MATH 152 Lab 4

Ram Paramathmuni 501

```
import sympy as sp
from sympy.plotting import (plot, plot_parametric)
x, u = sp.symbols('x,u')
```

Question 1

1a

```
# Enter your code here
sec, tan = sp.sec, sp.tan
integrand1a = (sec(x)**2 + sec(x)*tan(x)) / (sec(x) + tan(x))
u_expr = sec(x) + tan(x)
du_dx = sp.diff(u_expr, x)

print("Simplified Integrand:" , sp.simplify(du_dx/u_expr))

integral1a = sp.integrate(du_dx/u_expr, x)
simplified = sp.simplify(integral1a)
print("Integrated:", sp.simplify(integral1a))
```

```
Simplified Integrand: 1/cos(x)
Integrated: log(tan(x) + sec(x))
```

1b

```
# Enter your answer here

sin, cos = sp.sin, sp.cos

# Multiply by cos(x)/cos(x) and use cos^2(x) = 1 - sin^2(x)
expr_final = sp.simplify(cos(x) / (1 - sin(x)**2))

# Substitute u = sin(x)
expr_final_u = expr_final.subs(sin(x), u)

# Print only the final expression
print(expr_final_u)
```

```
1/cos(x)
```

1c

```
# Enter your code here
partial_fraction = sp.apart(1/(1 - u**2), u)
print("Partial Fraction Decomposition:", partial_fraction)
```

```
Partial Fraction Decomposition: 1/(2*(u + 1)) - 1/(2*(u - 1))
```

1d

```
# Enter your code here
Function_u = sp.integrate(sp.apart(1/(1 - u**2), u), u)
print(Function_u)  # integral in u

# optional: back-substitute u = sin(x) to express the ant ivative in x
Fd_x = sp.simplify(Function_u.subs(u, sp.sin(x)))
```

```
print(Fd_x)
```

```
-log(u - 1)/2 + log(u + 1)/2
-log(sin(x) - 1)/2 + log(sin(x) + 1)/2
```

Question 2

2a

```
# Enter your code here
oo = sp.oo

def function1(k):
    return sp.integrate(1/(x**k + 1), (x, -oo, oo))

for k in [2, 4, 6]:
    print(f"k={k} →", sp.simplify(function1(k)))

# odd k: real pole at x=-1 → improper integral diverges
try:
    print("k=3 →", function1(3))
except Exception as e:
    print("k=3 → diverges (real singularity at x = -1). Error:", e)
```

```
k=2 → pi
k=4 → sqrt(2)*pi/2
k=6 → 2*pi/3
k=3 → nan
```

2b

```
# Enter your code here
e, oo = sp.E, sp.oo
expr = 1/(x*sp.log(x))
result = sp.integrate(expr, (x, e, oo))
print(result)   # → oo (diverges)
```

```
oo
```

2c

```
# Enter your code here
lower_limit = sp.E**sp.E
expr = 1/(x*sp.log(x)*sp.log(sp.log(x)))
result = sp.integrate(expr, (x, lower_limit, sp.oo))
print(result)   # → oo (still diverges)
```

```
oo
```

2d

```
# Enter your code here
g = sp.log(sp.log(sp.log(x)))   # → ∞ as x→∞ (very slowly)
C = sp.E**(sp.E**sp.E)

print("lim g(x) as x→∞ =", sp.limit(g, x, sp.oo))  # → oo

integrand = 1/(x*sp.log(x)*sp.log(sp.log(x))*g)
result = sp.integrate(integrand, (x, C, sp.oo))
print(result)   # → oo (diverges while g(x)→∞)
```

```
lim g(x) as x→∞ = oo
oo
```