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

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

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 $\mathbf{u}_{k+1} \leftarrow \mathbf{u}_k + \Delta\mathbf{u}_k$.

Initial-value problems (IVPs): Given the ordinary-differential equation (ODE) and initial value

$$y^{(1)}(t) = f(t, y(t)) \text{ and } y(t_0) = y_0,$$

we will approximate $y_{k+1} \approx y(t_{k+1})$.

Given the system of ODEs and initial values

$$\mathbf{y}^{(1)}(t) = \mathbf{f}(t, \mathbf{y}(t)) \text{ and } \mathbf{y}(t_0) = \mathbf{y}_0$$

we will approximate $\mathbf{y}_{k+1} \approx \mathbf{y}(t_{k+1})$.

Euler's method:

$$y_{k+1} \leftarrow y_k + hf(t_k, y_k) \quad \mathbf{y}_{k+1} \leftarrow \mathbf{y}_k + h\mathbf{f}(t_k, \mathbf{y}_k)$$

Heun's method:

$$\begin{array}{ll} s_0 \leftarrow f(t_k, y_k) & \mathbf{s}_0 \leftarrow \mathbf{f}(t_k, y_k) \\ s_1 \leftarrow f(t_k + h, y_k + hs_0) & \mathbf{s}_1 \leftarrow \mathbf{f}(t_k + h, \mathbf{y}_k + h\mathbf{s}_0) \\ y_{k+1} \leftarrow y_k + h\frac{s_0+s_1}{2} & \mathbf{y}_{k+1} \leftarrow \mathbf{y}_k + h\frac{\mathbf{s}_0+\mathbf{s}_1}{2} \end{array}$$

The 4^{th}-order Runge-Kutta method:

$$\begin{array}{ll} s_0 \leftarrow f(t_k, y_k) & \mathbf{s}_0 \leftarrow \mathbf{f}(t_k, y_k) \\ s_1 \leftarrow f\left(t_k + \frac{1}{2}h, y_k + \frac{1}{2}hs_0\right) & \mathbf{s}_1 \leftarrow \mathbf{f}\left(t_k + \frac{1}{2}h, \mathbf{y}_k + \frac{1}{2}h\mathbf{s}_0\right) \\ s_2 \leftarrow f\left(t_k + \frac{1}{2}h, y_k + \frac{1}{2}hs_1\right) & \mathbf{s}_2 \leftarrow \mathbf{f}\left(t_k + \frac{1}{2}h, \mathbf{y}_k + \frac{1}{2}h\mathbf{s}_1\right) \\ s_3 \leftarrow f\left(t_k + h, y_k + hs_2\right) & \mathbf{s}_3 \leftarrow \mathbf{f}\left(t_k + h, \mathbf{y}_k + h\mathbf{s}_2\right) \\ y_{k+1} \leftarrow y_k + h\frac{s_0+2s_1+2s_2+s_3}{6} & \mathbf{y}_{k+1} \leftarrow \mathbf{y}_k + h\frac{\mathbf{s}_0+2\mathbf{s}_1+2\mathbf{s}_2+\mathbf{s}_3}{6} \end{array}$$

A single step of these three methods are $O(h^2)$, $O(h^3)$ and $O(h^5)$, respectively; however, multiple steps are one order less: $O(h)$, $O(h^2)$ and $O(h^4)$, respectively.

For Euler's method, the error of one step may be found from Taylor series:

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

where

$$y(t_{k+1}) = y_k + f(t_k, y_k)h + \frac{1}{2}y^2(\tau)h^2$$

assuming that y_k is exact.

The adaptive Euler-Heun method Given an ODE and an initial value together with a maximum absolute error per unit step ϵ_{abs}, start with an initial step size h and determine both minimum and maximum step sizes h_{min} and h_{max}, respectively. Also, let $k \leftarrow 0$.

