MSDM5004
Numerical Methods and Modeling in Science
Spring 2024

Lecture 2

Prof Yang Xiang
Hong Kong University of Science and Technology

4. Order of convergence

Definition

Suppose $\{p_n\}_{n=0}^{\infty}$ is a sequence that converges to p, with $p_n \neq p$ for all n. If positive constants λ and α exist with

$$\lim_{n\to\infty} \frac{|p_{n+1} - p|}{|p_n - p|^{\alpha}} = \lambda,$$

then $\{p_n\}_{n=0}^{\infty}$ **converges to p of order α, with asymptotic error constant λ.**

An iterative technique of the form $p_n = g(p_{n-1})$ is said to be of *order* α if the sequence $\{p_n\}_{n=0}^{\infty}$ converges to the solution $p = g(p)$ of order α.

In general, a sequence with a high order of convergence converges more rapidly than a sequence with a lower order. The asymptotic constant affects the speed of convergence but not to the extent of the order. Two cases of order are given special attention.

(i) If $\alpha = 1$ (and $\lambda < 1$), the sequence is **linearly convergent**.

(ii) If $\alpha = 2$, the sequence is **quadratically convergent**.

Newton's method $$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}$$

error $e_n = p_n - p$ satisfies $$e_{n+1} = \frac{f''(\xi)}{2f'(p_n)} e_n^2.$$

ξ between p_n and p.

When $f'(p) \neq 0$, $\{p_n\}_{n=0}^{\infty}$ converges to p,

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{e_n^2} = \lim_{n\to\infty} \frac{f''(\xi)}{2f'(p_n)^2} = \frac{f''(p)}{2f'(p)^2}.$$

It is at least quadratically convergent.

5. Secant method

Newton's method $$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Sometimes formulation of *f'(x)* is not available.

Approximating the derivative $f'(x_n)$ by

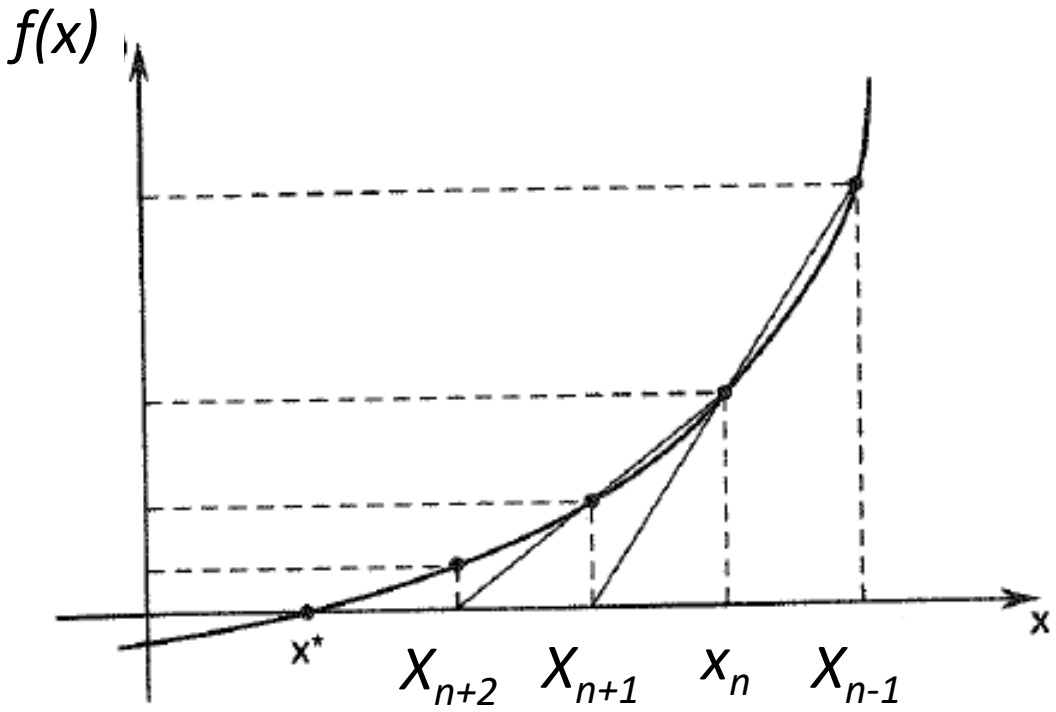
$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}},$$

we have the secant method for finding root:

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}.$$

Two initial estimates are needed: x_0, x_1

Idea of the secant method



Approximate the function *f(x)* near x_n by the secant line passing through $(x_n, f(x_n))$ and $(x_{n-1}, f(x_{n-1}))$.

An example

Solve for $f(x) = \cos x - x = 0$ using the secant method. The initial estimates are $x_0 = 0.5$, $x_1 = \frac{\pi}{4}$. Perform 4 iterations.

Solution. The secant method is

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}$$

$$= x_n - \frac{(\cos x_n - x_n)(x_n - x_{n-1})}{(\cos x_n - x_n) - (\cos x_{n-1} - x_{n-1})}.$$

$n = 1,$

$$\begin{aligned} x_2 =& x_1 - \frac{(\cos x_1 - x_1)(x_1 - x_0)}{(\cos x_1 - x_1) - (\cos x_0 - x_0)} \\ =& \frac{\pi}{4} - \frac{(\cos \frac{\pi}{4} - \frac{\pi}{4})(\frac{\pi}{4} - 0.5)}{(\cos \frac{\pi}{4} - \frac{\pi}{4}) - (\cos 0.5 - 0.5)} \\ =& 0.7363841388. \end{aligned}$$

$n = 2,$

$$x_3 = x_2 - \frac{(\cos x_2 - x_2)(x_2 - x_1)}{(\cos x_2 - x_2) - (\cos x_1 - x_1)} = 0.7390581392.$$

$n = 3,$

$$x_4 = x_3 - \frac{(\cos x_3 - x_3)(x_3 - x_2)}{(\cos x_3 - x_3) - (\cos x_2 - x_2)} = 0.7390851493.$$

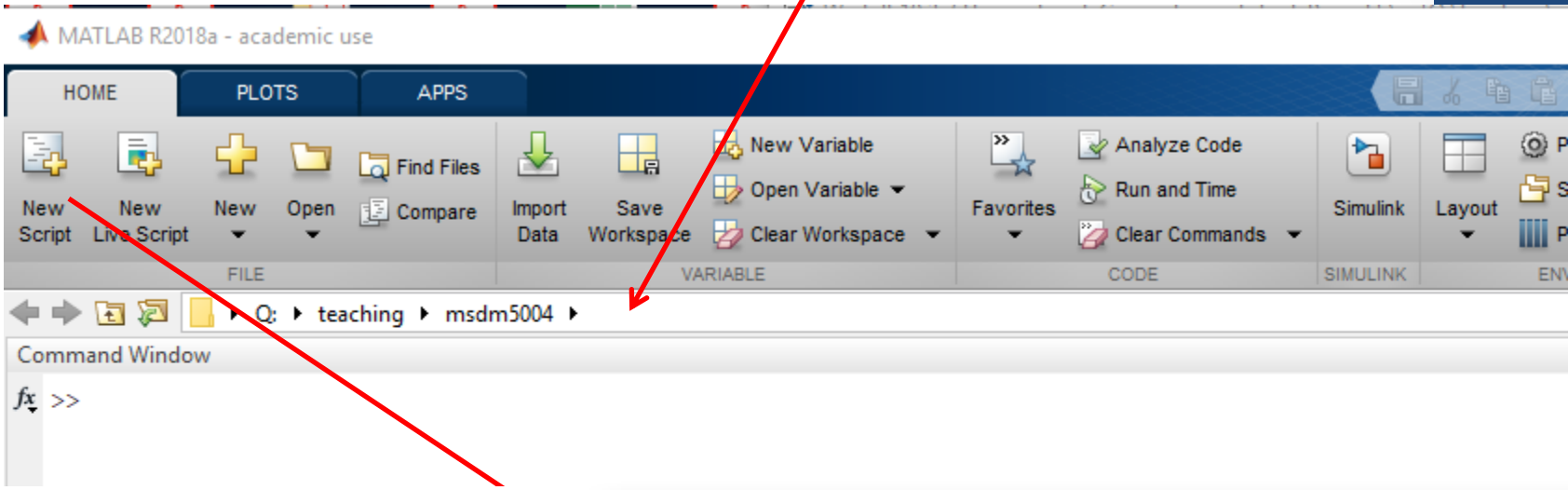
$n = 4,$

$$x_5 = x_4 - \frac{(\cos x_4 - x_4)(x_4 - x_3)}{(\cos x_4 - x_4) - (\cos x_3 - x_3)} = 0.7390851332.$$

