```
 1  module  tdma
 2      contains
 3      !onetdma code for solving one line using TDMA
 4      subroutine onetdma(n,a,b,c,d,line)
 5          implicit none
 6          integer, intent (in) :: n
 7          double precision, dimension (1:n), intent (inout) ::  line
 8          double precision, dimension (1:n), intent (in) :: a,b,c,d
 9          double precision, dimension (1:n) :: p,q
10          integer :: i,flag,rec
11          !flag to identify fixed point handling
12          flag = 0
13          do i = 2,n-1
14              if((a(i)==1).and.(b(i)==0).and.(c(i)==0)) then
15                  flag = 1
16                  rec = i
17              end if
18          end do
19          if (flag==0) then
20          !initialize boundaries
21          p(1) = b(1)/a(1)
22          q(1) = d(1)/a(1)
23          !compute p and q
24          do i = 2,n
25              p(i) = b(i)/(a(i)-c(i)*p(i-1))
26              q(i) = ((c(i)*q(i-1))+d(i))/(a(i)-(c(i)*p(i-1)))
27          end do
28          !calculate line values
29          do i = n-1,2,-1
30              line(i) = p(i)*line(i+1)+q(i)
31          end do
32          end if
33          !if flag found, break a line into two and solve for individual lines
34          if (flag==1) then
35              call onetdmarec(rec,a(1:rec),b(1:rec),c(1:rec),d(1:rec),line(1:rec))
```

```
36              call onetdmarec(n-rec+1,a(rec:n),b(rec:n),c(rec:n),d(rec:n),line(rec:n))
37          end if
38
39      end subroutine onetdma
40      !first recursion of onetdma required to handle fixed points
41      subroutine onetdmarec(n,a,b,c,d,line)
42          implicit none
43          integer, intent (in) :: n
44          double precision, dimension (1:n), intent (inout) ::  line
45          double precision, dimension (1:n), intent (in) :: a,b,c,d
46          double precision, dimension (1:n) :: p,q
47          integer :: i
48          !initialize boundaries
49          p(1) = b(1)/a(1)
50          q(1) = d(1)/a(1)
51          !compute p and q
52          do i = 2,n
53              p(i) = b(i)/(a(i)-c(i)*p(i-1))
54              q(i) = ((c(i)*q(i-1))+d(i))/(a(i)-(c(i)*p(i-1)))
55          end do
56          !calculate line values
57          do i = n-1,2,-1
58              line(i) = p(i)*line(i+1)+q(i)
59          end do
60      end subroutine onetdmarec
61      !code for onedtdma, constructs coefficients itself and solves for the line
62      subroutine onedtdma(n,mat,gmae,gmaw,Del_x,delxe,delxw,sc,sp)
63          implicit none
64          !delaratios
65          integer, intent (in) :: n
66          double precision, dimension (1:n), intent (in) :: gmae,gmaw,Del_x,delxe,delxw,sc,sp
67          double precision, dimension (1:n), intent (inout) :: mat
68          double precision, dimension (1:n) :: mat_dash
69          double precision, dimension (1:n) :: lane, a,b,c,d
70          double precision :: ae, aw, ap, delv,b_, eps
```

```
 71          integer :: i,c_
 72          c_ = 0
 73          do while (.true.)
 74              c_ = c_ + 1
 75              print*,c_
 76              eps = 0.0
 77              !backup matrix
 78              mat_dash = mat
 79              lane = mat_dash
 80              !setting boundaries
 81              a(1) = 1.0
 82              b(1) = 0.0
 83              c(1) = 0.0
 84              d(1) = mat_dash(1)
 85              a(n) = 1.0
 86              b(n) = 0.0
 87              c(n) = 0.0
 88              d(n) = mat_dash(n)
 89              !computing coefficients for inner points
 90              do i = 2,n-1
 91                  ae = gmae(i)/delxe(i)
 92                  aw = gmaw(i)/delxw(i)
 93                  delv = Del_x(i)
 94                  b_ = sc(i)*delv
 95                  ap = ae+aw-(sp(i)*delv)
 96                  a(i) = ap
 97                  b(i) = ae
 98                  c(i) = aw
 99                  d(i) = b_
100              end do
101              call onetdma(n,a,b,c,d,lane)
102              mat = lane
103              !error using eucleadian distance
104              eps = sqrt(sum((mat-mat_dash)**2))
105              if (eps<1e-10) exit
```

```
106         end do
107     end subroutine onedtdma
108     !code for extending onedtdma to two dimensions, builds all the coefficients itself, Dirichlet BC
109     subroutine twodtdma(n,m,mat,gmae,gmaw,gman,gmas,Del_x,Del_y,delxe,delxw,delyn,delys,sc,sp,sweep)
110         implicit none
111         !declarations
112         integer, intent (in) :: n,m
113         character(2), intent (in) :: sweep
114         double precision, dimension (1:n,1:m), intent (in) ::
              gmae,gmaw,gman,gmas,Del_x,Del_y,delxe,delxw,delyn,delys,sc,sp
115         double precision, dimension (1:n,1:m), intent (inout) :: mat
116         double precision, dimension (1:n,1:m) :: mat_dash
117         double precision, dimension (1:m) :: lanex, ax,bx,cx,dx
118         double precision, dimension (1:n) :: laney, ay,by,cy,dy
119         double precision :: ae, aw, an, as, ap, delv,b, eps
120         integer :: i,j,c
121         c = 0
122         !using if-statements for switching sweeps
123         !discretize to find a,b,c,d
124         if (sweep == '+x') then
125             do while (.true.)
