Click here to view the PDF version.

```
begin
    using Plots
    using PlutoUI
    using PlutoTeachingTools
    using LaTeXStrings
    using QuadGK
    using Printf
    using HypertextLiteral: @htl, @htl_str
end
```

Table of Contents

Numerical integration

Integration is an important operation in engineering and the physical sciences, but evaluating integrals analytically can be quite challenging. For such cases employing *numerical* integration techniques is a good alternative, which sometimes turns out to be no less accurate.

Let's start with an easy analytic case. The integral $\int_0^1 e^x dx$ can be computed rather elegantly by analytical means, since the anti-derivative of e^x is again e^x. Therefore

$$\int_0^1 e^x dx = [e^x]_0^1 = e^1 - 1 \simeq 1.718281828459045.$$

```
exact = 1.718281828459045
```

```
1 exact = exp(1) - 1
```

To perform the integration numerically, we can employ the Julia package `QuadGK`, which implements an all-purpose numerical integration routine:

```
1.7182818284590453
```

```
1 let
2     Q, error_estimate = quadgk(x -> exp(x), 0, 1)
3     Q
4 end
```

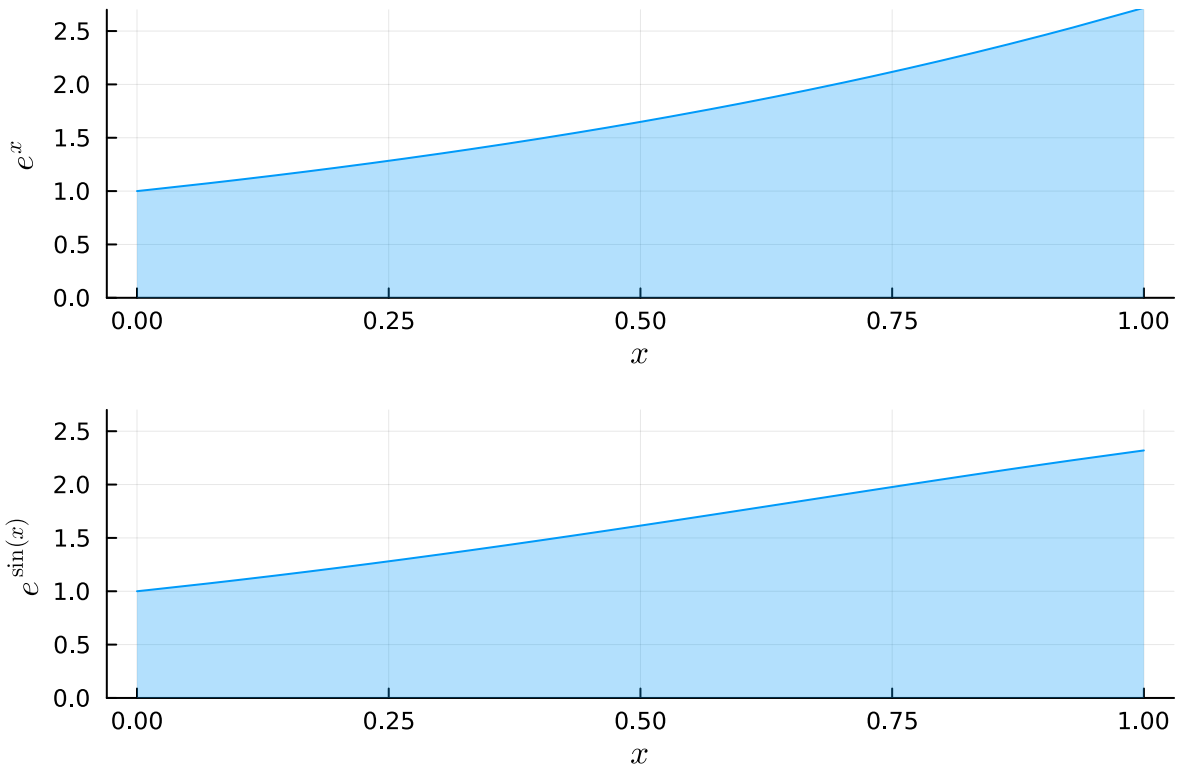
As we observe, the difference to the analytical result is tiny and on the order of the floating-point precision.

However, the numerical approach is much more flexible. For example $e^{\sin(x)}$ has no useful anti-derivative. Still the numerical approach works just as well to compute $\int_0^1 e^{\sin(x)} dx$:

```
1.6318696084180515
```

```
1 let
2     Q, error_estimate = quadgk(x -> exp(sin(x)), 0, 1)
3     Q
4 end
```

What is remarkable when looking at the graphs of these two functions is that they are very similar ... and for one case the area under the curve boils down to simple basic calculus and the other is impenetrable analytically. However, from a numerical standpoint they are basically the same problem.




```
begin
    p = plot(exp, 0, 1, fill=0, fillalpha=0.3, xlabel=L"x", ylabel=L"e^x", ylims=
    (0, 2.7), label="")
    q = plot(x -> exp(sin(x)), 0, 1, fill=0, fillalpha=0.3, xlabel=L"x",
    ylabel=L"e^{\sin(x)}", ylims=(0, 2.7), label="")

    plot(p, q, layout=(2, 1))
end
```

Trapezoidal rule

The task of **numerical integration** is to **approximate an integral** $\int_a^b f(x)\,dx$. We want to achieve this by **sampling the function** at n carefully selected points t_i, $i = 0, \ldots n$, followed by taking linear combinations of the results. We thus work towards approximations of the form

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{m} \alpha_i f(t_i).$$

In general the t_i can be distributed arbitrarily in the interval $[a, b]$. However, for simplicity we will assume **equally spaced nodes** t_i using the definition