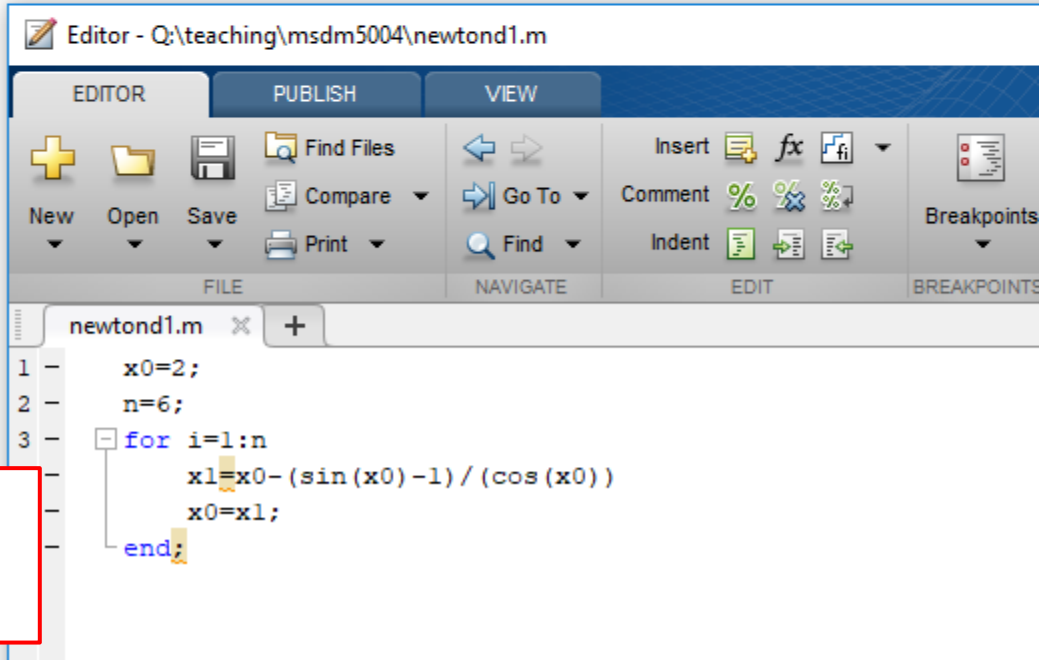
6. MATLAB codes

Running a script file: *Newton's method*

Save the file to the current directory (with .m at the end of file name)



Generate a script file

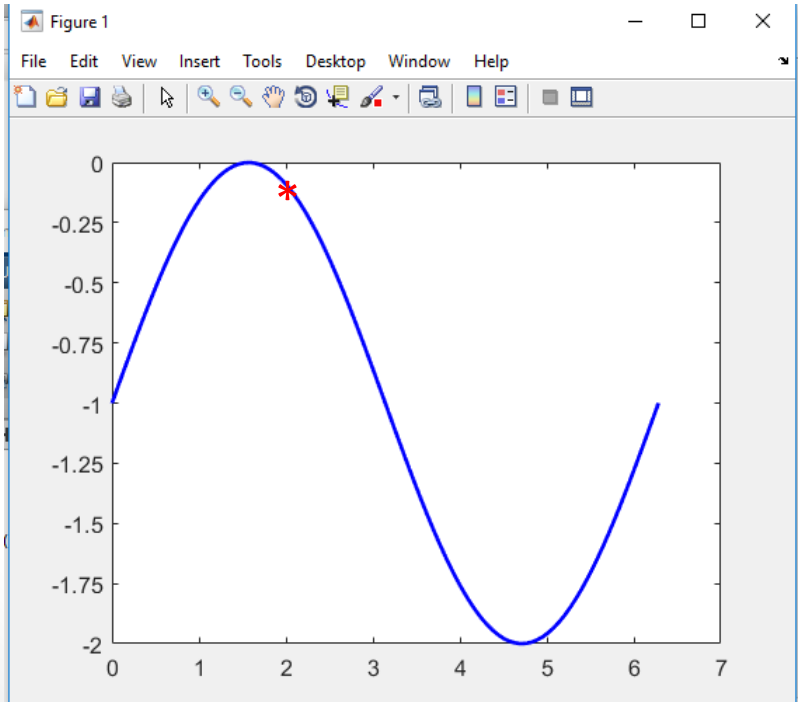


Problem: To *solve f(x)=sin(x)-1=0 using Newton's method*

The script file: newtond1.m

```
x0=2;
n=6;
for i=1:n
    x1=x0-(sin(x0)-1)/(cos(x0))
    x0=x1;
end;
```

Problem: To *solve f(x)=sin(x)-1=0*




```
>> newtond1
x1 =
   1.7820
x1 =
   1.6760
x1 =
   1.6234
x1 =
   1.5971
x1 =
   1.5839
x1 =
   1.5774
>>
```

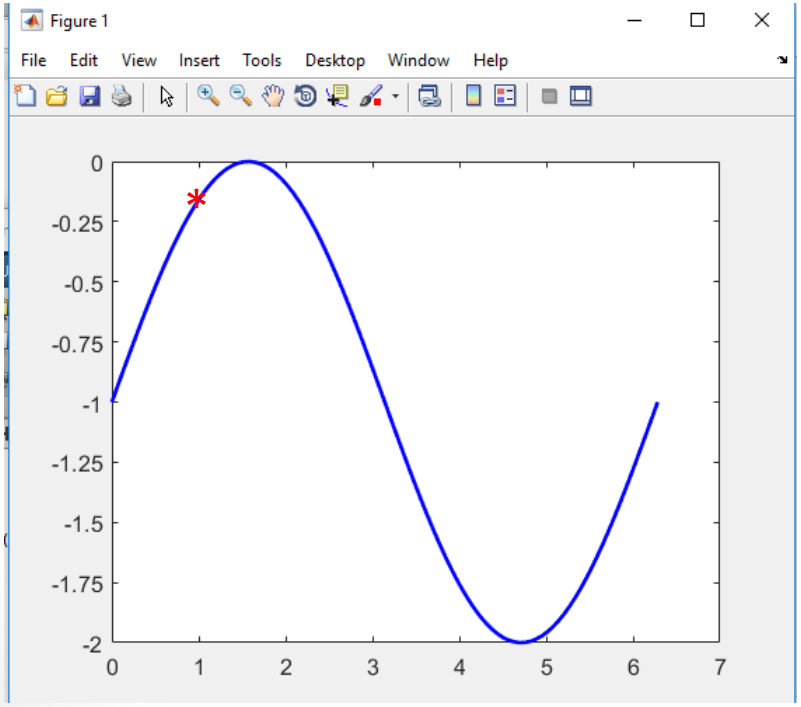
Save the output into a file

File name

```
>> diary newton1_output.dat
>> newtond1
x1 =
   1.7820
x1 =
   1.6760
x1 =
   1.6234
x1 =
   1.5971
x1 =
   1.5839
x1 =
   1.5774
>> diary off
```

Change initial estimate

```
x0=1;
n=6;
for i=1:n
    x1=x0-(sin(x0)-1)/(cos(x0))
    x0=x1;
end;
```

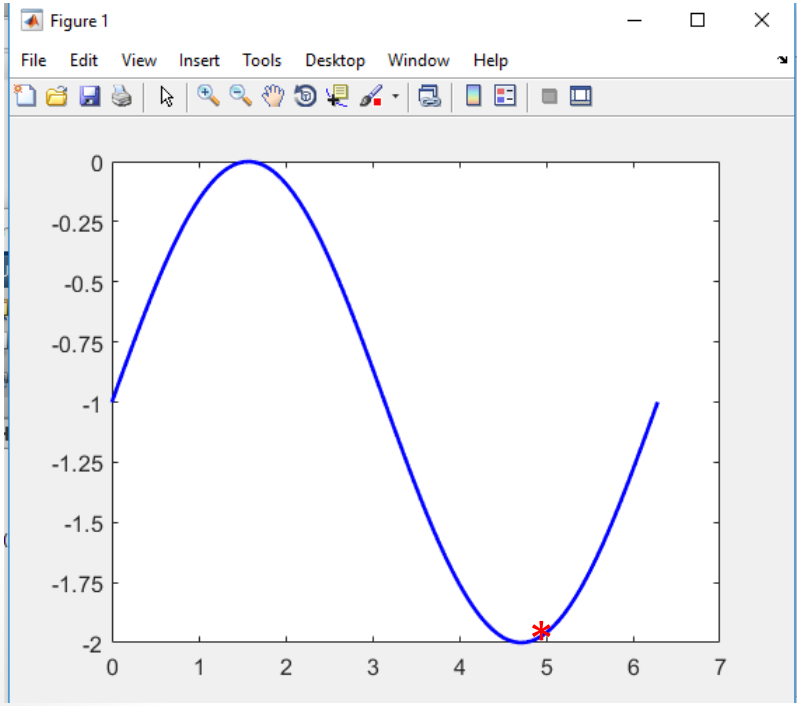



```
>> newtond1
x1 =
   1.2934
x1 =
   1.4330
x1 =
   1.5020
x1 =
   1.5364
x1 =
   1.5536
x1 =
   1.5622
>>
```