126                 c = c + 1
127                 print*,c
128                 eps = 0.0
129                 !backup matrix
130                 mat_dash = mat
131                 do j = 2,m-1
132                     laney = mat_dash(:,j)
133                     !setting boundaries
134                     ay(1) = 1.0
135                     by(1) = 0.0
136                     cy(1) = 0.0
137                     dy(1) = mat_dash(1,j)
138                     ay(n) = 1.0
139                     by(n) = 0.0
```

```
140                     cy(n) = 0.0
141                     dy(n) = mat_dash(n,j)
142                     !computing coefficients for inner points
143                     do i = 2,n-1
144                         ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
145                         aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
146                         an = gman(i,j)*Del_x(i,j)/delyn(i,j)
147                         as = gmas(i,j)*Del_x(i,j)/delys(i,j)
148                         delv = Del_x(i,j)*Del_y(i,j)
149                         b = sc(i,j)*delv
150                         ap = ae+aw+an+as-(sp(i,j)*delv)
151                         ay(i) = ap
152                         by(i) = an
153                         cy(i) = as
154                         dy(i) = ae*mat_dash(i,j+1)+aw*mat_dash(i,j-1)+b
155                     end do
156                     call onetdma(n,ay,by,cy,dy,laney)
157                     mat(:,j) = laney
158                 end do
159                 !error using eucleadian distance
160                 eps = sqrt(sum((mat-mat_dash)**2))
161                 if (eps<1e-10) exit
162             end do
163         end if
164         if (sweep == '-x') then
165             do while (.true.)
166                 c = c + 1
167                 print*,c
168                 eps = 0.0
169                 !backup matrix
170                 mat_dash = mat
171                 do j = m-1,2,-1
172                     laney = mat_dash(:,j)
173                     !setting boundaries
174                     ay(1) = 1.0
```

```
175                     by(1) = 0.0
176                     cy(1) = 0.0
177                     dy(1) = mat_dash(1,j)
178                     ay(n) = 1.0
179                     by(n) = 0.0
180                     cy(n) = 0.0
181                     dy(n) = mat_dash(n,j)
182                     !computing coefficients for inner points
183                     do i = 2,n-1
184                         ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
185                         aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
186                         an = gman(i,j)*Del_x(i,j)/delyn(i,j)
187                         as = gmas(i,j)*Del_x(i,j)/delys(i,j)
188                         delv = Del_x(i,j)*Del_y(i,j)
189                         b = sc(i,j)*delv
190                         ap = ae+aw+an+as-(sp(i,j)*delv)
191                         ay(i) = ap
192                         by(i) = an
193                         cy(i) = as
194                         dy(i) = ae*mat_dash(i,j+1)+aw*mat_dash(i,j-1)+b
195                     end do
196                     call onetdma(n,ay,by,cy,dy,laney)
197                     mat(:,j) = laney
198                 end do
199                 !error using eucleadian distance
200                 eps = sqrt(sum((mat-mat_dash)**2))
201                 if (eps<1e-10) exit
202             end do
203         end if
204         if (sweep == '+y') then
205             do while (.true.)