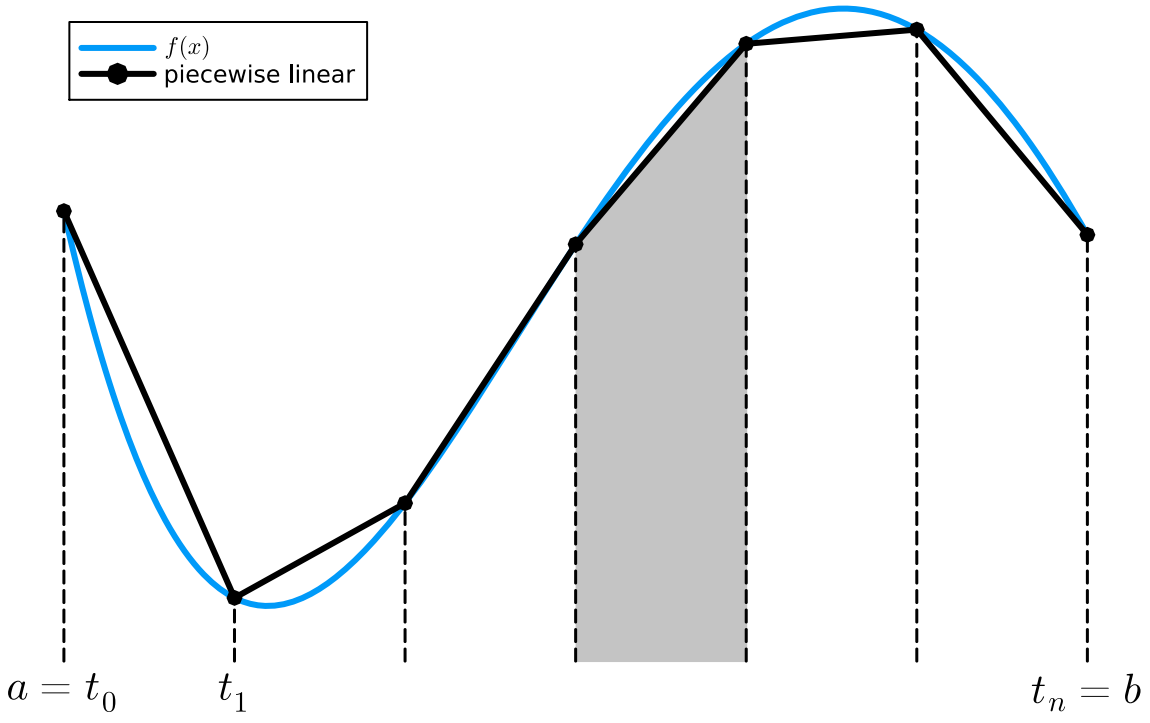
$$t_i = a + i\,h, \qquad h = \frac{b-a}{n} \qquad \text{where } i = 0, 1, \ldots, n. \tag{1}$$

A first idea goes back to **polynomial interpolation**: As **polynomials are easy to integrate analytically**, we could just **fit a n-th degree polynomial** through our $n+1$ nodes $t_0, t_1, \ldots, t_n$ and **then integrate that** instead of f itself.

Since the integration of the polynomial is essentially exact, the error of such a scheme is **dominated by the error of the polynomial interpolation**. Recall the chapter on Interpolation, where we noted polynomials through equispaced nodes to become numerically unstable and possibly inaccurate for large n due to Runge's phaenomenon.

Therefore we will pursue **piecewise linear polynomial interpolation** instead of fitting an n-th degree polynomial.

The following graphics illustrates the idea:



To approximate integral $I = \int_a^b f(x)dx$, that is the area under the blue curve, we evaluate the function f at $n+1$ equispaced nodes leading to data points $(t_i, f(t_i))$. From these we construct a piecewise linear polynomial interpolation $p_{1,h}$ (black line). Integrating $p_{1,h}$ on $[t_0, t_n]$ then yields an approximation to I:

$$I = \int_a^b f(x)\,dx \approx \int_a^b p_{1,h}(x)\,dx = \int_a^b \sum_{i=0}^{n} f(t_i) H_i(x)\,dx = \sum_{i=0}^{n} f(t_i) \int_a^b H_i(x)\,dx.$$

As a reminder the H_i are the hat functions defined as

$$H_i(x) = \begin{cases} \frac{x - x_{i-1}}{x_i - x_{i-1}} & \text{if } i > 1 \text{ and } x \in [x_{i-1}, x_i] \\ \frac{x_{i+1} - x}{x_{i+1} - x_i} & \text{if } i \leq n \text{ and } x \in [x_i, x_{i+1}] \\ 0 & \text{otherwise} \end{cases}$$

Recall that $t_{i+1} - t_i = h$, that is to say the distance between two nodes is always h. With this and using the formulas for computing the areas of triangles the integrals of the hat functions can be evaluated as

$$\int_a^b H_i(x)\,dx = \begin{cases} h & \text{if } i = 1, \ldots, n-1 \\ \frac{h}{2} & \text{if } i = 0 \text{ or } i = n \end{cases}$$

leading to the final formula

$$\begin{aligned} I = \int_a^b f(x)\,dx \approx T_n(f) &:= \frac{h}{2} f(t_0) + h f(t_1) + \cdots + h f(t_{n-1}) + \frac{h}{2} f(t_n) \\ &= \frac{h}{2} f(t_0) + \sum_{i=1}^{n-1} h\, f(t_i) + \frac{h}{2} f(t_n). \end{aligned}$$

This formula is called the **trapezoidal rule**. This name stems from a more geometric way of constructing the trapezoidal formula: as can be captured in the visualisation above, one can think of the formula as summing the areas of the trapezoids defined by the quadrature nodes (see shaded grey).

An implementation of the trapezoidal rule is:

```
trapezoid (generic function with 1 method)
```

```
function trapezoid(f, a, b, n)
    # f:  Function
    # [a, b]: Interval to integrate over
    # n:  Number of pieces to break the interval into
    h = (b - a) / n
    t = range(a, b, length=n+1)
    y = [f(tₙ) for tₙ in t]
    integral = h * (0.5y[1] + sum(y[2:n]) + 0.5y[n+1])
    (; integral, h)
end
```

Recall that in Theorem 4 of chapter 07 (Interpolation) we found that the piecewise polynomial interpolation shows quadratic convergence

$$\|f - p_{1,h}\|_\infty \leq \alpha h^2 \|f''\|_\infty,$$

where $\alpha > 0$. With this in mind we can bound the error of the trapezoidal rule as

$$\begin{aligned} I - T_n(f) &= \int_a^b f(x)\,dx - \int_a^b p_{1,h}(x)\,dx \\ &= \int_a^b [f(x) - p_{1,h}(x)]\,dx \\ &\leq \|f - p_{1,h}\|_\infty \int_a^b dx \\ &\leq \underbrace{(b-a)\,\alpha\,\|f''\|_\infty}_{=\tilde{\alpha}} h^2 = \tilde{\alpha}\,h^2 = O(h^2). \end{aligned}$$

We thus expect **quadratic convergence** with h.

Let us confirm this numerically on the simple integral from earlier, i.e.

```
f (generic function with 1 method)
```

```
1 f(x) = exp(x)
```

and

$$I = \int_0^1 e^x\,dx = e - 1$$

which is approximately 1.7182818.

