Floating-point representations: $\pm$`EENMMM` represents $\pm$`N.MMM` $\times 10^{\texttt{EE}-49}$
and the 64 bits `seeeeeeeeeeebbbbbb`$\cdots$`b` represents

$$(-1)^{\texttt{s}}\mathbf{1}.\texttt{bbbbbb}\cdots\texttt{b} \times 2^{\texttt{eeeeeeeeeee}-01111111111}$$

where `0b01111111111` $= 1023 =$ `0x3ff`. Recall 1 is `+491000` or `0x3ff0000000000000`.

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Given n real or complex numbers or vectors $x_1, \ldots, x_n$ and n real or complex numbers $w_1, \ldots, w_n$, then $\sum_{k=1}^{n} w_k x_k$ is:

1. a linear combination of the x-values if there are no restrictions on the weights,
2. a weighted average if $\sum_{k=1}^{n} w_k = 1$, and
3. a convex combination if the weights form a weighted average and each $w_k \geq 0$.

Fixed-point theorem: To approximate a solution to $x = f(x)$, choose x_0 and let $x_k \leftarrow f(x_{k-1})$.
Gaussian elimination with partial pivoting: This is the Gaussian elimination algorithm but always swapping appropriate rows so that the largest entry in absolute value is in the pivot position (the row that will be used to eliminate entries in that column in subsequent rows).
***n*th-order Taylor series:** If h is small, expanding around x yields:

$$f(x+h) = \left(\sum_{k=0}^{n} \frac{1}{k!} f^{(k)}(x) h^k \right) + \frac{1}{(n+1)!} f^{(n+1)}(\xi) h^{n+1}$$

where $x \leq \xi \leq x + h$. Otherwise, if x is close to x_0, expanding around x_0 yields:

$$f(x) = \left(\sum_{k=0}^{n} \frac{1}{k!} f^{(k)}(x_0) (x - x_0)^k \right) + \frac{1}{(n+1)!} f^{(n+1)}(\xi) (x - x_0)^{n+1}$$

where $x_0 \leq \xi \leq x$.

The examples of binary search and interpolation search are not required for this course: they are provided as examples of different bracketing algorithms.

```
double horner( double       const a[],
               unsigned int const degree,
               double       const x ) {
    // The coefficient of xˆk is a[k]
    double result{ a[degree] };

    for ( std::size_t k{degree - 1}; k < degree; --k ) {
        result = result*x + a[k];
    }

    return result;
}
```

Noise: Averaging noisy values with zero bias mitigates the effect, while differentiating noisy values magnifies the effect. Use interpolating polynomials if the data is accurate and precise, but use least squares best-fitting polynomials if the data is accurate but not precise (that is, the data has significant noise). If the data is not accurate, we cannot recover the underlying signal.
Evaluating interpolating polynomials: For interpolating between t_k and t_{k-1} where t_k is the time of the most recent data point, shift and scale to $\ldots, -2.5, -1.5, -0.5$ and 0.5 to ensure that $-0.5 < \delta < 0.5$ to evaluate the polynomial at the point $\frac{t_{k-1}+t_k}{2} + \delta h$ where h is the time step between readings. Note, you do not have to know these formulas explicitly; rather, you must understand the idea behind deriving these. For example, why to we shift and scale so that our choice of δ is such that $|\delta| < 0.5$.

Derivatives:
Centered three-point:

$$f^{(1)}(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{1}{6} f^{(3)}(\xi) h^2$$

Backward two-point:

$$y^{(1)}(t) = \frac{y(t) - y(t-h)}{h} + \frac{1}{2} y^{(2)}(\tau) h$$

Backward three-point:

$$y^{(1)}(t) = \frac{3y(t) - 4y(t-h) + y(t-2h)}{2h} + \frac{1}{3} y^{(3)}(t) h^2 + \mathrm{O}\left(h^3\right)$$

Second derivatives:
Centered three-point:

$$f^{(2)}(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} - \frac{1}{12} f^{(4)}(\xi) h^2$$

Backward three-point:

$$y^{(2)}(t) = \frac{y(t) - 2y(t-h) + y(t-2h)}{h^2} + y^{(3)}(\tau) h$$

Backward four-point:

$$y^{(2)}(t) = \frac{2y(t) - 5y(t-h) + 4y(t-2h) - y(t-3h)}{h^2} + \frac{11}{12} y^{(4)}(t) h^2 + \mathrm{O}\left(h^3\right)$$

Integrals:
Two-point (trapezoidal rule):

$$\int_{x_{k-1}}^{x_k} f(x)\,\mathrm{d}x = \left(\frac{1}{2}f(x_{k-1}) + \frac{1}{2}f(x_k)\right)h - \frac{1}{12}f^{(2)}(\xi)\,h^3$$

Centered four-point:

$$\int_{x_{k-1}}^{x_k} f(x)\,\mathrm{d}x = \left(-\frac{1}{24}f(x_{k-2}) + \frac{13}{24}f(x_{k-1}) + \frac{13}{24}f(x_k) - \frac{1}{24}f(x_{k+1})\right)h - \frac{11}{720}f^{(4)}(t_k)\,h^5 + \mathrm{O}\left(h^6\right)$$