206                 c = c + 1
207                 print*,c
208                 eps = 0.0
209                 !backup matrix
```

```
210                 mat_dash = mat
211                 do i = 2,n-1
212                     lanex = mat_dash(i,:)
213                     !setting boundaries
214                     ax(1) = 1.0
215                     bx(1) = 0.0
216                     cx(1) = 0.0
217                     dx(1) = mat_dash(i,1)
218                     ax(m) = 1.0
219                     bx(m) = 0.0
220                     cx(m) = 0.0
221                     dx(m) = mat_dash(i,m)
222                     !computing coefficients for inner points
223                     do j = 2,m-1
224                         ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
225                         aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
226                         an = gman(i,j)*Del_x(i,j)/delyn(i,j)
227                         as = gmas(i,j)*Del_x(i,j)/delys(i,j)
228                         delv = Del_x(i,j)*Del_y(i,j)
229                         b = sc(i,j)*delv
230                         ap = ae+aw+an+as-(sp(i,j)*delv)
231                         ax(j) = ap
232                         bx(j) = ae
233                         cx(j) = aw
234                         dx(j) = an*mat_dash(i+1,j)+as*mat_dash(i-1,j)+b
235                     end do
236                     call onetdma(m,ax,bx,cx,dx,lanex)
237                     mat(i,:) = lanex
238                 end do
239                 !error using eucleadian distance
240                 eps = sqrt(sum((mat-mat_dash)**2))
241                 if (eps<1e-10) exit
242             end do
243         end if
244         if (sweep == '-y') then
```

```
245             do while (.true.)
246                 c = c + 1
247                 print*,c
248                 eps = 0.0
249                 !backup matrix
250                 mat_dash = mat
251                 do i = n-1,2,-1
252                     lanex = mat_dash(i,:)
253                     !setting boundaries
254                     ax(1) = 1.0
255                     bx(1) = 0.0
256                     cx(1) = 0.0
257                     dx(1) = mat_dash(i,1)
258                     ax(m) = 1.0
259                     bx(m) = 0.0
260                     cx(m) = 0.0
261                     dx(m) = mat_dash(i,m)
262                     !computing coefficients for inner points
263                     do j = 2,m-1
264                         ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
265                         aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
266                         an = gman(i,j)*Del_x(i,j)/delyn(i,j)
267                         as = gmas(i,j)*Del_x(i,j)/delys(i,j)
268                         delv = Del_x(i,j)*Del_y(i,j)
269                         b = sc(i,j)*delv
270                         ap = ae+aw+an+as-(sp(i,j)*delv)
271                         ax(j) = ap
272                         bx(j) = ae
273                         cx(j) = aw
274                         dx(j) = an*mat_dash(i+1,j)+as*mat_dash(i-1,j)+b
275                     end do
276                     call onetdma(m,ax,bx,cx,dx,lanex)
277                     mat(i,:) = lanex
278                 end do
279                 !error using eucleadian distance
```

```
280                 eps = sqrt(sum((mat-mat_dash)**2))
281                 if (eps<1e-10) exit
282             end do
283         end if
284     end subroutine twodtdma
285     !subroutine to solve two dimensional TDMA as done before but with neuman boundary conditions
286     subroutine newtdma2d
          (n,m,mat,gmae,gmaw,gman,gmas,Del_x,Del_y,delxe,delxw,delyn,delys,sc,sp,p,q,phipq,quee,quew,quen,ques
          ,omega,sweep)
287         implicit none
288         !declarations
289         character(2), intent (in) :: sweep
290         integer, intent (in) :: n,m,p,q
291         double precision, dimension (1:n,1:m), intent (in) ::
              gmae,gmaw,gman,gmas,Del_x,Del_y,delxe,delxw,delyn,delys,sc,sp
292         double precision, dimension (1:n,1:m), intent (inout) :: mat
293         double precision, intent (in) :: phipq,quee,quew,quen,ques,omega
294         double precision, dimension (1:n,1:m) :: mat_dash,mat_star
295         double precision, dimension (1:m) :: lanex, ax,bx,cx,dx
296         double precision, dimension (1:n) :: laney, ay,by,cy,dy
297         double precision :: ae, aw, an, as, ap, delv,b, eps
298         integer :: i,j,c
299         c = 0
300         !using if-statements for switching sweeps
301         if(sweep == '+y') then
302         do while (.true.)
