2758         *args, scalex=scalex, scaley=scaley,
   2759         **({"data": data} if data is not None else {}), **kwarg
s)

File ~\anaconda3\lib\site-packages\matplotlib\axes\_axes.py:1632, in Ax
es.plot(self, scalex, scaley, data, *args, **kwargs)
   1390 """
   1391 Plot y versus x as lines and/or markers.
   1392
   (...)
   1629 (``'green'``) or hex strings (``'#008000'``).
   1630 """
   1631 kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1632 lines = [*self._get_lines(*args, data=data, **kwargs)]
   1633 for line in lines:
   1634     self.add_line(line)

File ~\anaconda3\lib\site-packages\matplotlib\axes\_base.py:312, in _pr
ocess_plot_var_args.__call__(self, data, *args, **kwargs)
    310     this += args[0],
    311     args = args[1:]
--> 312 yield from self._plot_args(this, kwargs)

File ~\anaconda3\lib\site-packages\matplotlib\axes\_base.py:498, in _pr
ocess_plot_var_args._plot_args(self, tup, kwargs, return_kwargs)
    495     self.axes.yaxis.update_units(y)
    497 if x.shape[0] != y.shape[0]:
--> 498     raise ValueError(f"x and y must have same first dimension,
but "
    499                      f"have shapes {x.shape} and {y.shape}")
    500 if x.ndim > 2 or y.ndim > 2:
    501     raise ValueError(f"x and y can be no greater than 2D, but h
ave "
    502                      f"shapes {x.shape} and {y.shape}")

ValueError: x and y must have same first dimension, but have shapes (20
```

```
00,) and (1,)
```



Problem 4:

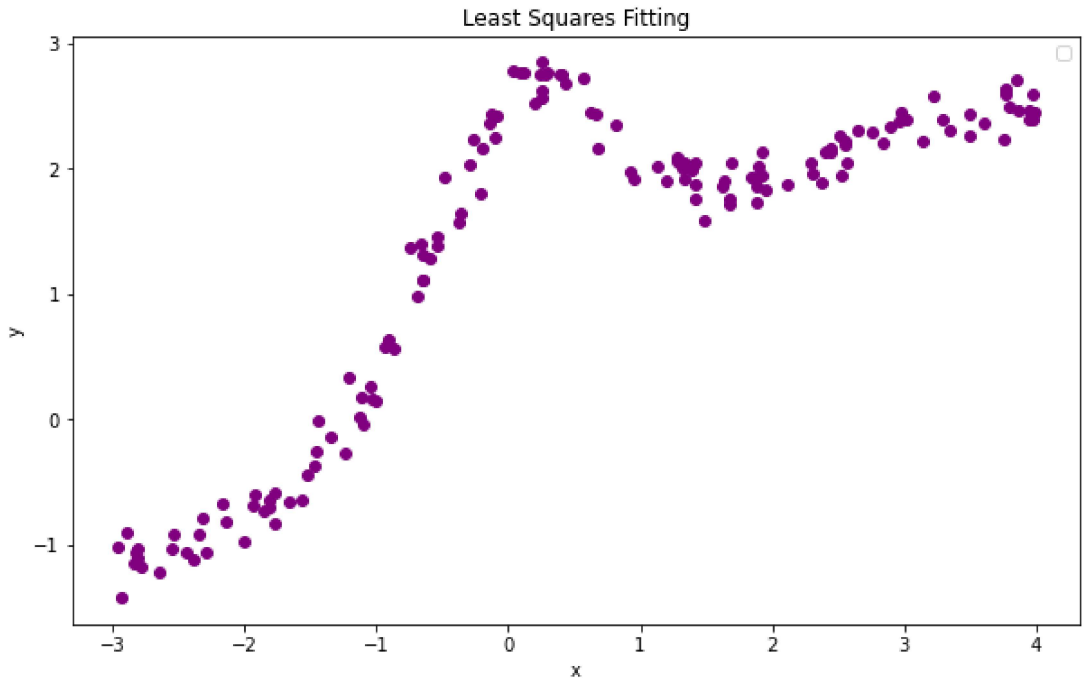
In [16]:

