# Quadrature based on polynomial interpolation

# Newton-Cotes rules

- *equidistant quadrature points* $x_i = a + ih$ where $h = (b-a)/n$ and $i = 0, \ldots, n$
- *quadrature rule:*

$$Q(f, \overset{\alpha}{a}, \overset{\beta}{b}) = \sum_{i=0}^{n} \overset{?}{w_i} f(x_i)$$

- Newton-Cotes method: choose *quadrature weights* $w_i$ such that

$$Q(p, a, b) = \int_a^b p(x)\, dx$$

for all polynomials $p$ of degree up to $n$

## special cases

- **rectangle rule** $n = 0$

$$Q(f, a, b) = (b - a)f(x_0)$$

  for any quadrature point $x_0 \in [a, b]$, weight $w_0 = b - a$

  - **midpoint rule** $x_0 = (a + b)/2$

- **trapezoidal rule** $n = 1$

$$Q(f, a, b) = \frac{b-a}{2} f(a) + \frac{b-a}{2} f(b)$$

  quadrature points $x_0 = a$, $x_1 = b$, weights
  $w_0 = w_1 = (b - a)/2$

# integer quadrature points for $n \geq 1$

- approximation of
$$I(f) = \int_0^n f(x)\, dx$$
- quadrature rule
$$Q(f) = \sum_{k=0}^{n} w_k f(k)$$
- note that this does not give a rule for $n = 0$ as $\int_0^0 f(x)\, dx = 0$, for this we consider "open" rules which approximate
$$I(f) = \int_0^{n+1} f(x)\, dx$$
by $Q(f)$ with the same points $x_k = k$ but adapted weights $w_k$ for $k = 0, \ldots, n$

# transformation formula

- introduce variable $z \in [0, n]$ such that

$$x = a + zh$$

  where $h = (b - a)/n$
- transformed function $g(z)$ satisfying

$$g(z) = f(a + zh)$$

- it follows that

$$\int_a^b f(x)\,dx = h \int_0^n g(z)\,dz$$

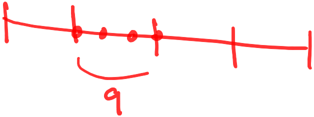
  and one gets the transformed quadrature rule

$$Q(f) = h \sum_{k=0}^{n} w_k f(a + kh)$$

q

  where $w_k$ are the Newton-Cotes weights for the interval $[0, a]$

# composite Newton-Cotes rules




- choose $N = nm$ quadrature points $x_k = a + kh$ where $h = (b - a)/N$
- composite formula

$$Q(f) = h \sum_{j=0}^{m} \sum_{k=0}^{n} w_k f(a + x_{k+jn})$$

where $w_k$ are the weights for the Newton Cotes on the interval $[0, n]$

# computing the weights $w_k$ from the Lagrange interpolation formula

- we only need to consider the interval $[0, n]$
- choose quadrature formula defined by

$$Q(f) = \int_0^n p(x)\,dx$$

  where $p$ is the interpolating polynomial at $x_k = k$ for $k = 0, \dots, n$
- Lagrange interpolation formula

$$p(x) = \sum_{k=0}^{n} l_k(x) f(k)$$

Lagrange polynomials

- integrate this formula to get the weights

$$w_k = \int_0^n l_k(x)\,dx$$

# example $n = 1$ – trapezoidal rule

- Lagrange functions

$$l_0(x) = 1 - x, \quad l_1(x) = x$$

- weights

$$w_0 = \int_0^1 (1 - x)\, dx = 1/2, \quad w_1 = \int_0^1 x\, dx = 1/2$$

# example $n = 2$ – Simpson's rule

- Lagrange (or cardinal) functions

$$l_0(x) = (x-1)(x-2)/2, \quad l_1(x) = -x(x-2), \quad l_2(x) = x(x-1)/2$$

- weights

$$w_0 = \int_0^2 l_0(x)\,dx = \int_0^2 (x^2 - 3x + 2)/2 = 1/3$$
$$w_1 = \int_0^2 l_1(x)\,dx = 4/3$$
$$w_2 = \int_0^2 l_2(x)\,dx = 1/3$$