303             c = c + 1
304             print*,c
305             eps = 0.0
306             mat(p,q) = phipq
307             !backup matrix
308             mat_dash = mat
309             mat_star = mat
310             do i = 2,n-1
311                 !fixing coefficients for boundary points and fixed point
```

```
312                 ax(1) = 1.0
313                 bx(1) = 0.0
314                 cx(1) = 0.0
315                 dx(1) = mat_dash(i,1)
316                 ax(m) = 1.0
317                 bx(m) = 0.0
318                 cx(m) = 0.0
319                 dx(m) = mat_dash(i,m)
320                 ax(q) = 1.0
321                 bx(q) = 0.0
322                 cx(q) = 0.0
323                 dx(q) = mat_dash(p,q)
324                 lanex = mat_dash(i,:)
325                 !computing coefficients for inner points
326                 do j = 2,m-1
327                     if ((i==p).and.(j==q)) cycle
328                     ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
329                     aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
330                     an = gman(i,j)*Del_x(i,j)/delyn(i,j)
331                     as = gmas(i,j)*Del_x(i,j)/delys(i,j)
332                     delv = Del_x(i,j)*Del_y(i,j)
333                     b = sc(i,j)*delv
334                     ap = ae+aw+an+as-(sp(i,j)*delv)
335                     ax(j) = ap
336                     bx(j) = ae
337                     cx(j) = aw
338                     dx(j) = an*mat_dash(i+1,j)+as*mat_dash(i-1,j)+b
339                 end do
340                 call onetdma(m,ax,bx,cx,dx,lanex)
341                 mat_star(i,:) = lanex
342             end do
343             !boundaries
344             do i = 1,m
345                 mat_star(1,i) = (ques+mat(2,i)*(gmas(2,i)/delys(2,i)))/(gmas(2,i)/delys(2,i))
346                 mat_star(n,i) = (quen+mat(n-1,i)*(gman(n-1,i)/delyn(n-1,i)))/(gman(n-1,i)/delyn(n-1,i))
```

```
347             end do
348             do i = 1,n
349                 mat_star(i,1) = (quew+mat(i,2)*(gmaw(i,2)/delxw(i,2)))/(gmaw(i,2)/delxw(i,2))
350                 mat_star(i,m) = (quee+mat(i,m-1)*(gmae(i,m-1)/delxe(i,m-1)))/(gmae(i,m-1)/delxe(i,m-1))
351             end do
352             !corner points
353             mat_star(1,1) = (mat_star(1,2)+mat_star(2,1))*0.5
354             mat_star(1,n) = (mat_star(1,n-1)+mat_star(2,n))*0.5
355             mat_star(n,1) = (mat_star(1,2)+mat_star(n-1,1))*0.5
356             mat_star(n,n) = (mat_star(n-1,n)+mat_star(n,n-1))*0.5
357             !relaxation
358             mat = omega*mat_star+(1-omega)*mat_dash
359             !error using eucleadian distance
360             eps = sqrt(sum((mat-mat_dash)**2))
361             if (eps<1e-10) exit
362         end do
363         end if
364 
365         if(sweep == '-y') then
366         do while (.true.)
367             c = c + 1
368             print*,c
369             eps = 0.0
370             mat(p,q) = phipq
371             !backup matrix
372             mat_dash = mat
373             mat_star = mat
374             do i = n-1,2,-1
375                 !fixing coefficients for boundary points and fixed point
376                 ax(1) = 1.0
377                 bx(1) = 0.0
378                 cx(1) = 0.0
379                 dx(1) = mat_dash(i,1)
380                 ax(m) = 1.0
381                 bx(m) = 0.0
```

```
382                 cx(m) = 0.0
383                 dx(m) = mat_dash(i,m)
384                 ax(q) = 1.0
385                 bx(q) = 0.0
386                 cx(q) = 0.0
387                 dx(q) = mat_dash(p,q)
388                 lanex = mat_dash(i,:)
389                 !computing coefficients for inner points
390                 do j = 2,m-1
391                     if ((i==p).and.(j==q)) cycle
392                     ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
393                     aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
394                     an = gman(i,j)*Del_x(i,j)/delyn(i,j)
395                     as = gmas(i,j)*Del_x(i,j)/delys(i,j)
396                     delv = Del_x(i,j)*Del_y(i,j)
397                     b = sc(i,j)*delv
398                     ap = ae+aw+an+as-(sp(i,j)*delv)
399                     ax(j) = ap
400                     bx(j) = ae