Change initial estimate

```
x0=5;
n=6;
for i=1:n
    x1=x0-(sin(x0)-1)/(cos(x0))
    x0=x1;
end;
```

Does not converge to the solution




```
>> newtond1
x1 =
   11.9058
x1 =
   13.9492
x1 =
   14.0434
x1 =
   14.0903
x1 =
   14.1138
x1 =
   14.1255
>>
```

Newton's method with stopping criterion

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| < \varepsilon$$

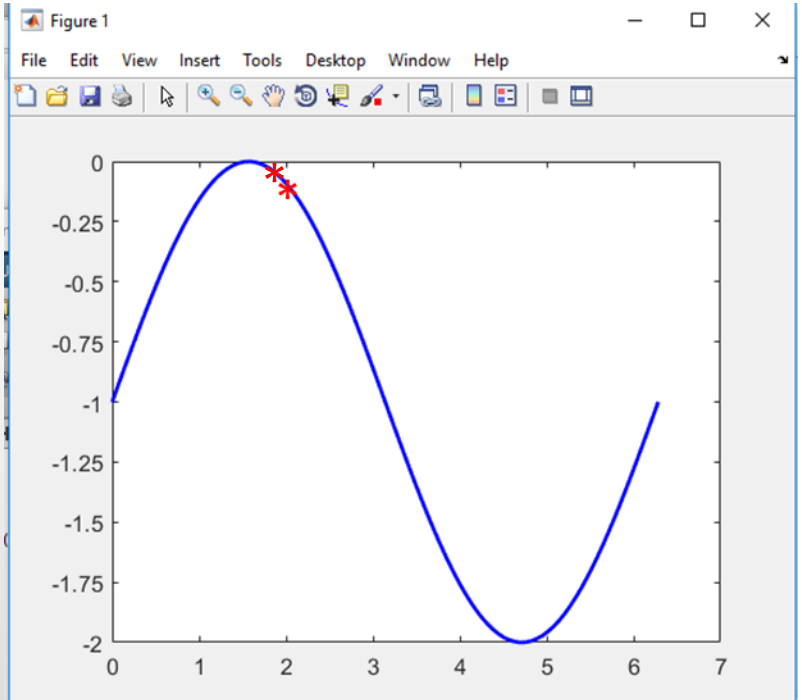
```
x0=2;
eps=0.001;
n=1
x1=x0-(sin(x0)-1)/(cos(x0))
while abs(x1-x0)>=eps
    n=n+1
    x0=x1;
    x1=x0-(sin(x0)-1)/(cos(x0))
end;
```

```
>> newtond1c
n =
   1
x1 =
   1.7820
n =
   2
x1 =
   1.6760
n =
   3
x1 =
   1.6234
n =
   4
x1 =
   1.5971
n =
   5
x1 =
   1.5839
n =
   6
x1 =
   1.5774
n =
   7
x1 =
   1.5741
n =
   8
x1 =
   1.5724
n =
   9
x1 =
   1.5716
```

Secant method

Problem: To *solve f(x)=sin(x)-1=0*

```
x0=1.8
x1=2;
n=6;
for i=1:n
    x2=x1-(x1-x0)*(sin(x1)-1)/((sin(x1)-1)-(sin(x0)-1))
    x0=x1;
    x1=x2;
end;
```




```
>> secantd1
x0 =
   1.8000
x2 =
   1.7190
x2 =
   1.6804
x2 =
   1.6337
x2 =
   1.6107
x2 =
   1.5952
x2 =
   1.5859
>>
```

7. Solving nonlinear systems

Nonlinear system

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0, \\ f_2(x_1, x_2, \ldots, x_n) &= 0, \\ &\vdots \qquad\qquad \vdots \\ f_n(x_1, x_2, \ldots, x_n) &= 0, \end{aligned}$$

Or

$$\mathbf{F}(\mathbf{x}) = \mathbf{0}.$$

where

$$\mathbf{x} = (x_1, x_2, \ldots, x_n)^t$$

$$\mathbf{F}(x_1, x_2, \ldots, x_n) = (f_1(x_1, x_2, \ldots, x_n), f_2(x_1, x_2, \ldots, x_n), \ldots, f_n(x_1, x_2, \ldots, x_n))^t.$$

Newton's method for nonlinear systems $\mathbf{F}(\mathbf{x}) = \mathbf{0}.$

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} - J\left(\mathbf{x}^{(k-1)}\right)^{-1}\mathbf{F}\left(\mathbf{x}^{(k-1)}\right)$$

where the Jacobian matrix

$$J(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_2}{\partial x_n}(\mathbf{x}) \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1}(\mathbf{x}) & \frac{\partial f_n}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_n}{\partial x_n}(\mathbf{x}) \end{bmatrix}$$

The idea

Linear approximation near $x^{(k-1)}$

Solution $\boldsymbol{p}$: $\boldsymbol{F(p)=0}$

$$\mathbf{0} = \mathbf{F}(\mathbf{p}) \approx \mathbf{F}(\mathbf{x}^{(k-1)}) + \mathbf{J}(\mathbf{x}^{(k-1)})(\mathbf{p} - \mathbf{x}^{(k-1)})$$

Solve for $\boldsymbol{p}$

$$\mathbf{p} \approx \mathbf{x}^{(k-1)} - \mathbf{J}(\mathbf{x}^{(k-1)})^{-1}\mathbf{F}(\mathbf{x}^{(k-1)}).$$

An example

$$3x_1 - \cos(x_2x_3) - \frac{1}{2} = 0,$$

$$x_1^2 - 81(x_2 + 0.1)^2 + \sin x_3 + 1.06 = 0,$$

$$e^{-x_1x_2} + 20x_3 + \frac{10\pi - 3}{3} = 0$$

Apply Newton's method to this problem with $\mathbf{x}^{(0)} = (0.1, 0.1, -0.1)^t$.

Solution Define

$$\mathbf{F}(x_1, x_2, x_3) = (f_1(x_1, x_2, x_3), f_2(x_1, x_2, x_3), f_3(x_1, x_2, x_3))^t,$$

where

$$f_1(x_1, x_2, x_3) = 3x_1 - \cos(x_2x_3) - \frac{1}{2},$$

$$f_2(x_1, x_2, x_3) = x_1^2 - 81(x_2 + 0.1)^2 + \sin x_3 + 1.06,$$

$$f_3(x_1, x_2, x_3) = e^{-x_1x_2} + 20x_3 + \frac{10\pi - 3}{3}.$$

The Jacobian matrix $J(\mathbf{x})$ for this system is

$$J(x_1, x_2, x_3) = \begin{bmatrix} 3 & x_3 \sin x_2 x_3 & x_2 \sin x_2 x_3 \\ 2x_1 & -162(x_2 + 0.1) & \cos x_3 \\ -x_2 e^{-x_1 x_2} & -x_1 e^{-x_1 x_2} & 20 \end{bmatrix}.$$

Let $\mathbf{x}^{(0)} = (0.1, 0.1, -0.1)^t$. Then $\mathbf{F}(\mathbf{x}^{(0)}) = (-0.199995, -2.269833417, 8.462025346)^t$ and

$$J(\mathbf{x}^{(0)}) = \begin{bmatrix} 3 & 9.999833334 \times 10^{-4} & 9.999833334 \times 10^{-4} \\ 0.2 & -32.4 & 0.9950041653 \\ -0.09900498337 & -0.09900498337 & 20 \end{bmatrix}.$$

Solving the linear system, $J(\mathbf{x}^{(0)})\mathbf{y}^{(0)} = -\mathbf{F}(\mathbf{x}^{(0)})$ gives

$$\mathbf{y}^{(0)} = \begin{bmatrix} 0.3998696728 \\ -0.08053315147 \\ -0.4215204718 \end{bmatrix} \quad \text{and} \quad \mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \mathbf{y}^{(0)} = \begin{bmatrix} 0.4998696782 \\ 0.01946684853 \\ -0.5215204718 \end{bmatrix}.$$

Continuing for $k = 2, 3, \ldots,$ we have

$$\begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \end{bmatrix} = \begin{bmatrix} x_1^{(k-1)} \\ x_2^{(k-1)} \\ x_3^{(k-1)} \end{bmatrix} + \begin{bmatrix} y_1^{(k-1)} \\ y_2^{(k-1)} \\ y_3^{(k-1)} \end{bmatrix},$$

where

$$\begin{bmatrix} y_1^{(k-1)} \\ y_2^{(k-1)} \\ y_3^{(k-1)} \end{bmatrix} = -\left(J\left(x_1^{(k-1)}, x_2^{(k-1)}, x_3^{(k-1)}\right)\right)^{-1} \mathbf{F}\left(x_1^{(k-1)}, x_2^{(k-1)}, x_3^{(k-1)}\right).$$

