1. 按照以下格式生成九九乘法表

$$
\begin{array}{lllllllll}
1\times1=1 \\
1\times2=2 & 2\times2=4 \\
1\times3=3 & 2\times3=6 & 3\times3=9 \\
1\times4=4 & 2\times4=8 & 3\times4=12 & 4\times4=16 \\
1\times5=5 & 2\times5=10 & 3\times5=15 & 4\times5=20 & 5\times5=25 \\
1\times6=6 & 2\times6=12 & 3\times6=18 & 4\times6=24 & 5\times6=30 & 6\times6=36 \\
1\times7=7 & 2\times7=14 & 3\times7=21 & 4\times7=28 & 5\times7=35 & 6\times7=42 & 7\times7=49 \\
1\times8=8 & 2\times8=16 & 3\times8=24 & 4\times8=32 & 5\times8=40 & 6\times8=48 & 7\times8=56 & 8\times8=64 \\
1\times9=9 & 2\times9=18 & 3\times9=27 & 4\times9=36 & 5\times9=45 & 6\times9=54 & 7\times9=63 & 8\times9=72 & 9\times9=81
\end{array}
$$

实现思想：两层循环并连接字符串

In [1]:
```
for i in range(1, 10): print('  '.join(f'{j}×{i}={j*i}' for j in ra
```

```
1×1=1
1×2=2  2×2=4
1×3=3  2×3=6  3×3=9
1×4=4  2×4=8  3×4=12  4×4=16
1×5=5  2×5=10  3×5=15  4×5=20  5×5=25
1×6=6  2×6=12  3×6=18  4×6=24  5×6=30  6×6=36
1×7=7  2×7=14  3×7=21  4×7=28  5×7=35  6×7=42  7×7=49
1×8=8  2×8=16  3×8=24  4×8=32  5×8=40  6×8=48  7×8=56  8×8=64
1×9=9  2×9=18  3×9=27  4×9=36  5×9=45  6×9=54  7×9=63  8×9=72  9×9=8
1
```

输出并不存在明显问题

2. 设函数$f(x)=x+sin(x+cosx)$,设计算法讨论$f(x)$在$[0,4\pi]$上的所有的极值点，精确到小数点后4位，并计算极值。

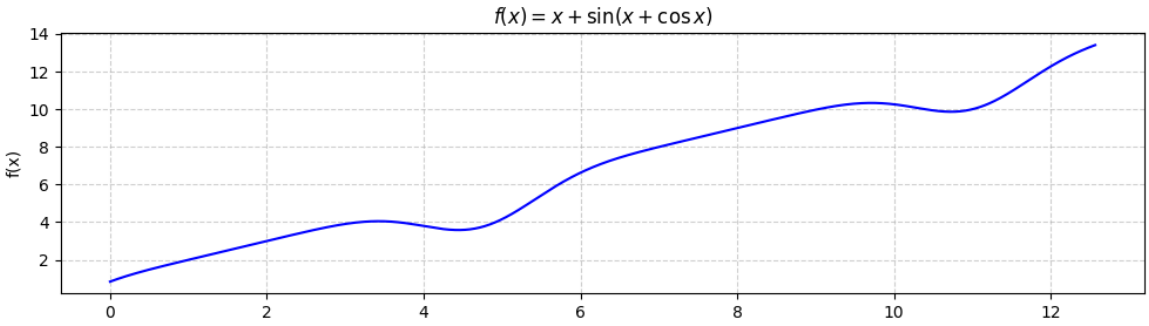
首先，导入numpy和matplotlib包，并定义函数与x范围并计算对应的y值。同时定义其一阶与二阶导函数

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

def f(x):
    return x+np.sin(x+np.cos(x))

def f_prime(x):
    return 1+np.cos(x+np.cos(x))*(1-np.sin(x))

def f_double_prime(x):
    u=x+np.cos(x)
    return -np.sin(u)*(1-np.sin(x))**2+np.cos(u)*(-np.cos(x))

