Ex

Computer Exercise 1.1

- 2
- 10
- 12

Computer Exercise 1.2

- 21
- 52

Computer Exercise 1.3

- 18
- 25
- 46

Computer Exercise 1.4

- 7
- 33
- 34

Com Ex

Computer Exercise 1.1

- 1
- 21

Computer Exercise 1.2

- 12
- 21
- 22

Computer Exercise 1.3

- 7

Computer Exercise 1.4

- 15

In [2]:
```
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
```

In [3]:
```
# 1.1 - 1

def First():
    i_min = 0
    n = 30
    e = 0
    y = 0
    x = 0.5
    h = 1
    emin = 1
    for i in range(1, n + 1):
        h = 0.25 * h
        y = (np.sin(x + h) - np.sin(x)) / h
        e = np.abs(y - np.cos(x))

        # Print the results
        print(f"i = {i}, h = {h}, y = {y}, e = {e}")

        if e < emin:
            emin = e
            i_min = i

    print(f"i_min = {i_min}, emin = {emin}")
    return i_min, emin

First()
```

```
i = 1, h = 0.25, y = 0.8088528856765245, e = 0.06872967621384829
i = 2, h = 0.0625, y = 0.8620341589090739, e = 0.015548402981298892
i = 3, h = 0.015625, y = 0.873801417582083, e = 0.0037811443082897966
i = 4, h = 0.00390625, y = 0.8766439532697916, e = 0.0009386086205811495
i = 5, h = 0.0009765625, y = 0.8773483279197194, e = 0.0002342339706533636
i = 6, h = 0.000244140625, y = 0.8775240295474305, e = 5.853234294228571e-05
i = 7, h = 6.103515625e-05, y = 0.8775679304389996, e = 1.4631451373148252e-
05
i = 8, h = 1.52587890625e-05, y = 0.8775789041283133, e = 3.6577620594613336
e-06
i = 9, h = 3.814697265625e-06, y = 0.8775816474517342, e = 9.144386385884218
e-07
i = 10, h = 9.5367431640625e-07, y = 0.8775823332834989, e = 2.2860687387549
206e-07
i = 11, h = 2.384185791015625e-07, y = 0.8775825046468526, e = 5.72435201462
4401e-08
i = 12, h = 5.960464477539063e-08, y = 0.8775825472548604, e = 1.46355123575
85867e-08
i = 13, h = 1.4901161193847656e-08, y = 0.8775825574994087, e = 4.3909640368
15603e-09
i = 14, h = 3.725290298461914e-09, y = 0.8775825649499893, e = 3.05961656010
8225e-09
i = 15, h = 9.313225746154785e-10, y = 0.8775825500488281, e = 1.18415446337
39431e-08
i = 16, h = 2.3283064365386963e-10, y = 0.8775825500488281, e = 1.1841544633
739431e-08
i = 17, h = 5.820766091346741e-11, y = 0.8775825500488281, e = 1.18415446337
39431e-08
i = 18, h = 1.4551915228366852e-11, y = 0.8775825500488281, e = 1.1841544633
739431e-08
i = 19, h = 3.637978807091713e-12, y = 0.8775787353515625, e = 3.82653881025
87394e-06
i = 20, h = 9.094947017729282e-13, y = 0.8775634765625, e = 1.90853278727587
4e-05
i = 21, h = 2.2737367544323206e-13, y = 0.87744140625, e = 0.000141155640372
75874
i = 22, h = 5.684341886080802e-14, y = 0.8779296875, e = 0.00034712560962724
126
i = 23, h = 1.4210854715202004e-14, y = 0.87890625, e = 0.001323688109627241
3
i = 24, h = 3.552713678800501e-15, y = 0.875, e = 0.0025825618903727587
i = 25, h = 8.881784197001252e-16, y = 0.875, e = 0.0025825618903727587
i = 26, h = 2.220446049250313e-16, y = 0.75, e = 0.12758256189037276
i = 27, h = 5.551115123125783e-17, y = 0.0, e = 0.8775825618903728
i = 28, h = 1.3877787807814457e-17, y = 0.0, e = 0.8775825618903728
i = 29, h = 3.469446951953614e-18, y = 0.0, e = 0.8775825618903728
i = 30, h = 8.673617379884035e-19, y = 0.0, e = 0.8775825618903728
i_min = 14, emin = 3.059616560108225e-09
```

```
Out[3]: (14, np.float64(3.059616560108225e-09))
```

Result

The results of the computer exercises are as follows: minimum error: 3.059616560108225e-09, iteration: 14, h at minimum error: 3.725290298461914e-09 error keep decreasing until iteration 14, then start to increase

Interpretation

The error decreases until the 14th iteration and then starts to increase. In the initial iterations, when h is sufficiently large, the dominant error is the truncation error. As h decreases, the truncation error decreases, leading to a reduction in the total error. However, when h becomes sufficiently small, the dominant error shifts to round-off error. At this point, as h decreases further, the round-off error increases, causing the total error to rise.