We consider a sequence of results where we double the number of integration points:

```
ns = [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048]
```

```
1 ns = [2^i for i in 1:11]
```

The corresponding approximate integrals using the trapezoidal formula $T_n(f)$ and quadrature node distances are:

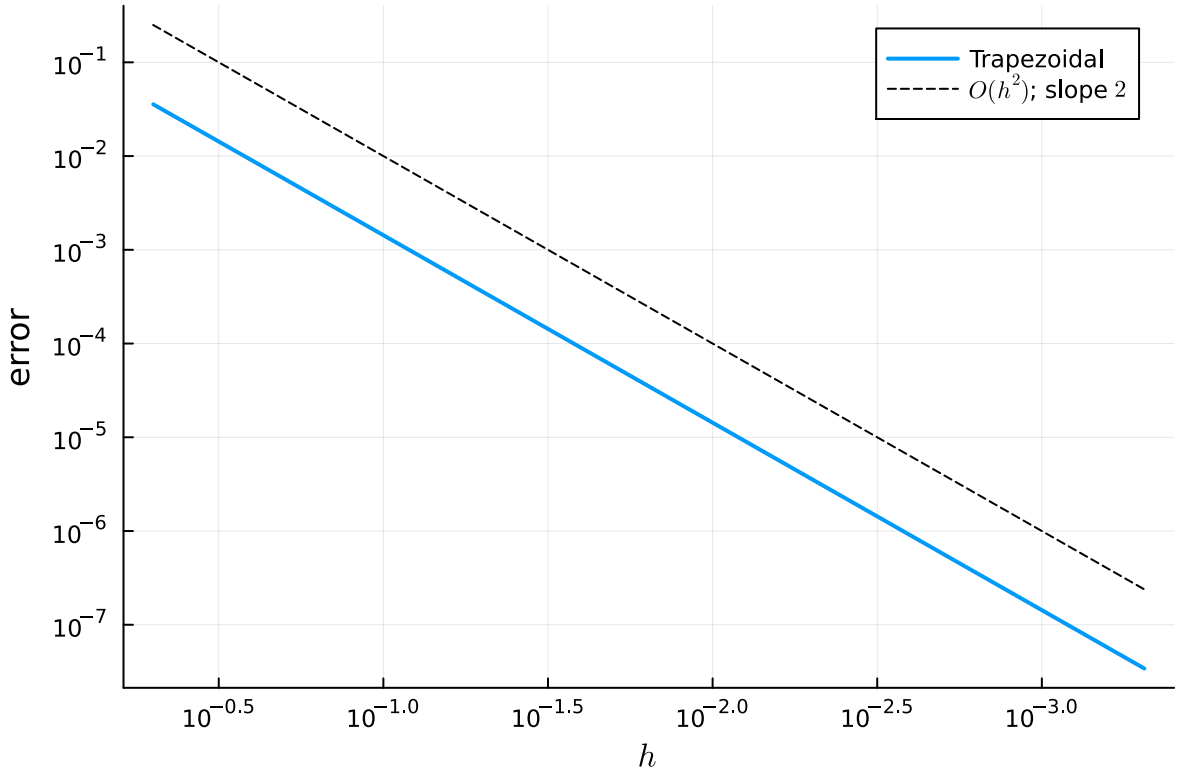
```julia
begin
    Tfs    = Float64[]
    hs     = Float64[]
    errors = Float64[]
    for n in ns
        res = trapezoid(f, 0, 1, n)
        Tf    = res.integral  # Trapezoidal approximation
        h     = res.h         # Quadrature node distances
        error = abs(Tf - exact)
        @printf "n =%4d   T(f) = %.10f   error = %.10f\n" n Tf error

        push!(Tfs, Tf)
        push!(hs, h)
        push!(errors, error)
    end
end
```

```
n =   2   T(f) = 1.7539310925   error = 0.0356492640
n =   4   T(f) = 1.7272219046   error = 0.0089400761
n =   8   T(f) = 1.7205185922   error = 0.0022367637
n =  16   T(f) = 1.7188411286   error = 0.0005593001
n =  32   T(f) = 1.7184216603   error = 0.0001398319
n =  64   T(f) = 1.7183167869   error = 0.0000349584
n = 128   T(f) = 1.7182905681   error = 0.0000087396
n = 256   T(f) = 1.7182840134   error = 0.0000021849
n = 512   T(f) = 1.7182823747   error = 0.0000005462
n =1024   T(f) = 1.7182819650   error = 0.0000001366
n =2048   T(f) = 1.7182818626   error = 0.0000000341
```

We notice that the error decreases roughly by a factor 4 when we double the number of quadrature nodes (i.e. half the distance h between the quadrature nodes). This again confirms the idea of a second-order convergence.

This becomes even clearer if we plot the error versus h in a log-log plot along with a line of slope 2:



The trapezoidal rule is just one representative of the wide class of numerical integration formulas. A general definition is:

Definition: Numerical integration formula

A **numerical integration formula** for the $N+1$ equispaced **quadrature nodes** t_i, $i = 0, \dots, N$ is the set of **weights** $w_0, w_1, \dots w_N$, such that for an integrand $f : [a, b] \to \mathbb{R}$

$$\int_a^b f(x)dx \approx Q_a^b(f) = \frac{b-a}{N} \sum_{i=0}^{N} w_i f(t_i). \tag{2}$$

The weights w_i are independent of f.

An older and still frequently used name for numerical integration is **quadrature**.

By comparing with our derivation above, we realise:

Definition: Trapezoid formula

The trapezoid formula is the numerical integration of the form (2) with $n+1$ nodes (i.e. $N = n$) and weights

$$w_i = \begin{cases} 1 & \text{if } 0 < i < n \\ \frac{1}{2} & \text{if } i = 0 \text{ or } i = n \end{cases}$$

Simpson's rule

⚠ TODO ⚠

Show an illustrative drawing as well

```
TODO("Show an illustrative drawing as well")
```

Considering the construction of the trapezoidal rule we may easily wonder: why stop at using only linear polynomials to proximate f within each interval $[t_i, t_{i+1}]$?

Indeed, Simpson's formula takes the idea one step further and constructs a quadratic polynomial within each interval $[t_i, t_{i+1}]$ by evaluate f on t_i, t_{i+1} as well as the midpoint $m_i = \frac{t_i + t_{i+1}}{2}$. This overall leads to a **piecewise quadratic interpolant** $p_{2,h}$ of the integrand f on $[a, b]$, which we again integrate exactly. This is a little harder to compute and will be done as an exercise. The resulting formula is Simpson's formula

$$\begin{aligned}\int_a^b f(x)\,dx \approx S_{2n}(f) &= \int_a^b p_{2,h}(x)\,dx \\ &= \sum_{i=1}^{n} \frac{h}{6}\left(f(t_{i-1}) + 4f(m_{i-1}) + f(t_i)\right) \\ &= \frac{h}{6}f(t_0) + \frac{h}{3}\sum_{i=1}^{n-1} f(t_i) + \frac{2h}{3}\sum_{i=0}^{n-1} f(m_i) + \frac{h}{6}f(t_n).\end{aligned} \tag{3}$$

While a little harder to see, this formula can also be brought into the form of (2): it employs $2n + 1$ **equispaced nodes** — namely the collection of both the t_i for $i = 0, \ldots, n$ *and* the m_i for $i = 0, \ldots n - 1$. Therefore $N = 2n$ in (2) leading to a **nodal distance** of $\frac{b-a}{2n} = \frac{h}{2}$, where we used that $h = t_{i+1} - t_i = \frac{b-a}{n}$

Exercise

Derive Simpson's rule, i.e. show that

$$\int_a^b p_{2,h}(x)\,dx = \sum_{i=1}^{n} \frac{h}{6}\left(f(t_{i-1}) + 4f(m_{i-1}) + f(t_i)\right)$$