# computing the weights using the method of unknown coefficients

- set up linear system of $n + 1$ equations for the $w_j$ from the conditions

$$Q(x^j) = \int_0^n x^j \, dx, \quad j = 0, \dots, n$$

- equations

$$\boxed{\sum_{k=0}^{n} k^j w_k = \frac{n^{j+1}}{j+1},} \quad j = 0, \dots, n$$

- matrix is Vandermonde matrix

# example $n = 1$ trapezoidal rule

- equations

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1/2 \end{bmatrix}$$

- weights

$$w_1 = 1/2, \quad w_0 = 1 - 1/2 = 1$$

# example $n = 2$ Simpson's rule

- equations

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 1 & 4 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 8/3 \end{bmatrix}$$

- with Gauss elimination we get

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 2/3 \end{bmatrix}$$

- the solution by back substitution is as before $w_0 = 1/3$, $w_1 = 4/3$ and $w_2 = 1/3$

```
# compute Newton-Cotes weights with sympy
w = np.zeros((4,9))
for i, n in enumerate((1,2,4,8)):
    x = sy.Symbol('x')
    print("\n n = {}:".format(n),end='    ')
    for j in range(n+1):
        lj = 1
        for k in range(n+1):  # compute Lagrangian polynom
            if (k!=j): lj *= (x-k)/(j-k)
        w[i,j] = float(sy.integrate(lj,(x,0,n)))
        print("w{} = {:4.2f}".format(j,w[i,j]),end='    ')
```

```
n = 1:   w0 = 0.50   w1 = 0.50
n = 2:   w0 = 0.33   w1 = 1.33   w2 = 0.33
n = 4:   w0 = 0.31   w1 = 1.42   w2 = 0.53   w3 = 1.42   w
n = 8:   w0 = 0.28   w1 = 1.66   w2 = -0.26   w3 = 2.96
```

# example: use weights to compute $I = \int_0^1 \exp(-x)\,dx$

- exact value $I = 1 - e^{-1}$
- approximations with Newton-Cotes

```
for i, n in enumerate((1,2,4,8)):
    Q = np.sum(w[i,:n+1]*np.exp(-np.linspace(0,1,n+1)))/n
    print(("n = {}, Q = {:8.7f}, Error = {:6.1e}"\
           .format(n,Q,Q-1+1.0/np.e)))
```

```
n = 1, Q = 0.6839397, Error = 5.2e-02
n = 2, Q = 0.6323337, Error = 2.1e-04
n = 4, Q = 0.6321209, Error = 3.2e-07
n = 8, Q = 0.6321206, Error = 3.6e-13
```

# Errors

# general formula for interval $[0, n]$

- error of $n$ degree interpolating polynomial $p$ for interpolation points $x_k = k$

$$e(x) = p(x) - f(x) = -\frac{1}{(n+1)!} f^{(n+1)}(\xi_x) w(x)$$

where $w(x) = \prod_{i=0}^{n}(x-k)$

- error of quadrature equals integral of interpolation error

$$E = \int_0^n e(x)\,dx = \frac{1}{(n+1)!}\int_0^n f^{(n+1)}(\xi_x) w(x)\,dx$$

- mean value theorem gives

$$|E| \leq \frac{1}{(n+1)!} |f^{(n+1)}(\xi)| \int_0^n |w(x)|\,dx$$

for some $\xi \in [0, n]$

## example: $\int_0^1 \exp(-x)\,dx$

- transformation from $[0, n]$ to $[0, 1]$

$$f(x) = \exp(-x/n)$$

- value of integral

$$I = \int_0^1 \exp(-x)\,dx = n^{-1} \int_0^n \exp(-x/n)\,dx$$

- quadrature ($w_k$ are weights for $[0, n]$)

$$Q = n^{-1} \sum_{k=0}^{n} w_k \exp(-k/n)$$

- derivatives for error bounds

$$f^{(k)}(x) = (-n)^{-k} \exp(-x/n)$$

## example: error for $n = 1$, $f(x) = \exp(-x)$ and $x \in [0, 1]$

- one has for some $\xi \in [0, 1]$:

$$|E| \leq \frac{1}{2} \exp(-\xi) \int_0^1 x(1 - x)\, dx \leq \frac{1}{12} \approx 0.08$$

where the actual error is 0.05 (see computation done previously)