Simpson's rule:

$$\int_{x_{k-1}}^{x_{k+1}} f(x)\,\mathrm{d}x = \left(\frac{1}{6}f(x_{k-1}) + \frac{4}{6}f(x_k) + \frac{1}{6}f(x_{k+1})\right)(2h) - \frac{1}{90}f^{(4)}(\xi)\,h^5$$

Backward three-point (half Simpson's rule):

$$\int_{t_{k-1}}^{t_k} y(t)\,\mathrm{d}x = \left(\frac{5}{12}y(t_k) + \frac{8}{12}y(t_{k-1}) - \frac{1}{12}y(t_{k-2})\right)h - \frac{1}{24}y^{(3)}(t_k)\,h^4 + \mathrm{O}\left(h^5\right)$$

Backward four-point:

$$\int_{t_{k-1}}^{t_k} y(t)\,\mathrm{d}x = \left(\frac{9}{24}y(t_k) + \frac{19}{24}y(t_{k-1}) - \frac{5}{24}y(t_{k-2}) + \frac{1}{24}y(t_{k-3})\right)h + \frac{19}{720}y^{(4)}(t_k)\,h^5 + \mathrm{O}\left(h^6\right)$$

As Simpson's rule spans two time intervals, it is less useful, but it is interesting with its comparison with the trapezoidal rule applied twice versus one application of Simpson's rule. It also corresponds with the 4th-order Runge Kutta method.
Any integral formula can be applied repeatedly on the interval $[a, b]$ by dividing the interval into n equally-spaced sub-intervals of width $h = \frac{b-a}{n}$ and then setting $x_k = a + kh$ or $t_k = a + kh$.

Least squares: In general, if we want to find the best approximation of an n-dimensional vector $\mathbf{y}$ by a linear combination of m vectors $\mathbf{v}_1, \ldots, \mathbf{v}_m$ (where $m < n$), we create the matrix $V = (\mathbf{v}_1 \cdots \mathbf{v}_m)$ and solve $V^\top V\boldsymbol{\alpha} = V^\top \mathbf{y}$. More specific to this course, having shifted and scaled the n most recent t-values onto $0, -1, -2, \ldots, -n+1$, with y values $\mathbf{y} = (y_k, y_{k-1}, y_{0-2}, \ldots, y_{k-n+1})$, we solve $V^\top V\boldsymbol{\alpha} = V^\top \mathbf{y}$ for the coefficients of the least-squares best-fitting polynomial, generally of degree one (linear or $\alpha_1 t + \alpha_0$) or two (quadratic or $\alpha_2 t^2 + \alpha_1 t + \alpha_0$). We can find the $2\times n$ or $3\times n$ matrix to calculate $\boldsymbol{\alpha} = \left(V^\top V\right)^{-1} V^{\mathrm{T}} \mathbf{y}$.

Value being estimated	Linear estimation
$y(t_k)$	α_0
$y(t_k + h)$	$\alpha_0 + \alpha_1$
$y^{(1)}(t_k)$	α_1/h
$\int_{t_k-h}^{t_k} y(\tau)\mathrm{d}\tau$	$(\alpha_0 - \alpha_1/2)h$
$\int_{t_k}^{t_k+h} y(\tau)\mathrm{d}\tau$	$(\alpha_0 + \alpha_1/2)h$

Value being estimated	Quadratic estimation
$y(t_k)$	α_0
$y(t_k + h)$	$\alpha_0 + \alpha_1 + \alpha_2$
$y^{(1)}(t_k)$	α_1/h
$y^{(2)}(t_k)$	$2\alpha_2/h^2$
$\int_{t_k-h}^{t_k} y(\tau)\mathrm{d}\tau$	$(\alpha_0 - \alpha_1/2 + \alpha_2/3)h$
$\int_{t_k}^{t_k+h} y(\tau)\mathrm{d}\tau$	$(\alpha_0 + \alpha_1/2 + \alpha_2/3)h$

Root finding:

- Bisection: Let $m_k \leftarrow \frac{a_k+b_k}{2}$ and update that endpoint that has the value of the function have the same sign as $f(m_k)$.
- Newton's method: $x_{k+1} \leftarrow x_k - \frac{f(x_k)}{f^{(1)}(x_k)}$.
- Secant method: $x_{k+1} \leftarrow x_k - \frac{f(x_k)}{\frac{f(x_k)-f(x_{k-1})}{x_k-x_{k-1}}} = x_k - \frac{f(x_k)(x_k-x_{k-1})}{f(x_k)-f(x_{k-1})}$.
- Inverse quadratic interpolation: Find the constant coefficient of the polynomial interpolating (y_{k-2}, x_{k-2}), (y_{k-1}, x_{k-1}) and (y_k, x_k).
- Newton's method in n dimensions: Given the approximation $\mathbf{u}_k$ to $\mathbf{f}(\mathbf{u}) = \mathbf{0}$, solve $J(\mathbf{f})(\mathbf{u}_k)\Delta\mathbf{u}_k = -\mathbf{f}(\mathbf{u}_k)$ and then let $\mathbf{u}_{k+1} \leftarrow \mathbf{u}_k + \Delta\mathbf{u}_k$.