A Julia implementation of Simpson's rule is given below:

```
simpson (generic function with 1 method)
```

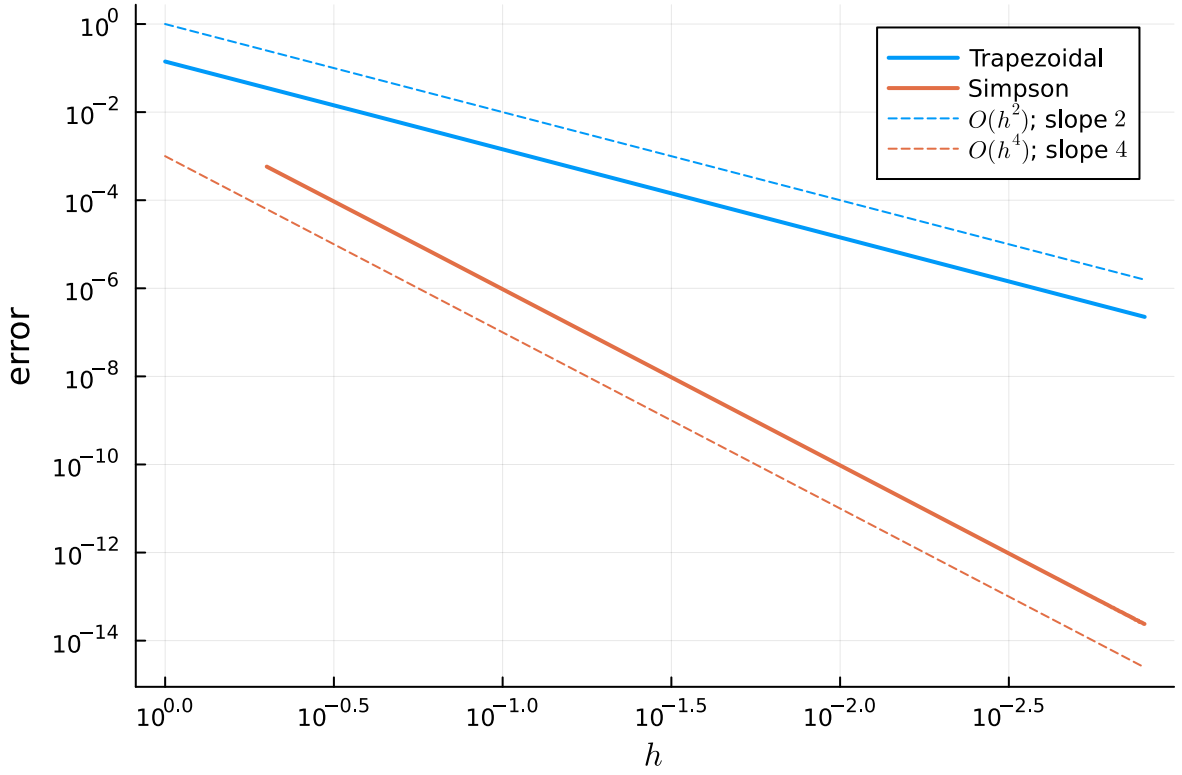
```julia
function simpson(f, a, b, n)
    # f:  Function
    # [a, b]: Interval to integrate over
    # n:  Number of pieces to break the interval into
    h = (b - a) / n
    t = range(a, b, length=n+1)        # Subinterval boundaries
    m = range(a+h/2, b-h/2, length=n)  # Subinterval midpoints

    # Evaluate f and compute approximation
    ft = f.(t)
    fm = f.(m)
    integral = h * (ft[1]/6 + sum(ft[2:n])/3 + 2sum(fm)/3 + ft[n+1]/6)

    (; integral, h=h/2)  # Note h/2 since the actual nodal distance is half of h
end
```

Comparing the error of the trapezoidal and Simpson's quadratures against the exact integral of $\int e^x dx$ we obtain a much faster convergence for Simpson's rule, numerically looking like an $O(h^4)$ convergence.

p_convergence =




```
p_convergence = let
    # Trapezoidal rule
    Tfs     = [trapezoid(f, 0, 1, n).integral for n in 1:3:800]
    hs_trap = [trapezoid(f, 0, 1, n).h        for n in 1:3:800]
    p = plot(hs_trap, abs.(Tfs .- exact);
             yaxis=:log, xflip=true, xaxis=:log,
             xlabel=L"h", ylabel="error", label="Trapezoidal", lw=2)

    # Simpson's rule
    Sfs      = [simpson(f, 0, 1, n).integral for n in 1:2:400]
    hs_simps = [simpson(f, 0, 1, n).h        for n in 1:2:400]
    plot!(p, hs_simps, abs.(Sfs .- exact), label="Simpson", lw=2)

    # Guiding lines
    plot!(p, hs_trap, hs_trap.^2, ls=:dash, c=1, label=L"$O(h^2)$; slope $2$")
    plot!(p, hs_trap, hs_trap.^4/1000, ls=:dash, c=2, label=L"$O(h^4)$; slope $4$")

    xticks!(p, 10.0 .^ (0:-0.5:-3))
    yticks!(p, 10.0 .^ (0:-2:-14))
end
```

In line with the definition of algebraic convergence (and convergence order) for other approximation techniques we define the accuracy of a quadrature formula as:

Definition: Convergence order of numerical integration

A numerical integration formula $Q_a^b(f)$ of the form (2) with equally spaced quadrature notes of separation h is of **order** p if a constant $\alpha > 0$ indepentent of h (but possibly dependent on f) exists, such that

$$\left| \int_a^b f(x)\,dx - Q_a^b(f) \right| \leq \alpha\, h^p$$

as long as the function f is sufficiently regular.

We notice that our numerical investigation suggests:

Convergence order of common numerical integration techniques

- **Trapezoidal rule:** Convergence order $p = 2$
- **Simpon's rule:** Convergence order $p = 4$

Error analysis

In this lecture we only consider so-called **composite quadrature formulas**, i.e. formulas which satisfy

$$Q_a^b(f) = \sum_{i=0}^{N-1} Q_{t_i}^{t_{i+1}}(f)$$

where $Q_{t_i}^{t_{i+1}}(f)$ is the quadrature formula applied to the subinterval $[t_{i-1}, t_i]$, i.e. the application of the quadrature formula to $\int_{t_{i-1}}^{t_i} f(x)dx$ with no no subdivision of the integration interval. As a consequence we can decompose the error

$$\int_a^b f(x)\,dx - Q_a^b(f) = \sum_{i=0}^{N-1} \int_{t_i}^{t_{i+1}} f(x)\,dx - Q_{t_i}^{t_{i+1}}(f) \tag{4}$$

into error contributions from each of the intervals $[t_{i-1}, t_i]$.

