PARALLEL COMPUTING USING R AND A RASPBERRY PI CLUSTER PART I: THE NEED FOR SPEED!

KEITH E EMMERT AND PETER WHITE
TARLETON STATE UNIVERSITY
DEPARTMENT OF MATHEMATICS
BOX T-0470, STEPHENVILLE, TX 76402
EMMERTTARLETON.EDU
WHITETARLETON.EDU

1 Introduction

A standard staple in any first calculus course is approximation of integrals using the midpoint method. See, for example, Hass et al. [2022]. Many times projects are based upon approximating strange functions, see Crandall [1994]. Improvements over these basic methods can be found in Press et al. [1992]. Of course, the need to approximate integrals of functions does not stop with calculus. The need for good approximations remains when studying differential equations, partial differential equations, and other, higher level applied mathematics courses Zill [2024] or DuChateau and Zachman [2002].

In this paper, we implement the classic midpoint method using R. Even when considering a very friendly function such as hyperboloids, the limits of a modern computer is quickly reached.

A project such as this could serve to introduce students into the exciting field of numerical approximation while also highlighting some of the limitations.

2 Midpoint Method on $\mathbb{R}^D$.

Consider a function $f : \mathbb{R}^D \mapsto \mathbb{R}$ which is continuous over $\Omega = \prod_{i=1}^{D}[a_i, b_i] \subseteq \mathbb{R}^D$. The goal is to approximate integrals of the form

$$\mathcal{J} = \int_{a_D}^{b_D} \cdots \int_{a_2}^{b_2} \int_{a_1}^{b_1} f(x_1, x_2, \ldots, x_D)\, \mathrm{d}x_1\, \mathrm{d}x_2 \cdots \mathrm{d}x_D\,.$$

Let N be a positive integer and, for simplicity, assume that for $i = 1, 2, \ldots, D$, $\Delta x_i = \dfrac{b_i - a_i}{N}$. A regular partition of Ω can then be formed with $x_{ij} = a_i + j\Delta x_i$ where $i = 1, 2, \ldots D$ and $j = 0, 1, 2, \ldots, N$. The midpoints of a regular partition of size N of $[a_i, b_i]$ are found via

$$m_{k,i_k} = \frac{x_{k,i_k} + x_{k,i_k-1}}{2},$$

for $k = 1, 2, \ldots, D$ and $i_k = 1, 2, \ldots, N$. Then, the D-dimensional midpoint rule with N^D points is given by

$$\mathcal{J} \approx M_{N^D} = \sum_{i_D=1}^{N} \cdots \sum_{i_2=1}^{N} \sum_{i_1=1}^{N} f(m_{1,i_1}, m_{2,i_2}, \ldots, m_{D,i_D})\Delta x_1 \Delta x_2 \cdots \Delta x_D. \tag{1}$$

The error bound for the midpoint method is $\mathcal{O}\left(\frac{1}{N^{2/D}}\right)$. Let $\epsilon = \frac{c}{N^{2/D}}$ be the error, where $c > 0$ is a constant which depends upon the domain Ω as well as the given function, f. Thus, we see that the number of points required to be within a tolerance of ϵ is given by

$$N > \left\lceil \left(\frac{c}{\epsilon}\right)^{D/2} \right\rceil, \tag{2}$$

Now, let $0 < \eta < 1$ be given. In order to reduce the error ϵ by a factor of η, we need to increase the number of points to $\hat{N}$, so that

$$\frac{c}{\hat{N}^{2/D}} < \eta\epsilon \Longrightarrow \hat{N} > \left\lceil \frac{N}{\eta^{D/2}} \right\rceil = \left\lceil N \cdot \eta^{-D/2} \right\rceil.$$

Thus, given $0 < \eta < 1$, we know that N increases by a factor of $\eta^{-D/2} > 1$. Of course, this is a worst case scenario. The actual $\hat{N}$ required to match the new error of $\eta\epsilon$ will vary based upon the function and Ω. See Table 1.