```
def data(a, b, N):
    c0 = 0.5
    c1 = 2.0
    c2 = 1.25
    xs = [a + (uniform(a, b) + (a + b) / 5.0) % (b - a) for i in range(N
    term1s = [c1 * exp(-x**2) for x in xs]
    term2s = [c2 * x / abs(x) * log(abs(x) + 1.0) for x in xs]
    ys = [c0 + term1 + term2 for term1, term2 in zip(term1s, term2s)]
    for i, y in enumerate(ys):
        ys[i] = gauss(y, 0.15)
    return xs, ys

def linear_theory(x, c0, c1):
    return c0 + c1 * np.array(x)

def f_theory(x, c0, c1, c2):
    theory = []
    for i in x:
        t = c0 + c1 * exp(-np.array(i)**2) + c2 * (np.array(i) / abs(np.
        theory.append(t)
    return theory

# part a
a, b, N = -3, 4, 150
xs, ys = data(a, b, N)

plt.figure(figsize=(10, 6))
plt.scatter(xs, ys, color = 'purple')
plt.xlabel("x")
plt.ylabel("y")
plt.title("Least Squares Fitting")
plt.legend()
```

```
No artists with labels found to put in legend.  Note that artists whose
label start with an underscore are ignored when legend() is called with
no argument.
```

Out[16]:

```
<matplotlib.legend.Legend at 0x22758c931c0>
```



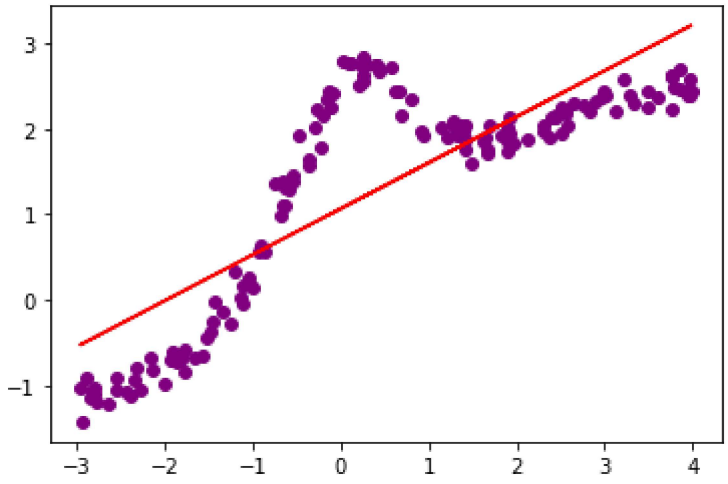
In [17]:

```
# part b
params_linear, covariance_linear = curve_fit(linear_theory, xs, ys)
c0, c1 = params_linear
chi2 = np.sum((ys - linear_theory(xs, c0, c1))**2)

print("c0 = ", c0)
print("c1 = ", c1)
print("chi squared value = ", chi2) # thats not good, right?
```

```
c0 =  1.0658664730493692
c1 =  0.5377558165502421
chi squared value =  81.92896836688638
```

In [18]:
```
# part c
plt.scatter(xs, ys, label="Generated Data", color = 'purple')
plt.plot(xs, linear_theory(xs, c0, c1), label="Linear Fit", color = 'red
# perfect
```

Out[18]:
```
[<matplotlib.lines.Line2D at 0x227590f9700>]
```



In [19]:
```
# part d
A = np.vstack((np.ones_like(xs), xs, np.exp(-np.array(xs)**2),
                    (np.array(xs) / abs(np.array(xs))) * np.log(abs(np.ar
params_general, residual, rank, singular_values = np.linalg.lstsq(A, ys,

c0, c1, c2, c3 = params_general
chi2 = np.sum(residual)

print("c0 = ", c0)
print("c1 = ", c1)
print("c2 = ", c2)
print("c3 = ", c3)
print("chi-squared = ", chi2)
```

```
c0 =  0.5403924844384849
c1 =  -0.022412316016291803
c2 =  1.9722176833263432
c3 =  1.2952748821027993
chi-squared =  2.9922288742183314
```

In [23]:
```
# part e
plt.scatter(xs, ys, label="Generated Data", color = 'purple')
plt.scatter(xs, f_theory(xs, c0, c1, c2), label="weirdo dots", color = '
plt.xlabel("x")
plt.ylabel("y")
plt.title("Least Squares Fit ")
plt.legend()
# i am sorry idk what is happening
```

Out[23]:
```
<matplotlib.legend.Legend at 0x22759342f70>
```