

---

---

# HW5 EMA 471 Intermediate Problem Solving for Engineers

---

---

SPRING 2016  
ENGINEERING MECHANICS DEPARTMENT  
UNIVERSITY OF WISCONSIN, MADISON

INSTRUCTOR: PROFESSOR ROBERT J. WITT

BY

NASSER M. ABBASI

DECEMBER 30, 2019

## Contents

0.1	Problem 1 . . . . .	3
0.2	Problem 2 . . . . .	5
0.2.1	part a . . . . .	7
0.2.2	part b . . . . .	9
0.3	Problem 3 . . . . .	16
0.3.1	shape functions . . . . .	18
0.3.2	$x(s, t, r)$ terms . . . . .	19
0.3.3	$y(r, s, t)$ terms . . . . .	25
0.3.4	$z(r, s, t)$ terms . . . . .	30
0.3.5	results . . . . .	35

## List of Tables

1	HW5, problem 1 result . . . . .	4
2	Gaussian quadrature using different points $\int_0^{2.5} \frac{\sin^2 x \sin x^2}{(1+x^2)^2} dx$ . Compared with Matlab Quad result of 0.115990989197426 for same integral . . . . .	7
3	Gaussian quadrature using 5 and 6 points $\int_0^{1.25} \frac{\sin^2 x \sin x^2}{(1+x^2)^2} dx + \int_{1.25}^{2.5} \frac{\sin^2 x \sin x^2}{(1+x^2)^2} dx$ . Compared with Matlab integral result of $\int_0^{2.5} \frac{\sin^2 x \sin x^2}{(1+x^2)^2} dx = 0.115990989197426$ . . . . .	9
4	Gaussian quadrature using 5 and 6 points. Comparing part(a) and part(b) relative error against Matlab's Quad . . . . .	10
5	work-equivalent conversion at each corner, problem 3 . . . . .	40

## List of Figures

1	problem 1 description . . . . .	3
2	problem 2 description . . . . .	6
3	Comparing Gaussian quadrature with Matlab's integral result . . . . .	8
4	Relative error for different $N$ values . . . . .	9
5	Comparing Gaussian quadrature with Matlab's integral result, part(b) . . . . .	10
6	Relative error for different $N$ values, part(b) . . . . .	11
7	problem 3 description . . . . .	16
8	more problem 3 description . . . . .	17
9	3D plot of the volume in physical coordinates . . . . .	35
10	3D plot of the aligned volume used for verification . . . . .	37

## 0.1 Problem 1

EP 471 – Homework #5  
Due: Thursday, April 7<sup>th</sup>, 2016

(1) (8 pts) It is stated without proof in Exercise 15 that Gaussian quadrature of order  $N$  produces an exact result when applied to the integration of a polynomial of order  $2N - 1$ . Consider the following polynomials in the interval  $0 \leq x \leq 4$  :

(a)  $f_1(x) = x^5$

(b)  $f_2(x) = x^7$

Integrate (a) and (b) over the interval  $0 \leq x \leq 4$  using Gaussian quadrature of  $N = 3$  and 4 respectively and compare your results to the analytical values. Does the quadrature of the appropriate order produce an exact match?

Figure 1: problem 1 description

A polynomial  $f(x)$  of order  $p$  is integrated exactly with the Gaussian quadrature method using  $\frac{p+1}{2}$  number of Gaussian points.

Hence  $f_1(x) = x^5$  needs  $\frac{5+1}{2} = 3$  Gaussian points and  $f_2(x) = x^7$  needs  $\frac{7+1}{2} = 4$  Gaussian points for exact result.

The integral  $\int_a^b f(x) dx$  is first converted to be in the domain  $\{-1, +1\}$  as follows

$$\int_a^b f(x) dx = \frac{b-a}{2} \int_{-1}^{+1} f\left(\frac{b-a}{2}t + \frac{b+a}{2}\right) dt$$

For a polynomial  $f(x)$  of order  $p = 3$ , two Gaussian points are needed to evaluate the above integral exactly. Therefore the above integral simplifies to

$$\int_a^b f(x) dx = A (w_1 f(At_1 + B) + w_2 f(At_2 + B))$$

Where

$$A = \frac{b-a}{2}$$

$$B = \frac{b+a}{2}$$

And  $w_i$  is the weight at location  $t_i$ . The weights and location of the weights are obtained from tables. For higher order polynomials, more points and weights are needed.

In general, using  $N$  points the integral is

$$\begin{aligned}\int_a^b f(x) dx &= \frac{b-a}{2} \sum_{i=1}^N w_i f\left(\frac{b-a}{2}t_i + \frac{a+b}{2}\right) \\ &= A \sum_{i=1}^N w_i f(At_i + B)\end{aligned}\quad (1)$$

The program `nma_EMA_471_HW5_problem_1.m` integrates the above two polynomials  $f_1(x), f_2(x)$  using Gaussian quadrature method (1) using  $N = 3$  and  $N = 4$  points respectively and compares the result of each to the analytical solution. The following table shows the result.

Table 1: HW5, problem 1 result

function	analytical result	$N$	Gaussian quadrature result
$f_1(x) = x^4$	$\int_0^4 x^4 dx = \frac{2048}{3} = 682.666666666667$	3	682.666666666667
$f_1(x) = x^7$	$\int_0^4 x^7 dx = 8192$	4	8192

The result is exact. Note: The above Matlab program used the exact weights and points for Gaussian quadrature as given in [https://en.wikipedia.org/wiki/Gaussian\\_quadrature](https://en.wikipedia.org/wiki/Gaussian_quadrature)

```

1 function nma_EMA_471_HW5_problem_1()
2 %Solves problem 1, HW5, EMA 471
3 %Nasser M. Abbasi
4
5 %reference https://en.wikipedia.org/wiki/Gaussian_quadrature for points
6 %and weights. These are exact.
7
8 gauss_3_points=[-sqrt(3/5) , 5/9;      %point, weight  per row
9                0           , 8/9;
10               sqrt(3/5)  , 5/9];
11
12 gauss_4_points=[-sqrt(3/7+ 2/7*sqrt(6/5)) , (18-sqrt(30))/36;
13                -sqrt(3/7- 2/7*sqrt(6/5)) , (18+sqrt(30))/36;
14                sqrt(3/7- 2/7*sqrt(6/5))  , (18+sqrt(30))/36;
15                sqrt(3/7+ 2/7*sqrt(6/5))  , (18-sqrt(30))/36];
16
17 f1=@(x) x.^5;
18 integrate(f1,0,4,gauss_3_points)
19
20 f1=@(x) x.^7;
21 integrate(f1,0,4,gauss_4_points)
22
23 end
24 %=====
25 function the_sum = integrate(f,from,to,g)
26 %INPUT:  f  is handle to function to integrate

```

```
27 %      from,to these are lower and upper integral bounds
28 %      g   This is matrix of Gaussian quadrature. first column is points
29 %          second column is corresponding weights
30
31 A = (to-from)/2;
32 B = (to+from)/2;
33 i = 1:size(g,1);
34 the_sum = A * sum( g(i,2) .* f(A*g(i,1)+B) ); %vectored sum
35 end
```

## 0.2 Problem 2

(2) (12 pts) In one of our exercises we evaluated the following function by Gaussian quadrature:

$$I = \int_0^{2.5} \frac{\sin^2 x \cdot \sin(x^2)}{(1+x^2)^2} dx$$

using 2, 3 and 4 gauss points. In this case, the domain is large enough and the function changes sharply enough that we don't get very good agreement with Matlab's `quad` utility even for 4 gauss points. To get better agreement, we could use a larger number of Gauss points, or break the domain into pieces,  $0 \leq x \leq 1.25$  and  $1.25 \leq x \leq 2.5$ , for example.

The following extends the table in Exercise 15 to 5, 6, 7 and 8 gauss points:

Number of terms, $N$	Values of $t(t_i)$	Weighting factor	Valid up to degree
5	0	0.56888889	9
	$\pm 0.53846931$	0.47862867	
	$\pm 0.90617985$	0.23692689	
6	$\pm 0.23861918$	0.46791393	11
	$\pm 0.66120939$	0.36076157	
	$\pm 0.93246951$	0.17132449	
7	0	0.41795918	13
	$\pm 0.40584515$	0.38183005	
	$\pm 0.74153119$	0.27970539	
	$\pm 0.94910791$	0.12948497	
8	$\pm 0.18343464$	0.36268378	15
	$\pm 0.52553241$	0.31370665	
	$\pm 0.79666648$	0.22238103	
	$\pm 0.96028986$	0.10122854	

- Keeping the original domain,  $0 \leq x \leq 2.5$ , how does the agreement with Matlab's `quad` utility improve with 5, 6, 7 and 8 point Gaussian quadrature?
- Breaking the domain into two pieces,  $0 \leq x \leq 1.25$  and  $1.25 \leq x \leq 2.5$ , evaluate the integral using 5 and 6 gauss points in each subdomain. How does the agreement compare with results from Matlab's `quad` utility now?

Figure 2: problem 2 description

### 0.2.1 part a

The program `nma_EMA_471_HW5_problem_2_part_a.m` implements the first part of this problem.

The following table shows the result of the computation. It shows the result of the integral using Gaussian quadrature for different number of points with the relative error against Matlab's `Quad (integral)` command.

Number of points	relative error (percentage)	value of integral
2	79.845442	0.023377470178383
3	23.892753	0.143704430262801
4	3.060319	0.112441294428254
5	4.692010	0.121433298541329
6	0.019015	0.116013045399658
7	0.011820	0.116004700249995
8	0.001535	0.115989208171974

Table 2: Gaussian quadrature using different points  $\int_0^{2.5} \frac{\sin^2 x \sin x^2}{(1+x^2)^2} dx$ . Compared with Matlab `Quad` result of 0.115990989197426 for same integral