x=np.linspace(0,4*np.pi,1000)
y=f(x)
y1=f_prime(x)
```

```
y2=f_double_prime(x)
```

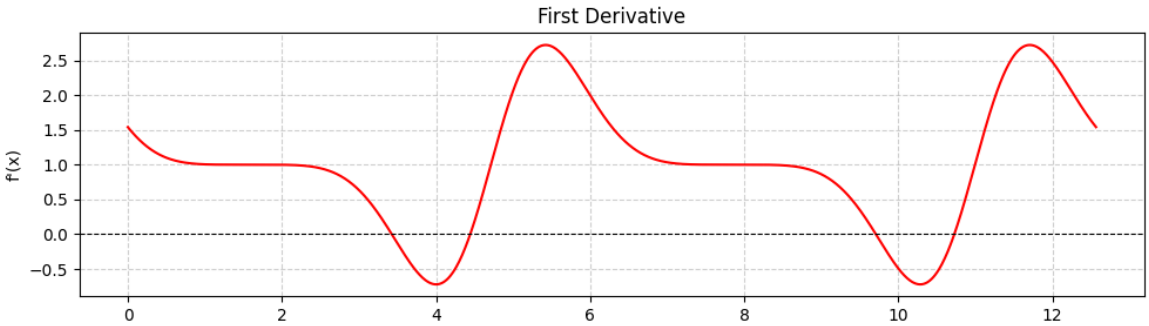
绘制原函数图

In [3]:
```
plt.figure(figsize=(10, 3))
plt.plot(x, y, 'b-', linewidth=1.5)
plt.ylabel('f(x)')
plt.grid(True, linestyle='--', alpha=0.6)
plt.title(r'$f(x)=x+\sin(x+\cos x)$')
plt.tight_layout()
plt.show()
```



可以看出我们不能直接看出极值点的位置，所以我们对一阶导数进行绘图

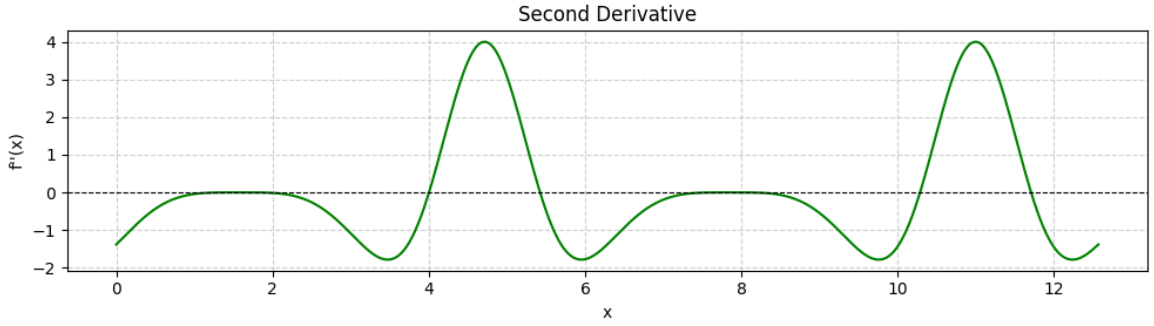
In [4]:
```
plt.figure(figsize=(10, 3))
plt.plot(x, y1, 'r-', linewidth=1.5)
plt.axhline(y=0, color='black', linestyle='--', linewidth=0.8)
plt.ylabel("f'(x)")
plt.grid(True, linestyle='--', alpha=0.6)
plt.title("First Derivative")
plt.tight_layout()
plt.show()
```



费马定理指出可导函数极值点导数为零。如图我们可以看出导函数和直线$y=0$在$[0,4\pi]$有四个交点。但是这样的话我们不能看出来是最大值还是最小值，所以我们需要将导函数的导数求出来，求导数之后再看导数在$[0,4\pi]$的取值范围，如果导数在$[0,4\pi]$的取值范围是正数，那么这个函数就是最大值，否则就是最小值。

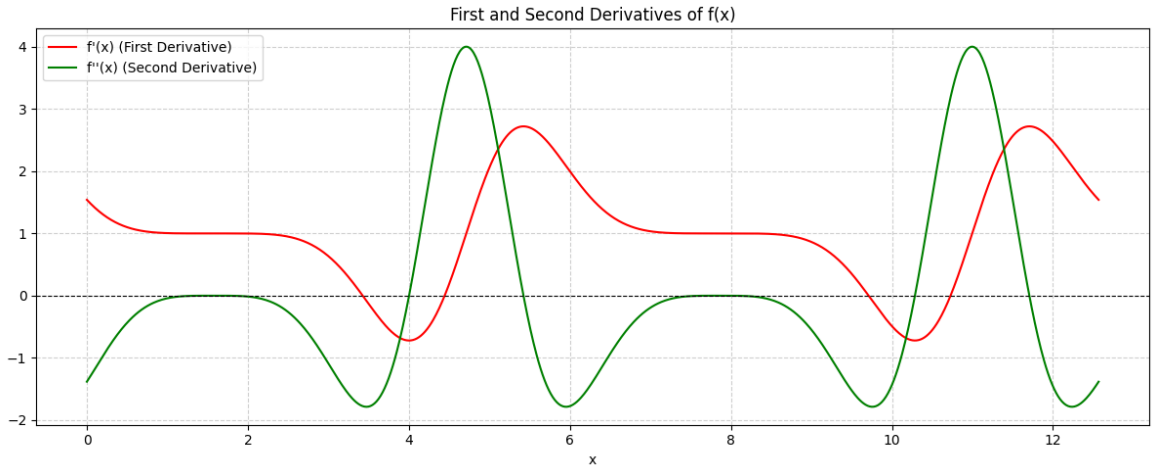
In [5]:
```
plt.figure(figsize=(10, 3))
plt.plot(x, y2, 'g-', linewidth=1.5)
plt.axhline(y=0, color='black', linestyle='--', linewidth=0.8)
plt.xlabel('x')
plt.ylabel("f''(x)")
plt.grid(True, linestyle='--', alpha=0.6)
```

