JOB MATRIK

1. TUJUAN

1.1. Mahasiswa mampu membuat fungsi dengan bahasa pemrograman operasi dasar matrik.
1.2. Mahasiswa mampu Menggunakan funsgsi fungsi operasi matrik yang tela dibuat.

2. TEORI DASAR

Pada bagian ini setiap mahasiswa melengkapi sebuah teori yang berhubungan dengan operasi matrik, antara lain

2.1. Matrik bujur sangkar
2.2. Matrik identitas
2.3. Zero matrik
2.4. Perkalian matrik
2.5. Transpose matrik
2.6. Minor matrik
2.7. Sub matrik
2.8. Determinan matrik
2.9. Invers matrik

3. LANGKAH PERCOBAAN

3.1. Agar anda terampil tentang sintak bahasa pemrogramanan C/C++, berikut ini adalah kode program untuk operasi matrik ordo 3x3. Ketiklah listing program no 1 hingga 444 dengan syarat semua mahasiswa harus kerja dengan cara membagi rata sehingga lebih cepat.

```
1: /*****************************************************************************************************************
2: *   JOB MATRIK
3: *   MEMBUAT FUNGSI FUNGSI DASAR OPERASI MATRIK
4: *
5: *
6: *****************************************************************************************************************/
7:
8: #include <stdio.h>
9: #include <stdlib.h>
10: #include <math.h>
11: #include <string.h>
12:
13: /*-------------------------------------------------------------------*/
14: #define MOrder 3                /* Matrix Order */
15: typedef double matrix_t[MOrder][MOrder];
16:
17: typedef struct
18: {
19:     matrix_t dummy_matrix;
20: } matrix_structure_t, *matrix_ptr_t;
21:
22: #define M_copy_matrix(src, dst)\
23: *((matrix_ptr_t) (dst)) = *((matrix_ptr_t) (src))
24: typedef double vector_t[MOrder];
25:
26: typedef struct
27: {
28:     vector_t dummy_vector;
29: } vector_structure_t, *vector_ptr_t;
30:
31: #define M_copy_vector(src, dst) *((vector_ptr_t) (dst)) = *((vector_ptr_t) (src))
32: #define M_round_to_zero(x) ((fabs(x) > 1e-33) ? (x) : 0.0)
33: /*-------------------------------------------------------------------*/
34: extern void M_print_matrix(char *name, matrix_t m);
35: extern void M_print_sub_matrix(char *name, matrix_t m, int order);
36: /*-------------------------------------------------------------------*/
```

```
37:
38: void M_set_identity_matrix(matrix_t m)
39: {
40:     int r, c;
41:     for (r = 0; r < MOrder; r++)
42:         for (c = 0; c < MOrder; c++)
43:             if (r == c)
44:                 m[r][c] = 1.0;
45:             else
46:                 m[r][c] = 0.0;
47: }                               /* M_set_identity_matrix */
48:
49: /*---------------------------------------------------------------------*/
50:
51: void M_set_rc_reversal_matrix(matrix_t m)
52: {
53:     int r, c;
54:
55:     for (r = 0; r < MOrder; r++)
56:         for (c = 0; c < MOrder; c++)
57:             if (r == (MOrder - (c + 1)))
58:                 m[r][c] = 1.0;
59:             else
60:                 m[r][c] = 0.0;
61: }                               /* M_set_rc_reversal_matrix */
62:
63: /*---------------------------------------------------------------------*/
64:
65: void M_set_scalar_matrix(matrix_t m, double s)
66: {
67:     int r, c;
68:     for (r = 0; r < MOrder; r++)
69:         for (c = 0; c < MOrder; c++)
70:             if (r == c)
71:                 m[r][c] = s;
72:             else
73:                 m[r][c] = 0.0;
74: }                               /* M_set_scalar_matrix */
