ENGSCI 233 – Computational Techniques and Computer Systems

Week 3: Sampled Data (Interpolation)

Integration

Lecture 6

Andreas W. Kempa-Liehr
based on material from David Dempsey and Colin Simpson

ENGINEERING

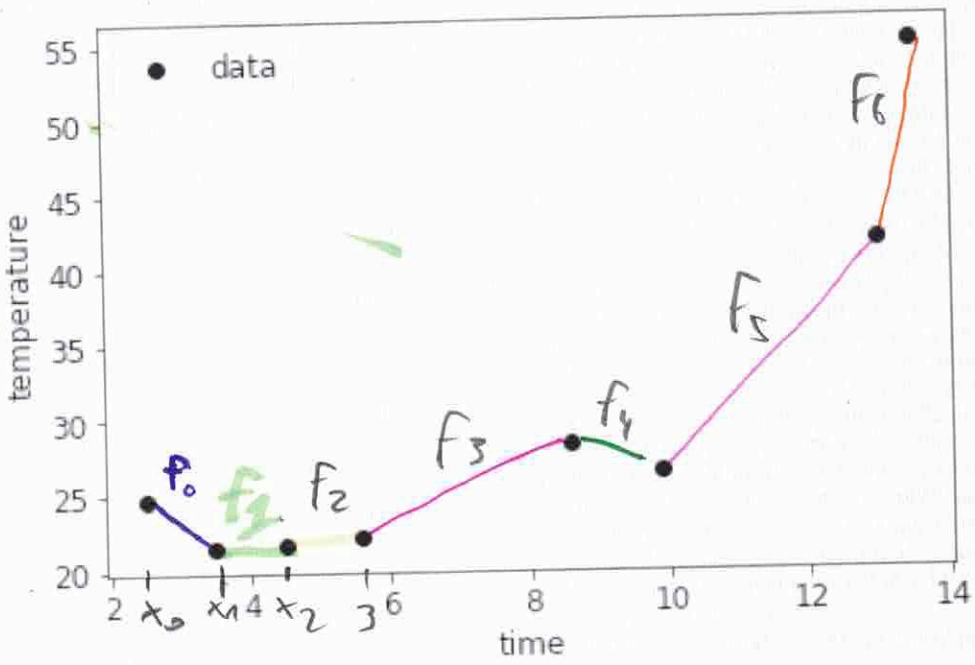
Department of Engineering Science

Week 3 term 1223 – slides˙sampleddata.pdf, Rev. 2c3afd4

Sections

Sampled Data

Consider again the sampled data example from before:



$$RSS = \sum_{i=0}^{n-1} \left(y_i - f(x_i) \right)^2$$

sum

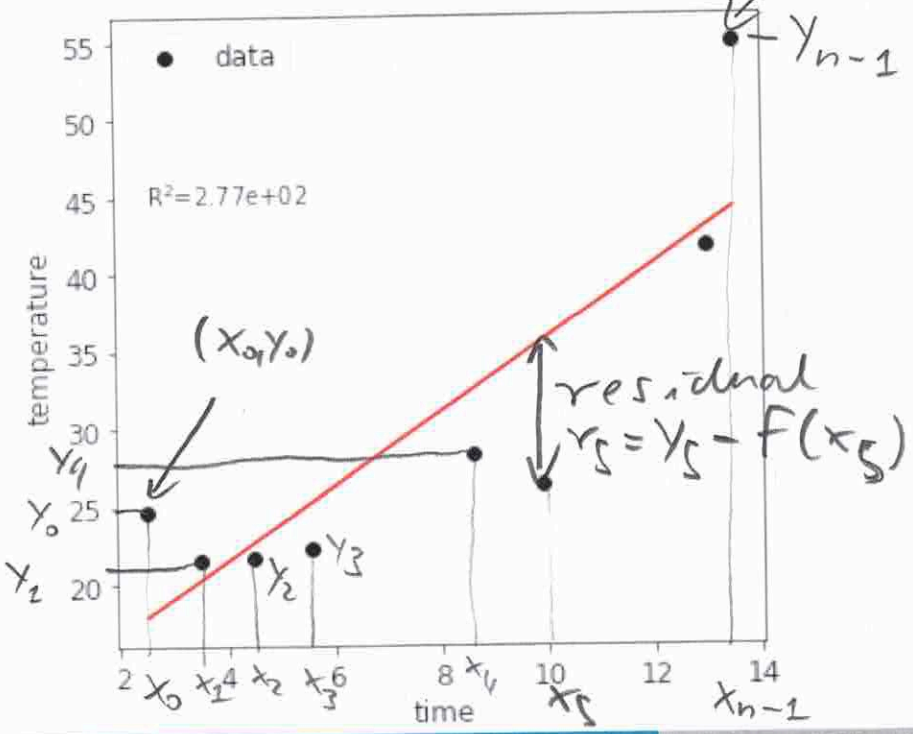
residual

$$y_i = f(x_i) + \varepsilon_i \qquad \varepsilon_i \sim N(0, \sigma)$$

$$y_i - f(x_i) = r_i$$

Order 1 Polynomial

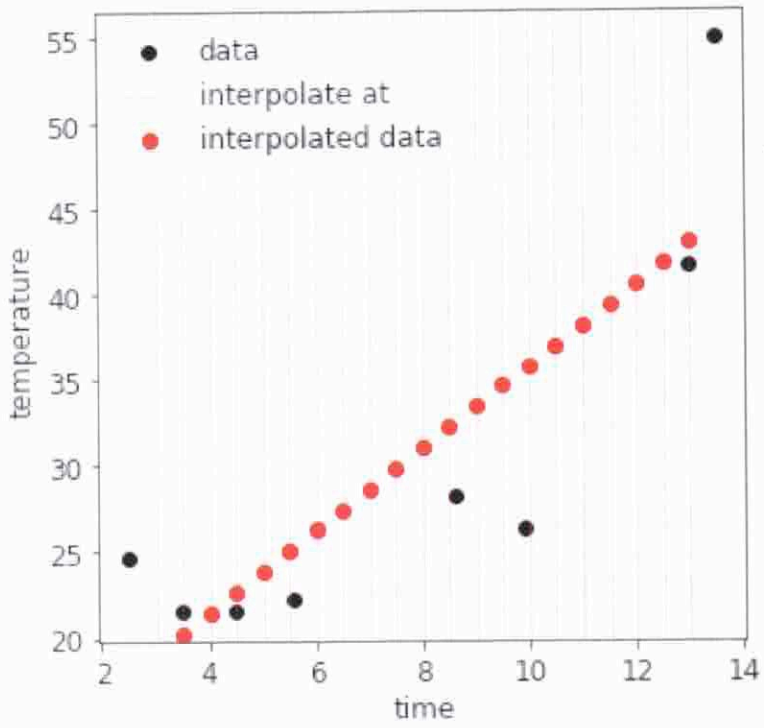
Fitting an order 1 best-fit polynomial (i.e. straight line) to the data, with an associated RSS ≈ 277:



n: number of samples

Interpolated Points

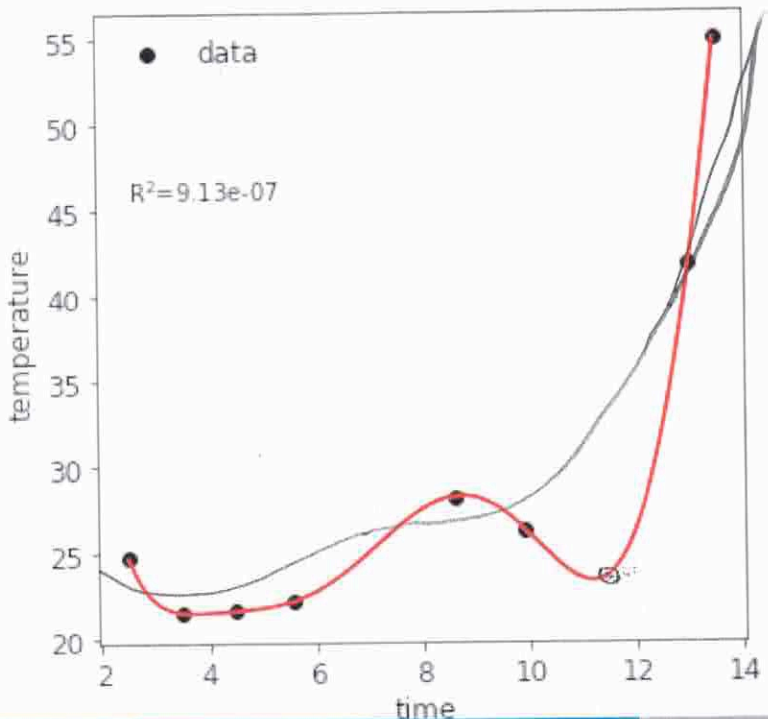
Evaluating the fitted polynomial at the interpolation points:



Order 7 Polynomial

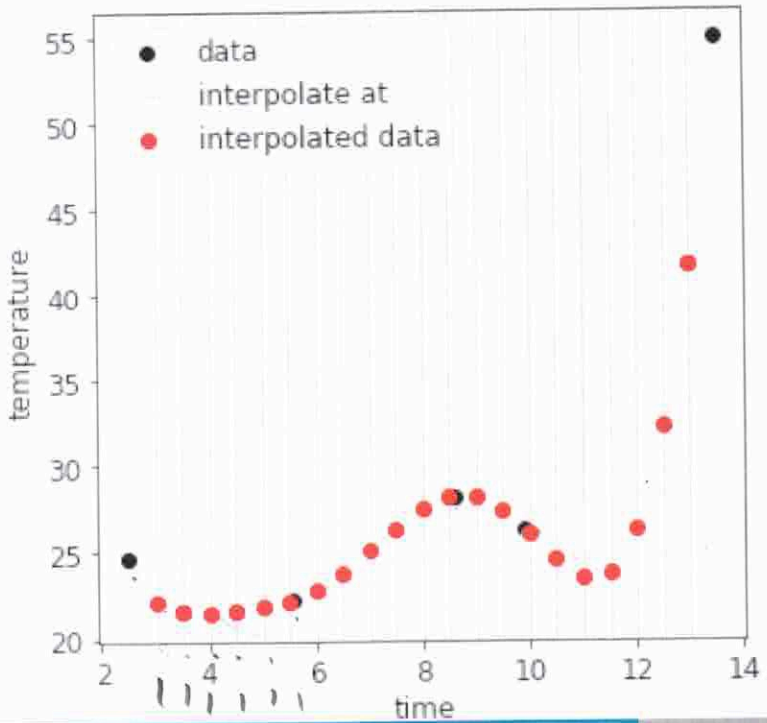
For $n = 8$ data points, there exists an order $m = n - 1$ polynomial that exactly matches the data (i.e. RSS $= 0$):