In [4]:
```
# 1.1 - 21

# a
def a():
    x = 0.1
    y = 0.01
    z = x - y
    p = 1.0 / 3.0
    q = 3.0 * p
    u = 7.6
    v = 2.9
    w = u - v
    print("Default formatting:")
    print(f"x = {x} y = {y} z = {z} p = {p} q = {q} u = {u} v = {v} w = {w}"

    print("\nExtremely large format field:")
    print(f"x = {x:.50f} y = {y:.50f} z = {z:.50f} p = {p:.50f} q = {q:.50f}

print("Part a:")
a()
print("\n")

# b
def b():
    x = 0
    y = 0
    z = 0
    for i in range(1, 11):
        x = (i - 1) / 2
        y = i * i / 3.0
        z = 1.0 + 1 / i
        print(f"x = {x} y = {y} z = {z}")

print("Part b:")
b()
print("\n")

# c
def c():
    f = 3.14159265358979323846
    x = 10 / 3
    i = int(x + 1 / 2)
    half = 1 / 2
    j = int(half)
    c = (5 / 9) * (f - 32)
    f = 9 / 5 * c + 32

    print(f"x = {x} i = {i} half = {half} j = {j} c = {c} f = {f}")

print("Part c:")
c()
print("\n")

# d
def d():
    radius = 1
    area = (22 / 7) * radius * radius
    circum = 2 * (3.1416) * radius
    print(f"area = {area} circum = {circum}")

print("Part d:")
d()
```

```
Part a:
Default formatting:
x = 0.1 y = 0.01 z = 0.09000000000000001 p = 0.3333333333333333 q = 1.0 u =
7.6 v = 2.9 w = 4.699999999999999

Extremely large format field:
x = 0.10000000000000000555111512312578270211815834045410 y = 0.0100000000000
0000020816681711721685132943093776703 z = 0.09000000000000001054711873393898
713402450084686279 p = 0.33333333333333331482961625624739099293947219848633
q = 1.000000000000000000000000000000000000000000000000 u = 7.5999999999999
9964472863211994990706443786621093750 v = 2.89999999999999991118215802998747
676610946655273438 w = 4.69999999999999928945726423989981412887573242187500


Part b:
x = 0.0 y = 0.3333333333333333 z = 2.0
x = 0.5 y = 1.3333333333333333 z = 1.5
x = 1.0 y = 3.0 z = 1.3333333333333333
x = 1.5 y = 5.333333333333333 z = 1.25
x = 2.0 y = 8.333333333333334 z = 1.2
x = 2.5 y = 12.0 z = 1.1666666666666667
x = 3.0 y = 16.333333333333332 z = 1.1428571428571428
x = 3.5 y = 21.333333333333332 z = 1.125
x = 4.0 y = 27.0 z = 1.1111111111111112
x = 4.5 y = 33.333333333333336 z = 1.1


Part c:
x = 3.3333333333333335 i = 3 half = 0.5 j = 0 c = -16.032448525783447 f = 3.
141592653589793


Part d:
area = 3.142857142857143 circum = 6.2832
```

Problem of d

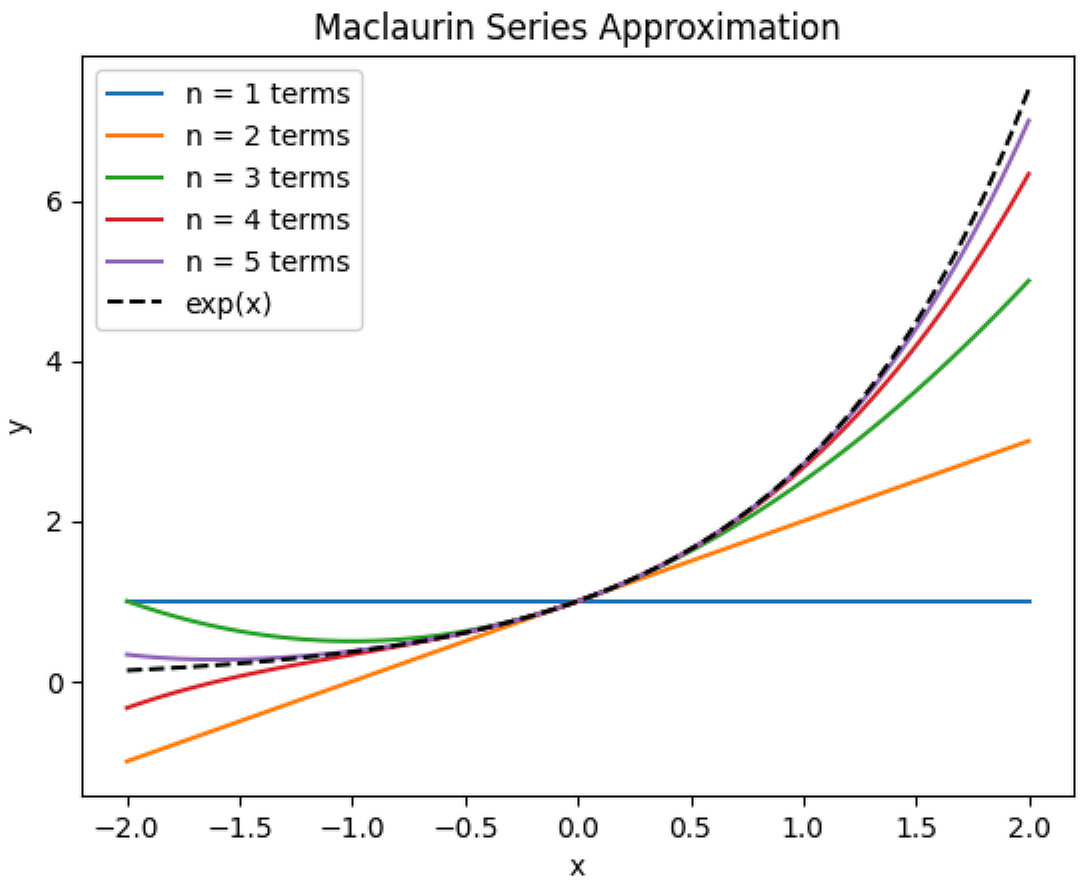
1. both of 22/7 and 3.1416 is not a perfect representation of pi, so the error would be accumalated and become larger as the iteration goes on
2. doesn't use consistent value of pi. in some case use 22/7, and other case use 3.1416. that would make the error become larger.

