# Chapter 6 Code

This repository contains an implementation of the backward Euler method as well as the trapezoidal method for solving ODEs and systems of ODEs, respectively.

## Important Notes

The backward Euler method expects two functions, $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ and $f_y : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, the partial derivative of $f$ with respect to $y$. Both of these functions has the signature `double(double t, double y)`.
These functions can be defined the usual way:

```
double f(double t, double y) {
    return ...;
}

double fy(double t, double y) {
    return ...;
}
```

Or, they can be defined via lambda functions:

```
auto f = [](double t, double y) { return ...; };
auto fy = [](double t, double y) { return ...; };
```

Which has the possible advantage of being easier to read if declaring a vector of such functions.

---

The trapezoidal method, on the other hand, expects a vector of $n$ functions (where $n$ is the number of systems) $f_i : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$. These functions have the signature `double(double t, const std::vector<double> &y)` where `y.size()` is $n$.

These functions may be defined in the usual way and then put into a vector:

```
double f1(double t, const std::vector<double> &y) {
    return ...;
}

...

double fn(double t, const std::vector<double> &y) {
    return ...;
}

// `funcn` is an alias for `std::function<double(double t, const std::vector<c
std::vector<funcn> f({f1, ..., fn});
```

Or, they can be defined in-line via lambda functions:

```
std::vector<funcn> f({
    [](double t, const std::vector<double> &y) { return ...; }, // f1
    ...
    [](double t, const std::vector<double> &y) { return ...; }  // fn
});
```