401                     cx(j) = aw
402                     dx(j) = an*mat_dash(i+1,j)+as*mat_dash(i-1,j)+b
403                 end do
404                 call onetdma(m,ax,bx,cx,dx,lanex)
405                 mat_star(i,:) = lanex
406             end do
407             !boundaries
408             do i = 1,m
409                 mat_star(1,i) = (ques+mat(2,i)*(gmas(2,i)/delys(2,i)))/(gmas(2,i)/delys(2,i))
410                 mat_star(n,i) = (quen+mat(n-1,i)*(gman(n-1,i)/delyn(n-1,i)))/(gman(n-1,i)/delyn(n-1,i))
411             end do
412             do i = 1,n
413                 mat_star(i,1) = (quew+mat(i,2)*(gmaw(i,2)/delxw(i,2)))/(gmaw(i,2)/delxw(i,2))
414                 mat_star(i,m) = (quee+mat(i,m-1)*(gmae(i,m-1)/delxe(i,m-1)))/(gmae(i,m-1)/delxe(i,m-1))
415             end do
416             !corner points
```

```
417             mat_star(1,1) = (mat_star(1,2)+mat_star(2,1))*0.5
418             mat_star(1,n) = (mat_star(1,n-1)+mat_star(2,n))*0.5
419             mat_star(n,1) = (mat_star(1,2)+mat_star(n-1,1))*0.5
420             mat_star(n,n) = (mat_star(n-1,n)+mat_star(n,n-1))*0.5
421             !relaxation
422             mat = omega*mat_star+(1-omega)*mat_dash
423             !error using eucleadian distance
424             eps = sqrt(sum((mat-mat_dash)**2))
425             if (eps<1e-10) exit
426         end do
427         end if
428 
429         if(sweep == '+x') then
430         do while (.true.)
431             c = c + 1
432             print*,c
433             eps = 0.0
434             mat(p,q) = phipq
435             !backup matrix
436             mat_dash = mat
437             mat_star = mat
438             do j = 2,m-1
439                 !fixing coefficients for boundary points and fixed point
440                 ay(1) = 1.0
441                 by(1) = 0.0
442                 cy(1) = 0.0
443                 dy(1) = mat_dash(1,j)
444                 ay(n) = 1.0
445                 by(n) = 0.0
446                 cy(n) = 0.0
447                 dy(n) = mat_dash(n,j)
448                 ay(p) = 1.0
449                 by(p) = 0.0
450                 cy(p) = 0.0
451                 dy(p) = mat_dash(p,q)
```

```
452                 laney = mat_dash(:,j)
453                 !computing coefficients for inner points
454                 do i = 2,n-1
455                     if ((i==p).and.(j==q)) cycle
456                     ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
457                     aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
458                     an = gman(i,j)*Del_x(i,j)/delyn(i,j)
459                     as = gmas(i,j)*Del_x(i,j)/delys(i,j)
460                     delv = Del_x(i,j)*Del_y(i,j)
461                     b = sc(i,j)*delv
462                     ap = ae+aw+an+as-(sp(i,j)*delv)
463                     ay(i) = ap
464                     by(i) = an
465                     cy(i) = as
466                     dy(i) = ae*mat_dash(i,j+1)+aw*mat_dash(i,j-1)+b
467                 end do
468                 call onetdma(n,ay,by,cy,dy,laney)
469                 mat_star(:,j) = laney
470             end do
471             !boundaries
472             do i = 1,m
473                 mat_star(1,i) = (ques+mat(2,i)*(gmas(2,i)/delys(2,i)))/(gmas(2,i)/delys(2,i))
474                 mat_star(n,i) = (quen+mat(n-1,i)*(gman(n-1,i)/delyn(n-1,i)))/(gman(n-1,i)/delyn(n-1,i))
475             end do
476             do i = 1,n
477                 mat_star(i,1) = (quew+mat(i,2)*(gmaw(i,2)/delxw(i,2)))/(gmaw(i,2)/delxw(i,2))
478                 mat_star(i,m) = (quee+mat(i,m-1)*(gmae(i,m-1)/delxe(i,m-1)))/(gmae(i,m-1)/delxe(i,m-1))
479             end do
480             !corner points
481             mat_star(1,1) = (mat_star(1,2)+mat_star(2,1))*0.5
482             mat_star(1,n) = (mat_star(1,n-1)+mat_star(2,n))*0.5
483             mat_star(n,1) = (mat_star(1,2)+mat_star(n-1,1))*0.5
484             mat_star(n,n) = (mat_star(n-1,n)+mat_star(n,n-1))*0.5
485             !relaxation
486             mat = omega*mat_star+(1-omega)*mat_dash
```

```
487             !error using eucleadian distance
488             eps = sqrt(sum((mat-mat_dash)**2))
489             if (eps<1e-10) exit
490         end do
491         end if
492 
493         if(sweep == '-x') then
494         do while (.true.)