```
plt.title("Second Derivative")
plt.tight_layout()
plt.show()
```



呃，两张图对比一下

In [6]:
```
plt.figure(figsize=(12, 5))
plt.plot(x, y1, 'r-', linewidth=1.5, label="f'(x) (First Derivative
plt.plot(x, y2, 'g-', linewidth=1.5, label="f''(x) (Second Derivati
plt.axhline(y=0, color='black', linestyle='--', linewidth=0.8)
plt.xlabel('x')
plt.title("First and Second Derivatives of f(x)")
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



如图所示，在$[0, 4\pi]$范围内共有4个极值点，分别为最大值最小值最大值最小值。然后使用二分法计算导数零点。 首先，编写二分法求解函数

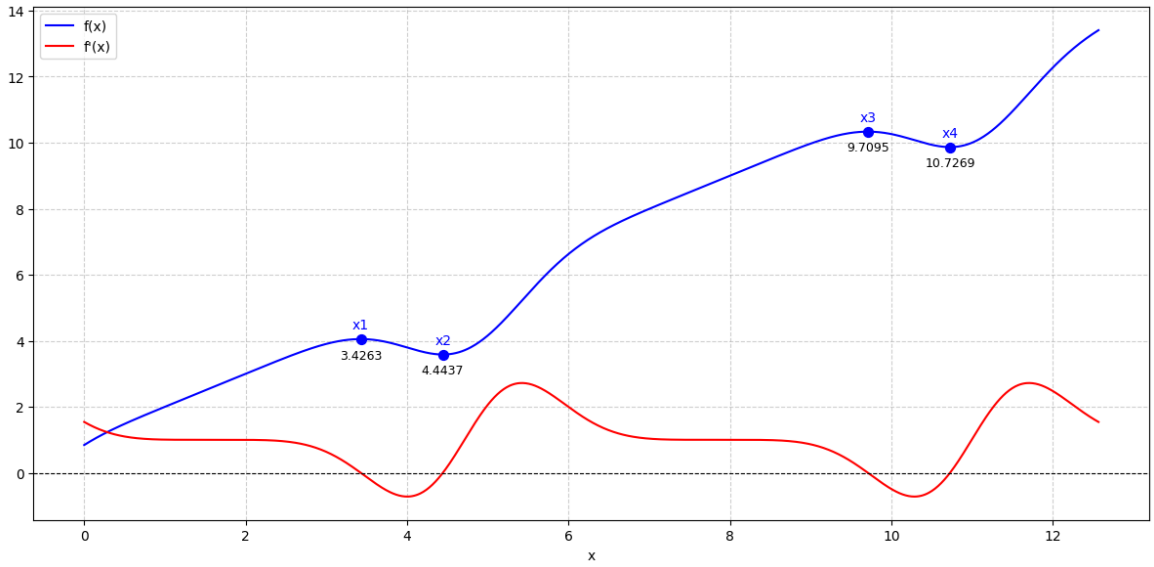
In [7]:
```
def bisect(f,a,b,tol=1e-9):
    while (b-a)/2>tol:
        c=(a+b)/2
        if f(a)*f(c)<0:b=c
        else:a=c
    return (a+b)/2
```

观察图像可知四个极值点范围分别为$(3, 4), (4, 5), (9, 10), (10, 11)$ 带入并求解得:

In [8]:
```
intervals = [(3, 4), (4, 5), (9, 10), (10, 11)]
i=0
for a, b in intervals:
    i+=1
    root = bisect(f_prime, a, b, 1e-9)
    print(f"x{i}={root:.4f}")
```

```
x1=3.4263
x2=4.4437
x3=9.7095
x4=10.7269
```