number of data points



Interpolated Points

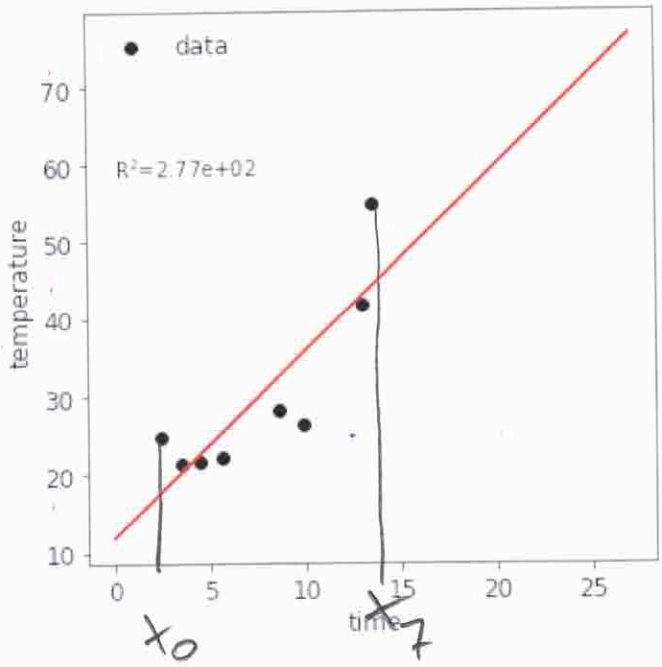
Evaluating this fitted polynomial gives a very different interpolation to the order 1 polynomial:

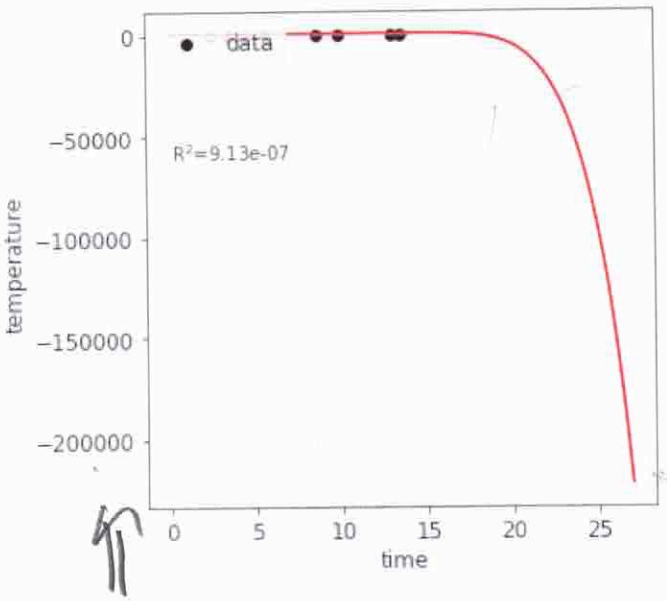


$f(x_1')$

Extrapolation

Consider extrapolating with order 1 and 7 polynomials:





Introduction

Rather than fit one interpolating function to n measurements, we can fit $n-1$ functions, f_i, to each pair of measurements:

- f_0 is an interpolating function in domain (x_0, x_1) f_0
- f_1 is an interpolating function in domain (x_1, x_2)
- ...
- f_{n-2} is an interpolating function in domain (x_{n-2}, x_{n-1})

Straight Line Equation

Piecewise linear uses straight lines as interpolating functions for each measurement pair.

Consider fitting a straight line between $y(x_0)$ and $y(x_1)$:

$$f_0(x) = m_0 x + c_0$$

$$f_0(x) = \underbrace{\frac{y_1 - y_0}{x_1 - x_0} x}_{m_0 x} + \underbrace{y_0 - \frac{y_1 - y_0}{x_1 - x_0} x_0}_{c_0}$$