Thus, at the kth step, the linear system $J\left(\mathbf{x}^{(k-1)}\right)\mathbf{y}^{(k-1)} = -\mathbf{F}\left(\mathbf{x}^{(k-1)}\right)$ must be solved, where

$$J\left(\mathbf{x}^{(k-1)}\right) = \begin{bmatrix} 3 & x_3^{(k-1)}\sin x_2^{(k-1)}x_3^{(k-1)} & x_2^{(k-1)}\sin x_2^{(k-1)}x_3^{(k-1)} \\ 2x_1^{(k-1)} & -162\left(x_2^{(k-1)}+0.1\right) & \cos x_3^{(k-1)} \\ -x_2^{(k-1)}e^{-x_1^{(k-1)}x_2^{(k-1)}} & -x_1^{(k-1)}e^{-x_1^{(k-1)}x_2^{(k-1)}} & 20 \end{bmatrix},$$

$$\mathbf{y}^{(k-1)} = \begin{bmatrix} y_1^{(k-1)} \\ y_2^{(k-1)} \\ y_3^{(k-1)} \end{bmatrix},$$

and

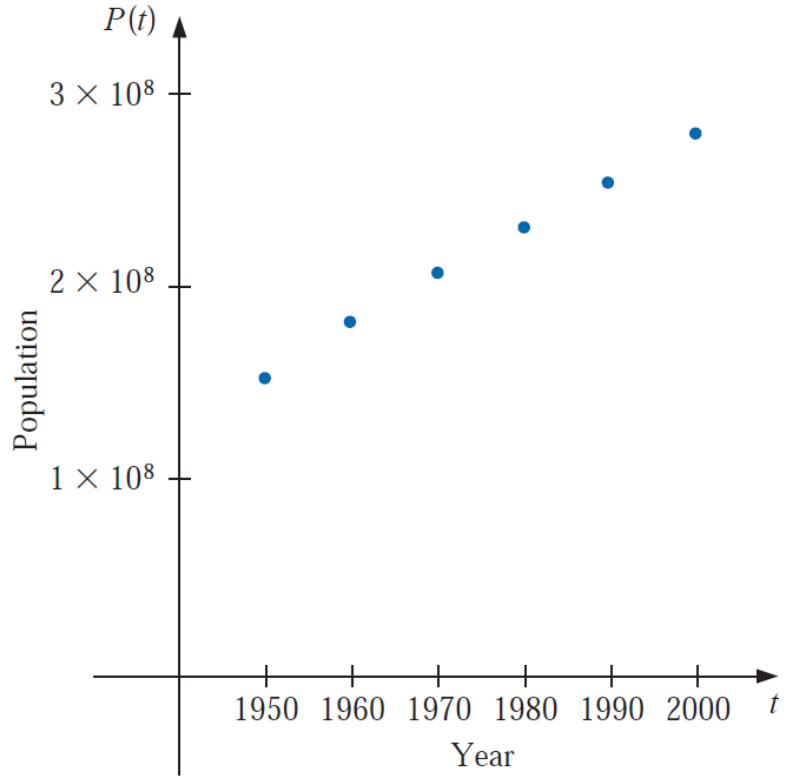
$$\mathbf{F}\left(\mathbf{x}^{(k-1)}\right) = \begin{bmatrix} 3x_1^{(k-1)} - \cos x_2^{(k-1)}x_3^{(k-1)} - \frac{1}{2} \\ \left(x_1^{(k-1)}\right)^2 - 81\left(x_2^{(k-1)}+0.1\right)^2 + \sin x_3^{(k-1)} + 1.06 \\ e^{-x_1^{(k-1)}x_2^{(k-1)}} + 20x_3^{(k-1)} + \frac{10\pi-3}{3} \end{bmatrix}.$$

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|_\infty$
0	0.1000000000	0.1000000000	−0.1000000000	
1	0.4998696728	0.0194668485	−0.5215204718	0.4215204718
2	0.5000142403	0.0015885914	−0.5235569638	1.788×10^{-2}
3	0.5000000113	0.0000124448	−0.5235984500	1.576×10^{-3}
4	0.5000000000	8.516×10^{-10}	−0.5235987755	1.244×10^{-5}
5	0.5000000000	-1.375×10^{-11}	−0.5235987756	8.654×10^{-10}

Chapter 3

Interpolation and Polynomial Approximation

1. Interpolation problems



Given a set of discrete data points, we hope to find a function f(x) to capture the behavior.

1. *To obtain the result at some other value of the independent variable.*
2. *To compute derivatives, integrals, …*

2. Polynomial Interpolation

To approximate the give data set by a polynomial

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0,$$

where n is a nonnegative integer and $a_0, \ldots, a_n$ are real constants.

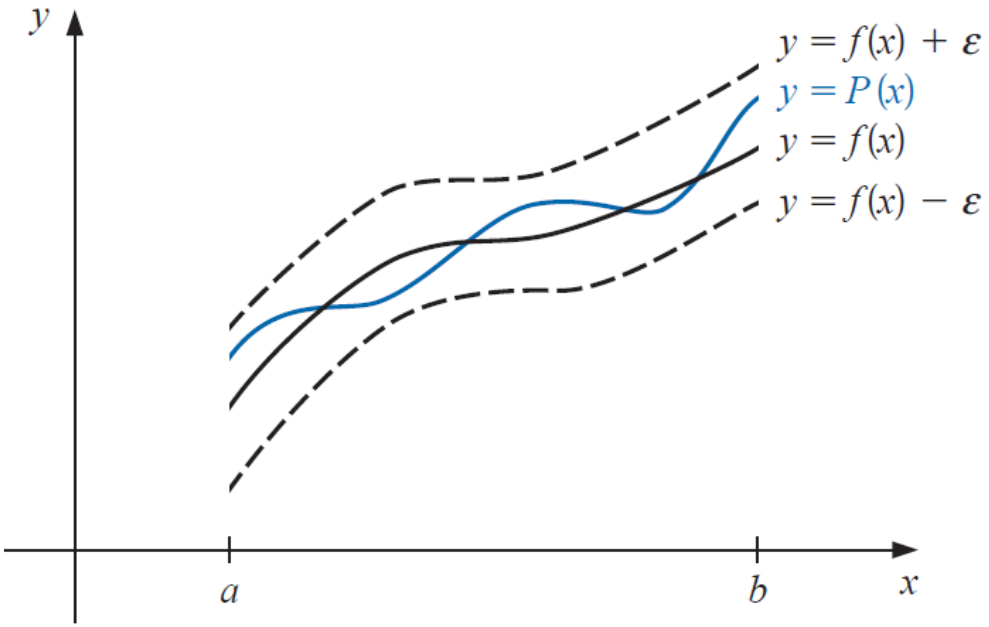
Advantages:

Simple formulation for derivatives, integrals, …

3. Weierstrass Approximation Theorem

Suppose that f is defined and continuous on $[a, b]$. For each $\epsilon > 0$, there exists a polynomial $P(x)$, with the property that

$$|f(x) - P(x)| < \epsilon, \quad \text{for all } x \text{ in } [a, b].$$



This theorem tells us that polynomials can uniformly approximate any continuous functions.

4. Taylor polynomials: Not good for interpolation

Taylor's Theorem

Suppose $f \in C^n[a,b]$, that $f^{(n+1)}$ exists on $[a,b]$, and $x_0 \in [a,b]$. For every $x \in [a,b]$, there exists a number $\xi(x)$ between x_0 and x with