The following is a plot of the above data

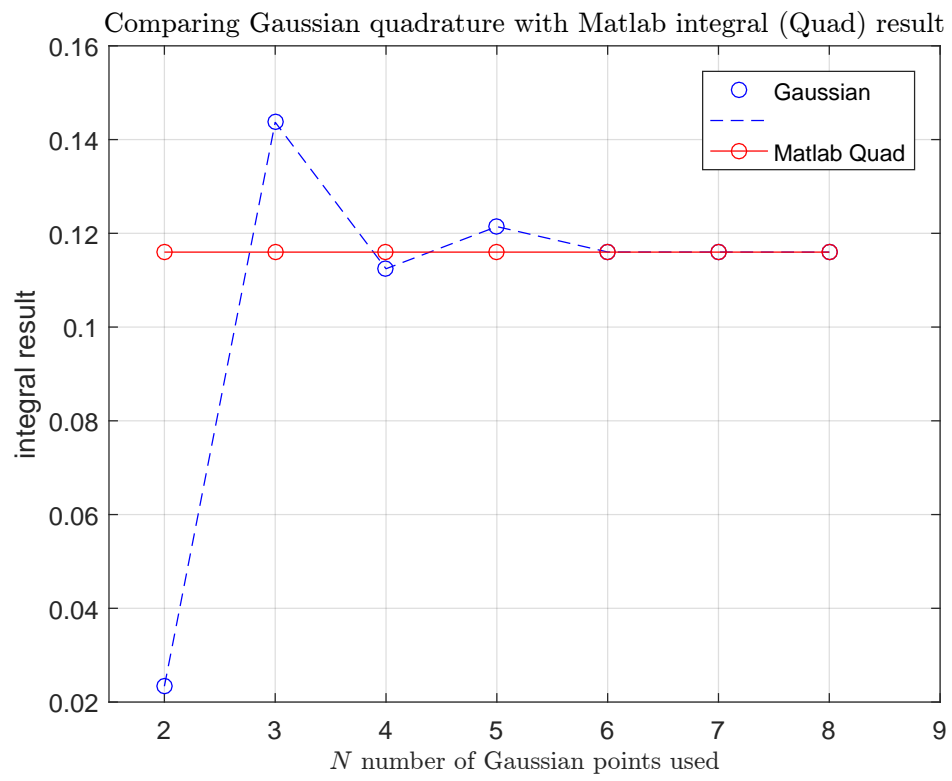


Figure 3: Comparing Gaussian quadrature with Matlab's integral result



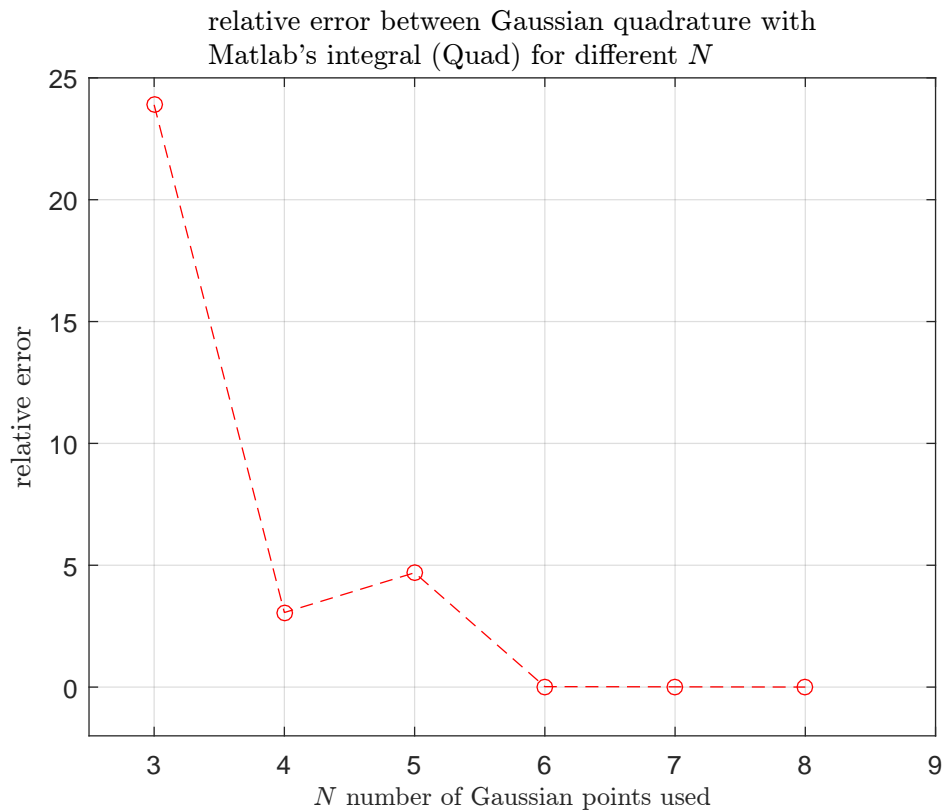


Figure 4: Relative error for different  $N$  values

### 0.2.2 part b

The program `nama_EMA_471_HW5_problem_2_part_b.m` implements the second part of this problem. By breaking the domain into 2 parts, the following table shows the result of the computation. It shows the result of the integral using Gaussian quadrature for 5 and 6 points with the relative error against Matlab's Quad (integral) command. The integration was done on each subdomain and the results added.

Number of points	relative error (percentage)	value of integral using Gaussian quadrature
5	1.231392	0.114562685084637
6	0.000303	0.115990636860983

Table 3: Gaussian quadrature using 5 and 6 points  $\int_0^{1.25} \frac{\sin^2 x \sin x^2}{(1+x^2)^2} dx + \int_{-1.25}^{2.5} \frac{\sin^2 x \sin x^2}{(1+x^2)^2} dx$ . Compared with Matlab integral result of  $\int_0^{2.5} \frac{\sin^2 x \sin x^2}{(1+x^2)^2} dx = 0.115990989197426$

The above shows clearly that by breaking the domain into two smaller parts, and adding each

result, the final result of Gaussian quadrature improved compared to part(a) where one large domain was used. This makes sense. Because we have effectively used more sampling points in part(b) compared to part(a) when looking at the whole domain.

This shows that, to obtain more accuracy using Gaussian quadrature, and still use the same number of points  $N$ , then we can break the domain into smaller regions, and use  $N$  on each region, and add the result obtained from each region.

To see the difference between part(a) and (b) more clearly, the following table shows the result for 5 and 6 points side by side from part(a) and part(b). The table below shows the relative error is much smaller for part(b).

Number of points	relative error part(b)	relative error part(a)
5	1.231392	4.692010
6	0.000303	0.019015

Table 4: Gaussian quadrature using 5 and 6 points. Comparing part(a) and part(b) relative error against Matlab's Quad

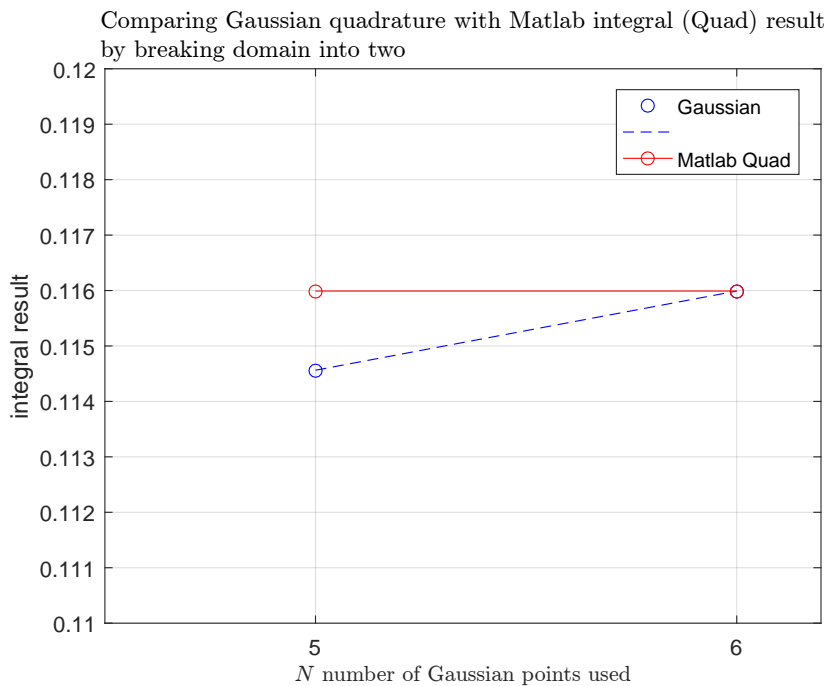
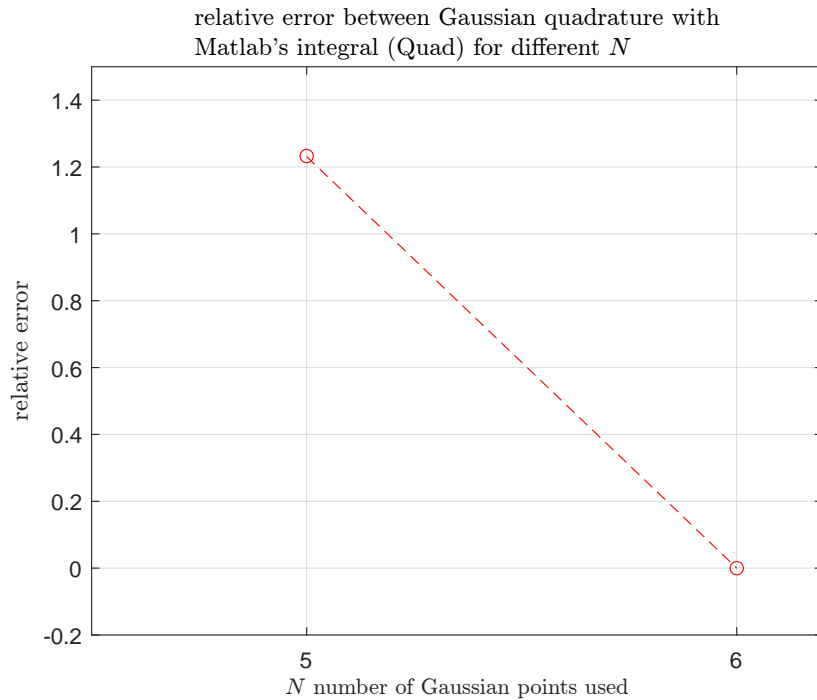


Figure 5: Comparing Gaussian quadrature with Matlab's integral result, part(b)

Figure 6: Relative error for different  $N$  values, part(b)

```

1 function nma_EMA_471_HW5_problem_2_part_a()
2 %Solves problem 2, part(a) HW5, EMA 471
3 %Nasser M. Abbasi
4
5 close all; clc;
6
7 gauss_2_points=[-0.57735027 , 1;      %point, weight  per row
8                0.57735027 , 1
9                ];
10
11 gauss_3_points=[-sqrt(3/5) , 5/9;    %point, weight  per row
12                0 , 8/9;
13                sqrt(3/5) , 5/9];
14
15 gauss_4_points=[-sqrt(3/7+ 2/7*sqrt(6/5)) , (18-sqrt(30))/36;
16                -sqrt(3/7- 2/7*sqrt(6/5)) , (18+sqrt(30))/36;
17                sqrt(3/7- 2/7*sqrt(6/5)) , (18+sqrt(30))/36;
18                sqrt(3/7+ 2/7*sqrt(6/5)) , (18-sqrt(30))/36];
19
20 gauss_5_points=[0 , 128/225;
21                -(1/3)*sqrt(5-2*sqrt(10/6)) , (322+13*sqrt(70))/900;
22                (1/3)*sqrt(5-2*sqrt(10/6)) , (322+13*sqrt(70))/900;
23                -(1/3)*sqrt(5+2*sqrt(10/6)) , (322-13*sqrt(70))/900;

```

```

24         (1/3)*sqrt(5+2*sqrt(10/6)) , (322-13*sqrt(70))/900];
25
26 gauss_6_points=[0.238619186083197 , 0.467913934572691;
27                -0.238619186083197 , 0.467913934572691;
28                0.661209386466265 , 0.360761573048139;
29                -0.661209386466265 , 0.360761573048139;
30                0.932469514203152 , 0.171324492379170;
31                -0.932469514203152 , 0.171324492379170];
32
33 gauss_7_points=[0 , 0.417959183673469;
34                0.405845151377397 , 0.381830050505119;
35                -0.405845151377397 , 0.381830050505119;
36                0.741531185599394 , 0.279705391489277;
37                -0.741531185599394 , 0.279705391489277;
38                0.949107912342759 , 0.129484966168870;
39                -0.949107912342759 , 0.129484966168870];
40
41
42 gauss_8_points=[0.183434642495650 , 0.362683783378361;
43                -0.183434642495650 , 0.362683783378361;
44                0.525532409916329 , 0.313706645877887;
45                -0.525532409916329 , 0.313706645877887;
46                0.796666477413627 , 0.222381034453374;
47                -0.796666477413627 , 0.222381034453374;
48                0.960289856497536 , 0.101228536290376;
49                -0.960289856497536 , 0.101228536290376];
50
51 f=@(x) sin(x).^2 .* sin(x.^2) ./ (1+x.^2).^2 ;
52 x_min    = 0;
53 x_max    = 2.5;
54 data     = zeros(7,4);
55 chk     = integral(f,x_min,x_max);
56 data(:,1) = chk;
57
58 for i=1:size(data,1)
59     switch i
60     case 1
61         data(i,2) = integrate(f,x_min,x_max,gauss_2_points);
62         data(i,3) = 100*abs(chk - data(i,2))/abs(chk);
63         data(i,4) = 2;
64     case 2
65         data(i,2) = integrate(f,x_min,x_max,gauss_3_points);
66         data(i,3) = 100*abs(chk - data(i,2))/abs(chk);
67         data(i,4) = 3;
68     case 3
69         data(i,2) = integrate(f,x_min,x_max,gauss_4_points);
70         data(i,3) = 100*abs(chk - data(i,2))/abs(chk);

```

```

71     data(i,4) = 4;
72     case 4
73         data(i,2) = integrate(f,x_min,x_max,gauss_5_points);
74         data(i,3) = 100*abs(chk - data(i,2))/abs(chk);
75         data(i,4) = 5;
76     case 5
77         data(i,2) = integrate(f,x_min,x_max,gauss_6_points);
78         data(i,3) = 100*abs(chk - data(i,2))/abs(chk);
79         data(i,4) = 6;
80     case 6
81         data(i,2) = integrate(f,x_min,x_max,gauss_7_points);
82         data(i,3) = 100*abs(chk - data(i,2))/abs(chk);
83         data(i,4) = 7;
84     case 7
85         data(i,2) = integrate(f,x_min,x_max,gauss_8_points);
86         data(i,3) = 100*abs(chk - data(i,2))/abs(chk);
87         data(i,4) = 8;
88     end
89 end
90
91 figure;
92 plot(data(:,4),data(:,2), 'bo',data(:,4),data(:,2), 'b--');
93 hold on;
94 plot(data(:,4),data(:,1), 'r-o');
95 xlim([1.5,9]);
96 xlabel('$N$ number of Gaussian points used', ' ...
97     'interpreter','Latex','FontSize',10);
98 ylabel('integral result');
99 title('Comparing Gaussian quadrature with Matlab integral (Quad) result',...
100     'interpreter','Latex');
101 legend('Gaussian',' ','Matlab Quad');
102 grid;
103
104 figure;
105 plot(data(2:end,4),data(2:end,3), 'ro',data(2:end,4),...
106     data(2:end,3), 'r--');
107 xlabel('$N$ number of Gaussian points used','interpreter',...
108     'Latex','FontSize',10);
109 ylabel('relative error','interpreter','Latex');
110 title({'relative error between Gaussian quadrature with',...
111     'Matlab's integral (Quad) for different $N$'},...
112     'interpreter','Latex');
113 grid;
114 xlim([2.5,9]);
115 ylim([-2,25]);
116
117