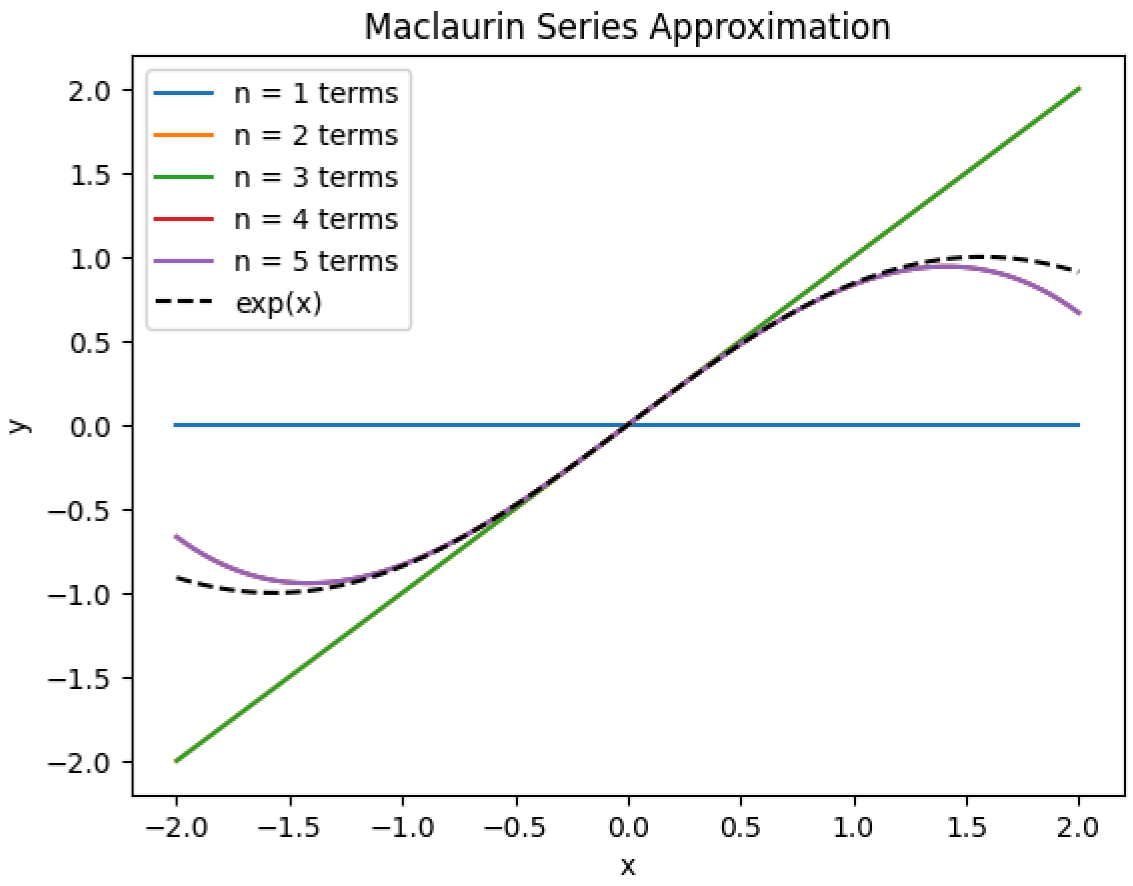
In [5]:
```
# 1.2 - 12
def pi_caculator_with_triangle(n):
    area = 0
    prev_sin = np.sin(np.radians(360 / pow(2, 2)))
    for i in range(3, n+1):
        count = pow(2, i)
        angle = 360 / count
        sin = np.sin(np.radians(angle))
        area = (1 / 2) * np.sin(np.radians(angle))
        derived_sin = prev_sin / np.sqrt(2*(1 + np.sqrt((1 - prev_sin * prev
        abs_error = np.abs(sin - derived_sin)
        rel_error = abs_error / sin
        pi = count * area
        print(f"i = {i}, count = {count}, angle = {angle}, area = {area}, ab
        prev_sin = np.sin(np.radians(angle))

pi_caculator_with_triangle(20)
```

```
i = 3, count = 8, angle = 45.0, area = 0.35355339059327373, abs_error = 0.0,
rel_error = 0.0, pi = 2.82842712474619
i = 4, count = 16, angle = 22.5, area = 0.1913417161825449, abs_error = 5.55
1115123125783e-17, rel_error = 1.450576286728263e-16, pi = 3.061467458920718
3
i = 5, count = 32, angle = 11.25, area = 0.09754516100806412, abs_error = 2.
7755575615628914e-17, rel_error = 1.4227038701250563e-16, pi = 3.12144515225
8052
i = 6, count = 64, angle = 5.625, area = 0.0490085701647803, abs_error = 1.3
877787807814457e-17, rel_error = 1.4158531621258808e-16, pi = 3.136548490545
9393
i = 7, count = 128, angle = 2.8125, area = 0.024533837163709007, abs_error =
0.0, rel_error = 0.0, pi = 3.140331156954753
i = 8, count = 256, angle = 1.40625, area = 0.012270614261456144, abs_error
= 0.0, rel_error = 0.0, pi = 3.141277250932773
i = 9, count = 512, angle = 0.703125, area = 0.006135769142859963, abs_error
= 1.734723475976807e-18, rel_error = 1.4136153394847493e-16, pi = 3.14151380