Table 1: Worst case scenario for computing the multiple of N, $\eta^{-D/2}$, required to reduce the error of M_N by a factor of η given the dimension of the domain space, D. The new number of rectangles is given by $\lceil \eta^{-D/2} \cdot N \rceil$.

Dimension	$\eta = 0.1$	$\eta = 0.01$	$\eta = 0.001$
1	3.16	10	31.62
2	10.00	100	1,000.00
3	31.62	1,000	31,622.78
4	100.00	10,000	1,000,000.00
5	316.23	100,000	31,622,776.60
6	1,000.00	1,000,000	1,000,000,000.00
7	3,162.28	10,000,000	31,622,776,601.68
8	10,000.00	100,000,000	1,000,000,000,000.00
9	31,622.78	1,000,000,000	31,622,776,601,683.79
10	100,000.00	10,000,000,000	1,000,000,000,000,000.00

Thus, for larger values of D, the curse of dimensionality cripples this method.

2.1 Example where the Domain is a Subset of $\mathbb{R}$

Consider the function $f(x) = 1 - x^2$ over $-1 \leq x \leq 1$. The exact value of the integral is

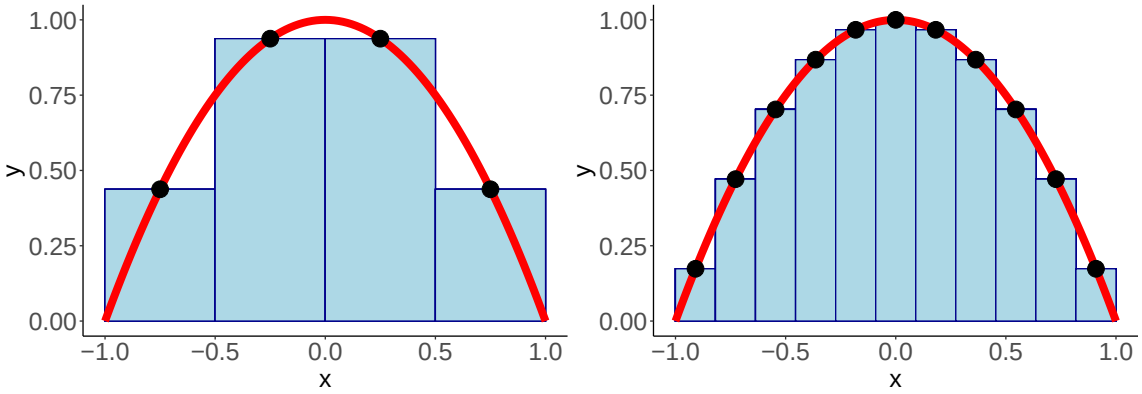
$$\int_{-1}^{1} (1 - x^2)\, \mathrm{d}x = \frac{4}{3}.$$

The figures and midpoint estimates are all based upon the code found in Listing 1. Here, Equation (1) reduces to

$$M_N = \sum_{i=1}^{N} f(m_i)\Delta x,$$

where m_i represent the midpoints of the regular partition of size N of $[-1, 1]$, for $i = 1, 2, \ldots, N$ and Δx the width of the rectangles. Note that most any calculus book such as Hass et al. [2022] will cover the midpoint method as well as the calculation of the constant, c, found in Equation (2).

Let the number of rectangles be $N = 4$. The approximating rectangles are shown in Figure 1 on the left. Applying the midpoint rule, we obtain $M_{11} = 1.375$. For Figure 1 on the right, we obtain $M_{11} = 1.338843$. However, $M_{10} = 1.34$, so at least 11 rectangles are needed for this example of two digits of accuracy after the decimal point. Note that since 4 rectangles gave us accuracy of $\epsilon = 0.1$, Table 1 guarantees a second digit of accuracy (i.e. $\eta = 0.1$) when $\lceil 4 \cdot 3.16 \rceil = \lceil 12.64 \rceil = 13$ rectangles are utilized. In fact, Note that $M_{13} = 1.3372781$, as predicted by the table, gives another digit of accuracy. But as we see in this example, we needed fewer rectangles.