Assume for simplicity that the function f is smooth and we can thus build a Taylor expansion

$$f(x) = \sum_{k=0}^{\infty} \frac{1}{k!} f^{(k)}(m_i)\,(x - m_i)^k$$

around the midpoint $m_i = \frac{t_{i+1}+t_i}{2}$ of the interval $[t_i, t_{i+1}]$. Based on this we can deduce a series for the exact integral:

$$\begin{aligned}
\int_{t_i}^{t_{i+1}} f(x)dx &= \int_{t_i}^{t_{i+1}} \left[\sum_{k=0}^{\infty} \frac{1}{k!} f^{(k)}(m_i)(x - m_i)^k\right] dx \\
&= \sum_{k=0}^{\infty} \frac{1}{k!} f^{(k)}(m_i) \int_{t_i}^{t_{i+1}} (x - m_i)^k dx \\
&= \sum_{k=0}^{\infty} \frac{1}{k!} f^{(k)}(m_i) \int_{t_i}^{t_{i+1}} q_k(x)\,dx,
\end{aligned}$$

where we defined $q_k(x) = (x - m_i)^k$.

If we apply a quadrature formula (2) to f we can follow through with similar steps. Here note that for $Q_{t_i}^{t_{i+1}}$ the number of nodes is exactly two — the beginning of the interval and the end, such that $N = 1$ and the term $\frac{b-a}{N} = \frac{t_{i+1}-t_i}{1} = h$. We obtain:

$$\begin{aligned}
Q_{t_i}^{t_{i+1}}(f) &= h \sum_{j=i}^{i+1} w_j f(t_j) \\
&= h \sum_{j=i}^{i+1} w_j \left[\sum_{k=0}^{\infty} \frac{1}{k!} f^{(k)}(m_j)\,(t_j - m_j)^k\right] \\
&= h \sum_{j=i}^{i+1} w_j \left[\sum_{k=0}^{\infty} \frac{1}{k!} f^{(k)}(m_j)\,q_k(t_j)\right] \qquad . \\
&= \sum_{k=0}^{\infty} \frac{1}{k!} f^{(k)}(m) \left[h \sum_{j=i}^{i+1} w_j\,q_k(t_j)\right] \\
&= \sum_{k=0}^{\infty} \frac{1}{k!} f^{(k)}(m) Q_{t_i}^{t_{i+1}}(q_k)
\end{aligned}$$

The difference between these expressions is exactly the error contribution from the interval $[t_i, t_{i+1}]$, namely

$$\int_{t_i}^{t_{i+1}} f(x)\,dx - Q_{t_i}^{t_{i+1}}(f) = \sum_{k=0}^{\infty} \frac{1}{k!} f^{(k)}(m) \left[\int_{t_i}^{t_{i+1}} q_k(x)\,dx - Q_{t_i}^{t_{i+1}}(q_k)\right]. \qquad (5)$$

The error of the integration formula can thus be completely understood by studying the error $\int_{t_i}^{t_{i+1}} q_k(x) - Q_{t_i}^{t_{i+1}}(q_k)$. Since q_k is just a polynomial of degree k we can thus understand the accuracy of quadrature formulas *for arbitrary functions* by studying the accuracy of quadrature formulas for *polynomials*, which is considerably simpler.

One property of quadrature formulas is their **degree of exactness**:

Definition: Degree of exactness

A numerical integration formula $Q_a^b(f) = \frac{b-a}{N} \sum_{i=1}^n w_i f(t_i)$ as given in (2) has a **degree of exactness** r if it integrates all monomials x^s with $0 \leq s \leq r$ (s integer) exactly, i.e. if

$$Q_a^b(x^s) = \int_a^b x^s \, dx \qquad \forall s \in \mathbb{N} \text{ with } 0 \leq s \leq r, \tag{6}$$

but not for $s = r+1$.

Note that the polynomial

$$q_k(x) = (x - m_i)^k = x^k + \binom{k}{1} x^{k-1} m_i + \binom{k}{2} x^{k-2} m_i^2 + \cdots + \binom{k}{k-1} x\, m_i^{k-1} + m_i^k$$

only features monomials x^s with $0 \leq s \leq k$. Therefore a formula with degree of exactness r will have $\int_{t_i}^{t_{i+1}} q_k(x)\,dx - Q_{t_i}^{t_{i+1}}(q_k) = 0$ for $k \leq r$. In (5) the first non-zero error term is thus

$$\left| \int_{t_i}^{t_{i+1}} q_{r+1}(x)\,dx - Q_{t_i}^{t_{i+1}}(q_{r+1}) \right| \overset{(*)}{=} \left| \int_{t_i}^{t_{i+1}} x^{r+1}\,dx - Q_{t_i}^{t_{i+1}}(x^{r+1}) \right| \overset{(\S)}{\leq} \widetilde{\alpha}_i h^{r+2}$$

where in $(*)$ all powers in x less than $r+1$ drop again because of Q's degree of exactness and in $(\S)$ we skipped a few non-trivial steps, which are optional and will be presented below. This is also the leading-order error term, such that

$$\left| \int_{t_i}^{t_{i+1}} f(x)\,dx - Q_{t_i}^{t_{i+1}}(f) \right| \leq \widetilde{\alpha}_i h^{r+2}$$

The error in each of the the N subintervals thus converges with $(r+2)$-th order, such that combining with (4) and using the triangle inequality we obtain the total error as

$$\begin{aligned}\left|\int_a^b f(x)\,dx - Q_a^b(f)\right| &\leq \sum_{i=0}^{N-1}\left|\int_{t_i}^{t_{i+1}} f(x)\,dx - Q_{t_i}^{t_{i+1}}(f)\right| \\ &\leq h^{r+2}\underbrace{\sum_{i=0}^{N-1}\widetilde{\alpha}_i}_{N \text{ terms}} \\ &\leq h^{r+1}\frac{b-a}{N} N \max_i \widetilde{\alpha}_i \\ &= C\,h^{r+1}\end{aligned}$$

where $\alpha = (b-a)\,\max_i \widetilde{\alpha}_i$.

We notice:

Theorem 1 (Convergence order, simple version)

A numerical integration formula $Q_a^b(f) = \frac{b-a}{N}\sum_{i=1}^{n} w_i f(t_i)$ as given in (2) with **degree of exactness r is of order $r+1$** as long as the function f is at least $r+1$ times differentiable.

▸ **Optional: More details on Theorem 1**

In short by **determining the degree of exactness** of a quadrature formula we thus **automatically obtain its convergence order**.

In fact in agreement with our expected convergence orders (2 and 4) one can show that:

Degree of exactness of common numerical integration techniques

- **Trapezoidal rule:** Degree of exactness $r = 1$
- **Simpson's rule:** Degree of exactness $r = 3$

We will show this now explicitly for the case of Simpson's rule.

Example: Simpson's formula has degree of exactness r=3.

We will confirm that Simpson's formula has degree of exactness $r = 3$.