$$f_0(x) = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x - x_0)$$

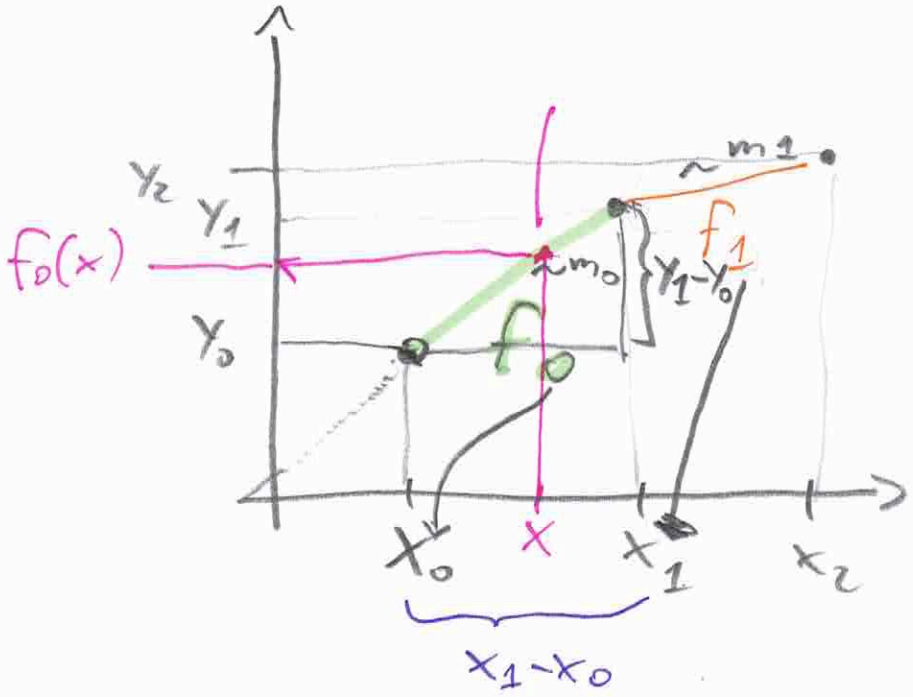
$f_1(x) = m_1 x + c_1$

$f_2(x) = m_2 x + c_2$

$\vdots$

$f_6(x) = m_6 x + c_6$

Straight Line Diagram



Subintervals

If there are n measurements in the sampled data, we can divide up the sampled data into $n-1$ subintervals.

The i^{th} subinterval is bounded by the points $[x_i, x_{i+1}]$. We can fit a straight line equation, $f_i(x)$, for each subinterval:

$$f_i(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i)$$

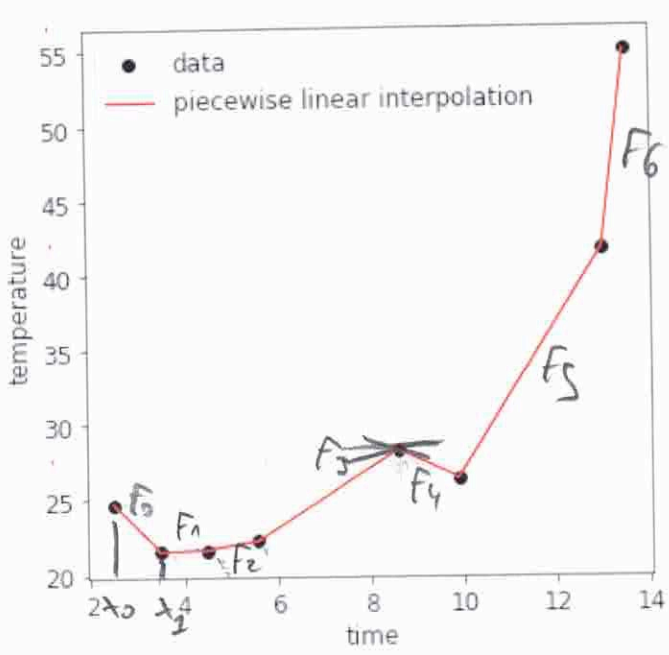
Algorithm

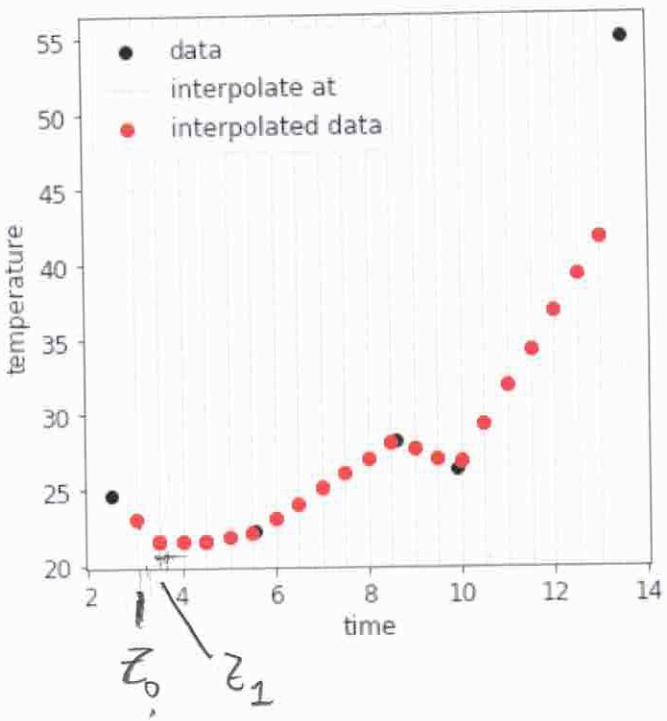
The algorithm steps for implementing piecewise linear interpolation:

- Compute the gradient, m_i, and intercept, c_i, for each subinterval.
- Find which subinterval each interpolation point, x_j, is in.
- Evaluate the relevant interpolating function, f_i, at the interpolation point, $f_i(x_j)$.