1144301
i = 10, count = 1024, angle = 0.3515625, area = 0.0030679423245772376, abs_e
rror = 0.0, rel_error = 0.0, pi = 3.1415729403670913
i = 11, count = 2048, angle = 0.17578125, area = 0.001533978381482988, abs_e
rror = 0.0, rel_error = 0.0, pi = 3.1415877252771596
i = 12, count = 4096, angle = 0.087890625, area = 0.0007669900931423828, abs
_error = 0.0, rel_error = 0.0, pi = 3.1415914215111997
i = 13, count = 8192, angle = 0.0439453125, area = 0.00038349515937135224, a
bs_error = 0.0, rel_error = 0.0, pi = 3.1415923455701176
i = 14, count = 16384, angle = 0.02197265625, area = 0.00019174759378569778,
abs_error = 0.0, rel_error = 0.0, pi = 3.1415925765848725
i = 15, count = 32768, angle = 0.010986328125, area = 9.587379865535165e-05,
abs_error = 2.710505431213761e-20, rel_error = 1.4135798670904448e-16, pi =
3.1415926343385627
i = 16, count = 65536, angle = 0.0054931640625, area = 4.793689954798867e-0
5, abs_error = 1.3552527156068805e-20, rel_error = 1.4135798605937833e-16, p
i = 3.1415926487769856
i = 17, count = 131072, angle = 0.00274658203125, area = 2.396844980153344e-
05, abs_error = 0.0, rel_error = 0.0, pi = 3.141592652386591
i = 18, count = 262144, angle = 0.001373291015625, area = 1.198422490420911e
-05, abs_error = 0.0, rel_error = 0.0, pi = 3.141592653288993
i = 19, count = 524288, angle = 0.0006866455078125, area = 5.992112452534853
e-06, abs_error = 1.6940658945086007e-21, rel_error = 1.4135798584620664e-1
6, pi = 3.141592653514593
i = 20, count = 1048576, angle = 0.00034332275390625, area = 2.9960562263212
138e-06, abs_error = 0.0, rel_error = 0.0, pi = 3.141592653570993
```