## example error for $f(x) = \exp(-x)$, $x \in [0,1]$ and general $n$

- error bound

$$|E| \leq \frac{h^{n+2}}{(n+1)!} \int_0^n |w(x)| \, dx$$

- compute $\int_0^n |w(x)| \, dx$ with sympy

```
x = sy.Symbol('x'); wn = 5*[1,]; wint = np.zeros(5)
for i,n in enumerate((1,2,3,4,5)):
    for j in range(n+1):
        wn[i] = wn[i]*(x-j)
    for j in range(n):
        wint[i] += np.abs(float(\
                sy.integrate(wn[i],(x,j,j+1))))
```

# compute the weights for $n = 1, 2, 3, 4, 5$

```
# compute Newton-Cotes weights with sympy
w = np.zeros((5,6));      x = sy.Symbol('x')
for i, n in enumerate((1,2,3,4,5)):
    for j in range(n+1):
        lj = 1
        for k in range(n+1):
            if (k!=j):
                lj *= (x-k)/(j-k)
        w[i,j] = float(sy.integrate(lj,(x,0,n)))
```

Lagrange

# compute the error bounds for $n = 1, 2, 3, 4, 5$

```
Ebound = np.zeros(5); E = np.zeros(5)
for i,n in enumerate((1,2,3,4,5)):
    h = 1.0/n
    Ebound[i] = h**(n+2)/math.factorial(n+1)*wint[n-1]
    E[i] = h*np.sum(w[n-1,:n+1]*\
        np.exp(-np.linspace(0,1,n+1))) - 1.0 + 1.0/np.e
    print("n = {}, error = {:4.2e}, bound = {:4.2e}"\
                .format(n,E[i], Ebound[i]))
```

```
n = 1, error = 5.18e-02, bound = 8.33e-02
n = 2, error = 2.13e-04, bound = 5.21e-03
n = 3, error = 9.50e-05, bound = 2.80e-04
n = 4, error = 3.16e-07, bound = 1.29e-05
n = 5, error = 1.78e-07, bound = 5.20e-07
```

# error bounds for even $n$

- note: error bounds for $n = 2$ and $n = 4$ are bad
- as $\int_0^n w(x)\,dx = 0$ for even $n$, quadrature exact for polynomials of degree $n + 1$ in this case
- Taylor expansion for $f^{(n+1)}(\xi)$:

$$f^{(n+1)}(\xi) = f^{(n+1)}(1/2) + (\xi - 1/2) f^{(n+2)}(\eta)$$

- in the case of even $n$ the first (constant) term does not contribute to the error and one gets for a general function $f(x)$

$$|E| \le \frac{h^{n+3}}{2(n+1)!} |f^{(n+2)}(\xi)| \int_0^n |w(x)|\,dx$$

and for our example $f(x) = \exp(-x)$:

$$|E| \le \frac{h^{n+3}}{2(n+1)!} \int_0^n |w(x)|\,dx$$

# recompute the error bounds for $n = 2, 4$

```
Ebound = np.zeros(2); E = np.zeros(2)
for i,n in enumerate((2,4)):
    h = 1.0/n
    Ebound[i] = h**(n+3)/(2*math.factorial(n+1))\
                *wint[n-1]
    E[i] = h*np.sum(w[n-1,:n+1]\
     *np.exp(-np.linspace(0,1,n+1))) - 1.0 + 1.0/np.e
    print("n = {}, error = {:4.2e}, bound = {:4.2e}"\
          .format(n,E[i], Ebound[i]))
```

```
n = 2, error = 2.13e-04, bound = 1.30e-03
n = 4, error = 3.16e-07, bound = 1.61e-06
```