(a) The comparison of $M_4 = 1.375$

(b) The comparison of $M_{11} = 1.3388$

Figure 1: Comparison of M_4 to M_{11}. The true value of the integral is $\frac{4}{3} \approx 1.3333$.

2.2 Example where the Domain is a Subset of $\mathbb{R}^2$

Consider the paraboloid $f(x, y) = 2 - x^2 - y^2$ over the domain $\Omega = [-1, 1] \times [-1, 1] = [-1, 1]^2$ with $N = 4$.

$$\int_{-1}^{1} \int_{-1}^{1} f(x, y)\, dx\, dy = \frac{16}{3} \approx 5.3333\cdots$$

with graph shown in Figure 2. Similarly, the midpoint method reduces Equation (1) to

$$M_{N^2} = \sum_{i=1}^{N} \sum_{j=1}^{N} f(m_{x,j}, m_{y,i}) \Delta x \Delta y.$$

For our example, $\Delta x = \Delta y$ and $m_{x,k} = m_{y,k}$ are midpoints of a regular partition of size N of $[-1, 1]$ for $k = 1, 2, \ldots, N$.

Note that Listing 3 calculates midpoints estimates using Ω. Listings 2 and 4 generate the three-dimensional plot of $y = f(x, y)$ and the two-dimensional grid of midpoints in Ω,

A typical $N \times N$ grid when $N = 4$ for the midpoint method $M_{N^2} = M_{16}$ is illustrated in Figure 3. Note that applying the midpoint method yields $M_{16} = 5.5$, a somewhat underwhelming estimate as it doesn't even match on digit after the decimal. From Table 1, we are guaranteed to match one place after the decimal if we increase N by a factor of 10 to 40. In fact, we have $M_{40^2} = M_{1600} = 5.335$. Again, we've extended more than is required due to the symmetry involved with our chosen function. In fact, if we consider Table 2, we see the progression of accuracy as N increases and, for this example, we need far fewer points than the worst case scenario predicts. In particular, we see we need $N^2 = 7^2 = 49$ points. And by $N^2 = 21^2 = 441$ points, we have our second place of accuracy.

2.3 Example where the Domain is a Subset of $\mathbb{R}^{10}$

The main example for this paper is using the midpoint rule to estimate the integral of the function

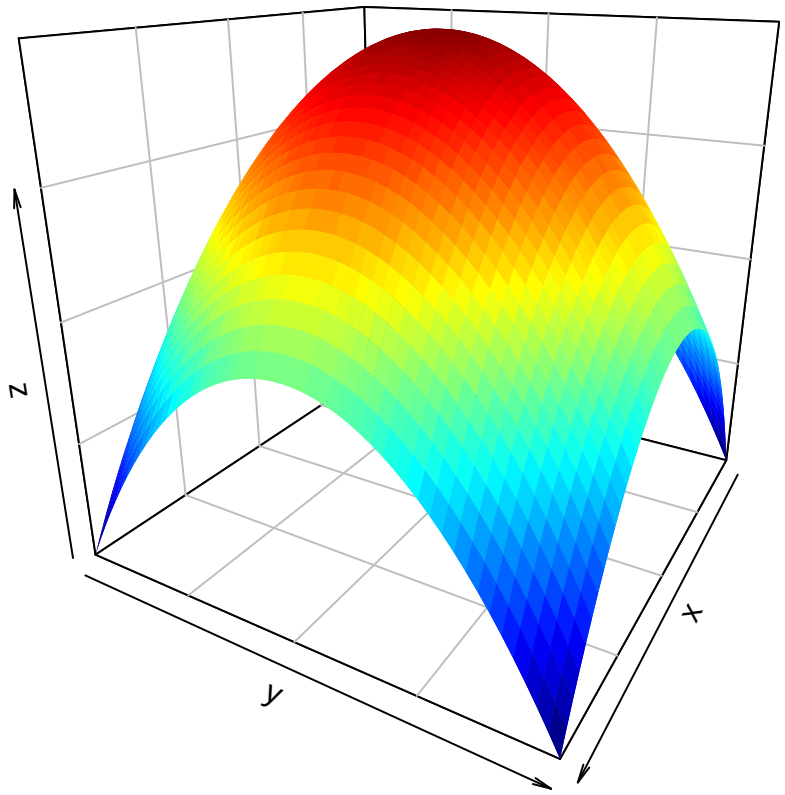


Figure 2: Plot of $f(x, y) = 2 - x^2 - y^2$.