In [6]:

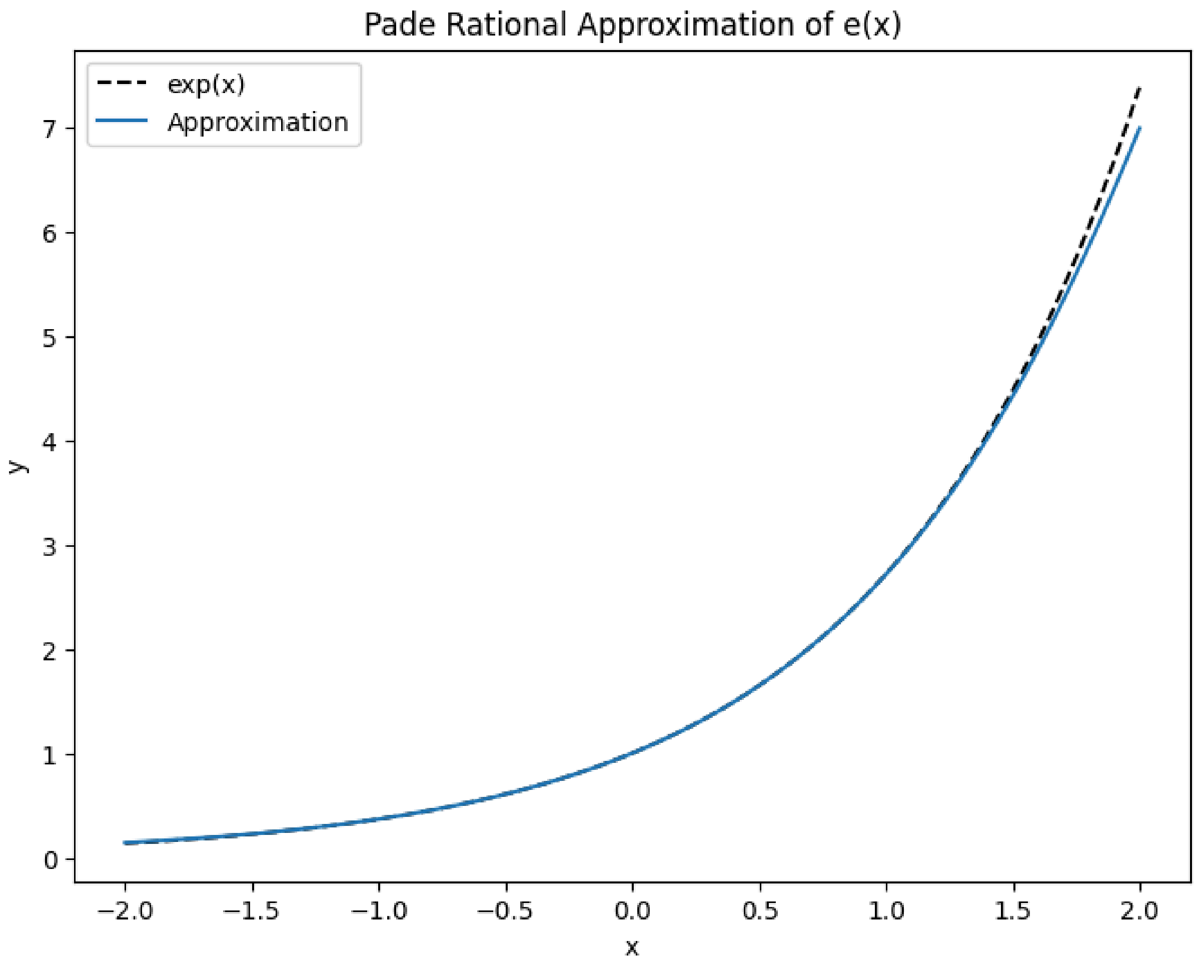
```python
# 1.2 - 21

def factorial(n):
    if n == 0:
        return 1
    return np.prod(np.arange(1, n+1))

def maclaurin_series(f, n, h=1e-3):
    if n % 2 == 0:
        m = n + 1
    else:
        m = n + 2
    half = m // 2
    nodes = np.arange(-half, half+1, dtype=float) * h

    M = np.array([[node**k for node in nodes] for k in range(m)])

    b = np.zeros(m)
    b[n] = factorial(n)

    a = np.linalg.solve(M, b)

    derivative_approx = np.sum(a * np.array([f(xi) for xi in nodes]))

    coeff = derivative_approx / factorial(n)
    return coeff


def plot_maclaurin(f, n, x_min, x_max):
    x = np.linspace(x_min, x_max, 100)
    coeffs = [maclaurin_series(f, i) for i in range(n+1)]
    for i in range(1, 6):
        y = np.zeros_like(x)
        for j in range(i):
            y += coeffs[j] * x**j
        plt.plot(x, y, label=f"n = {i} terms")

    plt.plot(x, f(x), 'k--', label="exp(x)")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("Maclaurin Series Approximation")
    plt.legend()
    plt.show()

plot_maclaurin(lambda x: np.exp(x), 5, -2, 2)
plot_maclaurin(lambda x: np.sin(x), 5, -2, 2)
```


Maclaurin Series Approximation
n = 1 terms
n = 2 terms
n = 3 terms
n = 4 terms
n = 5 terms
exp(x)
y
x
0
2
4
6
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Maclaurin Series Approximation
n = 1 terms
n = 2 terms
n = 3 terms
n = 4 terms
n = 5 terms
exp(x)
y
x
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