495             c = c + 1
496             print*,c
497             eps = 0.0
498             mat(p,q) = phipq
499             !backup matrix
500             mat_dash = mat
501             mat_star = mat
502             do j = m-1,2,-1
503                 !fixing coefficients for boundary points and fixed point
504                 ay(1) = 1.0
505                 by(1) = 0.0
506                 cy(1) = 0.0
507                 dy(1) = mat_dash(1,j)
508                 ay(n) = 1.0
509                 by(n) = 0.0
510                 cy(n) = 0.0
511                 dy(n) = mat_dash(n,j)
512                 ay(p) = 1.0
513                 by(p) = 0.0
514                 cy(p) = 0.0
515                 dy(p) = mat_dash(p,q)
516                 laney = mat_dash(:,j)
517                 !computing coefficients for inner points
518                 do i = 2,n-1
519                     if ((i==p).and.(j==q)) cycle
520                     ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
521                     aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
```

```
522                     an = gman(i,j)*Del_x(i,j)/delyn(i,j)
523                     as = gmas(i,j)*Del_x(i,j)/delys(i,j)
524                     delv = Del_x(i,j)*Del_y(i,j)
525                     b = sc(i,j)*delv
526                     ap = ae+aw+an+as-(sp(i,j)*delv)
527                     ay(i) = ap
528                     by(i) = an
529                     cy(i) = as
530                     dy(i) = ae*mat_dash(i,j+1)+aw*mat_dash(i,j-1)+b
531                 end do
532                 call onetdma(n,ay,by,cy,dy,laney)
533                 mat_star(:,j) = laney
534             end do
535             do i = 1,m
536                 mat_star(1,i) = (ques+mat(2,i)*(gmas(2,i)/delys(2,i)))/(gmas(2,i)/delys(2,i))
537                 mat_star(n,i) = (quen+mat(n-1,i)*(gman(n-1,i)/delyn(n-1,i)))/(gman(n-1,i)/delyn(n-1,i))
538             end do
539             do i = 1,n
540                 mat_star(i,1) = (quew+mat(i,2)*(gmaw(i,2)/delxw(i,2)))/(gmaw(i,2)/delxw(i,2))
541                 mat_star(i,m) = (quee+mat(i,m-1)*(gmae(i,m-1)/delxe(i,m-1)))/(gmae(i,m-1)/delxe(i,m-1))
542             end do
543             !corner points
544             mat_star(1,1) = (mat_star(1,2)+mat_star(2,1))*0.5
545             mat_star(1,n) = (mat_star(1,n-1)+mat_star(2,n))*0.5
546             mat_star(n,1) = (mat_star(1,2)+mat_star(n-1,1))*0.5
547             mat_star(n,n) = (mat_star(n-1,n)+mat_star(n,n-1))*0.5
548             !relaxation
549             mat = omega*mat_star+(1-omega)*mat_dash
550             !error using eucleadian distance
551             eps = sqrt(sum((mat-mat_dash)**2))
552             if (eps<1e-10) exit
553         end do
554         end if
555     end subroutine newtdma2d
556     !code for solving a line using matrix inversion, uses LU inversion method to inverse the matrix
```

```
557     subroutine inmatsolve(n,a,b,c,d,line)
558         double precision, dimension(n,n) :: matA,matAinv
559         integer, intent(in) :: n
560         double precision, dimension(n), intent(in) :: a,b,c,d
561         double precision, dimension(n), intent(inout) :: line
562         integer :: i,flag,rec
563         flag = 0
564         do i = 2,n-1
565             if((a(i)==1).and.(b(i)==0).and.(c(i)==0)) then
566                 flag = 1
567                 rec = i
568             end if
569         end do
570         if(flag==0) then
571         !build a matrix
572         matA = 0.0
573         matA(1,1) = a(1)
574         do i = 2,n-1
575             matA(i,i-1) = -1.0*c(i)
576             matA(i,i) = a(i)
577             matA(i,i+1) = -1.0*b(i)
578         end do
579         matA(n,n) = a(n)
580         call inverse(matA,matAinv,n)
581         line = matmul(matAinv,d)
582         end if
583         if(flag==1) then
584             call inmatsolverec(rec,a(1:rec),b(1:rec),c(1:rec),d(1:rec),line(1:rec))
585             call inmatsolverec(n-rec+1,a(rec:n),b(rec:n),c(rec:n),d(rec:n),line(rec:n))
586         end if
587     end subroutine inmatsolve
588     !recursive image of inmatsolve
589     subroutine inmatsolverec(n,a,b,c,d,line)
590         double precision, dimension(n,n) :: matA,matAinv
591         integer, intent(in) :: n
```

```
592         double precision, dimension(n), intent(in) :: a,b,c,d
593         double precision, dimension(n), intent(inout) :: line