Recall that the idea of Simpson's formula was to split the integral into n equally sized subintervals $[t_{i-1}, t_i]$ as

$$\int_a^b f(x)\,dx = \sum_{i=1}^{n} \int_{t_{i-1}}^{t_i} f(x)\,dx$$

and then on such a subinterval approximate

$$\int_{t_{i-1}}^{t_i} f(x)\,dx \approx \frac{h}{6}\big(f(t_{i-1}) + 4f(m_{i-1}) + f(t_i)\big). \tag{7}$$

If the approximation in (7) is an equality on all subintervals for the monomials with degree ≤ 3, then Simpson's formula is an equality for these monomials on the full interval $[a, b]$ as well. Inserting $f(x) = x^s$ into (7) we notice that our task is to show

$$\int_{t_{i-1}}^{t_i} x^s\,dx = \frac{h}{6}\big(t_{i-1}^s + 4m_{i-1}^s + t_i^s\big) \qquad \text{for } 0 \leq s \leq 3,$$

but that this does not hold for $s = 4$.

An additional trick we can employ to simplify our calculations is to assume that the integration interval is $[t_{i-1}, t_i] = [-1, 1]$ with a length of $h = 2$, $t_{i-1} = -1$, $t_i = 1$, $m_{i-1} = 0$. While perhaps suprising this does not actually change the generality of our result [1]. Inserting the values for h, t_{i-1}, t_i and m_i into the above expression we need to show that $0 \leq s \leq 3$

$$\int_{-1}^{1} x^s\,dx = \frac{2}{6}\big((-1)^s + 4\cdot 0^s + (+1)^s\big) = \frac{(-1)^s + 4\cdot 0^s + (+1)^s}{3},$$

but that this equality fails for $s \geq 4$.

We have

$$\begin{aligned}
s = 0: &\qquad \int_{-1}^{1} 1\,dx = 2 = \frac{(-1)^0 + 4\cdot 0^0 + 1^0}{3} &&\checkmark\\
s = 1: &\qquad \int_{-1}^{1} x\,dx = 0 = \frac{-1 + 0 + 1}{3} &&\checkmark\\
s = 2: &\qquad \int_{-1}^{1} x^2\,dx = \frac{2}{3} = \frac{(-1)^2 + 4\cdot 0^2 + 1^2}{3} &&\checkmark\\
s = 3: &\qquad \int_{-1}^{1} x^3\,dx = 0 = \frac{(-1)^3 + 4\cdot 0^3 + 1^3}{3} &&\checkmark
\end{aligned}$$

but in contrast

$$s = 4: \qquad \int_{-1}^{1} x^4 \, dx = \frac{2}{5} \neq \frac{(-1)^4 + 4 \cdot 0^4 + 1^4}{3} = \frac{2}{3}$$

Therefore Simpson's formula has **degree of exactness** $r = 3$.

[1]:

> The reason is that by an appropriate change of variables we can always rescale the integration coordinate, such that integration does run over $[-1, 1]$. Note that this in principle does generate some extra terms on top of x^s, but these are always of a lower power in x than x^s itself. So if we proceed as shown here, namely to consider integrating monomials with increasing powers of x, then all terms generated by this change of variables are already known to be integrated exactly at this stage.

Extrapolation techniques

In numerical integration the **computationally expensive step** is usually the **evaluation of the function** f at the points of the quadrature nodes. Generally higher-order numerical integration formulas are able to achieve higher accuracy with the same or less function evaluations. We will discuss one systematic technique to obtain higher-order quadrature formulas called **extrapolation**.

⚠ TODO ⚠

Make the general treatment optional and focus mostly directly applying this to the Trapezoidal rule.

```
TODO("Make the general treatment optional and focus mostly directly applying this to the Trapezoidal rule.")
```

⚠ TODO ⚠

What is confusing here is that before we did expansions (I - T_n(f) = stuff, but now we do I = T_n(f) + stuff.)

```
TODO("What is confusing here is that before we did expansions (I - T_n(f) = stuff, but now we do I = T_n(f) + stuff. )")
```

Suppose a quantity A_0 is approximated by an algorithm A_h with error expansion

$$A_0 = A_h + c_1 h + c_2 h^2 + c_3 h^3 + \cdots = A_h + c_1 h + c_2 h^2 + O(h^3). \tag{8}$$

Crucially it is not necessary to know the coefficients c_i, the only requirement is for them to be indepentent of h. If we now look at the better approximation $A_{h/2}$ with half the node spacing then

$$A_0 = A_{h/2} + \frac{c_1}{2} h + \frac{c_2}{4} h^2 + O(h^3) \tag{9}$$

Forming a **linear combination $2 \cdot (9) - (8)$ between both methods** we obtain

$$A_0 = \underbrace{2A_{h/2} - A_h}_{=\tilde{A}_{h/2}} - \frac{c_2}{2} h^2 + O(h^3)$$

which **defines a new approximation algorithm $\tilde{A}_{h/2} = 2A_{h/2} - A_h$**. Note that while A_h is a first-order algorithm, $\tilde{A}_{h/2}$ is a **second-order algorithm**.

This idea to cancel the leading-order error term by taking a linear combination of two formulas of simple and half node spacing is termed **Richardson extrapolation**. Importantly in contrast to simply reducing the node spacing this scheme is able to **increase the convergence order**.

Let us **apply this to the trapezoidal formula** for approximating the integral $I = \int_a^b f(x)\,dx$. We use $n+1$ quadrature nodes of equal separation $h = (b-a)/n$. As we have discussed above the trapezoidal formula is of order 2, so the leading-order error term is h^2. However, in this fortunate case one can even show (using the Euler–Maclaurin formula) that the odd powers of h are missing, i.e.

$$I = T_n(f) + c_2 h^2 + c_4 h^4 + O(h^6), \tag{10}$$

where

$$c_2 = -\frac{1}{12}\big(f'(b) - f'(a)\big) \qquad c_4 = \frac{1}{740}\big(f'''(b) - f'''(a)\big).$$

While it could happen that $f'(b) = f'(a)$ we will make the general assumption that $c_2 \neq 0$.

For convenience of notation we rewrite (10) in terms of the number of subintervals. Using $n = O(h^{-1})$ we obtain

$$I = T_n(f) + c_2\, n^{-2} + c_4\, n^{-4} + O(n^{-6}). \tag{11}$$

The Trapezoidal rule with twice the number of subintervals (half the spacing) similarly reads

$$I = T_{2n}(f) + \frac{1}{4}c_2\, n^{-2} + \frac{1}{16}c_4\, n^{-4} + O(n^{-6}). \tag{12}$$

Using an appropriate linear combination we can again cancel out the second-order term. Specifically, define

$$S_{2n}(f) = \frac{1}{3}\Big(4\,T_{2n}(f) - T_n(f)\Big) \tag{13}$$

and consider $\frac{4}{3}\cdot(12) - \frac{1}{3}(11)$:

$$I = S_{2n}(f) - \frac{1}{4}c_4\, n^{-4} + O(n^{-6}). \tag{14}$$

The quadrature formula $S_{2n}(f)$ is thus a 4-th order method. While not immediately clear (14) turns out to be identical to Simpson's formula (3), see the details at the end of this section. In other words we can view **Simpson's formula** as the numerical integration formula obtained either

- by integrating a piecewise quadratic approximation of the integrand or
- by applying Richardson extrapolation to the trapezoidal rule.