In [7]:
```
# 1.2 - 22
def pade_approximation(f, L, M, h=1e-3):
    N = L + M
    c = np.zeros(N+1)
    for i in range(N+1):
        c[i] = maclaurin_series(f, i, h)

    A = np.zeros((M, M))
    b_vec = np.zeros(M)
    for i in range(M):
        k = L + 1 + i
        for j in range(M):
            A[i, j] = c[k - (j+1)]
        b_vec[i] = -c[k]
    q = np.linalg.solve(A, b_vec)

    q_full = np.concatenate(([1.0], q))

    p = np.zeros(L+1)
    for i in range(L+1):
        s = 0.0
        for j in range(min(i, M) + 1):
            s += q_full[j] * c[i - j]
        p[i] = s

    def approximant(x):
        num = np.polyval(p[::-1], x)
        den = np.polyval(q_full[::-1], x)
        return num / den

    return approximant, p, q_full, c

f = lambda x: np.exp(x)
L = 3
M = 3
approx_f, p_coeff, q_coeff, c = pade_approximation(f, L, M)

print("Pade 근사 분자 계수 p:", p_coeff)
print("Pade 근사 분모 계수 q (q[0]=1):", q_coeff)

x = np.linspace(-2, 2, 200)
y_approx = approx_f(x)
y_true = np.exp(x)

plt.figure(figsize=(8, 6))
plt.plot(x, y_true, 'k--', label="exp(x)")
plt.plot(x, y_approx, label="Approximation")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Pade Rational Approximation of e(x)")
plt.legend()
plt.show()
```

```
Pade 근사 분자 계수 p: [   1.          -626.53026611 -313.26249498   -52.1677554
8]
Pade 근사 분모 계수 q (q[0]=1): [   1.          -627.53026628  313.76787585  -5
2.33719104]
```



In [8]:
```python
# 1.3 - 7

def euler_constant(n):
    s = 0
    e_max = -999999999999
    i_max = 0
    for i in range(1, n+1):
        s += 1 / i
        e = s - np.log(i)

        if e_max < e:
            e_max = e
            i_max = i

        if i % 100 == 0:
            print(f"i = {i}, e = {e}")

    print(f"i_max = {i_max}, e_max = {e_max}")

euler_constant(5000)
```

```
i = 100, e = 0.5822073316515288
i = 200, e = 0.5797135815734098
i = 300, e = 0.5788814056433012
i = 400, e = 0.5784651440685238
i = 500, e = 0.5782153315683285
i = 600, e = 0.5780487667534508
i = 700, e = 0.5779297805478292
i = 800, e = 0.5778405346932214
i = 900, e = 0.577771117576444
i = 1000, e = 0.5777155815682065
i = 1100, e = 0.5776701414855578
i = 1200, e = 0.5776322736978301
i = 1300, e = 0.5776002309764809
i = 1400, e = 0.5775727652416682
i = 1500, e = 0.5775489611978291
i = 1600, e = 0.5775281323494514
i = 1700, e = 0.5775097537135299
i = 1800, e = 0.5774934169591495
i = 1900, e = 0.5774787997122512
i = 2000, e = 0.5774656440682016
i = 2100, e = 0.577453741243179
i = 2200, e = 0.5774429204111771
i = 2300, e = 0.5774330404528918
i = 2400, e = 0.5774239837672805
i = 2500, e = 0.5774156515681996
i = 2600, e = 0.5774079602664202
i = 2700, e = 0.5774008386555254
i = 2800, e = 0.5773942257008384
i = 2900, e = 0.5773880687857824
i = 3000, e = 0.5773823223089227
i = 3100, e = 0.5773769465525795
i = 3200, e = 0.5773719067634975
i = 3300, e = 0.5773671724007521
i = 3400, e = 0.5773627165162711
i = 3500, e = 0.5773585152416505
i = 3600, e = 0.5773545473603523
i = 3700, e = 0.5773507939494777
i = 3800, e = 0.5773472380778735
i = 3900, e = 0.5773438645508655
i = 4000, e = 0.5773406596931707
i = 4100, e = 0.5773376111636459
i = 4200, e = 0.5773347077964406
i = 4300, e = 0.577331939464333
i = 4400, e = 0.5773292969607322
i = 4500, e = 0.5773267718973916
i = 4600, e = 0.5773243566154296
i = 4700, e = 0.5773220441077846
i = 4800, e = 0.5773198279512748
i = 4900, e = 0.5773177022470506
i = 5000, e = 0.577315661568166
i_max = 1, e_max = 1.0
```