75:
76: /*---------------------------------------------------------------------*/
77:
78: void M_set_zero_matrix(matrix_t m)
79: {
80:     int r, c;
81:     for (r = 0; r < MOrder; r++)
82:         for (c = 0; c < MOrder; c++)
83:             m[r][c] = 0.0;
84: }
85:
86: /*---------------------------------------------------------------------*/
87: void M_transpose_matrix(matrix_t m, matrix_t tm)
88: {
89:     int r, c;
90:     for (r = 0; r < MOrder; r++)
91:         for (c = 0; c < MOrder; c++)
92:             tm[c][r] = m[r][c];
93: }                               /* M_transpose_matrix */
94:
95: /*---------------------------------------------------------------------*/
 96:
 97: void M_matrix_add(matrix_t m1, matrix_t m2, matrix_t m3)
 98: {
 99:     int r, c;
100:     for (r = 0; r < MOrder; r++)
101:         for (c = 0; c < MOrder; c++)
102:             m3[r][c] = m1[r][c] + m2[r][c];
103: }
104:
105: /*---------------------------------------------------------------------*/
106:
107: void M_matrix_subtract(matrix_t m1, matrix_t m2, matrix_t m3)
108: {
109:     int r, c;
110:     for (r = 0; r < MOrder; r++)
```

```
111:         for (c = 0; c < MOrder; c++)
112:             m3[r][c] = m1[r][c] - m2[r][c];
113: }                               /* M_matrix_subtract */
114:
115: /*---------------------------------------------------------------------*/
116:
117: void M_matrix_by_scalar(matrix_t m1, double s, matrix_t m2)
118: {
119:     int r, c;
120:     for (r = 0; r < MOrder; r++)
121:         for (c = 0; c < MOrder; c++)
122:             m2[r][c] = m1[r][c] * s;
123: }                               /* M_matrix_by_scalar */
124:
125: /*---------------------------------------------------------------------*/

126:
127: double M_dot_product(vector_t v1, vector_t v2)
128: {
129:     double dp;
130:     int e;
131:     dp = 0;
132:     for (e = 0; e < MOrder; e++)
133:         dp += v1[e] * v2[e];
134:     return (dp);
135: }                               /* M_dot_product */
136:
137: /*---------------------------------------------------------------------*/

138:
139: void M_tensor_product(vector_t v1, vector_t v2, matrix_t m)
140: {
141:     int r, c;
142:     for (r = 0; r < MOrder; r++)
143:         for (c = 0; c < MOrder; c++)
144:             m[r][c] = v1[r] * v2[c];
145: }                               /* M_tensor_product */
146:
147: /*---------------------------------------------------------------------*/
148:
149: void M_matrix_by_vector(matrix_t m, vector_t v1, vector_t v2)
150: {
151:     int r, c;
152:     for (r = 0; r < MOrder; r++)
153:     {
154:         v2[r] = 0;
155:         for (c = 0; c < MOrder; c++)
156:             v2[r] += m[r][c] * v1[c];
157:     }
158:
159: }                               /* M_matrix_by_vector */
160:
161: /*---------------------------------------------------------------------*/
162:
163: void M_vector_by_matrix(vector_t v1, matrix_t m, vector_t v2)
164: {
165:     int r, c;
166:     for (c = 0; c < MOrder; c++)
167:     {
168:         v2[c] = 0;
169:         for (r = 0; r < MOrder; r++)
170:             v2[r] += v1[c] * m[r][c];
171:     }
172: }                               /* M_vector_by_matrix */
173:
174: /*---------------------------------------------------------------------*/
```