$$f(x) = P_n(x) + R_n(x),$$

where

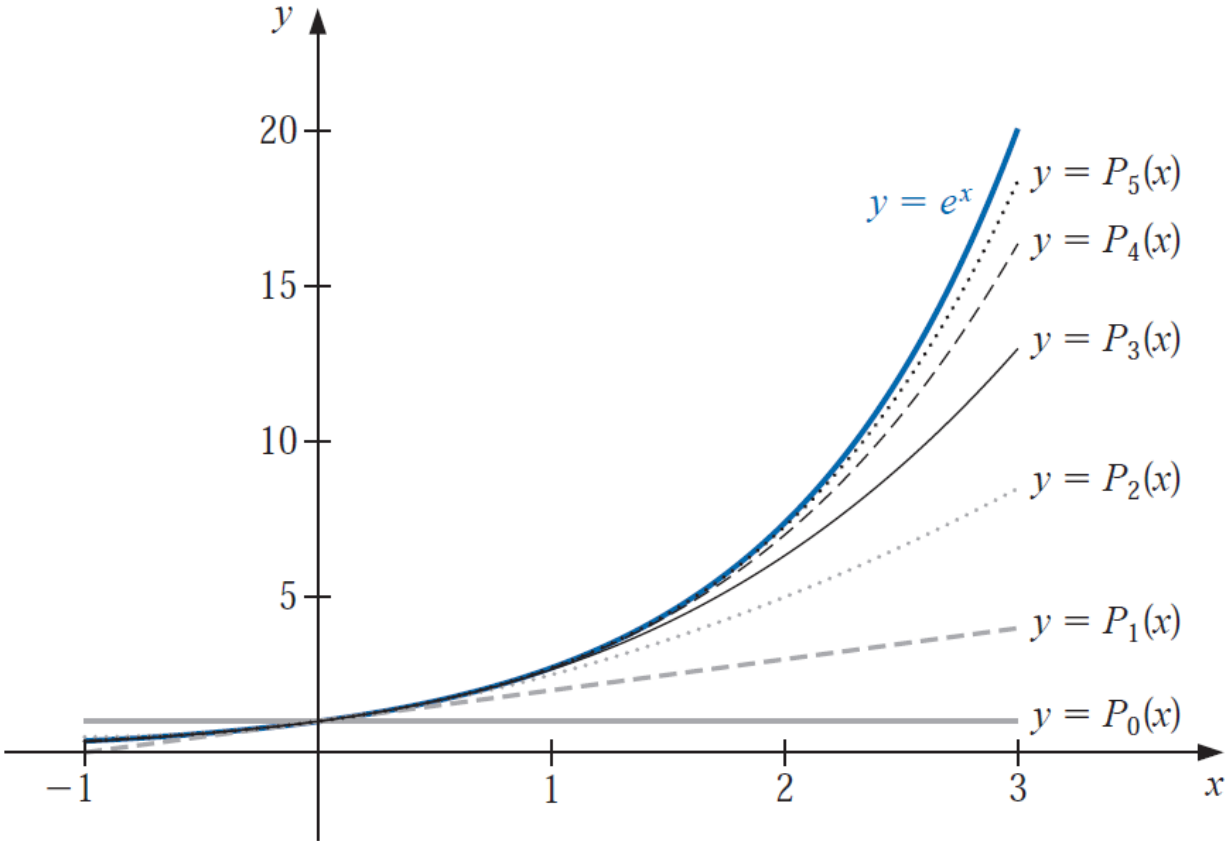
$$P_n(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

$$= \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k$$

$$R_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0)^{n+1}.$$

Here $P_n(x)$ is called the ***n*th Taylor polynomial** for f about x_0.

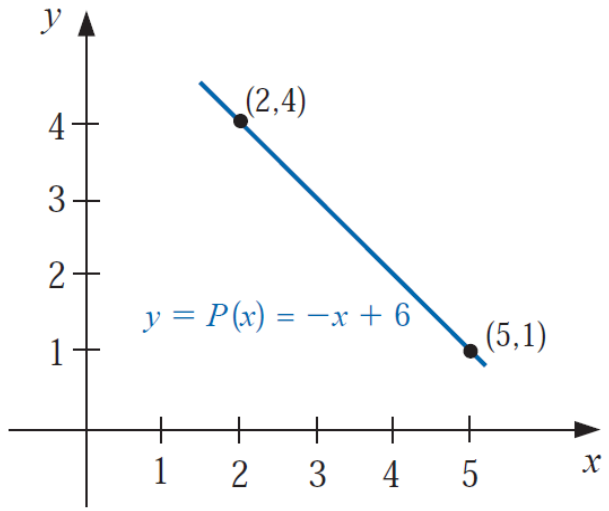
Taylor polynomial: Approximation near x_0, not necessarily accurate away from x_0.



Taylor polynomial of $f(x) = e^x$ about $x = 0$

$$P_n(x) = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \cdots + \frac{1}{n!}x^n$$

5. Polynomial interpolation: Lagrange polynomials



Linear case:

The problem of determining a polynomial of degree one that passes through the distinct points

$$(x_0, y_0) \quad \text{and} \quad (x_1, y_1)$$

is the same as approximating a function f for which

$$f(x_0) = y_0 \quad \text{and} \quad f(x_1) = y_1$$

by means of a first-degree polynomial interpolating, or agreeing with, the values of f at the given points.

Assume that the interpolating polynomial is

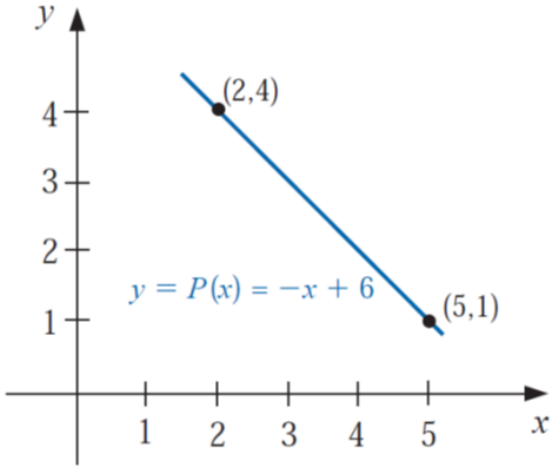
$$P(x) = ax + b.$$

We want to have $P(x_0) = f(x_0)$, $P(x_1) = f(x_1)$.

That is

$$ax_0 + b = f(x_0)$$

$$ax_1 + b = f(x_1)$$



The solution is

$$a = \frac{f(x_1) - f(x_0)}{x_1 - x_0}, \quad b = \frac{x_1 f(x_0) - x_0 f(x_1)}{x_1 - x_0}.$$

Therefore

$$P(x) = \frac{f(x_1) - f(x_0)}{x_1 - x_0} x + \frac{x_1 f(x_0) - x_0 f(x_1)}{x_1 - x_0}.$$

We write the interpolating polynomial as

$$P(x) = \frac{f(x_1) - f(x_0)}{x_1 - x_0} x + \frac{x_1 f(x_0) - x_0 f(x_1)}{x_1 - x_0}.$$

$$= \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1)$$

There is an alternative way to derive this interpolating polynomial.

Lagrange polynomial

Define the functions

$$L_0(x) = \frac{x - x_1}{x_0 - x_1} \quad \text{and} \quad L_1(x) = \frac{x - x_0}{x_1 - x_0}.$$

Definition

The linear Lagrange interpolating polynomial though (x_0, y_0) and (x_1, y_1) is

$$P(x) = L_0(x)f(x_0) + L_1(x)f(x_1) = \frac{x - x_1}{x_0 - x_1}f(x_0) + \frac{x - x_0}{x_1 - x_0}f(x_1).$$

$$L_0(x) = \frac{x - x_1}{x_0 - x_1} \quad \text{and} \quad L_1(x) = \frac{x - x_0}{x_1 - x_0}.$$

$$P(x) = L_0(x)f(x_0) + L_1(x)f(x_1) = \frac{x - x_1}{x_0 - x_1}f(x_0) + \frac{x - x_0}{x_1 - x_0}f(x_1).$$

Note that

$$L_0(x_0) = 1, \quad L_0(x_1) = 0, \quad L_1(x_0) = 0, \quad \text{and} \quad L_1(x_1) = 1,$$

which implies that

$$P(x_0) = 1 \cdot f(x_0) + 0 \cdot f(x_1) = f(x_0) = y_0$$

and

$$P(x_1) = 0 \cdot f(x_0) + 1 \cdot f(x_1) = f(x_1) = y_1.$$

An example

Determine the linear Lagrange interpolating polynomial that passes through the points $(2, 4)$ and $(5, 1)$.