```

```

118
119 end
120 %=====
121 function the_sum = integrate(f,from,to,g)
122 %INPUT:  f  is handle to function to integrate
123 % from,to these are lower and upper integral bounds
124 % g  This is matrix of Gaussian quadrature. first column is points
125 % second column is corresponding weights
126
127 A = (to-from)/2;
128 B = (to+from)/2;
129 i = 1:size(g,1);
130 the_sum = A * sum( g(i,2) .* f(A*g(i,1)+B) );  %vectored sum
131 end

1 function nma_EMA_471_HW5_problem_2_part_b()
2 %Solves problem 2, part(b) HW5, EMA 471
3 %Nasser M. Abbasi
4
5 close all; clc;
6
7
8 gauss_5_points=[0                , 128/225;
9                -(1/3)*sqrt(5-2*sqrt(10/6)) , (322+13*sqrt(70))/900;
10               (1/3)*sqrt(5-2*sqrt(10/6)) , (322+13*sqrt(70))/900;
11               -(1/3)*sqrt(5+2*sqrt(10/6)) , (322-13*sqrt(70))/900;
12               (1/3)*sqrt(5+2*sqrt(10/6)) , (322-13*sqrt(70))/900];
13
14 gauss_6_points=[0.238619186083197 , 0.467913934572691;
15                -0.238619186083197 , 0.467913934572691;
16                0.661209386466265 , 0.360761573048139;
17                -0.661209386466265 , 0.360761573048139;
18                0.932469514203152 , 0.171324492379170;
19                -0.932469514203152 , 0.171324492379170];
20
21
22 f=@(x) sin(x).^2 .* sin(x.^2) ./ (1+x.^2).^2 ;
23 data      = zeros(2,4);
24 chk      = integral(f,0,2.5);
25 data(:,1) = chk;
26
27 data(1,2) = integrate(f,0,1.25,gauss_5_points)+integrate(f,1.25,2.5,gauss_5_points);
28 data(1,3) = 100*abs(chk - data(1,2))/abs(chk);
29 data(1,4) = 5;
30
31 data(2,2) = integrate(f,0,1.25,gauss_6_points)+integrate(f,1.25,2.5,gauss_6_points);
32 data(2,3) = 100*abs(chk - data(2,2))/abs(chk);
33 data(2,4) = 6;

```

```

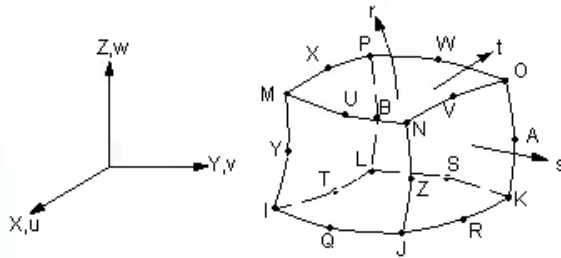
34
35 figure;
36 plot(data(:,4),data(:,2),'bo',data(:,4),data(:,2),'b--');
37 hold on;
38 plot(data(:,4),data(:,1),'r-o');
39 xlim([1.5,9]);
40 xlabel('$N$ number of Gaussian points used','interpreter','Latex','FontSize',10);
41 ylabel('integral result');
42 title({'Comparing Gaussian quadrature with Matlab integral (Quad) result', ...
43       'by breaking domain into two'},'interpreter','Latex');
44 legend('Gaussian',' ','Matlab Quad');
45 grid;
46 xlim([4.5,6.2]);
47 ylim([.11,.12]);
48 ax = gca;
49 ax.XTick = [5 6];
50
51
52 figure;
53 plot(data(:,4),data(:,3),'ro',data(:,4),data(:,3),'r--');
54 xlabel('$N$ number of Gaussian points used','interpreter','Latex','FontSize',10);
55 ylabel('relative error','interpreter','Latex');
56 title({'relative error between Gaussian quadrature with',...
57       'Matlab''s integral (Quad) for different $N$'},'interpreter','Latex');
58 grid;
59 xlim([4.5,6.2]);
60 ylim([-0.2,1.5]);
61 ax = gca;
62 ax.XTick = [5 6];
63
64
65
66 end
67 %=====
68 function the_sum = integrate(f,from,to,g)
69 %INPUT:  f  is handle to function to integrate
70 %        from,to these are lower and upper integral bounds
71 %        g  This is matrix of Gaussian quadrature. first column is points
72 %           second column is corresponding weights
73
74 A = (to-from)/2;
75 B = (to+from)/2;
76 i = 1:size(g,1);
77 the_sum = A * sum( g(i,2) .* f(A*g(i,1)+B) ); %vectored sum
78 end

```

### 0.3 Problem 3

(3) (20 pts) When using commercial software such as ANSYS, one can find the interpolation functions used for various element types in the Shape Functions section of the on-line Theory Manual. The most general 3D continuum elements are 20-node brick elements, and this figure from ANSYS lists the interpolation scheme:

Figure 12.17: 20-Node Brick Element



These shape functions are used for 20-node solid elements such as [SOLID90](#):

$$\begin{aligned}
 u = \frac{1}{8} & (u_I(1-s)(1-t)(1-r)(-s-t-r-2) + u_J(1+s)(1-t)(1-r)(s-t-r-2) \\
 & + u_K(1+s)(1+t)(1-r)(s+t-r-2) + u_L(1-s)(1+t)(1-r)(-s+t-r-2) \\
 & + u_M(1-s)(1-t)(1+r)(-s-t+r-2) + u_N(1+s)(1-t)(1+r)(s-t+r-2) \\
 & + u_O(1+s)(1+t)(1+r)(s+t+r-2) + u_P(1-s)(1+t)(1+r)(-s+t+r-2)) \\
 & + \frac{1}{4} (u_Q(1-s^2)(1-t)(1-r) + u_R(1+s)(1-t^2)(1-r) \\
 & + u_S(1-s^2)(1+t)(1-r) + u_T(1-s)(1-t^2)(1-r) \\
 & + u_U(1-s^2)(1-t)(1+r) + u_V(1+s)(1-t^2)(1+r) \\
 & + u_W(1-s^2)(1+t)(1+r) + u_X(1-s)(1-t^2)(1+r) \\
 & + u_Y(1-s)(1-t)(1-r^2) + u_Z(1+s)(1-t)(1-r^2) \\
 & + u_A(1+s)(1+t)(1-r^2) + u_B(1-s)(1+t)(1-r^2))
 \end{aligned}$$

Note that in this representation, “r”, “s” and “t” have replaced “xi” ( $\xi$ ), “eta” ( $\eta$ ) and “zeta” ( $\zeta$ ) as the natural coordinate system variables. The interpolation shown above is for displacement degree-of-freedom  $u$ , but this same interpolation holds for the other degrees-of-freedom as well as for the coordinates  $(x,y,z)$  within the element domain. Calculation of the volume of this element would be accomplished through:

Figure 7: problem 3 description



03/28/16

$$V = \int dx dy dz = \int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} \frac{\partial(x,y,z)}{\partial(r,s,t)} dr ds dt \cong \sum_{i=1}^N \sum_{j=1}^N \sum_{k=1}^N w_i w_j w_k \det J(r_i, s_j, t_k)$$

In the event that we need to find the response of a 3D structure to its own weight, we have to convert the continuously distributed weight density into a series of 20 discrete weights at each of the nodes. The work-equivalent finite element result is that the force at node  $i$  ( $i = 1$  through B in the figure above) is found from:

$$F_i = \int \gamma(x,y,z) f_i(r,s,t) dx dy dz$$

$$= \int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} \gamma[x(r,s,t), y(r,s,t), z(r,s,t)] f_i(r,s,t) \frac{\partial(x,y,z)}{\partial(r,s,t)} dr ds dt$$

Here  $\gamma = \rho g$  is the weight density and may be a function of position within the element. The  $f_i$  are the interpolation functions listed in the figure on the previous page. For example, for  $i = 1$ ,

$$f_i = f_1 = \frac{1}{8}(1-r)(1-s)(1-t)(-r-s-t-2)$$

Aside: The interpolation functions are defined so that they go to one when their associated node is approached from inside the element. All the other interpolation functions go to zero at the same location, so that the interpolated quantity goes to the nodal quantity. In the case of node I, you'll notice from the figure that this corresponds to  $r = s = t = -1$ .

Given the nodal coordinates listed below for nodes I through B (taken from one element in a mesh prepared using ANSYS), and given the spatial dependence of the mass density, find the  $z$ -component forces to be applied to each of the nodes so that the weight is applied to the element in a work-equivalent fashion (assumes  $\mathbf{g}$  points in the  $-z$  direction).

Nodal coordinates  $(x, y, z)$  are: (all values in units of cm):

Node I:	1	0	0
Node J:	1.1	0	0
Node K:	1.09066	0	-0.086305
Node L:	0.99692	0	-0.078459
Node M:	1.0077	0.16559	0
Node N:	1.1069	0.17203	0
Node O:	1.1035	0.17202	-0.08663
Node P:	1.0046	0.16557	-0.07882
Node Q:	1.05	0	0
Node R:	1.0992	0	-0.043186
Node S:	1.0468	0	-0.082382
Node T:	0.9923	0	-0.039260
Node U:	1.0573	0.16881	0
Node V:	1.1061	0.17202	-0.043328
Node W:	1.0540	0.16880	-0.082725
Node X:	1.0069	0.16558	-0.039418
Node Y:	1.0051	0.082737	0
Node Z:	1.1046	0.085964	0
Node A:	1.1012	0.085938	-0.086550
Node B:	1.0020	0.082709	-0.078731

03/28/16

The mass density (units of  $\text{g/cm}^3$ ) is due to a functionally graded material and has the functional form:

$$\rho = \rho_o(x^2 + z^2) \quad ; \quad \rho_o = 1$$

Figure 8: more problem 3 description

### 0.3.1 shape functions

The following are the shape functions

$$f_I = \frac{1}{8}(1-r)(1-s)(1-t)(-r-s-t-2)$$

$$f_J = \frac{1}{8}(1-r)(s+1)(1-t)(-r+s-t-2)$$

$$f_K = \frac{1}{8}(1-r)(s+1)(t+1)(-r+s+t-2)$$

$$f_L = \frac{1}{8}(1-r)(1-s)(t+1)(-r-s+t-2)$$

$$f_M = \frac{1}{8}(r+1)(1-s)(1-t)(r-s-t-2)$$

$$f_N = \frac{1}{8}(r+1)(s+1)(1-t)(r+s-t-2)$$

$$f_O = \frac{1}{8}(r+1)(s+1)(t+1)(r+s+t-2)$$

$$f_P = \frac{1}{8}(r+1)(1-s)(t+1)(r-s+t-2)$$

$$f_Q = \frac{1}{4}(1-r)(1-s^2)(1-t)$$

$$f_R = \frac{1}{4}(1-r)(s+1)(1-t^2)$$

$$f_S = \frac{1}{4}(1-r)(1-s^2)(t+1)$$

$$f_T = \frac{1}{4}(1-r)(1-s)(1-t^2)$$

$$f_U = \frac{1}{4}(r+1)(1-s^2)(1-t)$$

$$f_V = \frac{1}{4}(r+1)(s+1)(1-t^2)$$

$$f_W = \frac{1}{4}(r+1)(1-s^2)(t+1)$$

$$f_X = \frac{1}{4}(r+1)(1-s)(1-t^2)$$

$$f_Y = \frac{1}{4}(1-r^2)(1-s)(1-t)$$

$$f_Z = \frac{1}{4}(1-r^2)(s+1)(1-t)$$

$$f_A = \frac{1}{4}(1-r^2)(s+1)(t+1)$$

$$f_B = \frac{1}{4}(1-r^2)(1-s)(t+1)$$

To obtain the Jacobian, we need to obtain the determinant of

$$\begin{pmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} & \frac{\partial z}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} & \frac{\partial z}{\partial s} \\ \frac{\partial x}{\partial t} & \frac{\partial y}{\partial t} & \frac{\partial z}{\partial t} \end{pmatrix}$$

Where

$$x(r, s, t) = \sum_{i=I}^{I=B} x_i f_i(r, s, t)$$

$$y(r, s, t) = \sum_{i=I}^{I=B} y_i f_i(r, s, t)$$

$$z(r, s, t) = \sum_{i=I}^{I=B} z_i f_i(r, s, t)$$

Once we determine  $x(r, s, t), y(r, s, t), z(r, s, t)$  from the above, then we can determine the Jacobian determinant at each Gaussian point.