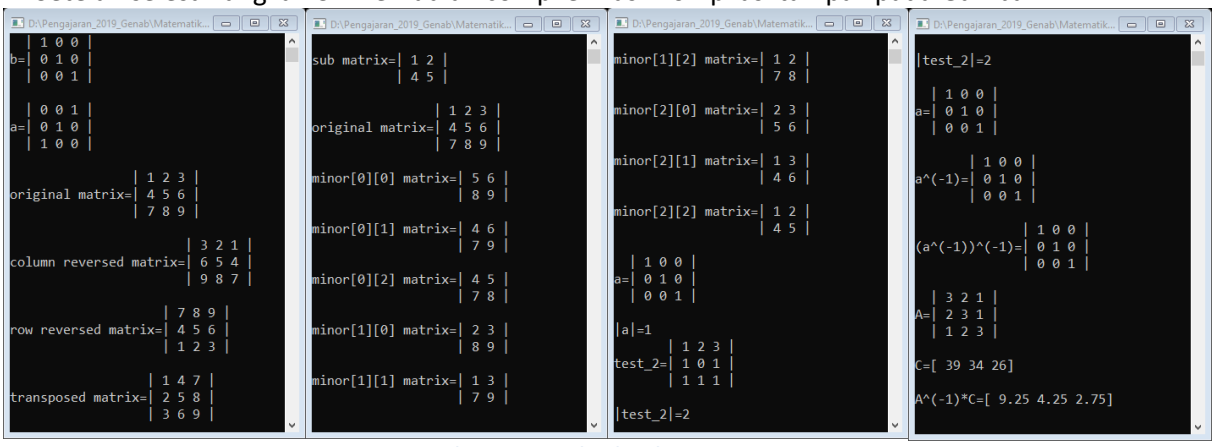
```
175: void M_matrix_multiply(matrix_t m1, matrix_t m2, matrix_t m3)
176: {
177:     int r1, c1, c2;
178:     for (r1 = 0; r1 < MOrder; r1++)
179:         for (c2 = 0; c2 < MOrder; c2++)
180:         {
181:             m3[r1][c2] = 0.0;
182:             for (c1 = 0; c1 < MOrder; c1++)
183:                 m3[r1][c2] += m1[r1][c1] * m2[c1][c2];
184:         }
185: }                               /* M_matrix_multiply */
186:
187: /*---------------------------------------------------------------------*/
188:
189: void M_minor_matrix(matrix_t m, int order, int i, int j, matrix_t mm)
190: {
191:     int r, c, mr, mc;
192:     mr = -1;
193:     for (r = 0; r < order; r++)
194:         if (r != i)
195:         {
196:             mr++;
197:             mc = -1;
198:             for (c = 0; c < order; c++)
199:                 if (c != j)
200:                 {
201:                     mc++;
202:                     mm[mr][mc] = m[r][c];
203:                 }
204:         }
205: }                               /* M_minor_matrix */
206:
207: /*---------------------------------------------------------------------*/
208:
209: double M_determinant(matrix_t m, int order)
210: {
211:     matrix_t mm;
212:     double d;
213:     int r;
214:     if (order == 1)
215:         d = m[0][0];
216:     else
217:     {
218:         d = 0;
219:         for (r = 0; r < order; r++)
220:         {
221:             M_minor_matrix(m, order, r, 0, mm);
222:             d = d + pow(-1.0, (double)r) * m[r][0] * M_determinant(mm, order - 1);
223:         }
224:     }
225:     return (d);
226: }                               /* M_determilnant */
227:
228: /*---------------------------------------------------------------------*/
229:
230: void M_cofactor_matrix(matrix_t m, matrix_t cm)
231: {
232:     matrix_t mm;
233:     int r, c;
234:     for (r = 0; r < MOrder; r++)
235:         for (c = 0; c < MOrder; c++)
236:         {
237:             M_minor_matrix(m, MOrder, r, c, mm);
238:             cm[r][c] = pow(-1.0, (double)(r + c)) * M_determinant(mm, MOrder - 1);
239:         }
240: }                               /* M_cofactor_matrix */
241:
242: /*---------------------------------------------------------------------*/
243:
244: void M_adjugate_matrix(matrix_t m, matrix_t am)
245: {
246:     matrix_t cm;
247:     M_cofactor_matrix(m, cm);
248:     M_transpose_matrix(cm, am);
249: }                               /* M_adjugate_matrix */
250:
251: /*---------------------------------------------------------------------*/
```

```
252:
253: void M_inverse_matrix(matrix_t m, matrix_t im)
254: {
255:     matrix_t am;
256:     double d;
257:     int r, c;
258:     d = M_determinant(m, MOrder);
259:     if (fabs(d) > 0.0)
260:     {
261:         M_adjugate_matrix(m, am);
262:         for (r = 0; r < MOrder; r++)
263:             for (c = 0; c < MOrder; c++)
264:                 im[r][c] = am[r][c] / d;
265:     }
266:     else
267:     {
268:         printf("M_inverse_matrix: Input matrix is singular!\n");
269:     }
270: }                               /* M_inverse_matrix */