We formulate a general procedure for Richardson extrapolation in the context of numerical integration:

Observation: General procedure for Richardson extrapolation

Given an error expansion of a quadrature Q_n with n subintervals

$$I = Q_n + c_p n^{-p} + O(n^{-q}), \tag{15}$$

where p is integer and $q > p$. Then a a **quadrature of at least order q** can be found by the linear combination

$$\widetilde{Q}_{2n} = \frac{2^p Q_{2n} - Q_n}{2^p - 1}. \tag{16}$$

This can be verified as follows: using twice the number of nodes in (15) we obtain

$$I = Q_{2n} + 2^{-p} c_p n^{-p} + O(n^{-q}). \tag{17}$$

Then considering the linear combination $\frac{2^p}{2^p-1}(17) - \frac{1}{2^p-1}(15)$ between the equations of single and double the number of subintervals we obtain

$$I = \widetilde{Q}_{2n} + O(n^{-q}),$$

i.e. that $\widetilde{Q}_{2n}$ is of order at least q.

Note that Simpson's formula (13) turns out to be another error expansion of the form (8), so **we can apply Richardson extrapolate another time**, this time using Simpson's rule with $2n$ and $4n$ quadrature subintervals. From (16) we note the appropriate linear combination to be

$$R_{4n}(f) = \frac{1}{15}\Big(16 S_{4n}(f) - S_{2n}(f)\Big), \tag{18}$$

which is a **6-th order numerical integration formula**.

This process can of course be repeated using $8n$, $16n$, $\ldots$ subintervals, leading to higher and higher quadrature orders. One usually calls this **Romberg integration**, which we will not present in full generality here.

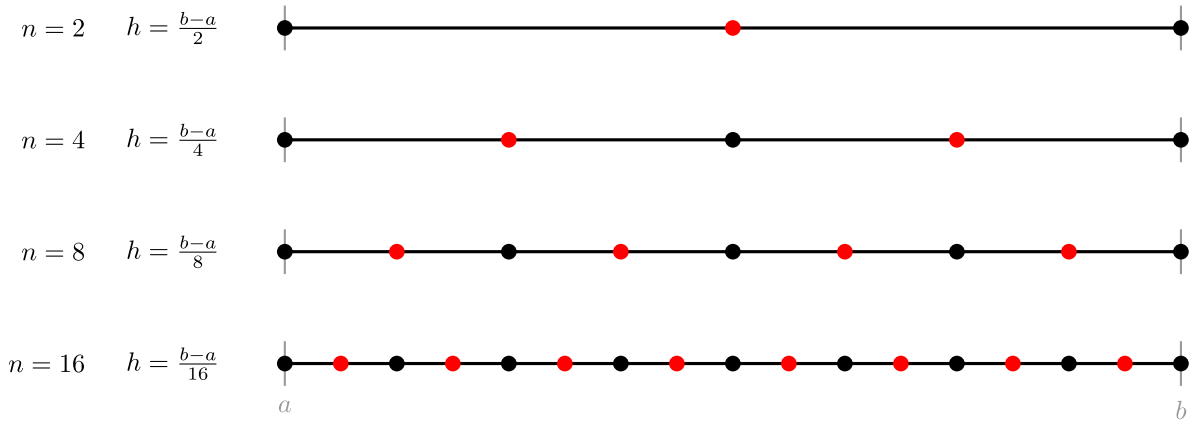
▸ **Optional: Details on the equality of the two forms of Simpson's formula**

Node doubling

If we consider the integration formula R_{4n} we notice that R_{4n} depends on S_{4n} and S_{2n}, which in turn depend on T_n, T_{2n} and T_{4n}. Continuing along the **Romberg integration procedure** we would **recursively perform more and more levels of extrapolation**. At the next level we also require T_{8n}, then T_{16n} and so on, such that **in each step** the **number of nodes** to consider roughly **doubles**.

This **node doubling is particularly advantageous in practice** and much preferred over other schemes. The reason is illustrated in the figure below, which divides the node spacing by half from

one level to the next. In each layer new nodes are only introduces at the midpoints (shown in red). Notably, about half of the total number of nodes in each layer is red. Therefore moving from h to $h/2$ (respectively from $n+1$ to $2n+1$ nodes) **only about half of the nodes are new** and need to be evaluated. On the others the function values are already known (as they were needed at the prevous level) and can be re-used without extra cost.



⚠ TODO ⚠

Using the drawing above this can be better explained, in particular T_{2n} easily re-written from T_n if one considers halfing the interval size ... perhaps some extra labels need to be introduced in the drawing above

```
TODO("Using the drawing above this can be better explained, in particular T_{2n} easily re-written from T_n if one considers halfing the interval size ... perhaps some extra labels need to be introduced in the drawing above")
```

To make this explicit we use a formulation for the trapezoidal rule, which is developed in the part *Details on the equality of the two forms of Simpson's formula*, equation (19). As it states the trapezoidal rule with $2n$ subintervals can be expressed in terms of quantities of the trapezoidal rule with n subintervals, that is in terms of the quadrature nodes $t_i = a + i\,h$ (for $i = 0, \ldots, n$), the nodal spacing $h = (b-a)/n$ and the midpoints $m_i = \frac{t_i + t_{i+1}}{2}$:

$$T_{2n}(f) = \frac{h}{4} f(t_0) + \frac{h}{2} \sum_{i=0}^{n-1} f(m_i) + \frac{h}{2} \sum_{i=1}^{n-1} f(t_i) + \frac{h}{4} f(t_n).$$

Noting $m_i = a + (2i+1)\,\frac{h}{2}$ for $i = 0, \ldots, n-1$ and using the definition of $T_n(f)$ this can be reformulated as

$$T_{2n}(f) = \frac{1}{2} T_n(f) + \frac{h}{2} \sum_{i=0}^{n-1} f\left(a + (2i+1)\,\frac{h}{2}\right). \tag{20}$$

Since $a + (2i+1)\,\frac{h}{2} = a + (2i+1)\,\frac{b-a}{2n}$ are just the *odd* quadrature nodes of the $T_{2n}(f)$ quadrature formula, we see that only these n odd nodes need to be evaluated additionally when considering $T_{2n}(f)$ instead of $T_n(f)$.

