Class Prep 10 | 5.2.1 – 5.3.2

Riley Primeau

2022-11-07

Section 5.2.1 Multipanel Interpolation Rules

```
midpt = function(f, a, b, m = 100) {
  nwidth = (b-a)/m
  x = seq(a, b - nwidth, length.out = m) + nwidth/2
  y = f(x)

  area = sum(y) * abs(b-a)/m
  return(area)
}

f = function(x) {x^2}
midpt(f, 0, 1, m =2)

## [1] 0.3125

midpt(f, 0, 1, m=10)

## [1] 0.3325

midpt(f, 0, 1, m=100)

## [1] 0.333325

midpt(f, 0, 1, m=1000)

## [1] 0.3333332

simp = function(f, a, b, m=100){
  x.ends = seq(a, b, length.out = m+1)
  y.ends = f(x.ends)
  x.mids = (x.ends[2:(m+1)] - x.ends[1:m])/2 + x.ends[1:m]
  y.mids = f(x.mids)

  p.area = sum(y.ends[2:(m+1)] + 4 * y.mids[1:m] + y.ends[1:m])
  p.area = p.area * abs(b-a)/(6*m)
  return(p.area)
}
```

```
library(cmna)

# The simp38 function returns different results then in the book. This
# commenting out and inclusion of the cmna library is just to convince you,
# and myself, that the function simp38 correctly defined. Both my definition
# and the cmna library definition returned the same results.

# simp38 = function(f, a, b, m=100){
#   x.ends = seq(a, b, length.out = m+1)
#   y.ends = f(x.ends)
#   x.midh = (2 * x.ends[2:(m+1)] + x.ends[1:m])/3
#   x.midl = (x.ends[2:(m+1)] + 2 * x.ends[1:m])/2
#   y.midh = f(x.midh)
#   y.midl = f(x.midl)
#
#   p.area = sum(y.ends[2:(m+1)] + 3 * y.midh[1:m] + 3 * y.midl[1:m] +
y.ends[1:m])
#   p.area = p.area * abs(b-a)/(8*m)
#
#   return(p.area)
# }

f = function(x) {x^2}
midpt(f, 0, 1, m =1)

## [1] 0.25

trap(f, 0, 1, m =1)

## [1] 0.5

simp(f, 0, 1, m =1)

## [1] 0.3333333

simp38(f, 0, 1, m =1)

## [1] 0.3333333

f = function(x) {x^4 - x^2 + 1}
midpt(f, 0, 1, m =1)

## [1] 0.8125
```

```r
trap(f, 0, 1, m =1)

## [1] 1

simp(f, 0, 1, m =1)

## [1] 0.875

simp38(f, 0, 1, m =1)

## [1] 0.8703704

f = function(x) {sin(x) + cos(x)}
midpt(f, 0, pi, m =10)

## [1] 2.008248

trap(f, 0, pi, m =10)

## [1] 1.983524

simp(f, 0, pi, m =10)

## [1] 2.000007

simp38(f, 0, pi, m =10)

## [1] 2.000003

simp(f, 0, pi, m =2)

## [1] 2.00456

simp(f, 0, pi, m =5)

## [1] 2.00011

simp(f, 0, pi, m =10)

## [1] 2.000007

simp(f, 0, pi, m =100)

## [1] 2

simp38(f, 0, pi, m =2)

## [1] 2.00201

simp38(f, 0, pi, m =5)

## [1] 2.000049

simp38(f, 0, pi, m =10)

## [1] 2.000003
```

```
simp38(f, 0, pi, m =100)

## [1] 2

simp38(f, 0, pi, m =1000)

## [1] 2
```

Section 5.2.2 Newton-Cotes Errors

No code in section.

Section 5.2.3 Newton-Cotes Forms, Generally

```
trap(log, 0, 1, m=10)

## [1] -Inf

simp(log, 0, 1, m=10)

## [1] -Inf

midpt(f, 0, 1, m=10)

## [1] 1.301711

midpt(f, 0, 1, m=100)

## [1] 1.301174

midpt(f, 0, 1, m=1000)

## [1] 1.301169

f = function(x) { 1/x }
trap(f, 0, 1, m=100)

## [1] Inf

midpt(f, 0, 1, m=10)

## [1] 4.266511

midpt(f, 0, 1, m=100)

## [1] 6.568684

midpt(f, 0, 1, m=1000)

## [1] 8.871265
```

Section 5.3.1 The Gaussian Method

```r
gaussint = function(f, x, w){
  y = f(x)

  return(sum(y*w))
}
```

Section 5.3.2 Implementation Details

```r
w = c(1,1)
x = c(-1/sqrt(3), 1/sqrt(3))
f = function(x) {x^3 + x + 1}

gaussint(f, x, w)

## [1] 2

trap(f, -1, 1, m=1)

## [1] 2

gaussint(cos, x, w)

## [1] 1.675824

trap(cos, -1, 1, m=1)

## [1] 1.080605

gauss.legendre = function(f, m=5){
  p = paste("gauss.legendre.", m, sep = " ")
  params = eval(parse(text = p))

  return(gaussint(f, params$x, params$w))
}
```