In [9]:
```
# 1.4 - 15

def lebesgue_constant(n):
    row = 1 /  (2*n + 1)
    valid_value = (4/(np.pi*np.pi))*np.log(2*n+1) + 1
    for i in range(1, n+1):
        row += 2/np.pi*(1/i)*np.tan(np.pi*i/(2*n+1))

    print(f"Validation = {valid_value}, row = {row}, error = {np.abs(row - v

for i in range(1, 101):
    lebesgue_constant(i)
```

```
Validation = 1.4452507898074822, row = 1.435991124176917, error = 0.00925966
563056524
Validation = 1.6522806171467048, row = 1.6421884352221212, error = 0.0100921
8192458361
Validation = 1.788647678255689, row = 1.7783228615258757, error = 0.01032481
6729813335
Validation = 1.8905015796149645, row = 1.8800805991023548, error = 0.0104209
80512609646
Validation = 1.9718303491611895, row = 1.961360593766015, error = 0.01046975
5395174474
Validation = 2.0395348195226264, row = 2.029036981624415, error = 0.01049783
7898211237
Validation = 2.097531406954187, row = 2.087015938756524, error = 0.010515468
197662692
Validation = 2.148258118124167, row = 2.13773086254819, error = 0.0105272555
75977001
Validation = 2.1933361701271217, row = 2.1828006472139836, error = 0.0105355
22913138085
Validation = 2.2338984680631713, row = 2.2233569241536824, error = 0.0105415
43909488915
Validation = 2.2707679410465813, row = 2.2602218767457516, error = 0.0105460
64300829716
Validation = 2.3045612342934096, row = 2.2940116900558927, error = 0.0105495
44237516972
Validation = 2.3357523694224467, row = 2.3252000892924096, error = 0.0105522
80130037062
Validation = 2.364713596672551, row = 2.3541591267780015, error = 0.01055446
9894549445
Validation = 2.3917425926843103, row = 2.381186342918065, error = 0.01055624
9766245251
Validation = 2.4170811389686717, row = 2.406523422962311, error = 0.01055771
6006360796
Validation = 2.4409282954023936, row = 2.430369357209375, error = 0.01055893
8193018452
Validation = 2.46344990777773, row = 2.4528899401492437, error = 0.010559967
628486433
Validation = 2.4847856093301086, row = 2.4742247665192183, error = 0.0105608
42810890314
Validation = 2.505054069358412, row = 2.494492476272467, error = 0.010561593
085944843
Validation = 2.524356990551078, row = 2.5137947494121162, error = 0.01056224
113896187
Validation = 2.542782196761669, row = 2.5322193920318794, error = 0.01056280
4729789654
Validation = 2.5604060488118847, row = 2.549842750889397, error = 0.01056329
7922487891
Validation = 2.577295356511378, row = 2.5667316245401457, error = 0.01056373
1971232393
Validation = 2.5935089079316493, row = 2.5829447919636572, error = 0.0105641
15967992116
Validation = 2.609098704346813, row = 2.5985342470242694, error = 0.01056445
7322543408
Validation = 2.624110966307894, row = 2.6135462041853272, error = 0.01056476
2122566851
Validation = 2.638586959934604, row = 2.6280219245276886, error = 0.01056503
5406915335
```

```
Validation = 2.65256368064992, row = 2.641998399274659, error = 0.0105652813
75261026
Validation = 2.6660744228895634, row = 2.6555089193392236, error = 0.0105655
03550339805
Validation = 2.6791492578706535, row = 2.6685835529658486, error = 0.0105657
04904804907
Validation = 2.6918154366693314, row = 2.6812495487081693, error = 0.0105658
8796116209
Validation = 2.704097732195577, row = 2.693531677324413, error = 0.010566054
87116391
Validation = 2.7160187308540635, row = 2.705452523374714, error = 0.01056620
7479349554
Validation = 2.7275990825208036, row = 2.717032735146631, error = 0.01056634
7374172835
Validation = 2.7388577157863923, row = 2.7282912398568744, error = 0.0105664
75929517871
Validation = 2.749812024100892, row = 2.739245429762283, error = 0.010566594
338608848
Validation = 2.7604780274168785, row = 2.749911323775151, error = 0.01056670
3641727404
Validation = 2.770870513101718, row = 2.7603037083525352, error = 0.01056680
4749182701
Validation = 2.781003159229929, row = 2.770436260769623, error = 0.010566898
460306007
Validation = 2.7908886428351147, row = 2.7803216573557954, error = 0.0105669
85479319335
Validation = 2.800538735270872, row = 2.789971668842332, error = 0.010567066
428539995
Validation = 2.8099643864800328, row = 2.7993972446203794, error = 0.0105671
4185965335
Validation = 2.819175799685226, row = 2.8086085874221838, error = 0.01056721
2263042247
Validation = 2.8281824977783154, row = 2.8176152197025655, error = 0.0105672
78075749975
Validation = 2.8369933824917926, row = 2.8264260428033636, error = 0.0105673
39688428934
Validation = 2.8456167872738267, row = 2.8350493898228617, error = 0.0105673
97450965021
Validation = 2.8540605246542397, row = 2.8434930729766092, error = 0.0105674
51677630402