594         integer :: i
595         !build a matrix
596         matA = 0.0
597         matA(1,1) = a(1)
598         do i = 2,N-1
599             matA(i,i-1) = -1.0*c(i)
600             matA(i,i) = a(i)
601             matA(i,i+1) = -1.0*b(i)
602         end do
603         matA(n,n) = a(n)
604         call inverse(matA,matAinv,n)
605         line = matmul(matAinv,d)
606     end subroutine inmatsolverec
607     !extending matrix inversion solver to two dimensions, neuman BC
608     subroutine newtdma2dinmat
          (n,m,mat,gmae,gmaw,gman,gmas,Del_x,Del_y,delxe,delxw,delyn,delys,sc,sp,p,q,phipq,quee,quew,quen,ques
          ,omega,sweep)
609         implicit none
610         !declarations
611         character(2), intent (in) :: sweep
612         integer, intent (in) :: n,m,p,q
613         double precision, dimension (1:n,1:m), intent (in) ::
              gmae,gmaw,gman,gmas,Del_x,Del_y,delxe,delxw,delyn,delys,sc,sp
614         double precision, dimension (1:n,1:m), intent (inout) :: mat
615         double precision, intent (in) :: phipq,quee,quew,quen,ques,omega
616         double precision, dimension (1:n,1:m) :: mat_dash,mat_star
617         double precision, dimension (1:m) :: lanex, ax,bx,cx,dx
618         double precision, dimension (1:n) :: laney, ay,by,cy,dy
619         double precision :: ae, aw, an, as, ap, delv,b, eps
620         integer :: i,j,c
621         c = 0
622         !using if-statements for switching sweeps
623         if(sweep == '+y') then
```

```
624         do while (.true.)
625             c = c + 1
626             print*,c
627             eps = 0.0
628             mat(p,q) = phipq
629             !backup matrix
630             mat_dash = mat
631             mat_star = mat
632             do i = 2,n-1
633                 !fixing coefficients for boundary points and a fixed point
634                 ax(1) = 1.0
635                 bx(1) = 0.0
636                 cx(1) = 0.0
637                 dx(1) = mat_dash(i,1)
638                 ax(m) = 1.0
639                 bx(m) = 0.0
640                 cx(m) = 0.0
641                 dx(m) = mat_dash(i,m)
642                 ax(q) = 1.0
643                 bx(q) = 0.0
644                 cx(q) = 0.0
645                 dx(q) = mat_dash(p,q)
646                 lanex = mat_dash(i,:)
647                 !computing coefficients for inner points
648                 do j = 2,m-1
649                     if ((i==p).and.(j==q)) cycle
650                     ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
651                     aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
652                     an = gman(i,j)*Del_x(i,j)/delyn(i,j)
653                     as = gmas(i,j)*Del_x(i,j)/delys(i,j)
654                     delv = Del_x(i,j)*Del_y(i,j)
655                     b = sc(i,j)*delv
656                     ap = ae+aw+an+as-(sp(i,j)*delv)
657                     ax(j) = ap
658                     bx(j) = ae
```

```
659                     cx(j) = aw
660                     dx(j) = an*mat_dash(i+1,j)+as*mat_dash(i-1,j)+b
661                 end do
662                 call inmatsolve(m,ax,bx,cx,dx,lanex)
663                 mat_star(i,:) = lanex
664             end do
665             !boundaries
666             do i = 1,m
667                 mat_star(1,i) = (ques+mat(2,i)*(gmas(2,i)/delys(2,i)))/(gmas(2,i)/delys(2,i))
668                 mat_star(n,i) = (quen+mat(n-1,i)*(gman(n-1,i)/delyn(n-1,i)))/(gman(n-1,i)/delyn(n-1,i))
669             end do
670             do i = 1,n
671                 mat_star(i,1) = (quew+mat(i,2)*(gmaw(i,2)/delxw(i,2)))/(gmaw(i,2)/delxw(i,2))
672                 mat_star(i,m) = (quee+mat(i,m-1)*(gmae(i,m-1)/delxe(i,m-1)))/(gmae(i,m-1)/delxe(i,m-1))
673             end do
674             !corner points
675             mat_star(1,1) = (mat_star(1,2)+mat_star(2,1))*0.5
676             mat_star(1,n) = (mat_star(1,n-1)+mat_star(2,n))*0.5
677             mat_star(n,1) = (mat_star(1,2)+mat_star(n-1,1))*0.5
678             mat_star(n,n) = (mat_star(n-1,n)+mat_star(n,n-1))*0.5
679             !relaxation
680             mat = omega*mat_star+(1-omega)*mat_dash
681             !error using eucleadian distance
682             eps = sqrt(sum((mat-mat_dash)**2))
683             if (eps<1e-10) exit
684         end do
685         end if
686         if(sweep == '-y') then
687         do while (.true.)