Example - Interpolating Functions

The interpolating functions and interpolated points:





Note: this method suffers from derivative discontinuity.


```
plt.plot(x,y) ⇐
plt.plot(x,y,'o') ⇐
                'x')
```

$z_0 \in [x_0, x_1]$

$f_0(z_0)$

Introduction

Cubic splines fit an order 3 polynomial to each adjacent data pair. For n measurements, there are $n-1$ subintervals and therefore also $n-1$ cubic polynomials to fit.

Cubic splines have first and second order derivative continuity at the subinterval boundaries. This provides a smoother fit through data points than linear piecewise interpolation.

Cubic Spline Equation

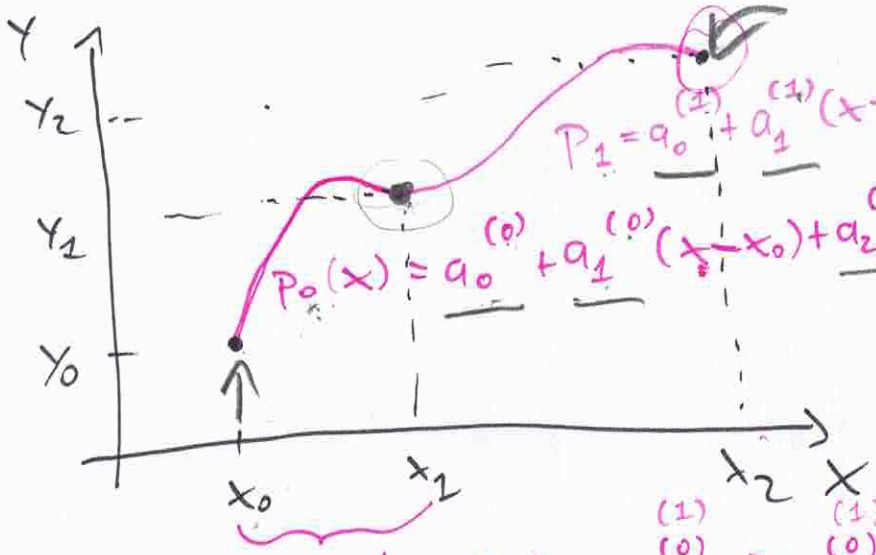
For subinterval i, which covers the interval $[x_i, x_{i+1}]$:

$$p_i(x) = a_0^{(i)} + a_1^{(i)}(x - x_i) + a_2^{(i)}(x - x_i)^2 + a_3^{(i)}(x - x_i)^3$$

$x \in [x_i, x_{i+1}]$

The notation used in the Jupyter Notebook:

- $p_i(x)$ is the cubic spline for subinterval i
- $a_0^{(i)}$ to $a_3^{(i)}$ are the polynomial coefficients for subinterval i



$P_1 = a_0^{(1)} + a_1^{(1)}(x-x_1) + a_2^{(1)}(x-x_1)^2 + a_3^{(1)}(x-x_1)^3$

$P_0(x) = a_0^{(0)} + a_1^{(0)}(x-x_0) + a_2^{(0)}(x-x_0)^2 + a_3^{(0)}(x-x_0)^3$

$\frac{d}{dx} P_0(x) = a_1^{(0)} + 2a_2^{(0)}(x-x_0) + 3a_3^{(0)}(x-x_0)$

$\frac{d^2}{dx^2} P_0(x) = 2a_2^{(0)} + 6a_3^{(0)}(x-x_0)$

$\frac{d}{dx} P_1(x) = a_1^{(1)} + 2a_2^{(1)}(x-x_1) + 3a_3^{(1)}(x-x_1)$

$\frac{d^2}{dx^2} P_1(x) = 2a_2^{(1)} + 6a_3^{(1)}(x-x_1)$

(1) =

(1) $P_0(x_0) = Y_0 = a_0^{(0)} + a_1^{(0)}\underbrace{(x_0 - x_0)}_{=0} + a_2^{(0)}\underbrace{(x_0 - x_0)^2}_{=0} + a_3^{(0)}\underbrace{(x_0 - x_0)^3}_{=0}$

$= a_0^{(0)}$

(2) $P_0(x_1) = Y_1 = a_0^{(0)} + a_1^{(0)}(x_1 - x_0) + a_2^{(0)}(x_1 - x_0)^2 + a_3^{(0)}(x_1 - x_0)^3$

(3) $P_1(x_1) = Y_1$

(4) $P_1(x_2) = Y_2$

(5) $\left.\frac{d}{dx} P_0(x)\right|_{x=x_1} = \left.\frac{d}{dx} P_1(x)\right|_{x=x_1}$

$a_1^{(0)} + 2a_2^{(0)}(x_1 - x_0) + 3a_3^{(0)}(x_1 - x_0)^2$

$= a_1^{(1)} + 2a_2^{(1)}\underbrace{(x_1 - x_1)}_{=0} + 3a_3^{(1)}\underbrace{(x_1 - x_1)^2}_{=0}$

