Imports

In [5]:
```
:dep rayon
```

In [6]:
```
use rayon::prelude::*;
use std::f64::consts::PI;
```

1. Implementation of trapezoidal and Simpson's quadratic rules

I have implemented both of these rules as methods on any function

$$f : \mathbb{R} \Rightarrow \mathbb{R}$$

These operations are also parallelized using the `rayon` library. As long as the function is thread-safe (which the type system will ensure via the `Sync` and `Send` bounds), we can easily parallelize operations by using `into_par_iter`.

In [ ]:
```
trait Quadrature {
    fn trapezoidal(&self, a: f64, n: usize) -> f64;
    fn simpsons(&self, a: f64, n: usize) -> f64;
}

impl<F> Quadrature for F
where
    F: Fn(f64) -> f64 + Sync + Send,
{
    fn trapezoidal(&self, a: f64, n: usize) -> f64 {
        let h = (2.0 * a) / (n as f64);

        let sum: f64 = (1..n)
            .into_par_iter()
            .map(|i| {
                self(-a + i as f64 * h)
            })
            .sum();

        h * (0.5 * (self(-a) + self(a)) + sum)
    }
    fn simpsons(&self, a: f64, n: usize) -> f64 {
        let h = (2.0 * a) / (n as f64);

        let sum: f64 = (1..n)
            .into_par_iter()
            .map(|i| {
                let x_h = -a + (i - 1) as f64 * h;
                let x_i = -a + i as f64 * h;
                let x_j = -a + (i + 1) as f64 * h;
```

```
                self(x_h) + 4.0 * self((x_i + x_j) / 2.0) + self(x_j)
            })
            .sum();

        (h / 6.0) * (self(-a) + self(a) + sum)
    }
}
```

This is the function we are integrating, which is also known as the `sinc` function.

In [22]:
```
fn sinc(x: f64) -> f64 {
    if x.abs() < 1e-10 {
        1.0
    } else {
        x.sin() / x
    }
}
```

In [23]:
```
let a = 1e6; // Large interval
let n = 1_000_000; // Number of intervals (should be large)

let integral_trap = sinc.trapezoidal(a, n);
let integral_simp = sinc.simpsons(a, n);

println!("Trapezoidal Rule Approximation: {}", integral_trap);
println!("Simpson's Rule Approximation: {}", integral_simp);
```

```
Trapezoidal Rule Approximation: 3.1415914506275415
Simpson's Rule Approximation: 3.1415923088615667
```

2. Truncation error

In [25]:
```
fn compute_error(a: f64, n: usize) {
    let integral_trap = sinc.trapezoidal(a, n);
    let integral_simp = sinc.simpsons(a, n);

    let error_trap = integral_trap - PI;
    let error_simp = integral_simp - PI;

    println!(
        "a = {:e}, N = {:e} | Trapezoidal Error: {:.5e}, Simpson's Error: {:
        a, n as f64, error_trap, error_simp
    );
}
```

In [ ]:
```
let a_values = (6..10).map(|i| 10.0_f64.powi(i)).collect::<Vec<f64>>(); // D
let n_values = (6..10).map(|i| 10_usize.pow(i)).collect::<Vec<usize>>(); //

for &a in &a_values {
    for &n in &n_values {
        compute_error(a, n);
    }
}
```

```
a = 1e6, N = 1e6 | Trapezoidal Error: -1.20296e-6, Simpson's Error: -3.44728
e-7
a = 1e6, N = 1e7 | Trapezoidal Error: -1.86725e-6, Simpson's Error: -1.80266
e-6
a = 1e6, N = 1e8 | Trapezoidal Error: -1.87344e-6, Simpson's Error: -1.86859
e-6
a = 1e6, N = 1e9 | Trapezoidal Error: -1.87350e-6, Simpson's Error: -1.87303
e-6
a = 1e7, N = 1e6 | Trapezoidal Error: 1.88496e1, Simpson's Error: 2.09439e0
a = 1e7, N = 1e7 | Trapezoidal Error: 1.16510e-7, Simpson's Error: 3.52128e-
8
a = 1e7, N = 1e8 | Trapezoidal Error: 1.80849e-7, Simpson's Error: 1.73522e-
7
a = 1e7, N = 1e9 | Trapezoidal Error: 1.81448e-7, Simpson's Error: 1.80869e-
7
a = 1e8, N = 1e6 | Trapezoidal Error: 1.94779e2, Simpson's Error: 6.07375e1
a = 1e8, N = 1e7 | Trapezoidal Error: 1.88496e1, Simpson's Error: 2.09440e0
a = 1e8, N = 1e8 | Trapezoidal Error: 4.66654e-9, Simpson's Error: 3.11766e-
9
a = 1e8, N = 1e9 | Trapezoidal Error: 7.24299e-9, Simpson's Error: 5.94708e-
9
a = 1e9, N = 1e6 | Trapezoidal Error: 1.99805e3, Simpson's Error: 6.66018e2
a = 1e9, N = 1e7 | Trapezoidal Error: 1.94779e2, Simpson's Error: 6.07375e1
a = 1e9, N = 1e8 | Trapezoidal Error: 1.88496e1, Simpson's Error: 2.09440e0
a = 1e9, N = 1e9 | Trapezoidal Error: -1.07584e-9, Simpson's Error: -1.16280
e-10
```