In [9]:
```
x_points=[3.4263,4.4437,9.7095,10.7269]
x_labels=['x1','x2','x3','x4']

y_f_vals=f(np.array(x_points))
y_fp_vals=f_prime(np.array(x_points))

plt.figure(figsize=(12,6))
plt.plot(x,y,'b-',linewidth=1.5,label="f(x)")
plt.plot(x,y1,'r-',linewidth=1.5,label="f'(x)")
plt.axhline(y=0,color='black',linestyle='--',linewidth=0.8)

plt.scatter(x_points,y_f_vals,color='blue',zorder=5,s=50)
for i,(xi,yi) in enumerate(zip(x_points,y_f_vals)):
    plt.text(xi,yi+0.3,x_labels[i],fontsize=10,ha='center',color='b
    plt.text(xi,yi-0.6,f'{xi:.4f}',fontsize=9,ha='center',color='bl

plt.xlabel('x')
plt.legend()
plt.grid(True,linestyle='--',alpha=0.6)
plt.tight_layout()
plt.show()
```



3. 编写Gauss消去法求解以下方程组

(1)

$$\begin{cases} x_1 + 0.5x_2 + 0.33x_3 = 1.83 \\ 0.5x_1 + 0.33x_2 + 0.25x_3 = 1.08 \\ 0.33x_1 + 0.25x_2 + 0.2x_3 = 0.783 \end{cases}$$

(2)

$$\begin{cases} x_1 + 0.5x_2 + 0.33x_3 + 0.25x_4 = 2.08 \\ 0.5x_1 + 0.33x_2 + 0.25x_3 + 0.2x_4 = 1.28 \\ 0.33x_1 + 0.25x_2 + 0.2x_3 + 0.17x_4 = 0.95 \\ 0.25x_1 + 0.2x_2 + 0.167x_3 + 0.143x_4 = 0.76 \end{cases}$$

实现高斯消元

In [10]:
```python
def Gauss_Elimination(a:list[list[float]])->list:
    n=len(a)
    for i in range(n):
        pivot=i
        while(pivot<n and a[pivot][i]==0): pivot+=1
        if pivot==n: continue
        if pivot !=i: a[i],a[pivot]=a[pivot],a[i]
        for j in range(i+1,n):
            p=a[j][i]/a[i][i]
            for k in range(i,n+1): a[j][k]-=p*a[i][k]
    ans=[]
    for i in range(n-1,-1,-1):
        for j in range(i+1,n): a[i][n]-=a[i][j]*ans[n-1-j]
        if a[i][i]==0:
            raise ValueError("No Solution")
        else: ans.append(a[i][n]/a[i][i])
    ans.reverse()
    return ans
```

对于传入的矩阵，不能保证其有解。所以加入异常及其处理，用于应对无解或者无穷解的情况。

In [11]:
```python
input_1=[[1,0.5,0.33,1.83],[0.5,0.33,0.25,1.08],[0.33,0.25,0.2,0.78
input_2=[[1,0.5,0.33,0.25,2.08],[0.5,0.33,0.25,0.2,1.28],[0.33,0.25
try:
    ans_1=Gauss_Elimination(input_1)
    print("The answer of input_1 is:")
    for an_1 in ans_1:print(f"{an_1:.4f}")
except ValueError as e:
    print(e)
try:
    ans_2=Gauss_Elimination(input_2)
    print("The answer of input_2 is:")
    for an_2 in ans_2:print(f"{an_2:.4f}")
except ValueError as e:
    print(e)
```

```
The answer of input_1 is:
1.7667
-3.0476
4.8095
The answer of input_2 is:
1.0000
1.0000
1.0000
1.0000
```

虽然，都用python了。更加常见的解法应该是：

In [12]:
```
A1=[[1,0.5,0.33],[0.5,0.33,0.25],[0.33,0.25,0.2]]
A2=[[1,0.5,0.33,0.25],[0.5,0.33,0.25,0.2],[0.33,0.25,0.2,0.17],[0.2
B1=[1.83,1.08,0.783]
B2=[2.08, 1.28, 0.95, 0.76]
try:
    Ans_1=np.linalg.solve(A1,B1)
    print("The answer of input_1 is:")
    for An_1 in Ans_1:print(f"{An_1:.4f}")
except:
    print("No Solution")
try:
    Ans_2=np.linalg.solve(A2,B2)
    print("The answer of input_2 is:")
    for An_2 in Ans_2:print(f"{An_2:.4f}")
except:
    print("No Solution")
```

```
The answer of input_1 is:
1.7667
-3.0476
4.8095
The answer of input_2 is:
1.0000
1.0000
1.0000
1.0000
```