### 0.3.2 $x(s, t, r)$ terms

From the above sum, expanding  $x(r, s, t)$  gives

$$x = x_I f_I + x_J f_J + x_K f_K + x_L f_L + x_M f_M + x_N f_N + x_O f_O + x_P f_P + x_Q f_Q + x_R f_R + x_S f_S + x_T f_T + x_U f_U + x_V f_V +$$

Substituting the values of  $f_i$  into the above results in

$$\begin{aligned}
x(r, s, t) = & x_I \frac{1}{8} (1-r)(1-s)(1-t)(-r-s-t-2) + \\
& x_J \frac{1}{8} (1-r)(s+1)(1-t)(-r+s-t-2) + \\
& x_K \frac{1}{8} (1-r)(s+1)(t+1)(-r+s+t-2) + \\
& x_L \frac{1}{8} (1-r)(1-s)(t+1)(-r-s+t-2) + \\
& x_M \frac{1}{8} (r+1)(1-s)(1-t)(r-s-t-2) + \\
& x_N \frac{1}{8} (r+1)(s+1)(1-t)(r+s-t-2) + \\
& x_O \frac{1}{8} (r+1)(s+1)(t+1)(r+s+t-2) + \\
& x_P \frac{1}{8} (r+1)(1-s)(t+1)(r-s+t-2) + \\
& x_Q \frac{1}{4} (1-r) (1-s^2) (1-t) + \\
& x_R \frac{1}{4} (1-r)(s+1) (1-t^2) + \\
& x_S \frac{1}{4} (1-r) (1-s^2) (t+1) + \\
& x_T \frac{1}{4} (1-r)(1-s) (1-t^2) + \\
& x_U \frac{1}{4} (r+1) (1-s^2) (1-t) + \\
& x_V \frac{1}{4} (r+1)(s+1) (1-t^2) + \\
& x_W \frac{1}{4} (r+1) (1-s^2) (t+1) + \\
& x_X \frac{1}{4} (r+1)(1-s) (1-t^2) + \\
& x_Y \frac{1}{4} (1-r^2) (1-s)(1-t) + \\
& x_Z \frac{1}{4} (1-r^2) (s+1)(1-t) + \\
& x_A \frac{1}{4} (1-r^2) (s+1)(t+1) + \\
& x_B \frac{1}{4} (1-r^2) (1-s)(t+1)
\end{aligned}$$

Taking partial derivative of the above w.r.t.  $r, s, t$  in turn gives the following

$$\begin{aligned}
\frac{\partial x}{\partial r} = & x_I \left( \frac{1}{8}(s-1)(t-1)(2r+s+t+1) \right) + \\
& x_J \left( \frac{1}{8}(s+1)(t-1)(-2r+s-t-1) \right) + \\
& x_K \left( -\frac{1}{8}(s+1)(t+1)(-2r+s+t-1) \right) + \\
& x_L \left( -\frac{1}{8}(s-1)(t+1)(2r+s-t+1) \right) + \\
& x_M \left( -\frac{1}{8}(s-1)(t-1)(-2r+s+t+1) \right) + \\
& x_N \left( -\frac{1}{8}(s+1)(t-1)(2r+s-t-1) \right) + \\
& x_O \left( \frac{1}{8}(s+1)(t+1)(2r+s+t-1) \right) + \\
& x_P \left( \frac{1}{8}(s-1)(t+1)(-2r+s-t+1) \right) + \\
& x_Q \left( -\frac{1}{4}(s^2-1)(t-1) \right) + \\
& x_R \left( \frac{1}{4}(s+1)(t^2-1) \right) + \\
& x_S \left( \frac{1}{4}(s^2-1)(t+1) \right) + \\
& x_T \left( -\frac{1}{4}(s-1)(t^2-1) \right) + \\
& x_U \left( \frac{1}{4}(s^2-1)(t-1) \right) + \\
& x_V \left( -\frac{1}{4}(s+1)(t^2-1) \right) + \\
& x_W \left( -\frac{1}{4}(s^2-1)(t+1) \right) + \\
& x_X \left( \frac{1}{4}(s-1)(t^2-1) \right) + \\
& x_Y \left( -\frac{1}{2}r(s-1)(t-1) \right) + \\
& x_Z \left( \frac{1}{2}r(s+1)(t-1) \right) + \\
& x_A \left( -\frac{1}{2}r(s+1)(t+1) \right) + \\
& x_B \left( \frac{1}{2}r(s-1)(t+1) \right)
\end{aligned}$$

In the Matlab implementation, the terms  $r, s, t$  in the above expression are the Gaussian integration points along the three directions. Similarly, we now find  $\frac{\partial x}{\partial s}$  as above. This results

in

$$\begin{aligned}
\frac{\partial x}{\partial s} = & x_I \left( \frac{1}{8}(r-1)(t-1)(r+2s+t+1) \right) + \\
& x_J \left( -\frac{1}{8}(r-1)(t-1)(r-2s+t+1) \right) + \\
& x_K \left( \frac{1}{8}(r-1)(t+1)(r-2s-t+1) \right) + \\
& x_L \left( -\frac{1}{8}(r-1)(t+1)(r+2s-t+1) \right) + \\
& x_M \left( \frac{1}{8}(r+1)(t-1)(r-2s-t-1) \right) + \\
& x_N \left( -\frac{1}{8}(r+1)(t-1)(r+2s-t-1) \right) + \\
& x_O \left( \frac{1}{8}(r+1)(t+1)(r+2s+t-1) \right) + \\
& x_P \left( -\frac{1}{8}(r+1)(t+1)(r-2s+t-1) \right) + \\
& x_Q \left( -\frac{1}{2}(r-1)s(t-1) \right) + \\
& x_R \left( \frac{1}{4}(r-1)(t^2-1) \right) + \\
& x_S \left( \frac{1}{2}(r-1)s(t+1) \right) + \\
& x_T \left( -\frac{1}{4}(r-1)(t^2-1) \right) + \\
& x_U \left( \frac{1}{2}(r+1)s(t-1) \right) + \\
& x_V \left( -\frac{1}{4}(r+1)(t^2-1) \right) + \\
& x_W \left( -\frac{1}{2}(r+1)s(t+1) \right) + \\
& x_X \left( \frac{1}{4}(r+1)(t^2-1) \right) + \\
& x_Y \left( -\frac{1}{4}(r^2-1)(t-1) \right) + \\
& x_Z \left( \frac{1}{4}(r^2-1)(t-1) \right) + \\
& x_A \left( -\frac{1}{4}(r^2-1)(t+1) \right) + \\
& x_B \left( \frac{1}{4}(r^2-1)(t+1) \right)
\end{aligned}$$

Similarly, we now find  $\frac{\partial x}{\partial t}$  as above. This results in

$$\begin{aligned}
\frac{\partial x}{\partial t} = & x_I \left( \frac{1}{8}(r-1)(s-1)(r+s+2t+1) \right) + \\
& x_J \left( -\frac{1}{8}(r-1)(s+1)(r-s+2t+1) \right) + \\
& x_K \left( \frac{1}{8}(r-1)(s+1)(r-s-2t+1) \right) + \\
& x_L \left( -\frac{1}{8}(r-1)(s-1)(r+s-2t+1) \right) + \\
& x_M \left( \frac{1}{8}(r+1)(s-1)(r-s-2t-1) \right) + \\
& x_N \left( -\frac{1}{8}(r+1)(s+1)(r+s-2t-1) \right) + \\
& x_O \left( \frac{1}{8}(r+1)(s+1)(r+s+2t-1) \right) + \\
& x_P \left( -\frac{1}{8}(r+1)(s-1)(r-s+2t-1) \right) + \\
& x_Q \left( -\frac{1}{4}(r-1)(s^2-1) \right) + \\
& x_R \left( \frac{1}{2}(r-1)(s+1)t \right) + \\
& x_S \left( \frac{1}{4}(r-1)(s^2-1) \right) + \\
& x_T \left( -\frac{1}{2}(r-1)(s-1)t \right) + \\
& x_U \left( \frac{1}{4}(r+1)(s^2-1) \right) + \\
& x_V \left( -\frac{1}{2}(r+1)(s+1)t \right) + \\
& x_W \left( -\frac{1}{4}(r+1)(s^2-1) \right) + \\
& x_X \left( \frac{1}{2}(r+1)(s-1)t \right) + \\
& x_Y \left( -\frac{1}{4}(r^2-1)(s-1) \right) + \\
& x_Z \left( \frac{1}{4}(r^2-1)(s+1) \right) + \\
& x_A \left( -\frac{1}{4}(r^2-1)(s+1) \right) + \\
& x_B \left( \frac{1}{4}(r^2-1)(s-1) \right)
\end{aligned}$$



### 0.3.3 $y(r, s, t)$ terms

We now repeat all the above to find  $\frac{\partial y}{\partial r}$ ,  $\frac{\partial y}{\partial s}$  and  $\frac{\partial y}{\partial t}$ . We first need to expand  $y(r, s, t) = \sum_{i=I}^{I=B} y_i f_i(r, s, t)$ , which gives

$$y = y_I f_I + y_J f_J + y_K f_K + y_L f_L + y_M f_M + y_N f_N + y_O f_O + y_P f_P + y_Q f_Q + y_R f_R + y_S f_S + y_T f_T + y_U f_U + y_V f_V + \dots$$

Expanding the above gives

$$\begin{aligned}
y(r, s, t) = & y_I \frac{1}{8} (1-r)(1-s)(1-t)(-r-s-t-2) + \\
& y_J \frac{1}{8} (1-r)(s+1)(1-t)(-r+s-t-2) + \\
& y_K \frac{1}{8} (1-r)(s+1)(t+1)(-r+s+t-2) + \\
& y_L \frac{1}{8} (1-r)(1-s)(t+1)(-r-s+t-2) + \\
& y_M \frac{1}{8} (r+1)(1-s)(1-t)(r-s-t-2) + \\
& y_N \frac{1}{8} (r+1)(s+1)(1-t)(r+s-t-2) + \\
& y_O \frac{1}{8} (r+1)(s+1)(t+1)(r+s+t-2) + \\
& y_P \frac{1}{8} (r+1)(1-s)(t+1)(r-s+t-2) + \\
& y_Q \frac{1}{4} (1-r) (1-s^2) (1-t) + \\
& y_R \frac{1}{4} (1-r)(s+1) (1-t^2) + \\
& y_S \frac{1}{4} (1-r) (1-s^2) (t+1) + \\
& y_T \frac{1}{4} (1-r)(1-s) (1-t^2) + \\
& y_U \frac{1}{4} (r+1) (1-s^2) (1-t) + \\
& y_V \frac{1}{4} (r+1)(s+1) (1-t^2) + \\
& y_W \frac{1}{4} (r+1) (1-s^2) (t+1) + \\
& y_X \frac{1}{4} (r+1)(1-s) (1-t^2) + \\
& y_Y \frac{1}{4} (1-r^2) (1-s)(1-t) + \\
& y_Z \frac{1}{4} (1-r^2) (s+1)(1-t) + \\
& y_A \frac{1}{4} (1-r^2) (s+1)(t+1) + \\
& y_B \frac{1}{4} (1-r^2) (1-s)(t+1)
\end{aligned}$$