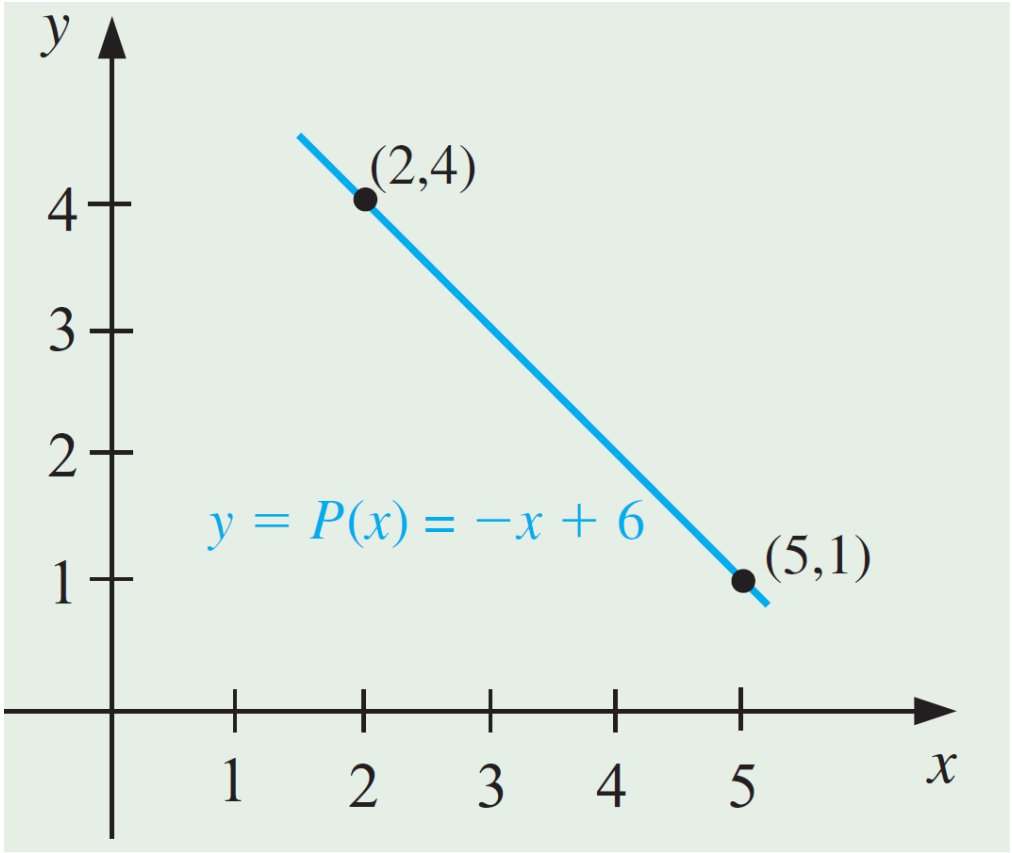
Solution

In this case we have

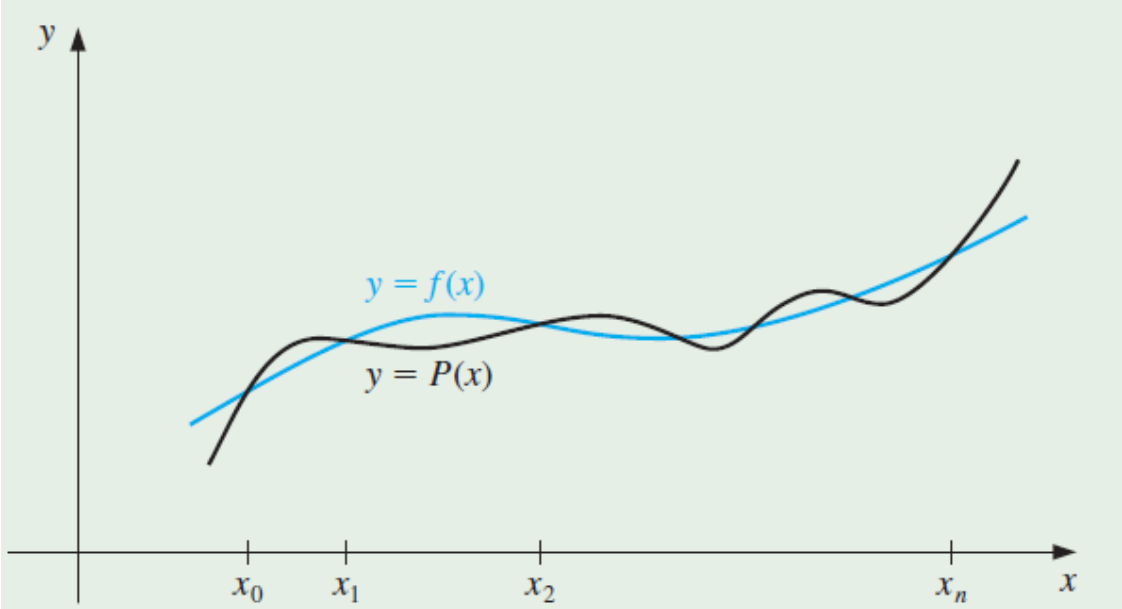
$$L_0(x) = \frac{x-5}{2-5} = -\frac{1}{3}(x-5) \quad \text{and} \quad L_1(x) = \frac{x-2}{5-2} = \frac{1}{3}(x-2),$$

so

$$P(x) = -\frac{1}{3}(x-5)\cdot 4 + \frac{1}{3}(x-2)\cdot 1 = -\frac{4}{3}x + \frac{20}{3} + \frac{1}{3}x - \frac{2}{3} = -x + 6.$$


y
4
3
2
1
(2,4)
y = P(x) = −x + 6
(5,1)
1
2
3
4
5
x

Lagrange polynomial: General case



To generalize the concept of linear interpolation, consider the construction of a polynomial of degree at most n that passes through the $n + 1$ points

$$(x_0, f(x_0)),\ (x_1, f(x_1)), \ldots, (x_n, f(x_n)).$$

Constructing the degree n polynomial

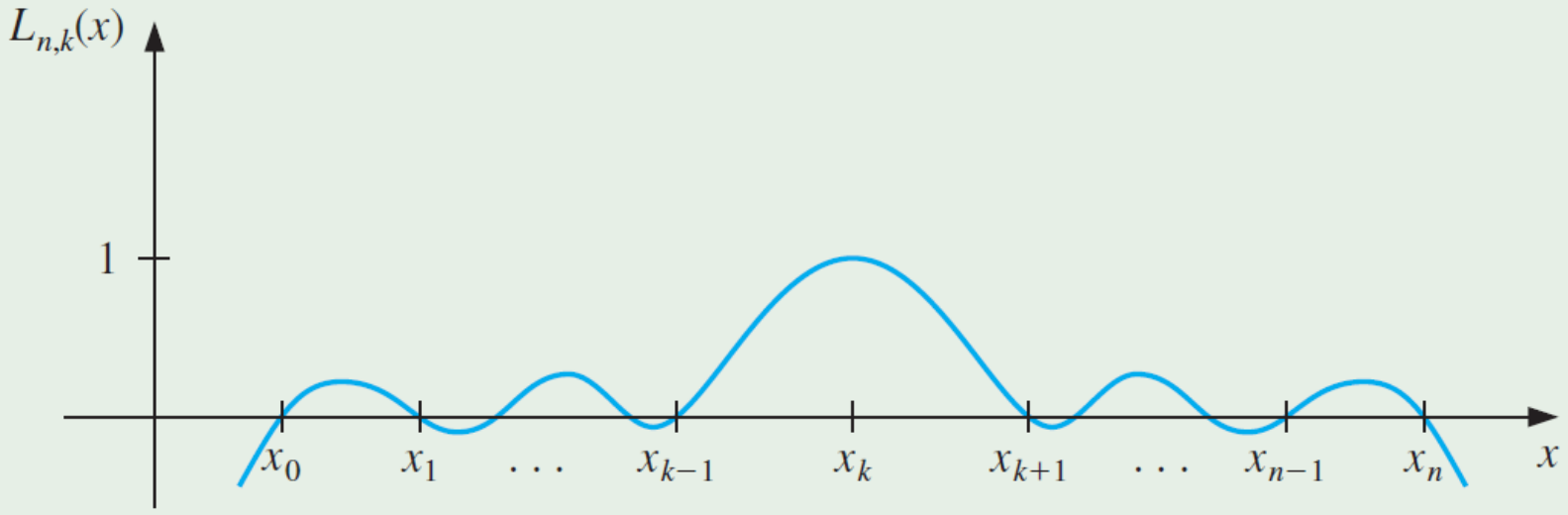
- We first construct, for each $k = 0, 1, \ldots, n$, a function $L_{n,k}(x)$ with the property that $L_{n,k}(x_i) = 0$ when $i \neq k$ and $L_{n,k}(x_k) = 1$.
- To satisfy $L_{n,k}(x_i) = 0$ for each $i \neq k$ requires that the numerator of $L_{n,k}(x)$ contain the term

$$(x - x_0)(x - x_1) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_n).$$

- To satisfy $L_{n,k}(x_k) = 1$, the denominator of $L_{n,k}(x)$ must be this same term but evaluated at $x = x_k$.
- Thus

$$L_{n,k}(x) = \frac{(x - x_0) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_n)}{(x_k - x_0) \cdots (x_k - x_{k-1})(x_k - x_{k+1}) \cdots (x_k - x_n)}.$$

$$L_{n,k}(x) = \frac{(x - x_0)\cdots(x - x_{k-1})(x - x_{k+1})\cdots(x - x_n)}{(x_k - x_0)\cdots(x_k - x_{k-1})(x_k - x_{k+1})\cdots(x_k - x_n)}.$$



If $x_0, x_1, \ldots, x_n$ are $n+1$ distinct numbers and f is a function whose values are given at these numbers, then a unique polynomial $P(x)$ of degree at most n exists with

$$f(x_k) = P(x_k), \quad \text{for each } k = 0, 1, \ldots, n.$$

This polynomial is given by

$$P(x) = f(x_0)L_{n,0}(x) + \cdots + f(x_n)L_{n,n}(x) = \sum_{k=0}^{n} f(x_k)L_{n,k}(x)$$

where, for each $k = 0, 1, \ldots, n$, $L_{n,k}(x)$ is defined as follows:

n-th Lagrange interpolating polynomial

6. Hermite interpolation

Problem:

Find a polynomial $H(x)$, such that

$$H(x_i) = f(x_i), \quad H'(x_i) = f'(x_i), \quad \text{for } i = 0, 1, 2, \cdots, n.$$

Theorem:

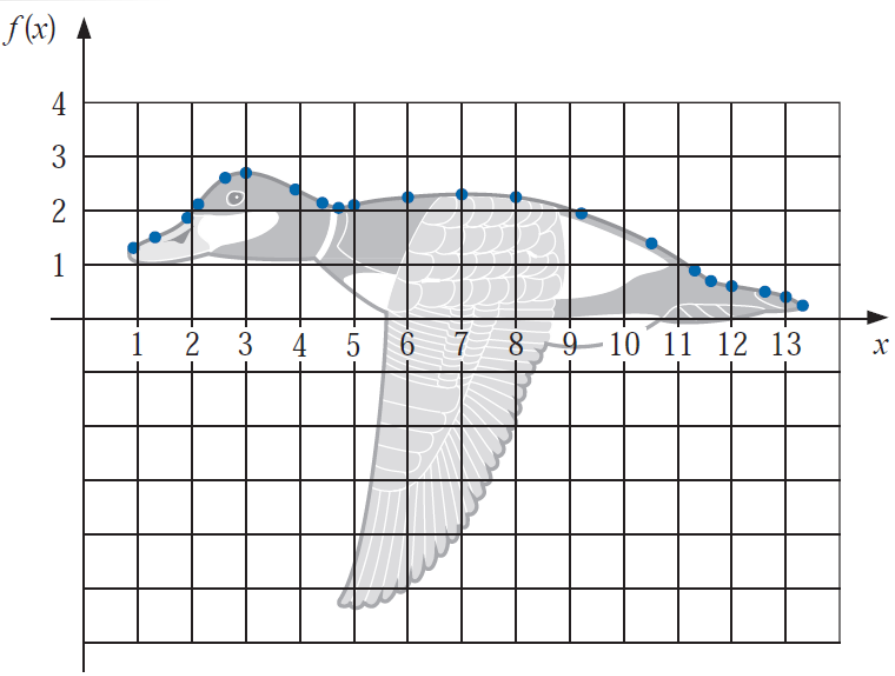
If $f \in C^1[a, b]$ and $x_0, \ldots, x_n \in [a, b]$ are distinct, the unique polynomial of least degree agreeing with f and f' at $x_0, \ldots, x_n$ is the Hermite polynomial of degree at most $2n + 1$ given by

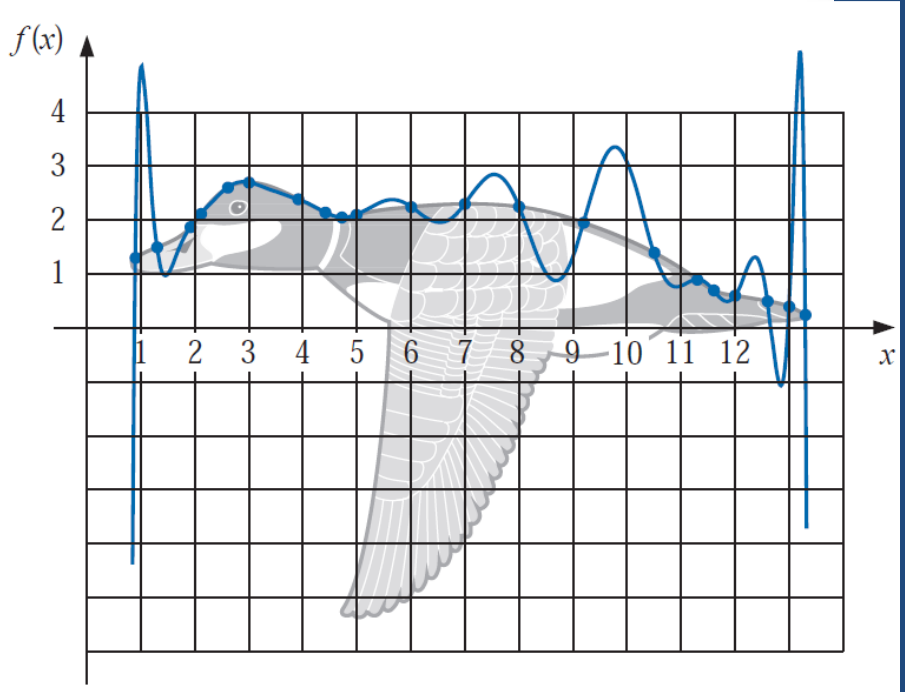
$$H_{2n+1}(x) = \sum_{j=0}^{n} f(x_j) H_{n,j}(x) + \sum_{j=0}^{n} f'(x_j) \hat{H}_{n,j}(x)$$

where, for $L_{n,j}(x)$ denoting the jth Lagrange coefficient polynomial of degree n, we have

$$H_{n,j}(x) = [1 - 2(x - x_j) L'_{n,j}(x_j)] L^2_{n,j}(x) \quad \text{and} \quad \hat{H}_{n,j}(x) = (x - x_j) L^2_{n,j}(x)$$

7. High-degree polynomials *vs* piecewise polynomial interpolation





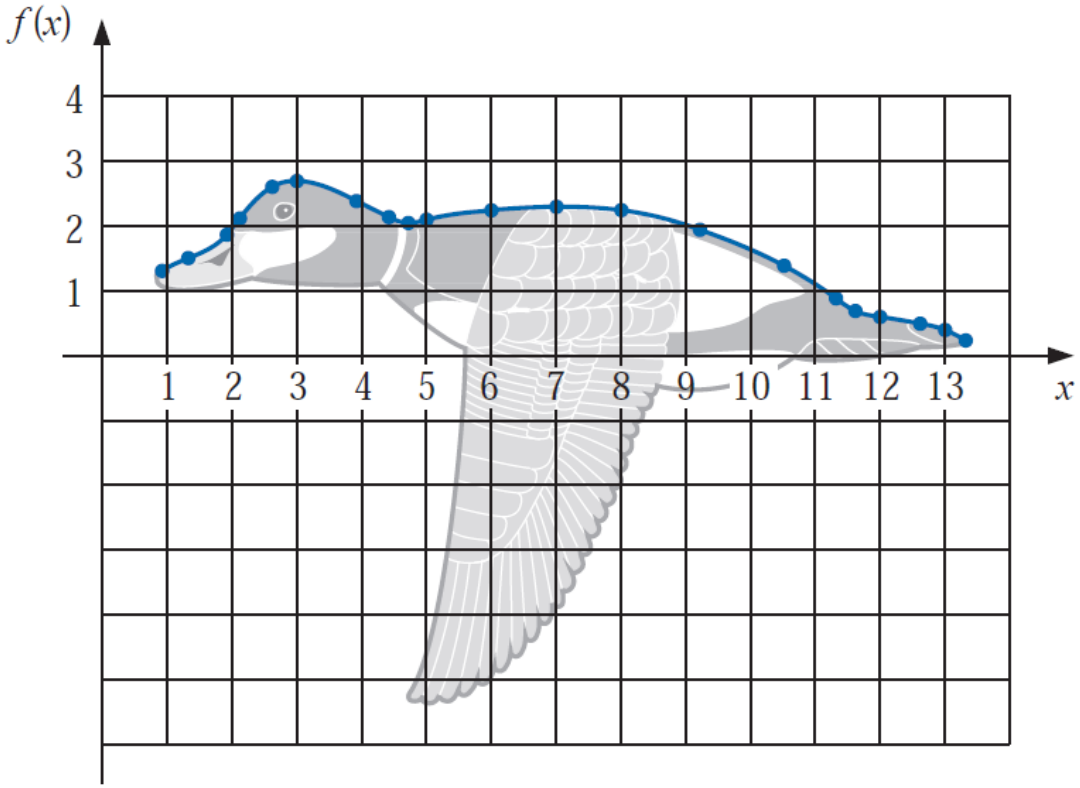
21 data points

x	0.9	1.3	1.9	2.1	2.6	3.0	3.9	4.4	4.7	5.0	6.0	7.0	8.0	9.2	10.5	11.3	11.6	12.0	12.6	13.0	13.3
$f(x)$	1.3	1.5	1.85	2.1	2.6	2.7	2.4	2.15	2.05	2.1	2.25	2.3	2.25	1.95	1.4	0.9	0.7	0.6	0.5	0.4	0.25

High-degree Lagrange interpolating polynomials (here 20-degree) are oscillating.

Alternative solution

- Piecewise low degree polynomial interpolation
- Dividing the interval into a collection of subintervals and use polynomial approximation n each subinterval.