Let's see this in an example. We want to compute

$$V = \int_0^2 x^2 e^{-2x}\,dx$$

using extrapolation. First we use `quadgk` to get an accurate value:

```
begin
    g(x) = x^2 * exp(-2x)
    a = 0
    b = 2
    V, _ = quadgk(g, a, b, atol=1e-14, rtol=1e-14)
end;
```

```
0.1904741736116139
```

```
V
```

Based on the trapezoidal rule on $n = 20$ subintervals we get a first estimate:

```
T20 = 0.19041144993926787
```

```
T20 = let
    n = 20
    h = (b - a) / n
    t = h * (0:n)
    y = g.(t)
    T20 = h*y[1]/2 + h*sum(y[2:n]) + h*y[n+1]/2
end
```

Now we double to $n = 40$, but we only need to evaluate the odd nodes:

```
T40 = 0.19045880585951175
```

```
T40 = let
    n = 40
    h = (b - a) / n
    t = h * (0:n)
    y_odd = g.(t[2:2:n])
    T40 = T20/2 + h*sum(y_odd)
end
```

We repeat a second time and double n again:

```
T80 = 0.1904703513046443
```

```
T80 = let
    n = 80
    h = (b - a) / n
    t = h * (0:n)
    y_odd = g.(t[2:2:n])
    T80 = T40/2 + h*sum(y_odd)
end
```

Using (13) we can perform a first level of extrapolation and get the two Simpson values $S_{40}(f)$ and $S_{80}(f)$:

```
S40 = 0.19047459116625973
```

```
S40 = (4T40 - T20) / 3
```

```
S80 = 0.19047419978635513
```

```
S80 = (4T80 - T40) / 3
```

Finally, we perform one more level of extrapolation to get the sxth-order accurate result $R_{80}(f)$:

```
R80 = 0.1904741736943615
```

```
R80 = (16S80 - S40) / 15
```

We compute all errors to 10 digits:

```
begin
    eT = round.(V .- [T20 T40 T80]; digits=10)
    eS = round.(V .- [S40 S80];     digits=10)
    eR = round.(V .- [R80];         digits=10)
end;
```

and summarise them in a table along with the number of function evaluations:

number of evals	order 2	order 4	order 6
21	6.27237e-5		
41	1.53678e-5	-4.176e-7	
81	3.8223e-6	-2.62e-8	-1.0e-10

We notice that the 6th order **result obtained using Richardson extrapolation** is about **twice as accurate** as the order 2 result, even though it uses the **same number of function evaluations**.

Since the cost of function evaluation is usually dominating in numerical integration, the take-away is that one should **never just use low-order quadratures**, but always **employ extrapolation techniques.**

Optional: *A posteriori* error estimation

If we want to numerically evaluate an integral $I = \int_a^b f(x)\,dx$ using a chosen quadrature formula Q_n, a natural question to ask is how many quadrature nodes n are necessary to obtain an approximation of the the integral I within a predefined error tolerance ϵ.

In the previous sections we discussed, that Richardson extrapolation provides a receipe to obtain a higher-order quadrature $\tilde{Q}_{2n}(f)$ from the numerical integration values $Q_n(f)$ and $Q_{2n}(f)$. As a

result $\widetilde{Q}_{2n}(f)$ is in general more accurate than $Q_{2n}(f)$, which motivates to employ the difference

$$\eta_{2n} = |\widetilde{Q}_{2n}(f) - Q_{2n}(f)|$$

as an estimate of the error committed by the formula $Q_{2n}(f)$. Indeed assuming an error expansion (15), i.e. $I = Q_n(f) + c_p n^{-p} + O(n^{-q})$, and and a corresponding Richardson extrapolation (16) one verifies

$$\begin{aligned} |I - Q_{2n}(f)| &\leq |I - \widetilde{Q}_{2n}(f)| + |\widetilde{Q}_{2n}(f) - Q_{2n}(f)| = \underbrace{\eta_{2n}}_{=O(n^{-p})} + \underbrace{|I - \widetilde{Q}_{2n}(f)|}_{=O(n^{-q})} \\ &= \eta_{2n} + O(n^{-q}) \end{aligned}$$

Therefore a simple adaptive strategy is to start with $2n$ nodes by computing and $Q_n(f)$ and $Q_{2n}(f)$. Then check whether $\eta_{2n} \leq \epsilon$ is satisfied and if this is not the case keep doubling the number of nodes until it is. Sticking to a single layer of extrapolation for simplicity we obtain the following algorithm:

Algorithm: Adaptive quadrature based on Richardson extrapolation

Given the problem to compute $\int_a^b f(x)\,dx$, quadrature formula $Q_n(f)$ an initial number of subintervals n_0 and a requested tolerance ϵ, compute:

- Initialise $n = n_0$, $h = (b-a)/n_0$
- Initial estimate of integral $I_n = Q_n(f)$
- While $\eta_n > \epsilon$ iterate:
  - Update $h = \frac{b-a}{2n}$
  - $I_{2n} = Q_{2n}(f)$ (using updated node distance h)
  - Extrapolate $\tilde{I}_{2n} = \frac{2^p I_{2n} - I_n}{2^p - 1}$ using (14)
  - $\eta_{2n} = |\tilde{I}_{2n} - I_{2n}|$
  - Update $n = 2n$

Applied to the trapezoidal formula this is implemented as follows:

```
trapezoid_adaptive (generic function with 1 method)
```

```
function trapezoid_adaptive(f, a, b; n₀=2, tol=1e-12)
    # f:  Function
    # [a, b]: Interval to integrate over
    # n₀: Initial number of subintervals (i.e. n₀+1 quadrature nodes)
    # tol: Desired tolerance
    n = n₀
    h = (b - a) / n₀  # Node separation
    t = (0:n) .* h   # Quadrature nodes
    y = f.(t)        # Evaluate function on all nodes
    Tₙ = h * (0.5y[1] + sum(y[2:n]) + 0.5y[n+1])  # Trapezoidal formula with n₀ nodes

    extrapolated = 0.0
    ηₙ = 10tol
    while ηₙ > tol
        h = (b - a) / 2n  # New node separation
        t = (0:2n) .* h   # New set of quadrature nodes
        y_odd = f.(t[2:2:2n])  # Evaluate only at odd nodes

        # Use node doubling formula (20) to evaluate Trapezoidal formula with 2n nodes.
        T₂ₙ = Tₙ/2 + h * sum(y_odd)

        # Perform extrapolation: Note that p = 2 for the Trapezoidal formula
        extrapolated = (4T₂ₙ - Tₙ) / 3

        n  = 2n
        Tₙ = T₂ₙ
        ηₙ = abs(extrapolated - T₂ₙ)
    end

    (; integral=extrapolated, h, n)
end
```

We test this works as expected using our example from the node doubling section:

```
▸ (integral = 0.190474, h = 0.03125, n = 64)
```

```
(; integral, h, n) = trapezoid_adaptive(g, 0, 2; tol=1e-5)
```

The error is indeed below the requested tolerance:

```
6.386761788879092e-8
```

```
abs(integral - V)
```

The next step to improve this approach would be to additionally employ Romberg integration, i.e. to recursive extrapolate not only from $T_{2n}(f)$ & $T_{4n}(f)$ to $S_{4n}(f)$, but to also employ $T_n(f)$ and

evaluate the 6-th order formula $R_{4n}(f)$ like we did in the numerical example of the node doubling section. However, these ideas are out of scope for us.

Numerical analysis