Taking partial derivatives of the above w.r.t.  $r, s, t$  in turn, we see that it gives similar results to earlier ones, but the only difference is in the multipliers now being the  $y_i$  values of

coordinates instead of the  $x_i$  coordinates. This is reproduced again for completion

$$\begin{aligned}
\frac{\partial y}{\partial r} = & y_I \left( \frac{1}{8}(s-1)(t-1)(2r+s+t+1) \right) + \\
& y_J \left( \frac{1}{8}(s+1)(t-1)(-2r+s-t-1) \right) + \\
& y_K \left( -\frac{1}{8}(s+1)(t+1)(-2r+s+t-1) \right) + \\
& y_L \left( -\frac{1}{8}(s-1)(t+1)(2r+s-t+1) \right) + \\
& y_M \left( -\frac{1}{8}(s-1)(t-1)(-2r+s+t+1) \right) + \\
& y_N \left( -\frac{1}{8}(s+1)(t-1)(2r+s-t-1) \right) + \\
& y_O \left( \frac{1}{8}(s+1)(t+1)(2r+s+t-1) \right) + \\
& y_P \left( \frac{1}{8}(s-1)(t+1)(-2r+s-t+1) \right) + \\
& y_Q \left( -\frac{1}{4}(s^2-1)(t-1) \right) + \\
& y_R \left( \frac{1}{4}(s+1)(t^2-1) \right) + \\
& y_S \left( \frac{1}{4}(s^2-1)(t+1) \right) + \\
& y_T \left( -\frac{1}{4}(s-1)(t^2-1) \right) + \\
& y_U \left( \frac{1}{4}(s^2-1)(t-1) \right) + \\
& y_V \left( -\frac{1}{4}(s+1)(t^2-1) \right) + \\
& y_W \left( -\frac{1}{4}(s^2-1)(t+1) \right) + \\
& y_X \left( \frac{1}{4}(s-1)(t^2-1) \right) + \\
& y_Y \left( -\frac{1}{2}r(s-1)(t-1) \right) + \\
& y_Z \left( \frac{1}{2}r(s+1)(t-1) \right) + \\
& y_A \left( -\frac{1}{2}r(s+1)(t+1) \right) + \\
& y_B \left( \frac{1}{2}r(s-1)(t+1) \right)
\end{aligned}$$

Similarly, we now find  $\frac{\partial y}{\partial s}$  as above. This results in

$$\begin{aligned}
\frac{\partial y}{\partial s} = & y_I \left( \frac{1}{8}(r-1)(t-1)(r+2s+t+1) \right) + \\
& y_J \left( -\frac{1}{8}(r-1)(t-1)(r-2s+t+1) \right) + \\
& y_K \left( \frac{1}{8}(r-1)(t+1)(r-2s-t+1) \right) + \\
& y_L \left( -\frac{1}{8}(r-1)(t+1)(r+2s-t+1) \right) + \\
& y_M \left( \frac{1}{8}(r+1)(t-1)(r-2s-t-1) \right) + \\
& y_N \left( -\frac{1}{8}(r+1)(t-1)(r+2s-t-1) \right) + \\
& y_O \left( \frac{1}{8}(r+1)(t+1)(r+2s+t-1) \right) + \\
& y_P \left( -\frac{1}{8}(r+1)(t+1)(r-2s+t-1) \right) + \\
& y_Q \left( -\frac{1}{2}(r-1)s(t-1) \right) + \\
& y_R \left( \frac{1}{4}(r-1)(t^2-1) \right) + \\
& y_S \left( \frac{1}{2}(r-1)s(t+1) \right) + \\
& y_T \left( -\frac{1}{4}(r-1)(t^2-1) \right) + \\
& y_U \left( \frac{1}{2}(r+1)s(t-1) \right) + \\
& y_V \left( -\frac{1}{4}(r+1)(t^2-1) \right) + \\
& y_W \left( -\frac{1}{2}(r+1)s(t+1) \right) + \\
& y_X \left( \frac{1}{4}(r+1)(t^2-1) \right) + \\
& y_Y \left( -\frac{1}{4}(r^2-1)(t-1) \right) + \\
& y_Z \left( \frac{1}{4}(r^2-1)(t-1) \right) + \\
& y_A \left( -\frac{1}{4}(r^2-1)(t+1) \right) + \\
& y_B \left( \frac{1}{4}(r^2-1)(t+1) \right)
\end{aligned}$$

We now find  $\frac{\partial y}{\partial t}$  as above. This results in

$$\begin{aligned}
\frac{\partial y}{\partial t} = & y_I \left( \frac{1}{8}(r-1)(s-1)(r+s+2t+1) \right) + \\
& y_J \left( -\frac{1}{8}(r-1)(s+1)(r-s+2t+1) \right) + \\
& y_K \left( \frac{1}{8}(r-1)(s+1)(r-s-2t+1) \right) + \\
& y_L \left( -\frac{1}{8}(r-1)(s-1)(r+s-2t+1) \right) + \\
& y_M \left( \frac{1}{8}(r+1)(s-1)(r-s-2t-1) \right) + \\
& y_N \left( -\frac{1}{8}(r+1)(s+1)(r+s-2t-1) \right) + \\
& y_O \left( \frac{1}{8}(r+1)(s+1)(r+s+2t-1) \right) + \\
& y_P \left( -\frac{1}{8}(r+1)(s-1)(r-s+2t-1) \right) + \\
& y_Q \left( -\frac{1}{4}(r-1)(s^2-1) \right) + \\
& y_R \left( \frac{1}{2}(r-1)(s+1)t \right) + \\
& y_S \left( \frac{1}{4}(r-1)(s^2-1) \right) + \\
& y_T \left( -\frac{1}{2}(r-1)(s-1)t \right) + \\
& y_U \left( \frac{1}{4}(r+1)(s^2-1) \right) + \\
& y_V \left( -\frac{1}{2}(r+1)(s+1)t \right) + \\
& y_W \left( -\frac{1}{4}(r+1)(s^2-1) \right) + \\
& y_X \left( \frac{1}{2}(r+1)(s-1)t \right) + \\
& y_Y \left( -\frac{1}{4}(r^2-1)(s-1) \right) + \\
& y_Z \left( \frac{1}{4}(r^2-1)(s+1) \right) + \\
& y_A \left( -\frac{1}{4}(r^2-1)(s+1) \right) + \\
& y_B \left( \frac{1}{4}(r^2-1)(s-1) \right)
\end{aligned}$$

### 0.3.4 $z(r, s, t)$ terms

We now repeat all the above to find  $\frac{\partial z}{\partial r}$ ,  $\frac{\partial z}{\partial s}$  and  $\frac{\partial z}{\partial t}$ . These produce similar results to the above, but will have  $z_i$  as multipliers. We first need to expand  $z(r, s, t) = \sum_{i=I}^{I=B} z_i f_i(r, s, t)$ , which gives

$$z = z_I f_I + z_J f_J + z_K f_K + z_L f_L + z_M f_M + z_N f_N + z_O f_O + z_P f_P + z_Q f_Q + z_R f_R + z_S f_S + z_T f_T + z_U f_U + z_V f_V + z$$

Expanding the above gives

$$\begin{aligned}
z(r, s, t) = & z_I \frac{1}{8} (1-r)(1-s)(1-t)(-r-s-t-2) + \\
& z_J \frac{1}{8} (1-r)(s+1)(1-t)(-r+s-t-2) + \\
& z_K \frac{1}{8} (1-r)(s+1)(t+1)(-r+s+t-2) + \\
& z_L \frac{1}{8} (1-r)(1-s)(t+1)(-r-s+t-2) + \\
& z_M \frac{1}{8} (r+1)(1-s)(1-t)(r-s-t-2) + \\
& z_N \frac{1}{8} (r+1)(s+1)(1-t)(r+s-t-2) + \\
& z_O \frac{1}{8} (r+1)(s+1)(t+1)(r+s+t-2) + \\
& z_P \frac{1}{8} (r+1)(1-s)(t+1)(r-s+t-2) + \\
& z_Q \frac{1}{4} (1-r) (1-s^2) (1-t) + \\
& z_R \frac{1}{4} (1-r)(s+1) (1-t^2) + \\
& z_S \frac{1}{4} (1-r) (1-s^2) (t+1) + \\
& z_T \frac{1}{4} (1-r)(1-s) (1-t^2) + \\
& z_U \frac{1}{4} (r+1) (1-s^2) (1-t) + \\
& z_V \frac{1}{4} (r+1)(s+1) (1-t^2) + \\
& z_W \frac{1}{4} (r+1) (1-s^2) (t+1) + \\
& z_X \frac{1}{4} (r+1)(1-s) (1-t^2) + \\
& z_Y \frac{1}{4} (1-r^2) (1-s)(1-t) + \\
& z_Z \frac{1}{4} (1-r^2) (s+1)(1-t) + \\
& z_A \frac{1}{4} (1-r^2) (s+1)(t+1) + \\
& z_B \frac{1}{4} (1-r^2) (1-s)(t+1)
\end{aligned}$$

Taking partial derivative of the above w.r.t.  $r, s, t$  in turns gives the following

$$\begin{aligned}
\frac{\partial z}{\partial r} = & z_I \left( \frac{1}{8} (s-1)(t-1)(2r+s+t+1) \right) + \\
& z_J \left( \frac{1}{8} (s+1)(t-1)(-2r+s-t-1) \right) + \\
& z_K \left( -\frac{1}{8} (s+1)(t+1)(-2r+s+t-1) \right) + \\
& z_L \left( -\frac{1}{8} (s-1)(t+1)(2r+s-t+1) \right) + \\
& z_M \left( -\frac{1}{8} (s-1)(t-1)(-2r+s+t+1) \right) + \\
& z_N \left( -\frac{1}{8} (s+1)(t-1)(2r+s-t-1) \right) + \\
& z_O \left( \frac{1}{8} (s+1)(t+1)(2r+s+t-1) \right) + \\
& z_P \left( \frac{1}{8} (s-1)(t+1)(-2r+s-t+1) \right) + \\
& z_Q \left( -\frac{1}{4} (s^2-1)(t-1) \right) + \\
& z_R \left( \frac{1}{4} (s+1)(t^2-1) \right) + \\
& z_S \left( \frac{1}{4} (s^2-1)(t+1) \right) + \\
& z_T \left( -\frac{1}{4} (s-1)(t^2-1) \right) + \\
& z_U \left( \frac{1}{4} (s^2-1)(t-1) \right) + \\
& z_V \left( -\frac{1}{4} (s+1)(t^2-1) \right) + \\
& z_W \left( -\frac{1}{4} (s^2-1)(t+1) \right) + \\
& z_X \left( \frac{1}{4} (s-1)(t^2-1) \right) + \\
& z_Y \left( -\frac{1}{2} r(s-1)(t-1) \right) + \\
& z_Z \left( \frac{1}{2} r(s+1)(t-1) \right) + \\
& z_A \left( -\frac{1}{2} r(s+1)(t+1) \right) + \\
& z_B \left( \frac{1}{2} r(s-1)(t+1) \right)
\end{aligned}$$



Similarly,  $\frac{\partial z}{\partial s}$  results in

$$\begin{aligned}
\frac{\partial z}{\partial s} = & z_I \left( \frac{1}{8}(r-1)(t-1)(r+2s+t+1) \right) + \\
& z_J \left( -\frac{1}{8}(r-1)(t-1)(r-2s+t+1) \right) + \\
& z_K \left( \frac{1}{8}(r-1)(t+1)(r-2s-t+1) \right) + \\
& z_L \left( -\frac{1}{8}(r-1)(t+1)(r+2s-t+1) \right) + \\
& z_M \left( \frac{1}{8}(r+1)(t-1)(r-2s-t-1) \right) + \\
& z_N \left( -\frac{1}{8}(r+1)(t-1)(r+2s-t-1) \right) + \\
& z_O \left( \frac{1}{8}(r+1)(t+1)(r+2s+t-1) \right) + \\
& z_P \left( -\frac{1}{8}(r+1)(t+1)(r-2s+t-1) \right) + \\
& z_Q \left( -\frac{1}{2}(r-1)s(t-1) \right) + \\
& z_R \left( \frac{1}{4}(r-1)(t^2-1) \right) + \\
& z_S \left( \frac{1}{2}(r-1)s(t+1) \right) + \\
& z_T \left( -\frac{1}{4}(r-1)(t^2-1) \right) + \\
& z_U \left( \frac{1}{2}(r+1)s(t-1) \right) + \\
& z_V \left( -\frac{1}{4}(r+1)(t^2-1) \right) + \\
& z_W \left( -\frac{1}{2}(r+1)s(t+1) \right) + \\
& z_X \left( \frac{1}{4}(r+1)(t^2-1) \right) + \\
& z_Y \left( -\frac{1}{4}(r^2-1)(t-1) \right) + \\
& z_Z \left( \frac{1}{4}(r^2-1)(t-1) \right) + \\
& z_A \left( -\frac{1}{4}(r^2-1)(t+1) \right) + \\
& z_B \left( \frac{1}{4}(r^2-1)(t+1) \right)
\end{aligned}$$