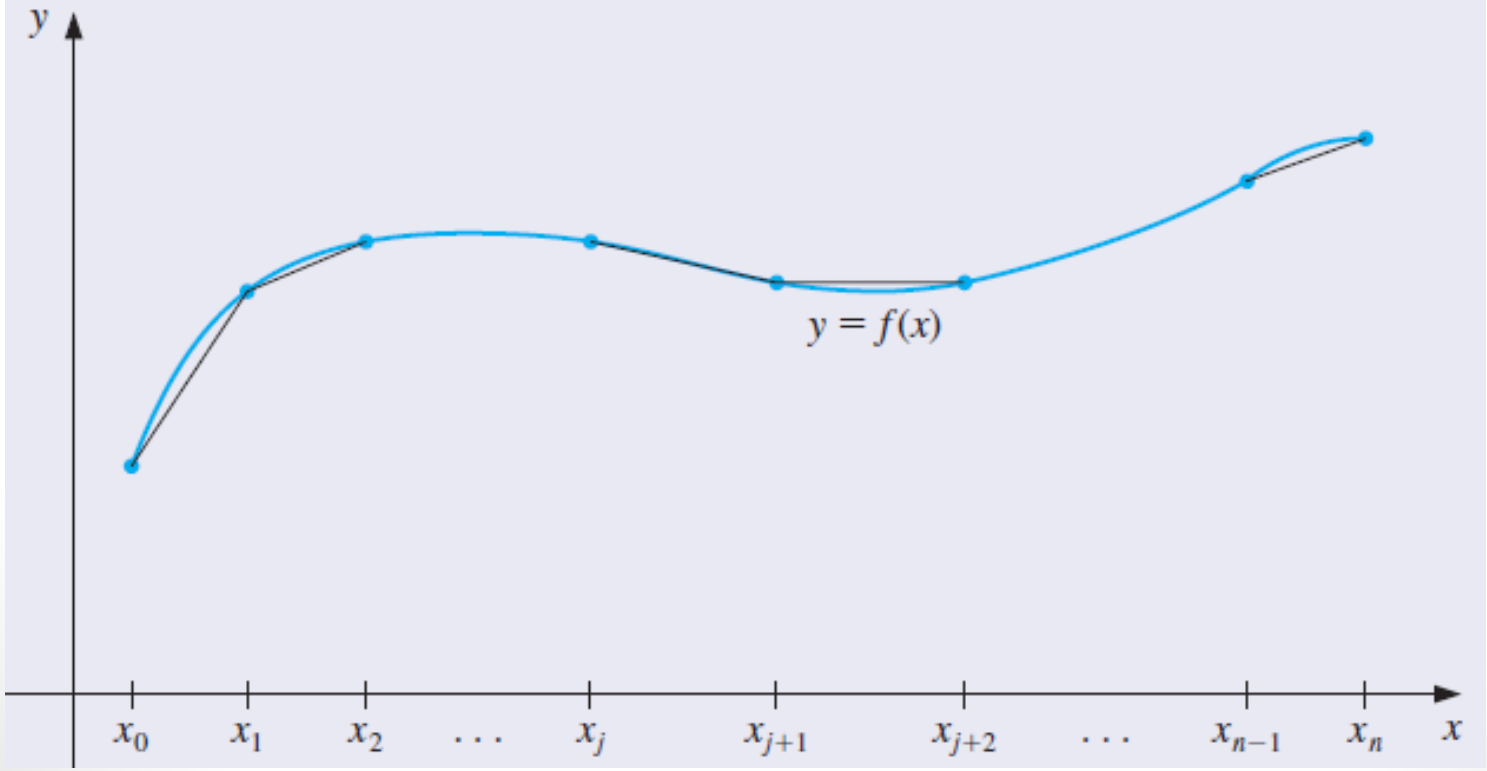


8. Piecewise-linear interpolation

This is the simplest piecewise-polynomial approximation and which consists of joining a set of data points

$$\{(x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_n, f(x_n))\}$$

by a series of straight lines:



Advantage: Simple

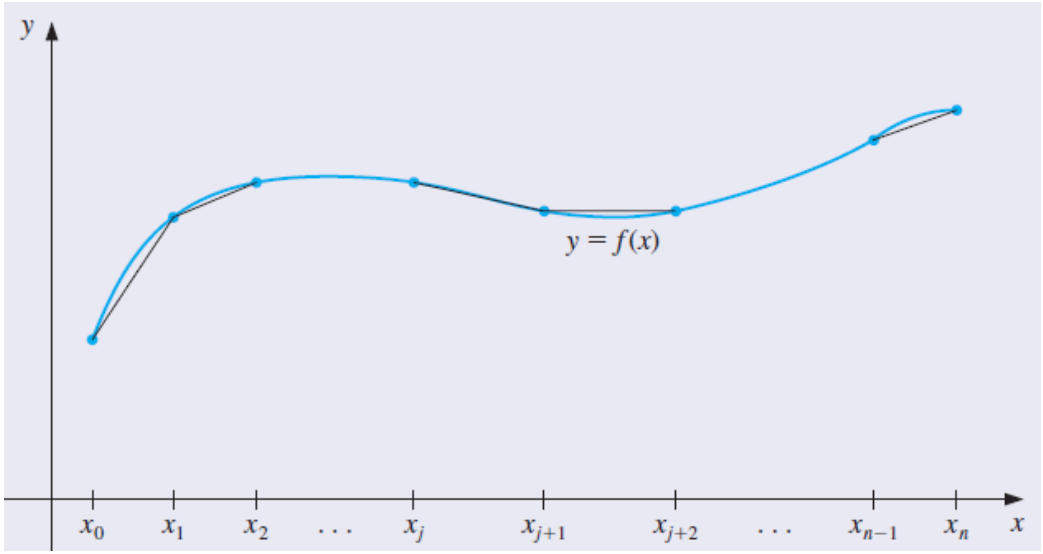
Disadvantage:

- There is likely no differentiability at the endpoints of the subintervals, which, in a geometrical context, means that the interpolating function is not “smooth.”
- Often it is clear from physical conditions that smoothness is required, so the approximating function must be continuously differentiable.

Piecewise polynomial of Hermite type?

if the values of f and of f' are known at each of the points $x_0 < x_1 < \cdots < x_n$

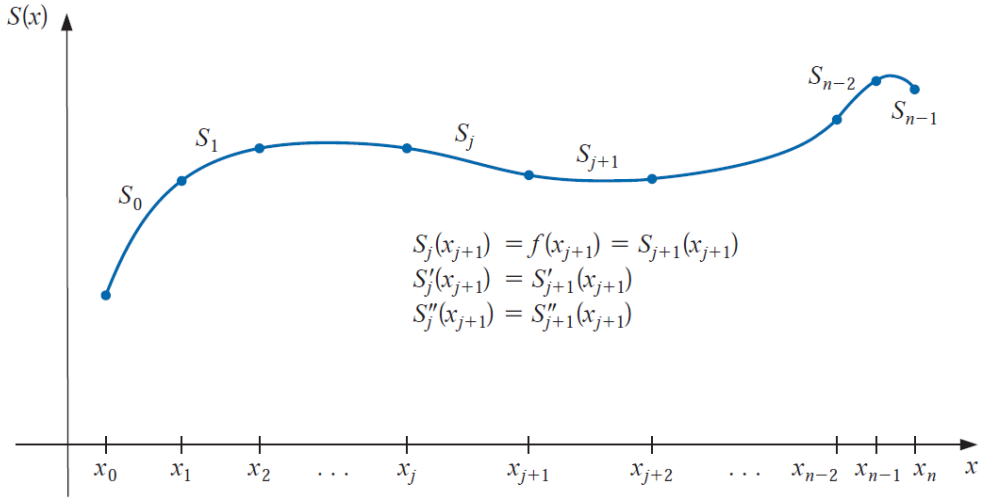
However, in general we don't know the values of derivative *f'* at these points.



9. Cubic spline interpolation

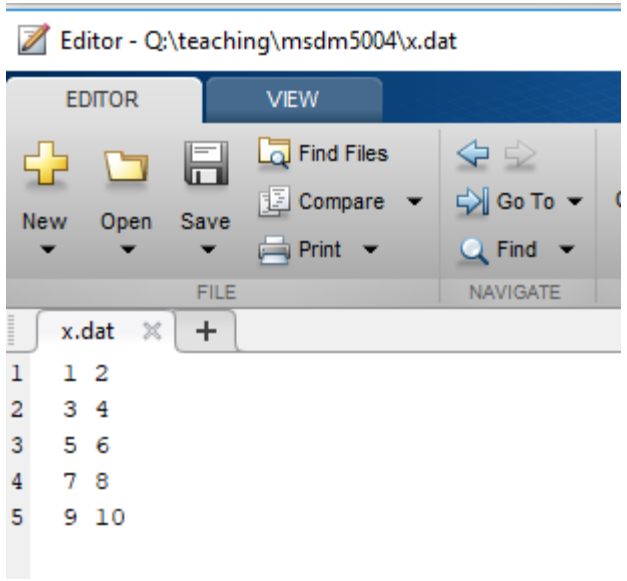
Cubic spline *S(x)* to approximate *f(x)*

- On each subinterval, *S(x)* is a polynomial of degree ≤ *3*.
- *S(x)=f(x)* at each node.
- *S'(x)*, *S''(x)* are continuous at each node (i.e. continuous across adjacent subintervals).
- Values of *f'(x)* and *f''(x)* at the nodes are not required.



MATLAB: Input data from a file

Generate a file x.dat



- variable name
- those after the dot (extension) will be neglected

```
>> load x.dat
>> x

x =

     1     2
     3     4
     5     6
     7     8
     9    10
>>
```

For more information

```
>> help load
```