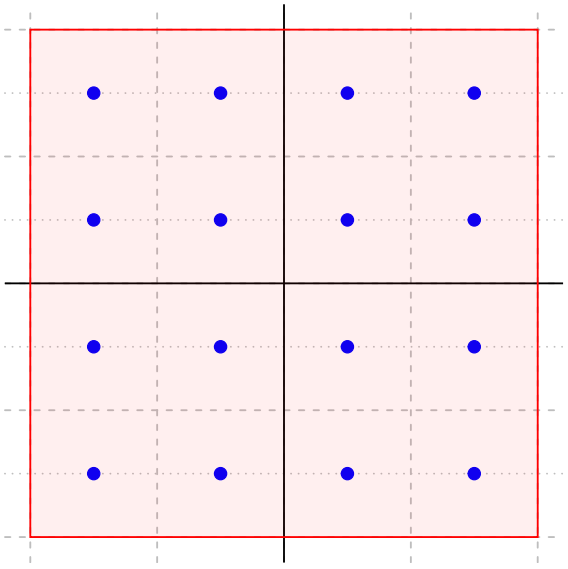


Figure 3: Four by four grids for midpoint rule $M_{16} = 5.5$ with the true value of the integral equal to $16/3 \approx 5.3333...$. Red square represents $\Omega = [-1, 1]^2$.

Table 2: Estimates for M_{N^2} for variuous values of N when considering $\Omega = [-1,1]^2$. The true value is $\frac{16}{3} = 5.333\cdots$.

N	M_{N^2}	N	M_{N^2}	N	M_{N^2}	N	M_{N^2}
4	5.500000	9	5.366255	14	5.346939	19	5.340720
5	5.440000	10	5.360000	15	5.345185	20	5.340000
6	5.407407	11	5.355372	16	5.343750	21	5.339380
7	5.387755	12	5.351852	17	5.342561	22	5.338843
8	5.375000	13	5.349112	18	5.341564	23	5.338374

$$f(x_1, x_2, \ldots, x_{10}) = 10 - x_1^2 - x_2^2 - \cdots - x_{10}^2$$

over the domain $\Omega = [-1,1]^{10}$. Again, we have a paraboloid and a lot of symmetry. However, $D = 10$ and this is crippling to the midpoint method. Please note that the truth is given by

$$\begin{aligned}\int_{-1}^{1}\cdots\int_{-1}^{1}\int_{-1}^{1}(10 - x_1^2 - x_2^2 - \cdots - x_{10}^2)\,dx_1\,dx_2\cdots dx_{10} \\ = \frac{20,480}{3} = 6,826.6666\cdots.\end{aligned}$$

Once again, Equation (1) reduces to

$$M_{N^{10}} = \sum_{i_{10}=1}^{N}\cdots\sum_{i_2=1}^{N}\sum_{i_1=1}^{N} f(m_{1,i_1}, m_{2,i_2}, \ldots, m_{10,i_{10}})\Delta x_1\Delta x_2\cdots\Delta x_{10}, \tag{3}$$

where m_{k,i_k} represents midpoints of a regular partition of size N of $[-1,1]$ which yields a total number of points of N^{10}.

The code used is found in Listing 5.