And  $\frac{\partial z}{\partial t}$  gives

$$\begin{aligned}
\frac{\partial z}{\partial t} = & z_I \left( \frac{1}{8}(r-1)(s-1)(r+s+2t+1) \right) + \\
& z_J \left( -\frac{1}{8}(r-1)(s+1)(r-s+2t+1) \right) + \\
& z_K \left( \frac{1}{8}(r-1)(s+1)(r-s-2t+1) \right) + \\
& z_L \left( -\frac{1}{8}(r-1)(s-1)(r+s-2t+1) \right) + \\
& z_M \left( \frac{1}{8}(r+1)(s-1)(r-s-2t-1) \right) + \\
& z_N \left( -\frac{1}{8}(r+1)(s+1)(r+s-2t-1) \right) + \\
& z_O \left( \frac{1}{8}(r+1)(s+1)(r+s+2t-1) \right) + \\
& z_P \left( -\frac{1}{8}(r+1)(s-1)(r-s+2t-1) \right) + \\
& z_Q \left( -\frac{1}{4}(r-1)(s^2-1) \right) + \\
& z_R \left( \frac{1}{2}(r-1)(s+1)t \right) + \\
& z_S \left( \frac{1}{4}(r-1)(s^2-1) \right) + \\
& z_T \left( -\frac{1}{2}(r-1)(s-1)t \right) + \\
& z_U \left( \frac{1}{4}(r+1)(s^2-1) \right) + \\
& z_V \left( -\frac{1}{2}(r+1)(s+1)t \right) + \\
& z_W \left( -\frac{1}{4}(r+1)(s^2-1) \right) + \\
& z_X \left( \frac{1}{2}(r+1)(s-1)t \right) + \\
& z_Y \left( -\frac{1}{4}(r^2-1)(s-1) \right) + \\
& z_Z \left( \frac{1}{4}(r^2-1)(s+1) \right) + \\
& z_A \left( -\frac{1}{4}(r^2-1)(s+1) \right) + \\
& z_B \left( \frac{1}{4}(r^2-1)(s-1) \right)
\end{aligned}$$

Finally now we can determine the Jacobian and its determinant using the above expressions. This is done in the Matlab code provided. The following Jacobian Matrix is evaluated at each Gaussian integration point then its determinant is found using `det()` command.

$$\begin{pmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} & \frac{\partial z}{\partial r} \\ \frac{\partial x}{\partial x} & \frac{\partial y}{\partial y} & \frac{\partial z}{\partial z} \\ \frac{\partial s}{\partial x} & \frac{\partial s}{\partial y} & \frac{\partial s}{\partial z} \\ \frac{\partial t}{\partial x} & \frac{\partial t}{\partial y} & \frac{\partial t}{\partial z} \end{pmatrix}$$

### 0.3.5 results

The first step was to obtain estimate of the volume in order to verify that the volume calculation was valid and that the Jacobian was correct.

An independent small piece of code was written to plot the 3D shape and obtain its volume using a build-in function in the computer algebra program Mathematica. This is a plot of the 3D shape generated and below it is the code used to generate the plot, with the volume found shown in the title.

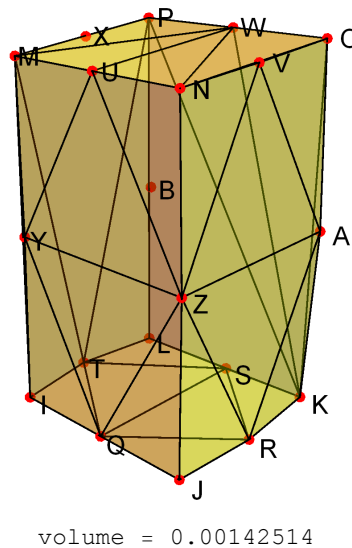


Figure 9: 3D plot of the volume in physical coordinates

```

1 xI=1;xJ=1.1;xK=1.09066;xL=0.99692;xM=1.0077;
2 xN=1.1069;xO=1.1035;xP=1.0046;xQ=1.05;xR=1.0992;
3 xS=1.0468;xT=0.9923;xU=1.0573;xV=1.1061;xW=1.0540;
4 xX=1.0069;xY=1.0051;xZ=1.1046;xA=1.1012;xB=1.0020;
5 xCoordinates={xI,xJ,xK,xL,xM,xN,xO,xP,xQ,xR,xS,xT,xU,xV,xW,xX,xY,xZ,xA,xB};
6 yI=0;yJ=0;yK=0;yL=0;yM=0.16559;
7 yN=0.17203;yO=0.17202;yP=0.16557;yQ=0;yR=0;
8 yS=0;yT=0;yU=0.16881;yV=0.17202;yW=0.16880;

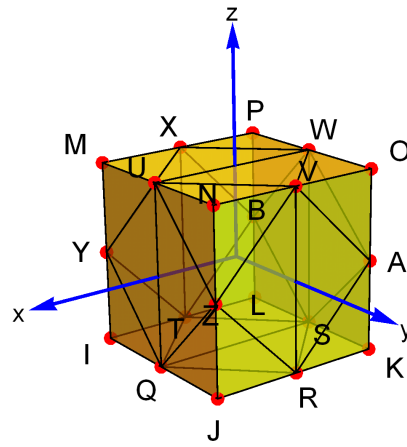
```

```

9 yX=0.16558;yY=0.082737;yZ=0.085964;yA=0.085938;yB=0.082709;
10 yCoordinates={yI,yJ,yK,yL,yM,yN,yO,yP,yQ,yR,yS,yT,yU,yV,yW,yX,yY,yZ,yA,yB};
11 zI=0;zJ=0;zK=-0.086305;zL=-0.078459;zM=0;
12 zN=0;zO=-0.08663;zP=-0.07882;zQ=0;zR=-0.043186;
13 zS=-0.082382;zT=-0.039260;zU=0;zV=-0.043328;zW=-0.082725;
14 zX=-0.039418;zY=0;zZ=0;zA=-0.086550;zB=-0.078731;
15 zCoordinates={zI,zJ,zK,zL,zM,zN,zO,zP,zQ,zR,zS,zT,zU,zV,zW,zX,zY,zZ,zA,zB};
16 data3D=Table[{xCoordinates[[i]],yCoordinates[[i]],
17             zCoordinates[[i]]},{i,1,Length@yCoordinates}];
18 nodes={"I","J","K","L","M","N","O","P","Q",
19        "R","S","T","U","V","W","X","Y","Z","A","B"};
20 Needs["TetGenLink`"];
21 {pts,surface}=TetGenConvexHull[data3D];
22 c=First@Last@Reap@Do[Sow@{nodes[[i]],data3D[[i]]},{i,1,Length[nodes]}];
23 Labeled[Graphics3D[{
24     {Red,PointSize[0.02],Point[data3D]},
25     {Yellow,Opacity[.4],EdgeForm[{Thin,Black]},
26     GraphicsComplex[data3D,Polygon[surface]]},
27     {Text[Style#[[1]],14],1.01*#[[2]]&/@c}
28 },Boxed->False,Axes->False,SphericalRegion->True,
29     ImageSize->300,ImageMargins->5],
30     Row[{"volume = ",RegionMeasure@ConvexHullMesh[data3D]}]]

```

We now know the volume should be  $0.0042514\text{cm}^3$  from the above independent verification. The Matlab code was now implemented, and the volume was verified to be the same. Also, a separate test was run to verify that  $\|J\| = 1$  for a test 3D volume which was aligned along the same orientation as the natural coordinates as shown below.



volume = 8.

Figure 10: 3D plot of the aligned volume used for verification

The code used to plot the above is

```

1 xI=1;xJ=1.1;xK=1.09066;xL=0.99692;xM=1.0077; xN=1.1069;xO=1.1035;xP=1.0046;
2 xQ=1.05;xR=1.0992;xS=1.0468;xT=0.9923;xU=1.0573;xV=1.1061;xW=1.0540;
3 xX=1.0069;xY=1.0051;xZ=1.1046;xA=1.1012;xB=1.0020;
4 xCoordinates={xI,xJ,xK,xL,xM,xN,xO,xP,xQ,xR,xS,xT,xU,xV,xW,xX,xY,xZ,xA,xB};
5 yI=0;yJ=0;yK=0;yL=0;yM=0.16559; yN=0.17203;yO=0.17202;yP=0.16557;yQ=0;yR=0;
6 yS=0;yT=0;yU=0.16881;yV=0.17202;yW=0.16880; yX=0.16558;yY=0.082737;yZ=0.085964;
7 yA=0.085938;yB=0.082709;
8 yCoordinates={yI,yJ,yK,yL,yM,yN,yO,yP,yQ,yR,yS,yT,yU,yV,yW,yX,yY,yZ,yA,yB};
9 zI=0;zJ=0;zK=-0.086305;zL=-0.078459;zM=0; zN=0;zO=-0.08663;zP=-0.07882;zQ=0;
10 zR=-0.043186; zS=-0.082382;zT=-0.039260;zU=0;zV=-0.043328;zW=-0.082725;
11 zX=-0.039418;zY=0;zZ=0;zA=-0.086550;zB=-0.078731;
12 zCoordinates={zI,zJ,zK,zL,zM,zN,zO,zP,zQ,zR,zS,zT,zU,zV,zW,zX,zY,zZ,zA,zB};
13 data3D=Table[{xCoordinates[[i]],yCoordinates[[i]],zCoordinates[[i]]},{i,1,
    Length@yCoordinates}];
14 nodes={"I","J","K","L","M","N","O","P","Q",
15        "R","S","T","U","V","W","X","Y","Z","A","B"};
16 Needs["TetGenLink`"];
17 {pts,surface}=TetGenConvexHull[data3D];
18 c=First@Last@Reap@Do[Sow@{nodes[[i]],data3D[[i]]},{i,1,Length[nodes]}];
19 Labeled[Graphics3D[{
20     {Red,PointSize[0.02],Point[data3D]},
21     {Yellow,Opacity[.4],EdgeForm[{Thin,Black]}},

```

```

22 GraphicsComplex[data3D,Polygon[surface]]},
23 {Text[Style#[[1]],14],1.01*#[[2]]&/@c}
24 },Boxed->False,Axes->False,SphericalRegion->True,
25 ImageSize->300,ImageMargins->5],
26 Row[{"volume = ",RegionMeasure@ConvexHullMesh[data3D]}]]

```

This test also passed in Matlab and gave a volume of  $8\text{cm}^3$  as expected. Here is the small code segment in Matlab which verifies the above.

```

1 wt(1) = 5/9;          wt(2) = 8/9;   wt(3) = 5/9;
2 gs(1) = -sqrt(3/5); gs(2) = 0;     gs(3) = sqrt(3/5) ;
3
4 %set nodal coordinates as cube of side 2, centered
5 %with natural coordinates origin
6 xI=1;      xJ=1;      xK=-1;      xL=-1;      xM=1;
7 xN=1;      xO=-1;     xP=-1;      xQ=1;      xR=0;
8 xS=-1;     xT=0;      xU=1;      xV=0;      xW=-1;
9 xX=0;      xY=1;      xZ=1;      xA=-1;     xB=-1;
10
11 xCoordinates=[xI,xJ,xK,xL,xM,xN,xO,xP,xQ,xR,xS,xT,xU,xV,...
12              xW,xX,xY,xZ,xA,xB];
13 yI=-1;     yJ=1;      yK=1;      yL=-1;     yM=-1;
14 yN=1;     yO=1;      yP=-1;     yQ=0;     yR=1;
15 yS=0;     yT=-1;     yU=0;     yV=1;     yW=0;
16 yX=-1;     yY=-1;     yZ=1;     yA=1;     yB=-1;
17 yCoordinates=[yI,yJ,yK,yL,yM,yN,yO,yP,yQ,yR,yS,yT,yU,yV,yW,...
18              yX,yY,yZ,yA,yB];
19
20 zI=-1;     zJ=-1;     zK=-1;     zL=-1;     zM=1;
21 zN=1;     zO=1;     zP=1;     zQ=-1;     zR=-1;
22 zS=-1;     zT=-1;     zU=1;     zV=1;     zW=1;
23 zX=1;     zY=0;     zZ=0;     zA=0;     zB=0;
24 zCoordinates=[zI,zJ,zK,zL,zM,zN,zO,zP,zQ,zR,zS,zT,zU,zV,...
25              zW,zX,zY,zZ,zA,zB];
26
27 %to collect sum of integrals at each node
28 the_sum = zeros(20,1);
29
30 %find the volume first, to use for verification.
31 for i=1:3
32     s = gs(i);
33     for j=1:3
34         t = gs(j);
35         for k=1:3
36             r = gs(k);
37             J = get_jacobian(r,s,t,xCoordinates,yCoordinates,zCoordinates);

```

```

38     detJ = det(J);
39     fprintf('|J|= %3.3f at Gaussian point [r=%3.3f,s=%3.3f,t=%3.3f]\n',
    detJ,r,s,t);
40     for ii=1:length(xCoordinates)
41         the_sum(ii)=the_sum(ii)+ wt(i)*wt(j)*wt(k)*f(ii,r,s,t)*detJ;
42     end
43 end
44 end
45 end
46
47 fprintf('volume for test is [%3.3f] (is it 8?)\n',sum(the_sum));
48 end

```

The output of the above is given below, with the rest of the program output.