271:
272: /*---------------------------------------------------------------------*/
273:
274: void M_solve_system(matrix_t A, vector_t C, vector_t X)
275: {
276:     matrix_t im;
277:     int r, c;
278:     M_set_identity_matrix(im);
279:     M_inverse_matrix(A, im);
280:     for (r = 0; r < MOrder; r++)
281:     {
282:         X[r] = 0;
283:         for (c = 0; c < MOrder; c++)
284:             X[r] += im[r][c] * C[c];
285:     }
286: }                               /* M_solve_system */
287:
288: /*---------------------------------------------------------------------*/
289:
290: void M_print_matrix(char *name, matrix_t m)
291: {
292:     char fs[512];
293:     int r, c, l, ln;
294:     l = strlen(name);
295:     ln = (MOrder - 1) / 2;
296:     sprintf(fs, "%%%ds", l);
297:     for (r = 0; r < MOrder; r++)
298:     {
299:         if (r != ln)
300:         {
301:             printf(fs, " ");
302:             printf(" |");
303:         }
304:         else
305:         {
306:             printf(fs, name);
307:             printf("=|");
308:         }
309:         for (c = 0; c < MOrder; c++)
310:             //printf(" %e", M_round_to_zero(m[r][c]));
311:             printf(" %g", M_round_to_zero(m[r][c]));
312:         printf(" |\n");
313:     }
314:     printf("\n");
315: }                               /* M_print_matrix */
316:
317: /*---------------------------------------------------------------------*/
318:
319: void M_print_sub_matrix(char *name, matrix_t m, int order)
320: {
321:     char fs[512];
322:     int r, c, l, ln;
323:     //int le;                       // coba ditambah le
324:     l = strlen(name);
325:     ln = (order - 1) / 2;
326:     sprintf(fs, "%%%ds", l);
327:     for (r = 0; r < order; r++)
328:     {
329:         if (r != ln)
330:         {
```

```
331:             printf(fs, " ");
332:             printf(" |");
333:         }
334:         else
335:         {
336:             printf(fs, name);
337:             printf("=|");
338:         }
339:         for (c = 0; c < order; c++)
340:             //printf(" %e", M_round_to_zero(m[r][c]));
341:             printf(" %g", M_round_to_zero(m[r][c]));
342:         printf(" |\n");
343:     }
344:     printf("\n");
345: }                               /* M_print_sub_matrix */
346:
347: /*---------------------------------------------------------------------*/
348:
349: void M_print_vector(char *name, vector_t v)
350: {
351:     int e;
352:     printf("%s=[", name);
353:     for (e = 0; e < MOrder; e++)
354:         //printf(" %e", M_round_to_zero(v[e]));
355:         printf(" %g", M_round_to_zero(v[e]));
356:     printf("]\n\n");
357: }                               /* M_print_vector */
358:
359: /*---------------------------------------------------------------------*/
360:
361: void M_print_sub_vector(char *name, vector_t v, int order)
362: {
363:     int e;
364:     //int le;                       // coba ditambah le
365:     printf("%s=[", name);
366:     for (e = 0; e < order; e++)
367:         //printf(" %e", M_round_to_zero(v[e]));
368:         printf(" %g", M_round_to_zero(v[e]));