Note that with $D = 10$, we have an $\mathcal{O}\left(\frac{1}{N^{2/D}}\right) = \mathcal{O}\left(\frac{1}{N^{1/5}}\right)$ rate of convergence. Choosing our typical $N = 4$, we have $N^{10} = 4^{10} = 1048576$ points. The estimate is $M_{4^{10}} = 7040$ which is somewhat underwhelming. Consulting Table 2, we need to increase N by a factor of 100,000 to guarantee an increase of one place of accuracy. That is, we need $400,000^{10} = 1.6 \cdot 10^{11}$. Don't try this at home! Consider Table 3 which lists midpoint estimates for N^{10} points where $N = 2, 3, 4, 5, 6$, and 7. With $N^{10} = 7^{10}$ points, we have the hundreds digit correct.

Table 3: Midpoint estimates for various values of N. Number of points required by the method, N^{10}, are shown as well. The true value is $\frac{20,480}{3} = 6,826.6666\cdots$.

N	N^{10}	M_{N^D}
2	1,024	7,680.000
3	59,049	7,205.926
4	1,048,576	7,040.000
5	9,765,625	6,963.200
6	60,466,176	6,921.481
7	282,475,249	6,896.327

Trying $N = 8$, R throws *Error: cannot allocate vector of size 4.0 Gb*. This is due to the midpoint function we have written. In the code used within the paper up until this point, vectors containing all of the points needed are constructed and utilizing R's amazing ability to iterate over vectors, the entire domain is evaluated at once. The trade off for speed is RAM usage, and with 10-dimensional spaces, vector allocation limitations are quickly exceeded. Additionally, the time to process continues to increase as the number of points increases.

3 Conclusion

Choosing to utilize R's ability to iterate over vectors is a limitation which needs to be overcome. One method is rewriting the code to use the more brute force method of nested for loops. In this case, large chunks of memory do not need to be allocated, but the evaluation of the code slows drastically as the dimension increases. Thus we have a function whose execution is $\mathcal{K}N^D$ or $\mathcal{O}(N^D)$, for some appropriate constant, $\mathcal{K}$ which is dependent upon the function, f, and its domain Ω.

This process will still be crippled, not by the limitation of RAM, but rather by the time it takes to execute the extreme number of evaluations. Discussions with students can then highlight the need to change how we think about solving problems. One way to overcome this limitation is the implementation of parallel processing. A problem like this one is ripe for distributed computing (i.e. multiple computers) or parallel computing (i.e. a single computer).

The next part will implement parallel processing using Raspberry Pi compute modules.

A Code Listings

A.1 One Dimensional Midpoint Method

Listing 1: Code utilized for generating midpoint estimates M_N assuming $\Omega \subseteq \mathbb{R}$ and utilizing various values of N. Additionally, the code generates the graphs of the function as well as the rectangles.

```
# The function we seek to estimate
myFun = function(x) {
  1 - x**2
}

a = -1
b = 1

oneDMidPoint = function(n) {
# Midpoint method begins
myDelta = (b-a)/n
myInterval = seq(a, b, by = myDelta)

myMidpoint = (myInterval[1:n] + myInterval[2:(n+1)])/2
myMidpointY = myFun(myMidpoint)
midPointDF = data.frame(x = myMidpoint, y = myMidpointY)

midPointApproximation = sum(myMidpointY) * myDelta # Midpoint method

# Deterministic graph using ggplot2 library
myDF = data.frame(x = myInterval)
detGraph = ggplot(data = myDF, aes(x = x)) +
  theme_classic() +
  theme(
    axis.text.x = element_text(size=20),
```

```
    axis.title.x = element_text(size=20),
    axis.text.y = element_text(size=20),
    axis.title.y = element_text(size=20)
    )

# Draw the rectangles on the plot
for(i in 1:n)
{
  detGraph = detGraph +
    geom_rect(
      xmin = myInterval[i], xmax = myInterval[i+1],
      ymin = 0, ymax = myMidpointY[i],
       fill = 'lightblue', color = 'darkblue')
}

# Add the graph y = f(x) as well as points to the plot
detGraph = detGraph +
  geom_function(fun = myFun, color = 'red', linewidth = 3) +
  geom_point(data = midPointDF, aes(x = x, y = y), color = 'black', size = 6)

return(list(detGraph, midPointApproximation))
}
```