The program `nma_EMA_471_HW5_problem_3.m` implements the main solution to this problem and included in the zip file. It runs both the Jacobian verification and the load calculations after that.

The main loop of the Matlab function iterates over three indices  $i, j, k$  from 1 to 3 each. In the inner most loop, it finds the determinant of the Jacobian, the mass density, and evaluates the shape function at the Gaussian points, then sums the result. At the end it prints the work-equivalent conversion for each of the twenty nodes. The following shows the main core of the program

```

1  for i=1:3
2      s = gs(i);
3      for j=1:3
4          t = gs(j);
5          for k=1:3
6              r = gs(k);
7              J = get_jacobian(r,s,t,xCoordinates,...
8                  yCoordinates,zCoordinates);
9              detJ = det(J);
10             mg = find_mass_density(r,s,t,xCoordinates,zCoordinates);
11
12             for ii=1:length(xCoordinates)
13                 the_sum(ii)=the_sum(ii)+...
14                     wt(i)*wt(j)*wt(k)*mg*g*f(ii,r,s,t)*detJ;
15             end
16         end
17     end
18 end

```

The final result is in the following table

Corner	work-equivalent load (z direction) $\frac{\text{gram-cm}}{\text{sec}^2}$	Newton units
I	-0.1097835	-0.000001934
J	-0.1336398	-0.000001990
K	-0.1159590	-0.000002
L	-0.1143340	-0.000001928
M	-0.1373666	-0.000001927
N	-0.1611689	-0.000001984
O	-0.1418730	-0.00000198
P	-0.1160910	-0.000001924
Q	0.1701952	0.000002614
R	0.1812529	0.000002739
S	0.0808510	0.000002592
T	0.0741811	0.000002513
U	0.2242564	0.000002589
V	0.2374809	0.000002728
W	0.1905987	0.000002589
X	0.1802555	0.000002476
Y	0.1723161	0.000002482
Z	0.2334560	0.000002734
A	0.1929484	0.000002710
B	0.0986489	0.000002487

Table 5: work-equivalent conversion at each corner, problem 3

We also see that the load on the corners is negative while on the middle nodes it is positive. This agrees with what one would expect as per class notes on the 8-node element. Only difference is that this is a 3D element.

The following is the console output from running the above program. It is implemented using Matlab 2016a. It starts with the Jacobian verification then it will run the main task next only if the verification passes.

```
>>nma_EMA_471_HW5_problem_3()
```

```
starting verification of Jacobian....
```



```

|J|= 1.000 at Gaussian point [r=-0.775,s=-0.775,t=-0.775]
|J|= 1.000 at Gaussian point [r=0.000,s=-0.775,t=-0.775]
|J|= 1.000 at Gaussian point [r=0.775,s=-0.775,t=-0.775]
|J|= 1.000 at Gaussian point [r=-0.775,s=-0.775,t=0.000]
|J|= 1.000 at Gaussian point [r=0.000,s=-0.775,t=0.000]
|J|= 1.000 at Gaussian point [r=0.775,s=-0.775,t=0.000]
|J|= 1.000 at Gaussian point [r=-0.775,s=-0.775,t=0.775]
|J|= 1.000 at Gaussian point [r=0.000,s=-0.775,t=0.775]
|J|= 1.000 at Gaussian point [r=0.775,s=-0.775,t=0.775]
|J|= 1.000 at Gaussian point [r=-0.775,s=0.000,t=-0.775]
|J|= 1.000 at Gaussian point [r=0.000,s=0.000,t=-0.775]
|J|= 1.000 at Gaussian point [r=0.775,s=0.000,t=-0.775]
|J|= 1.000 at Gaussian point [r=-0.775,s=0.000,t=0.000]
|J|= 1.000 at Gaussian point [r=0.000,s=0.000,t=0.000]
|J|= 1.000 at Gaussian point [r=0.775,s=0.000,t=0.000]
|J|= 1.000 at Gaussian point [r=-0.775,s=0.000,t=0.775]
|J|= 1.000 at Gaussian point [r=0.000,s=0.000,t=0.775]
|J|= 1.000 at Gaussian point [r=0.775,s=0.000,t=0.775]
|J|= 1.000 at Gaussian point [r=-0.775,s=0.775,t=-0.775]
|J|= 1.000 at Gaussian point [r=0.000,s=0.775,t=-0.775]
|J|= 1.000 at Gaussian point [r=0.775,s=0.775,t=-0.775]
|J|= 1.000 at Gaussian point [r=-0.775,s=0.775,t=0.000]
|J|= 1.000 at Gaussian point [r=0.000,s=0.775,t=0.000]
|J|= 1.000 at Gaussian point [r=0.775,s=0.775,t=0.000]
|J|= 1.000 at Gaussian point [r=-0.775,s=0.775,t=0.775]
|J|= 1.000 at Gaussian point [r=0.000,s=0.775,t=0.775]
|J|= 1.000 at Gaussian point [r=0.775,s=0.775,t=0.775]
volume for test is [8.000] (is it 8?)
!! passed Jacobian test. Will run main program now

volume is 0.001426 cm^3

load at corner I = -0.1934002 [gram-cm/sec^2] = -0.000001934 N
load at corner J = -0.1990333 [gram-cm/sec^2] = -0.000001990 N
load at corner K = -0.2000363 [gram-cm/sec^2] = -0.000002000 N
load at corner L = -0.1928213 [gram-cm/sec^2] = -0.000001928 N
load at corner M = -0.1927113 [gram-cm/sec^2] = -0.000001927 N
load at corner N = -0.1984265 [gram-cm/sec^2] = -0.000001984 N
load at corner O = -0.1980087 [gram-cm/sec^2] = -0.000001980 N
load at corner P = -0.1923718 [gram-cm/sec^2] = -0.000001924 N
load at corner Q = 0.2614284 [gram-cm/sec^2] = 0.000002614 N
load at corner R = 0.2739156 [gram-cm/sec^2] = 0.000002739 N
load at corner S = 0.2592383 [gram-cm/sec^2] = 0.000002592 N
load at corner T = 0.2513418 [gram-cm/sec^2] = 0.000002513 N
load at corner U = 0.2589067 [gram-cm/sec^2] = 0.000002589 N

```

```

load at corner V = 0.2727979 [gram-cm/sec^2] = 0.000002728 N
load at corner W = 0.2588535 [gram-cm/sec^2] = 0.000002589 N
load at corner X = 0.2475648 [gram-cm/sec^2] = 0.000002476 N
load at corner Y = 0.2481743 [gram-cm/sec^2] = 0.000002482 N
load at corner Z = 0.2733760 [gram-cm/sec^2] = 0.000002734 N
load at corner A = 0.2710102 [gram-cm/sec^2] = 0.000002710 N
load at corner B = 0.2487254 [gram-cm/sec^2] = 0.000002487 N
>>

```

```

1 function nma_EMA_471_HW5_problem_3()
2 %Solves problem 3, HW5, EMA 471
3 %Nasser M. Abbasi
4
5 close all; clc;
6
7 status = do_jacobian_test();
8 if ~status
9     error('failed jacobian test. Internal code error\n');
10 else
11     fprintf('!! passed Jacobian test. Will run main program now\n\n');
12     do_main_program();
13 end
14
15 end
16 %=====
17 function status = do_jacobian_test()
18 %This function checks that |J|=1 at each Gaussian point.
19 %This verifies the code is ok before
20 %running the main program. This also checks that volume is
21 % 2*2*2=8 cm^3 since we are using a cube with nodal coordinates
22 % with side length = 2 cm and it is aligned along the natural
23 %coordinates and centered at the natural coordinates origin also.
24 %
25
26 status = true;
27
28 wt(1) = 5/9;      wt(2) = 8/9;      wt(3) = 5/9;
29 gs(1) = -sqrt(3/5); gs(2) = 0;      gs(3) = sqrt(3/5) ;
30
31 %set nodal coordinates as cube of side 2,
32 %centered with natural coordinates origin
33 xI=1;      xJ=1;      xK=-1;      xL=-1;      xM=1;
34 xN=1;      xO=-1;     xP=-1;      xQ=1;      xR=0;
35 xS=-1;     xT=0;      xU=1;      xV=0;      xW=-1;
36 xX=0;      xY=1;      xZ=1;      xA=-1;     xB=-1;
37
38 xCoordinates=[xI,xJ,xK,xL,xM,xN,xO,xP,xQ,xR,xS,xT,xU,xV, ...
39              xW,xX,xY,xZ,xA,xB];

```

```

40
41 yI=-1;      yJ=1;      yK=1;      yL=-1;      yM=-1;
42 yN=1;      yO=1;      yP=-1;      yQ=0;      yR=1;
43 yS=0;      yT=-1;      yU=0;      yV=1;      yW=0;
44 yX=-1;      yY=-1;      yZ=1;      yA=1;      yB=-1;
45 yCoordinates=[yI,yJ,yK,yL,yM,yN,yO,yP,yQ,yR,yS,yT,yU,yV,yW,...
46              yX,yY,yZ,yA,yB];
47
48 zI=-1;      zJ=-1;      zK=-1;      zL=-1;      zM=1;
49 zN=1;      zO=1;      zP=1;      zQ=-1;      zR=-1;
50 zS=-1;      zT=-1;      zU=1;      zV=1;      zW=1;
51 zX=1;      zY=0;      zZ=0;      zA=0;      zB=0;
52 zCoordinates=[zI,zJ,zK,zL,zM,zN,zO,zP,zQ,zR,zS,zT,zU,zV,zW,...
53              zX,zY,zZ,zA,zB];
54
55 the_sum = zeros(20,1); %to collect sum of integrals at each node
56
57 %find the volume first, to use for verification.
58 format short;
59 format compact;
60 fprintf('starting verification of Jacobian...\n');
61 for i=1:3
62     s = gs(i);
63
64     for j=1:3
65         t = gs(j);
66
67         for k=1:3
68             r = gs(k);
69
70             J = get_jacobian(r,s,t,xCoordinates,...
71                             yCoordinates,zCoordinates);
72             detJ = det(J);
73             fprintf('|J|= %3.3f at Gaussian point [r=%3.3f,s=%3.3f,t=%3.3f]\n',...
74                     detJ,r,s,t);
75             if detJ <=0
76                 status = false; %FAILED TEST
77                 return;
78             end
79             for ii=1:length(xCoordinates)
80                 the_sum(ii)=the_sum(ii)+ ...
81                     wt(i)*wt(j)*wt(k)*f(ii,r,s,t)*detJ;
82             end
83         end
84     end
85 end
86