Out[ ]: ()

For the trapezoidal rule:

When $N = a$, the truncation error is in the magnitude of $\frac{1}{N}$.

When $N < a$, the truncation error is in the magnitude of $\frac{a}{N}$.

When $N > a$, the truncation error remains in the magnitude $\frac{1}{N}$ and grows very slowly.

Simpson's rule is generally better than the trapezoidal rule, with an order of magnitude less error when $N < a$ and $N = a$. When $N > a$, Simpson's rule becomes very similar to the trapezoidal rule.

This seems to indicate that the optimal values for these parameters when approximating is that $N = a$, and that Simpson's rule is more accurate than the trapezoidal rule.

3. Convergence

In [36]:
```rust
fn convergence_steps(a: f64, epsilon: f64) {
    let mut n = 10_000;
    let mut prev_trap = sinc.trapezoidal(a, n);
    let mut prev_simp = sinc.simpsons(a, n);
    let mut iter_trap = 0;
```

```
    let mut iter_simp = 0;

    loop {
        n += 10_000;
        let current_trap = sinc.trapezoidal(a, n);

        iter_trap += 1;

        if (current_trap - prev_trap).abs() < epsilon {
            println!(
                "Converged for a = {:e}, n = {:e} | Trapezoidal in {} iterat
                a, n, iter_trap
            );
            break;
        }

        prev_trap = current_trap;
    }
    let mut n = 10_000;
    loop {
        n += 10_000;
        let current_simp = sinc.simpsons(a, n);

        iter_simp += 1;

        if (current_simp - prev_simp).abs() < epsilon {
            println!(
                "Converged for a = {:e}, n = {:e} | Simpson's in {} iteratic
                a, n, iter_simp
            );
            break;
        }

        prev_simp = current_simp;
    }
}
```

In [37]:
```
let epsilon = 1e-4;

for &a in &a_values {
    convergence_steps(a, epsilon);
}
```

```
Converged for a = 1e6, n = 1e5 | Trapezoidal in 9 iterations
Converged for a = 1e6, n = 1e5 | Simpson's in 9 iterations
Converged for a = 1e7, n = 2.1e5 | Trapezoidal in 20 iterations
Converged for a = 1e7, n = 2.1e5 | Simpson's in 20 iterations
Converged for a = 1e8, n = 6e5 | Trapezoidal in 59 iterations
Converged for a = 1e8, n = 6e5 | Simpson's in 59 iterations
Converged for a = 1e9, n = 1.85e6 | Trapezoidal in 184 iterations
Converged for a = 1e9, n = 1.85e6 | Simpson's in 184 iterations
```

Out[37]: ()

In this context, the steps to convergence involve making the intervals smaller and smaller, i.e. making n larger and larger. I am increasing it by 10,000 at every step.

The number of steps doubles between a = 1e6 and 1e7, and triples between 1e7 and 1e8, and 1e8 and 1e9. In general n increases slowly in order to converge.

The number of steps for convergence is the same between the trapezoidal rule and Simpson's rule.