Validation = 2.862331928776154, row = 2.851764426124932, error = 0.010567502
651221972
Validation = 2.8704378936735764, row = 2.8598703430467736, error = 0.0105675
50626802813
Validation = 2.8783849077955255, row = 2.8678173119605335, error = 0.0105675
95834992005
Validation = 2.886179085209876, row = 2.875611446725291, error = 0.010567638
484584752
Validation = 2.8938261938629037, row = 2.8832585150978227, error = 0.0105676
78765081023
Validation = 2.901331681221726, row = 2.8907639643729612, error = 0.01056771
6848764785
Validation = 2.908700697585212, row = 2.8981329446927306, error = 0.01056775
2892481241
Validation = 2.915938117313215, row = 2.9053703302737963, error = 0.01056778
703941852
```

```
Validation = 2.923048558193286, row = 2.912480738772947, error = 0.010567819
420338687
Validation = 2.930036399137591, row = 2.919468548982643, error = 0.010567850
154947767
Validation = 2.9369057963798566, row = 2.9263379170268413, error = 0.0105678
79353015286
Validation = 2.943660698322379, row = 2.933092791207115, error = 0.010567907
115263786
Validation = 2.950304859165894, row = 2.9397369256315895, error = 0.01056793
3534304519
Validation = 2.9568418514401147, row = 2.946273892744683, error = 0.01056795
8695431479
Validation = 2.9632750775396484, row = 2.9527070948624243, error = 0.0105679
8267722403
Validation = 2.96960778035856, row = 2.959039774806259, error = 0.0105680055
52300974
Validation = 2.9758430531067894, row = 2.965275025719104, error = 0.01056802
7387685586
Validation = 2.9819838483828107, row = 2.9714158001373736, error = 0.0105680
48245437112
Validation = 2.9880329865691513, row = 2.9774649183860538, error = 0.0105680
68183097523
Validation = 2.993993163610522, row = 2.9834250763565704, error = 0.01056808
7253951752
Validation = 2.999866958228254, row = 2.9892988527206885, error = 0.01056810
5507565484
Validation = 3.005656838619367, row = 2.9950887156293993, error = 0.01056812
2989967677
Validation = 3.011365168683816, row = 3.0007970289398442, error = 0.01056813
9743971638
Validation = 3.0169942138192556, row = 3.006426058009748, error = 0.01056815
5809507651
Validation = 3.0225461463188603, row = 3.011977975095052, error = 0.01056817
1223808154
Validation = 3.0280230504043906, row = 3.0174548643828456, error = 0.0105681
86021544967
Validation = 3.0334269269236938, row = 3.0228587266884595, error = 0.0105682
00235234304
Validation = 3.0387596977391316, row = 3.028191483843957, error = 0.01056821
3895174594
Validation = 3.0440232098310154, row = 3.03345498280126, error = 0.010568227
029755572
Validation = 3.049219239137984, row = 3.0386509994723903, error = 0.01056823
9665593726
Validation = 3.054349494154295, row = 3.0437812423266903, error = 0.01056825
182760468
Validation = 3.0594156193022704, row = 3.0488473557630655, error = 0.0105682
63539204814
Validation = 3.0644191980965476, row = 3.0538509232741737, error = 0.0105682
7482237388
Validation = 3.0693617561153763, row = 3.0587934704176525, error = 0.0105682
85697723834
Validation = 3.074244763792906, row = 3.0636764676082304, error = 0.01056829
6184675585
Validation = 3.079069639045253, row = 3.0685013327437205, error = 0.01056830
6301532715
```

```
Validation = 3.0838377497420866, row = 3.0732694336765713, error = 0.0105683
16065515226
Validation = 3.0885504160345008, row = 3.0779820905416777, error = 0.0105683
2549282305
Validation = 3.093208912549099, row = 3.0826405779503556, error = 0.01056833
4598743512
Validation = 3.097814470457402, row = 3.087246127059648, error = 0.010568343
397753921
Validation = 3.10236827942899, row = 3.0917999275255372, error = 0.010568351
903452733
Validation = 3.1068714894761302, row = 3.0963031293473615, error = 0.0105683
60128768717
Validation = 3.1113252126970457, row = 3.1007568446111593, error = 0.0105683
68085886348
Validation = 3.1157305249244347, row = 3.1051621491380867, error = 0.0105683
7578634795
Validation = 3.1200884672853566, row = 3.109520084044303, error = 0.01056838
3241053692
Validation = 3.1244000476781357, row = 3.113831657217819, error = 0.01056839
0460316657
Validation = 3.1286662421715343, row = 3.1180978447175125, error = 0.0105683
97454021827
Validation = 3.132887996331045, row = 3.1223195920995863, error = 0.01056840
4231458661
Validation = 3.1370662264768137, row = 3.1264978156754024, error = 0.0105684
10801411243
Validation = 3.1412018208773818, row = 3.1306334037051107, error = 0.0105684
17172271083
Validation = 3.1452956408831314, row = 3.1347272175311263, error = 0.0105684
2335200514
Validation = 3.1493485220030593, row = 3.1387800926548395, error = 0.0105684
29348219777
```