A.2 Two Dimensional Midpoint Method

Listing 2: Code utilized for generating the plot of $y = f(x, y)$ for midpoint method when $\Omega \subseteq \mathbb{R}^2$.

```
# Plot generated using plot3D library,
myMesh = mesh(seq(-1, 1, length.out = 50), seq(-1, 1, length.out = 50))
x = myMesh$x
y = myMesh$y
z = 2 - x * x - y * y
par(mai = c(0.1,0.1,0.1,0.1)) # Make the plot larger by shrinking the margins

myParaboloid = surf3D(x, y, z,
       colvar = z, colkey = FALSE, # Color key options
       box = TRUE,
       bty = "b2", # Box type: f: full, b: default, b2: gridlines, n: none
       phi = 20, # Tilts the box towards you if phi > 0
       theta = 120 # Rotates the TOP (z-direction) of the box clockwise
       )
```

Listing 3: Code utilized for generating midpoint estimates M_N assuming $\Omega \subseteq \mathbb{R}^2$ and utilizing various values of N.

```
# Function to approximate
myGridFun = function(x, y)
{
  2 - x*x - y*y
}

gridEstimateFun2D = function(n)
{
  # Create the midpoint method's points
  gridA = -1
```

```
  gridB = 1
  gridN = n

  myGridDelta = (gridB - gridA)/gridN # For N = 4, 4^2 = 16 points to choose
  xp = seq(-1,1,by=myGridDelta)
  yp = seq(-1,1,by=myGridDelta)

  xmid = (xp[1:(length(xp) - 1)] + xp[2:length(xp)])/2
  ymid = (yp[1:(length(yp) - 1)] + yp[2:length(yp)])/2

  # Full outer join i.e. product of xmid and ymid
  gridPtsDF = merge(x=xmid, y=ymid, all=TRUE)

  # Estimate for midpoint method using grid
  midpointGridApproximation = sum(
    myGridDelta *
    myGridDelta *
    myGridFun(gridPtsDF$x, gridPtsDF$y)
    )

  return(midpointGridApproximation)
}
```

Listing 4: Code utilized to generate the grid of midpoints when $\Omega \subseteq \mathbb{R}^2$.

```
# Create the midpoint method's points
gridA = -1
gridB = 1
gridN = 4

myGridDelta = (gridB - gridA)/gridN # For N = 4, 4^2 = 16 points to choose
xp = seq(-1,1,by=myGridDelta)
yp = seq(-1,1,by=myGridDelta)

xmid = (xp[1:(length(xp) - 1)] + xp[2:length(xp)])/2
ymid = (yp[1:(length(yp) - 1)] + yp[2:length(yp)])/2

gridPtsDF = merge(x=xmid, y=ymid, all=TRUE) # Full outer join i.e. xmid x ymid

# Empty axis begins
nullDF = data.frame(x = 1, y = 1)

ggplot(data = nullDF, aes(x = x, y = y)) +
  coord_fixed() + # Preserves the aspect ratio
  geom_hline(aes(yintercept = 0)) +
  geom_vline(aes(xintercept = 0)) +
  scale_x_continuous(limits = c(-1, 1),
                     breaks = seq(-1, 1, by = 0.5),
                     minor_breaks = FALSE
                     ) +
  scale_y_continuous(limits = c(-1, 1),
                     breaks = seq(-1, 1, by = 0.5)
                     ) +
  geom_point(data = gridPtsDF, aes(x = x, y = y), color = 'blue', size = 3) +
  geom_rect(
```

```
    xmin = gridA, xmax = gridB,
    ymin = gridA, ymax = gridB,
    fill = 'red', color = 'red',
    alpha = 0.0625
  )+
  theme_void() +
  theme(panel.grid = element_line(
          color = 'gray', linewidth = 0.5, linetype = 2
          ),
        panel.grid.minor = element_line(
          color = 'gray', linewidth = 0.5, linetype = 3
          )
        )
```