(6) $\left.\frac{d^2}{dx^2} P_0(x)\right|_{x=x_1} = \left.\frac{d^2}{dx^2} P_1(x)\right|_{x=x_1}$

$2a_2^{(0)} + 6a_3^{(0)}(x_1 - x_0) = 2a_2^{(1)} + 6a_3^{(1)}\underbrace{(x_1 - x_1)}_{=0}$

boundary conditions

(7) $\left.\frac{d^2}{dx^2} P_0(x)\right|_{x=x_0} = 0$

$2a_2^{(0)} + 6a_3^{(0)}(x_0 - x_0) = 0$

(8) $\left.\frac{d^2}{dx^2} P_1(x)\right|_{x=x_2} = 0$

$2a_2^{(1)} + 6a_3^{(1)}(x_2 - x_1) = 0$

Derivative Equations

Adjacent cubic splines, $p_i(x)$ and $p_{i+1}(x)$, must have first and second order derivative continuity at $x = x_{i+1}$.

The general equations for the first and second order derivatives:

$$\frac{dp_i}{dx} = a_1^{(i)} + 2a_2^{(i)}(x - x_i) + 3a_3^{(i)}(x - x_i)^2$$

$$\frac{d^2p_i}{dx^2} = 2a_2^{(i)} + 6a_3^{(i)}(x - x_i)$$

Polynomial Coefficients

For n data points, there are $n-1$ cubic splines and $4\,(n-1)$ unknown polynomial coefficients.

We therefore require $4\,(n-1)$ equations:

- Boundary conditions on function value
- Boundary conditions on function first order derivative
- Boundary conditions on function second order derivative
- Second derivative zero at first/last data points

Conditions: Function Value

Each cubic spline must exactly match the sampled data at either end of its subinterval i.e. $p_i(x_i) = y(x_i)$ and $p_i(x_{i+1}) = y(x_{i+1})$:

$$a_0^{(i)} = y_i$$

$$a_0^{(i)} + a_1^{(i)}(\Delta x_i) + a_2^{(i)}(\Delta x_i)^2 + a_3^{(i)}(\Delta x_i)^3 = y_{i+1}$$

where $\Delta x_i = x_{i+1} - x_i$.

These boundary conditions apply to all $n-1$ cubic polynomials. This gives $2(n-1) = 2n-2$ equations.

Conditions: First Order Derivative Continuity

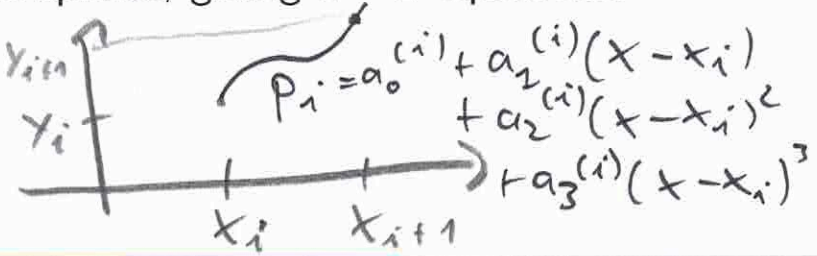
Equating the first order derivative of adjacent cubic splines:

$$\left.\frac{dp_i}{dx}\right|_{x=x_{i+1}} - \left.\frac{dp_{i+1}}{dx}\right|_{x=x_{i+1}} = 0$$

$$a_1^{(i)} + 2a_2^{(i)}(\Delta x_i) + 3a_3^{(i)}(\Delta x_i)^2 - a_1^{(i+1)} = 0$$

where $\Delta x_i = x_{i+1} - x_i$.

There are $n-2$ adjacent cubic splines, giving $n-2$ equations.



Conditions: Second Order Derivative Continuity

Equating the second order derivative of adjacent cubic splines:

$$\left.\frac{d^2 p_i}{dx^2}\right|_{x=x_{i+1}} - \left.\frac{d^2 p_{i+1}}{dx^2}\right|_{x=x_{i+1}} = 0$$

$$2a_2^{(i)} + 6a_3^{(i)}(\Delta x_i) - 2a_2^{(i+1)} = 0$$

where $\Delta x_i = x_{i+1} - x_i$.

There are $n-2$ adjacent cubic splines, giving $n-2$ equations.

Conditions: End Points

The second derivative is zero at the first data point:

$$\left.\frac{d^2 p_0}{dx^2}\right|_{x=x_0} = 0$$

$$2a_2^{(0)} = 0$$

The second derivative is zero at the last data point:

$$\left.\frac{d^2 p_{n-2}}{dx^2}\right|_{x=x_{n-1}} = 0$$

$$2a_2^{(n-2)} + 6a_3^{(n-2)}(\Delta x_{n-2}) = 0$$

This gives the last two equations.

System of Equations

This system of $4\,(n-1)$ equations can be solved as a linear algebra problem i.e. $A\vec{x} = \vec{b}$.

In this course we construct the rows of A and $\vec{b}$ using:

- First $2\,(n-1)$ rows from function value condition.
- Next $n-2$ rows from first derivative continuity.
- Next $n-2$ rows from second derivative continuity.
- Two rows from zero second derivative at first/last data point.