```

```

87 fprintf('volume for test is [%3.3f] (is it 8?)\n',sum(the_sum));
88
89 end
90
91 %======
92 function do_main_program()
93 wt(1) = 5/9;          wt(2) = 8/9;   wt(3) = 5/9;
94 gs(1) = -sqrt(3/5);  gs(2) = 0;    gs(3) = sqrt(3/5) ;
95
96 xI=1;      xJ=1.1;      xK=1.09066;    xL=0.99692;    xM=1.0077;
97 xN=1.1069; xO=1.1035;    xP=1.0046;    xQ=1.05;       xR=1.0992;
98 xS=1.0468; xT=0.9923;    xU=1.0573;    xV=1.1061;    xW=1.0540;
99 xX=1.0069; xY=1.0051;    xZ=1.1046;    xA=1.1012;    xB=1.0020;
100
101 xCoordinates=[xI,xJ,xK,xL,xM,xN,xO,xP,xQ,xR,xS,xT,xU,...
102              xV,xW,xX,xY,xZ,xA,xB];
103
104 yI=0;      yJ=0;      yK=0;      yL=0;      yM=0.16559;
105 yN=0.17203; yO=0.17202; yP=0.16557; yQ=0;      yR=0;
106 yS=0;      yT=0;      yU=0.16881; yV=0.17202; yW=0.16880;
107 yX=0.16558; yY=0.082737; yZ=0.085964; yA=0.085938; yB=0.082709;
108 yCoordinates=[yI,yJ,yK,yL,yM,yN,yO,yP,yQ,yR,yS,yT,yU,yV,...
109              yX,yY,yZ,yA,yB];
110
111 zI=0;      zJ=0;      zK=-0.086305; zL=-0.078459; zM=0;
112 zN=0;      zO=-0.08663; zP=-0.07882; zQ=0;      zR=-0.043186;
113 zS=-0.082382; zT=-0.039260; zU=0;      zV=-0.043328; zW=-0.082725;
114 zX=-0.039418; zY=0;      zZ=0;      zA=-0.086550; zB=-0.078731;
115 zCoordinates=[zI,zJ,zK,zL,zM,zN,zO,zP,zQ,zR,zS,zT,zU,zV,zW,zX,...
116              zY,zZ,zA,zB];
117
118 the_sum = zeros(20,1);
119 g        = 9.81*100; %acceleration g in cm per sec^2
120
121 %find the volume first, to use for verification.
122 for i=1:3
123     s = gs(i);
124
125     for j=1:3
126         t = gs(j);
127
128         for k=1:3
129             r = gs(k);
130
131             J = get_jacobian(r,s,t,xCoordinates,...
132                             yCoordinates,zCoordinates);
133             detJ = det(J);

```

```

134         if detJ <=0
135             error('code internal error, invalid jacobian det. %7.6f\n',detJ);
136         end
137         for ii=1:length(xCoordinates)
138             the_sum(ii)=the_sum(ii)+ ...
139                 wt(i)*wt(j)*wt(k)*f(ii,r,s,t)*detJ;
140         end
141     end
142 end
143 end
144
145 fprintf('volume is %7.6f cm^3\n', sum(the_sum));
146
147 for i=1:3
148     s = gs(i);
149
150     for j=1:3
151         t = gs(j);
152
153         for k=1:3
154             r = gs(k);
155
156             J = get_jacobian(r,s,t,xCoordinates,...
157                             yCoordinates,zCoordinates);
158
159             detJ = det(J);
160             if detJ <=0
161                 error('code internal error, invalid jacobian det. %7.6f\n',detJ);
162             end
163             mg = find_mass_density(r,s,t,xCoordinates,zCoordinates);
164
165             for ii=1:length(xCoordinates)
166                 the_sum(ii)=the_sum(ii)+...
167                     wt(i)*wt(j)*wt(k)*mg*g*f(ii,r,s,t)*detJ;
168             end
169         end
170     end
171 end
172
173 map_node={'I','J','K','L','M','N','O','P','Q','R','S',...
174           'T','U','V',...
175           'W','X','Y','Z','A','B'};
176
177 for i=1:length(xCoordinates)
178     fprintf('load at corner %c = %9.7f [gram-cm/sec^2] = %10.9f N\n',...
179           map_node{i},the_sum(i),the_sum(i)*10^(-3)*10^(-2));
180 end

```

```

181 end
182 %=====
183 function mass_density = find_mass_density(r,s,t,...
184         xCoordinates,zCoordinates)
185
186     p0 = 1;
187     X = 0;
188     for i=1:20
189         X = X + xCoordinates(i)*f(i,r,s,t);
190     end
191
192     Z = 0;
193     for i=1:20
194         Z = Z + zCoordinates(i)*f(i,r,s,t);
195     end
196
197     mass_density = p0*(X^2+Z^2);
198 end
199 %=====
200 function the_shape_function=f(idx,r,s,t)
201 switch idx
202     case 1 %I
203         the_shape_function=(1/8)*(1-r)*(1-s)*(1-t)*(-r-s-t-2);
204     case 2 %J
205         the_shape_function=(1/8)*(1-r)*(s+1)*(1-t)*(-r+s-t-2);
206     case 3 %K
207         the_shape_function=(1/8)*(1-r)*(s+1)*(t+1)*(-r+s+t-2);
208     case 4 %L
209         the_shape_function=(1/8)*(1-r)*(1-s)*(t+1)*(-r-s+t-2);
210     case 5 %M
211         the_shape_function=(1/8)*(r+1)*(1-s)*(1-t)*(r-s-t-2);
212     case 6 %N
213         the_shape_function=(1/8)*(r+1)*(s+1)*(1-t)*(r+s-t-2);
214     case 7 %O
215         the_shape_function=(1/8)*(r+1)*(s+1)*(t+1)*(r+s+t-2);
216     case 8 %P
217         the_shape_function=(1/8)*(r+1)*(1-s)*(t+1)*(r-s+t-2);
218     case 9 %Q
219         the_shape_function=(1/4)*(1-r)*(1-s^2)*(1-t);
220     case 10 %R
221         the_shape_function=(1/4)*(1-r)*(s+1)*(1-t^2);
222     case 11 %S
223         the_shape_function=(1/4)*(1-r)*(1-s^2)*(t+1);
224     case 12 %T
225         the_shape_function=(1/4)*(1-r)*(1-s)*(1-t^2);
226     case 13 %U
227         the_shape_function=(1/4)*(1+r)*(1-s^2)*(1-t);

```

```

228     case 14 %V
229         the_shape_function=(1/4)*(r+1)*(s+1)*(1-t^2);
230     case 15 %W
231         the_shape_function=(1/4)*(r+1)*(1-s^2)*(t+1);
232     case 16 %X
233         the_shape_function=(1/4)*(r+1)*(1-s)*(1-t^2);
234     case 17 %Y
235         the_shape_function=(1/4)*(1-r^2)*(1-s)*(1-t);
236     case 18 %Z
237         the_shape_function=(1/4)*(1-r^2)*(s+1)*(1-t);
238     case 19 %A
239         the_shape_function=(1/4)*(1-r^2)*(s+1)*(t+1);
240     case 20 %B
241         the_shape_function=(1/4)*(1-r^2)*(1-s)*(t+1);
242 end
243 end
244
245 %----- internal function
246 function the_result=dds(c,r,s,t)
247 %find dx/ds or dy/ds or dz/ds. These all have same
248 %form, except for the multiplier c, which is the nodal
249 %coordinates, passed in.
250 the_result=c(1)*((1/8)*(r-1)*(t-1)*(r+2*s+t+1))+...
251     c(2)*(-(1/8)*(r-1)*(t-1)*(r-2*s+t+1))+...
252     c(3)*((1/8)*(r-1)*(t+1)*(r-2*s-t+1))+...
253     c(4)*(-(1/8)*(r-1)*(t+1)*(r+2*s-t+1))+...
254     c(5)*((1/8)*(r+1)*(t-1)*(r-2*s-t-1))+...
255     c(6)*(-(1/8)*(r+1)*(t-1)*(r+2*s-t-1))+...
256     c(7)*((1/8)*(r+1)*(t+1)*(r+2*s+t-1))+...
257     c(8)*(-(1/8)*(r+1)*(t+1)*(r-2*s+t-1))+...
258     c(9)*(-(1/2)*(r-1)*s*(t-1))+...
259     c(10)*((1/4)*(r-1)*(t^2-1))+...
260     c(11)*((1/2)*(r-1)*s*(t+1))+...
261     c(12)*(-(1/4)*(r-1)*(t^2-1))+...
262     c(13)*((1/2)*(r+1)*s*(t-1))+...
263     c(14)*(-(1/4)*(r+1)*(t^2-1))+...
264     c(15)*(-(1/2)*(r+1)*s*(t+1))+...
265     c(16)*((1/4)*(r+1)*(t^2-1))+...
266     c(17)*(-(1/4)*(r^2-1)*(t-1))+...
267     c(18)*((1/4)*(r^2-1)*(t-1))+...
268     c(19)*(-(1/4)*(r^2-1)*(t+1))+...
269     c(20)*((1/4)*(r^2-1)*(t+1));
270 end
271 %----- internal function
272 function the_result=ddt(c,r,s,t)
273 %find dx/dt or dy/dt or dz/dt. These all have same form,
274 %except for the multiplier c, which is the nodal coordinates,

```

```

275 %passed in.
276
277 the_result=c(1)*((1/8)*(r-1)*(s-1)*(r+s+2*t+1))+...
278     c(2)*(-(1/8)*(r-1)*(s+1)*(r-s+2*t+1))+...
279     c(3)*((1/8)*(r-1)*(s+1)*(r-s-2*t+1))+...
280     c(4)*(-(1/8)*(r-1)*(s-1)*(r+s-2*t+1))+...
281     c(5)*((1/8)*(r+1)*(s-1)*(r-s-2*t-1))+...
282     c(6)*(-(1/8)*(r+1)*(s+1)*(r+s-2*t-1))+...
283     c(7)*((1/8)*(r+1)*(s+1)*(r+s+2*t-1))+...
284     c(8)*(-(1/8)*(r+1)*(s-1)*(r-s+2*t-1))+...
285     c(9)*(-(1/4)*(r-1)*(s^2-1))+...
286     c(10)*((1/2)*(r-1)*(s+1)*t)+...
287     c(11)*((1/4)*(r-1)*(s^2-1))+...
288     c(12)*(-(1/2)*(r-1)*(s-1)*t)+...
289     c(13)*((1/4)*(r+1)*(s^2-1))+...
290     c(14)*(-(1/2)*(r+1)*(s+1)*t)+...
291     c(15)*(-(1/4)*(r+1)*(s^2-1))+...
292     c(16)*((1/2)*(r+1)*(s-1)*t)+...
293     c(17)*(-(1/4)*(r^2-1)*(s-1))+...
294     c(18)*((1/4)*(r^2-1)*(s+1))+...
295     c(19)*(-(1/4)*(r^2-1)*(s+1))+...
296     c(20)*((1/4)*(r^2-1)*(s-1));
297 end
298 %----- internal function
299 function the_result=ddr(c,r,s,t)
300 %find dx/dr or dy/dr or dz/dr. These all have same form,
301 %except for the multiplier c, which is the nodal coordinates,
302 %passed in.
303
304 the_result=c(1)*((1/8)*(s-1)*(t-1)*(2*r+s+t+1))+...
305     c(2)*((1/8)*(s+1)*(t-1)*(-2*r+s-t-1))+...
306     c(3)*(-(1/8)*(s+1)*(t+1)*(-2*r+s+t-1))+...
307     c(4)*(-(1/8)*(s-1)*(t+1)*(2*r+s-t+1))+...
308     c(5)*(-(1/8)*(s-1)*(t-1)*(-2*r+s+t+1))+...
309     c(6)*(-(1/8)*(s+1)*(t-1)*(2*r+s-t-1))+...
310     c(7)*((1/8)*(s+1)*(t+1)*(2*r+s+t-1))+...
311     c(8)*((1/8)*(s-1)*(t+1)*(-2*r+s-t+1))+...
312     c(9)*(-(1/4)*(s^2-1)*(t-1))+...
313     c(10)*((1/4)*(s+1)*(t^2-1))+...
314     c(11)*((1/4)*(s^2-1)*(t+1))+...
315     c(12)*(-(1/4)*(s-1)*(t^2-1))+...
316     c(13)*((1/4)*(s^2-1)*(t-1))+...
317     c(14)*(-(1/4)*(s+1)*(t^2-1))+...
318     c(15)*(-(1/4)*(s^2-1)*(t+1))+...
319     c(16)*((1/4)*(s-1)*(t^2-1))+...
320     c(17)*(-(1/2)*r*(s-1)*(t-1))+...
321     c(18)*((1/2)*r*(s+1)*(t-1))+...

```



```
322     c(19)*(-(1/2)*r*(s+1)*(t+1))+...
323     c(20)*((1/2)*r*(s-1)*(t+1));
324 end
325 %=====
326 function J=get_jacobian(r,s,t,xCoordinates,yCoordinates,zCoordinates)
327
328 J = [ddr(xCoordinates,r,s,t),ddr(yCoordinates,r,s,t),...
329     ddr(zCoordinates,r,s,t);
330     dds(xCoordinates,r,s,t),dds(yCoordinates,r,s,t),...
331     dds(zCoordinates,r,s,t);
332     ddt(xCoordinates,r,s,t),ddt(yCoordinates,r,s,t),...
333     ddt(zCoordinates,r,s,t)
334 ];
335 end
```