A.3 Ten Dimensional Midpoint Method

Listing 5: Code utilized for generating midpoint estimates M_N assuming $\Omega \subseteq \mathbb{R}^{10}$ and utilizing various values of N.

```
# Function
my10GridFun = function(x1, x2, x3, x4, x5, x6, x7, x8, x9, x10)
{
  10 - x1*x1  - x2*x2  - x3*x3  - x4*x4  - x5*x5  - x6*x6  - x7*x7  - x8*x8
- x9*x9  - x10*x10
}

a = -1
b = 1

# Midpoint method begins
midPointRuleFor10DFun = function(my10N)
{
  deltaX10D = (b-a) / my10N

  xp1 = seq(-1,1,by=deltaX10D)
  xp2 = seq(-1,1,by=deltaX10D)
  xp3 = seq(-1,1,by=deltaX10D)
  xp4 = seq(-1,1,by=deltaX10D)
  xp5 = seq(-1,1,by=deltaX10D)
  xp6 = seq(-1,1,by=deltaX10D)
  xp7 = seq(-1,1,by=deltaX10D)
  xp8 = seq(-1,1,by=deltaX10D)
  xp9 = seq(-1,1,by=deltaX10D)
  xp10 = seq(-1,1,by=deltaX10D)

  xm1 = (xp1[1:(length(xp1) - 1)] + xp1[2:length(xp1)])/2
  xm2 = (xp2[1:(length(xp2) - 1)] + xp2[2:length(xp2)])/2
  xm3 = (xp3[1:(length(xp3) - 1)] + xp3[2:length(xp3)])/2
  xm4 = (xp4[1:(length(xp4) - 1)] + xp4[2:length(xp4)])/2
  xm5 = (xp5[1:(length(xp5) - 1)] + xp5[2:length(xp5)])/2
  xm6 = (xp6[1:(length(xp6) - 1)] + xp6[2:length(xp6)])/2
  xm7 = (xp7[1:(length(xp7) - 1)] + xp7[2:length(xp7)])/2
  xm8 = (xp8[1:(length(xp8) - 1)] + xp8[2:length(xp8)])/2
  xm9 = (xp9[1:(length(xp9) - 1)] + xp9[2:length(xp9)])/2
```

```
  xm10 = (xp10[1:(length(xp10) - 1)] + xp10[2:length(xp10)])/2

  # Cartesian product/full outer join
  gridPts10DF = expand.grid(xm1, xm2, xm3, xm4, xm5, xm6, xm7, xm8, xm9, xm10)
  colnames(gridPts10DF) = c('xm1', 'xm2', 'xm3', 'xm4', 'xm5',
                            'xm6', 'xm7', 'xm8', 'xm9', 'xm10')

  # Estimate for midpoint method using grid
  midPoint10D = deltaX10D^10 * sum(my10GridFun(gridPts10DF$xm1,
                                               gridPts10DF$xm2,
                                               gridPts10DF$xm3,
                                               gridPts10DF$xm4,
                                               gridPts10DF$xm5,
                                               gridPts10DF$xm6,
                                               gridPts10DF$xm7,
                                               gridPts10DF$xm8,
                                               gridPts10DF$xm9,
                                               gridPts10DF$xm10
                                               )
                                   )
  return(midPoint10D)
}
```

References

Richard E. Crandall. *Projects in Scientific Computation*. The Electronic Library of Science (TELOS), 1994.

Paul DuChateau and David Zachman. *Applied Partial Differential Equations*. Dover Publications, Inc., 1 edition, 2002.

Joel R. Hass, Christopher E. Heil, Maurice D. Weir, and Przemyslaw Bogacki. *Thomas' Calculus*. Pearson, 15 edition, 2022.

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brin P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Pearson, 2 edition, 1992.

Dennis G. Zill. *A First Course in Differential Equations with Modeling Applications*. Cengage, 12 edition, 2024.