Learning Outcomes – Integration

- Develop algorithms to apply concepts of numerical integration (e.g. Newton-Cotes and Gaussian quadrature methods) to discrete data.
- Know the mathematics associated with the trapezium rule. Gaussian quadrature.

Numerical vs Analytical Integration

Not all integrals can be solved analytically, but can be either approximated or precisely calculated using numerical methods.

Consider a generalised integral:

$$I = \int_a^b g(x)\ dx$$

The value of the integral, I, corresponds to the area under the graph of the *integrand*, $g(x)$, between $x = a$ and $x = b$.

Numerical Integration

Numerical integration divides the integration range into a number of subintervals and sums the area of these individual subintervals.

- Known integrand: the subintervals can be chosen.
- Unknown integrand: the subintervals coincide with the sampled data. The integrand is approximated by an interpolating function, $f(x) \approx g(x)$.

Newton-Cotes Methods

Newton-Cotes methods fit a polynomial to each subinterval, $p_i(x)$, and evaluates the geometric area under each polynomial.

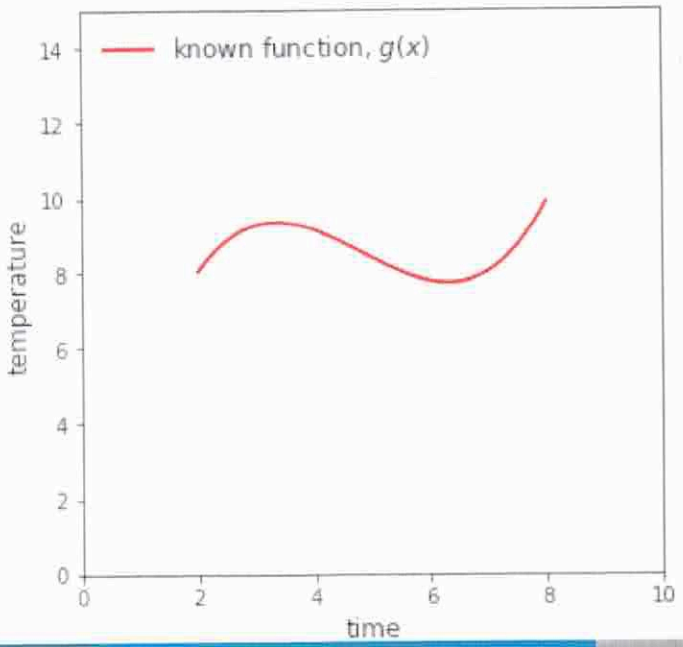
There are two commonly used variants:

- Trapezium method: order 1 polynomial
- Simpson's method: order 2 polynomial

Numerical Integration - Example

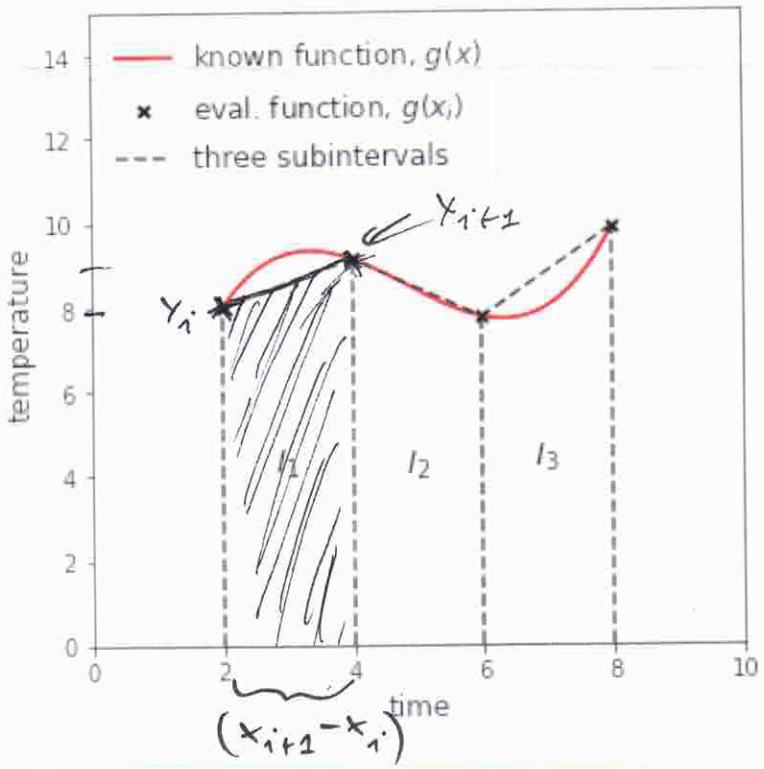
Consider numerical integration of a known integrand:

$$I = \int_2^8 g(x)\ dx$$



Numerical Integration - Example

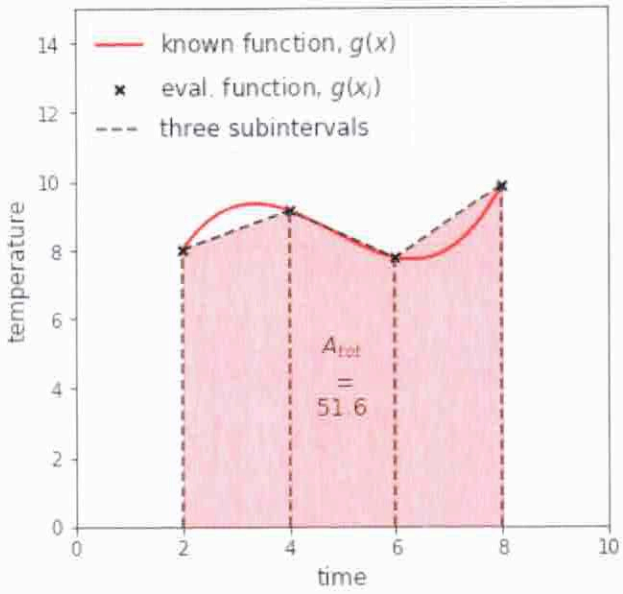
Numerical approximation of the integral using subintervals:



Numerical Integration - Example

The subinterval area sum, A_{tot}, approximates the integral, I:

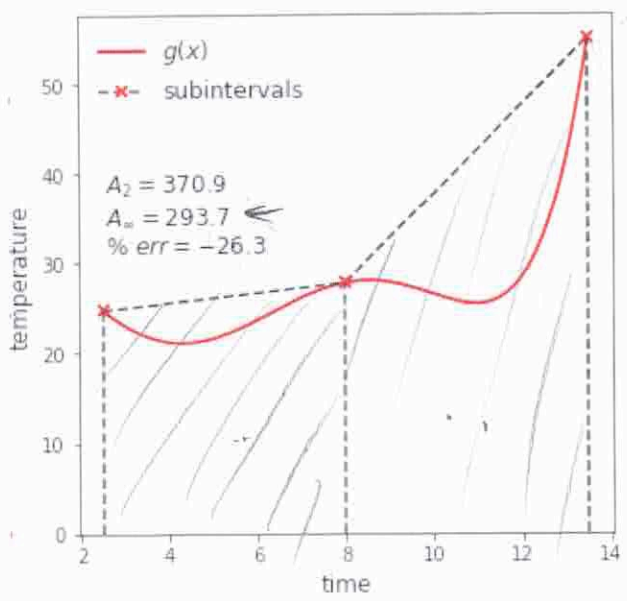
$$I \approx A_{tot} = \sum_{i=0}^{n-2} I_i$$

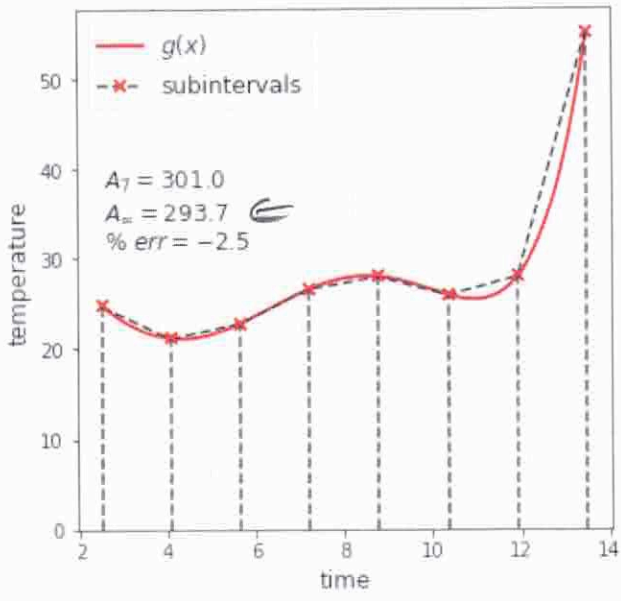


Numerical Accuracy of Newton-Cotes

There is typically a trade-off between numerical accuracy and computational expense for Newton-Cotes methods.

For example:





Trapezium Method

Fit an order 1 polynomial to each subinterval i.e. the same as in piecewise linear interpolation. The area for subinterval i is:

$$I_i = \frac{y(x_i) + y(x_{i+1})}{2}(x_{i+1} - x_i)$$

Simpson's Method

Fit an order 2 polynomial to each subinterval. This polynomial will exactly match the data at either end of the subinterval, as well as an mid-point.

We generally won't go into much detail on this method.

Simpson's Method

Fit an order 2 polynomial, $p_i(x)$, to subinterval i that exactly matches the data points at either end of the subinterval, as well as an imaginary mid-point:

$$f_i(x_i) = ax_i^2 + bx_i + c$$

$$f_i\left(\frac{x_i + x_{i+1}}{2}\right) = ax_i^2 + bx_i + c = a\left(\frac{x_i + x_{i+1}}{2}\right)^2 + \frac{x_i + x_{i+1}}{2} + c$$

$$f_i(x_{i+1}) = ax_{i+1}^2 + bx_{i+1} + c$$

i.e. the same as in piecewise linear interpolation. The area for subinterval i is:

$$I_i = \frac{y(x_i) + y(x_{i+1})}{2}(x_{i+1} - x_i)$$