1. If $h < h_{min}$, set $h \leftarrow h_{min}$, and if $h > h_{max}$, set $h \leftarrow h_{max}$.
2. Given y_k, calculate $s_0 \leftarrow f(t_k, y_k)$ and $s_1 \leftarrow f(t_k + h, y_k + hs_0)$.
3. Estimate $y(t_k + h)$ with
   - $y \leftarrow y_k + hs_0$ (the worse approximation), and
   - $z \leftarrow y_k + h\frac{s_0+s_1}{2}$ (the better approximation).
4. Let $a \leftarrow \frac{h\epsilon_{\text{abs}}}{2|y-z|}$, and
   - if $a \geq 1$ or $h = h_{min}$, let $t_{k+1} \leftarrow t_k + h$ and let $y_{k+1} \leftarrow z$ and then set $h \leftarrow 0.9ah$ and $k \leftarrow k + 1$, and we will continue with the next step;
   - otherwise $a < 1$ and we will try again.
5. If $0.9a \leq 0.5$, set $h \leftarrow 0.5h$ (don't shrink h by more than a factor of two),
   else if $0.9a \geq 2$, set $h \leftarrow 2h$ (don't grow h by more than a factor of two),
   else set $h \leftarrow 0.9ah$.

Given a system of ODEs and initial values with a similar set-up:

1. If $h < h_{min}$, set $h \leftarrow h_{min}$, and if $h > h_{max}$, set $h \leftarrow h_{max}$.
2. Given y_k, calculate $\mathbf{s}_0 \leftarrow \mathbf{f}(t_k, \mathbf{y}_k)$ and $\mathbf{s}_1 \leftarrow \mathbf{f}(t_k + h, \mathbf{y}_k + h\mathbf{s}_0)$.
3. Estimate $\mathbf{y}(t_k + h)$ with
   - $\mathbf{y} \leftarrow \mathbf{y}_k + h\mathbf{s}_0$ (the worse approximation), and
   - $\mathbf{z} \leftarrow \mathbf{y}_k + h\frac{\mathbf{s}_0+\mathbf{s}_1}{2}$ (the better approximation).

4. Let $a \to \frac{h\epsilon_{\text{abs}}}{2\|\mathbf{y}-\mathbf{z}\|_2}$ where $\|\cdot\|_2$ is the 2-norm (or Euclidean norm), and
   - if $a \geq 1$ or $h = h_{min}$, let $t_{k+1} \leftarrow t_k + h$ and let $\mathbf{y}_{k+1} \leftarrow \mathbf{z}$ and then set $h \leftarrow 0.9ah$ and $k \leftarrow k+1$, and we will continue with the next step;
   - otherwise $a < 1$ and we will try again.
5. If $0.9a \leq 0.5$, set $h \leftarrow 0.5h$ (don't shrink h by more than a factor of two), else if $0.9a \geq 2$, set $h \leftarrow 2h$ (don't grow h by more than a factor of two), else set $h \leftarrow 0.9ah$.

Boundary-value problems (BVPs) Given a 2nd-order ODE $u^{(1)}(x) = f(x, u(x), u^{(1)}(x))$ with two boundary conditions $u(a) = u_a$ and $u(b) = u_b$, we can approximate a solution to this BVP as follows. First, create the IVP $u^{(1)}(x) = f(x, u(x), u^{(1)}(x))$ with the two initial conditions $u(a) = u_a$ and $u^{(1)}(a) = s$ where s is an initial slope we can choose. Let us use any technique you wish (preferably Dormand Prince), and let $u_s(x)$ be an approximation to the solution of this IVP with the initial slope s. Then the approximation of $u(b)$ using the this technique and initial slope is $u_s(b)$. Proceed as follows:

1. Let $s_0 = \frac{u_b - u_a}{b-a}$ and find $u_{s_0}(b)$. If $u_{s_0}(b) = u_b$, we are done, otherwise, continue.
2. Let $s_1 = \frac{2u_b - u_{s_0}(b) - u_a}{b-a}$ and find $u_{s_1}(b)$. If $u_{s_1}(b) = u_b$, we are done, otherwise, continue.
3. Define the function $f(s) = u_b - u_s(b)$, which is a function of a single variable s, and use s_0 and s_1 as the first two approximations for the secant method.