369:     printf("]\n\n");
370: }                               /* M_print_sub_vector */
371:
372: /*---------------------------------------------------------------------*/
373:
374: /* run this program using the console pauser or add your own getch, system("pause") or input loop */
375:
376: int main(int argc, char *argv[]) {
377:
378:     int i, j;
379:
380:     char s[25];
381:     matrix_t a, b, c;
382:     matrix_t test = {
383:         {1.0, 2.0, 3.0},
384:         {4.0, 5.0, 6.0},
385:         {7.0, 8.0, 9.0}
386:     };
387:     matrix_t test_2 = {
388:         {1.0, 2.0, 3.0},
389:         {1.0, 0.0, 1.0},
390:         {1.0, 1.0, 1.0}
391:     };
392:
393:     matrix_t A = {
394:         {3.0, 2.0, 1.0},
395:         {2.0, 3.0, 1.0},
396:         {1.0, 2.0, 3.0}
397:     };
398:
399:     vector_t C = { 39.0, 34.0, 26.0 };
400:     vector_t X;
401:
402:     M_set_identity_matrix(a);
403:     M_set_zero_matrix(b);
404:     M_copy_matrix(a, b);
405:     M_print_matrix("b", b);
406:     M_set_rc_reversal_matrix(a);
407:     M_print_matrix("a", a);
```

```
408:
409:     M_print_matrix("original matrix", test);
410:     M_set_rc_reversal_matrix(b);
411:     M_matrix_multiply(test, b, c);
412:     M_print_matrix("column reversed matrix", c);
413:     M_matrix_multiply(b, test, c);
414:     M_print_matrix("row reversed matrix", c);
415:     M_transpose_matrix(test, c);
416:     M_print_matrix("transposed matrix", c);
417:     M_print_sub_matrix("sub matrix", test, 2);
418:     M_print_matrix("original matrix", test);
419:     for (i = 0; i < MOrder; i++)
420:         for (j = 0; j < MOrder; j++)
421:         {
422:             M_minor_matrix(test, MOrder, i, j, c);
423:             sprintf(s, "minor[%d][%d] matrix", i, j);
424:             M_print_sub_matrix(s, c, MOrder - 1);
425:         }
426:
427:     M_set_identity_matrix(a);
428:     M_print_matrix("a", a);
429:     printf("|a|=%g\n", M_determinant(a, MOrder));
430:     M_print_matrix("test_2", test_2);
431:     printf("|test_2|=%g\n\n", M_determinant(test_2, MOrder));
432:     M_set_identity_matrix(a);
433:     M_print_matrix("a", a);
434:     M_inverse_matrix(a, b);
435:     M_print_matrix("a^(-1)", b);
436:     M_inverse_matrix(b, c);
437:     M_print_matrix("(a^(-1))^(-1)", c);
438:     M_print_matrix("A", A);
439:     M_print_vector("C", C);
440:     M_solve_system(A, C, X);
441:     M_print_vector("A^(-1)*C", X);
442:
443:     return 0;
444: }
```

3.2. Setelah selesai langkah 3.1 kemudian compile. Hasil kompilasi tampak pada Gambar 1.



Gambar 1. Tampilan hasil program

4. HASIL DAN PEMBAHASAN

Pada bagian ini setiap mahasiswa diharapkan mampu melakukan analisa berdasar hasil pengamatan percobaan laboratorium dan teori, serta mampu memberikan kesimpulan.

5. TUGAS

Dengan buku yang telah anda dapat "MATLAB Programming - David Kuncicky" pada pertemuan sebelumnya lalukan

5.1. Percobaan mencari hasil perkalian matrik dengan fungsi matlab.

5.2. Percobaan mencari hasil invers matrik dengan fungsi matlab.

5.3. Percobaan mencari hasil transpose matrik dengan fungsi matlab.