Linear boundary-value problems (BVPs) Given a 2nd-order linear ODE (LODE) $c_2(x)u^{(2)}(x) + c_1(x)u^{(1)}(x) + c_0(x)u(x) = g(x)$ with two boundary conditions $u(a) = u_a$ and $u(b) = u_b$, we convert the LODE into a linear finite-difference equation and let $x_k = a + hk$ for $h = \frac{b-a}{n}$, so if we define for each $k = 1, ..., n-1$ the three values $p_k = 2c_2(x_k) - hc_1(x_k)$, $q_k = -4c_2(x_k) + 2h^2c_0(x_k)$ and $r_k = 2c_2(x_k) + hc_1(x_k)$, we have

$$p_k u_{k-1} + q_k u_k + r_k u_{k+1} = 2h^2 g(x_k)$$

For Dirichlet boundary conditions, the corresponding linear equations are:

$$q_1 u_1 + r_1 u_2 = 2h^2 g(x_1) - p_1 u_a$$

$$p_{n-1} u_{n-2} + q_{n-1} u_{n-1} = 2h^2 g(x_{n-1}) - r_{n-1} u_b$$

For Neumann boundary conditions, the corresponding linear equations are:

$$\left(q_1 + \frac{4}{3}p_1\right) u_1 + \left(r_1 - \frac{1}{3}p_1\right) u_2 = 2h^2 g(x_1) + \frac{2}{3}hp_1 u_a^{(1)}$$

$$\left(p_{n-1} - \frac{1}{3}r_{n-1}\right) u_{n-2} + \left(q_{n-1} + \frac{4}{3}r_{n-1}\right) u_{n-1} = 2h^2 g(x_{n-1}) - \frac{2}{3}hr_{n-1}u_b^{(1)}$$

after which

$$u_0 \leftarrow -\frac{2}{3}hu_a^{(1)} + \frac{4}{3}u_1 - \frac{1}{3}u_2$$

$$u_n \leftarrow \frac{2}{3}hu_b^{(1)} + \frac{4}{3}u_{n-1} - \frac{1}{3}u_{n-2}$$

and this simplifies for insulated boundary conditions where $u_a^{(1)} = 0$ or $u_b^{(1)} = 0$.

Heat equation: For the heat equation $\frac{\partial u}{\partial t} = \alpha \nabla^2 u$, we have $u_{init}(x)$, $u_a(t)$ and $u_b(t)$, and we convert the partial-differential equation (PDE) into a finite-difference equation with $x_k \leftarrow a + kh$ and $t_\ell \leftarrow t_0 + \ell\Delta t$ so that $u_{k,0} \leftarrow u_{init}(x_k)$,

$$u_{k,\ell+1} \leftarrow u_{k,\ell} + \frac{\alpha\Delta t}{h^2}\left(u_{k-1,\ell} - 2u_{k,\ell} + u_{k+1,\ell}\right)$$

where $u_{0,\ell} \leftarrow u_a(t_\ell)$ and $u_{n,\ell} \leftarrow u_b(t_\ell)$.

Laplace's equation: On a grid $x_j = a_x + jh$ and $y_k \leftarrow a_y + kh$, we convert the PDE into the linear finite difference equation:

$$4u_{j,k} - u_{j-1,k} - u_{j+1,k} - u_{j,k-1} - u_{j,k+1} = 0$$

replacing any points on the boundary with their boundary value. Note that each value is the average of the surrounding points. If there are insulated boundaries, each entry must be the average of all the points around it that are not insulated boundary points.

Newton's: Use Newton's method on $f^{(1)}(x)$.
Golden ratio search: To minimize, given an interval $[a, b]$ with a minimum on that interval, let $m_1 \leftarrow b-(b-a)/\phi$ and $m_2 \leftarrow a+(b-a)/\phi$ and update $a \leftarrow m_1$ if $f(m_2) < f(m_1)$ and update $b \leftarrow m_2$ otherwise.
Successive parabolic interpolation: Formula not necessary: given three points, find the interpolating polynomial $ax^2 + bx + c$ and let the next point be $-\frac{b}{2a}$.
Newton's: Use Newton's method on $\vec{\nabla} f$.
Gradient descent: Calculate or approximate $\vec{\nabla} f(\mathbf{x}_k)$, and then use a one-dimensional algorithm to find a minimum of $f(\mathbf{x}_k - s\vec{\nabla} f(\mathbf{x}_k))$ and when we find the value s_k that gives us the minimum, set $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k - s_k \vec{\nabla} f(\mathbf{x}_k)$.