688             c = c + 1
689             print*,c
690             eps = 0.0
691             mat(p,q) = phipq
692             !backup matrix
693             mat_dash = mat
```

```
694             mat_star = mat
695             do i = n-1,2,-1
696                 !fixing coefficients for boundary points and a fixed point
697                 ax(1) = 1.0
698                 bx(1) = 0.0
699                 cx(1) = 0.0
700                 dx(1) = mat_dash(i,1)
701                 ax(m) = 1.0
702                 bx(m) = 0.0
703                 cx(m) = 0.0
704                 dx(m) = mat_dash(i,m)
705                 ax(q) = 1.0
706                 bx(q) = 0.0
707                 cx(q) = 0.0
708                 dx(q) = mat_dash(p,q)
709                 lanex = mat_dash(i,:)
710                 !computing coefficients for inner points
711                 do j = 2,m-1
712                     if ((i==p).and.(j==q)) cycle
713                     ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
714                     aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
715                     an = gman(i,j)*Del_x(i,j)/delyn(i,j)
716                     as = gmas(i,j)*Del_x(i,j)/delys(i,j)
717                     delv = Del_x(i,j)*Del_y(i,j)
718                     b = sc(i,j)*delv
719                     ap = ae+aw+an+as-(sp(i,j)*delv)
720                     ax(j) = ap
721                     bx(j) = ae
722                     cx(j) = aw
723                     dx(j) = an*mat_dash(i+1,j)+as*mat_dash(i-1,j)+b
724                 end do
725                 call inmatsolve(m,ax,bx,cx,dx,lanex)
726                 mat_star(i,:) = lanex
727             end do
728             !boundaries
```

```
729             do i = 1,m
730                 mat_star(1,i) = (ques+mat(2,i)*(gmas(2,i)/delys(2,i)))/(gmas(2,i)/delys(2,i))
731                 mat_star(n,i) = (quen+mat(n-1,i)*(gman(n-1,i)/delyn(n-1,i)))/(gman(n-1,i)/delyn(n-1,i))
732             end do
733             do i = 1,n
734                 mat_star(i,1) = (quew+mat(i,2)*(gmaw(i,2)/delxw(i,2)))/(gmaw(i,2)/delxw(i,2))
735                 mat_star(i,m) = (quee+mat(i,m-1)*(gmae(i,m-1)/delxe(i,m-1)))/(gmae(i,m-1)/delxe(i,m-1))
736             end do
737             !corner points
738             mat_star(1,1) = (mat_star(1,2)+mat_star(2,1))*0.5
739             mat_star(1,n) = (mat_star(1,n-1)+mat_star(2,n))*0.5
740             mat_star(n,1) = (mat_star(1,2)+mat_star(n-1,1))*0.5
741             mat_star(n,n) = (mat_star(n-1,n)+mat_star(n,n-1))*0.5
742             !relaxation
743             mat = omega*mat_star+(1-omega)*mat_dash
744             !error using eucleadian distance
745             eps = sqrt(sum((mat-mat_dash)**2))
746             if (eps<1e-10) exit
747         end do
748         end if
749 
750         if(sweep == '+x') then
751         do while (.true.)
752             c = c + 1
753             print*,c
754             eps = 0.0
755             mat(p,q) = phipq
756             !backup matrix
757             mat_dash = mat
758             mat_star = mat
759             do j = 2,m-1
760                 !fixing coefficients for boundary points and a fixed point
761                 ay(1) = 1.0
762                 by(1) = 0.0
763                 cy(1) = 0.0
```

```
764                 dy(1) = mat_dash(1,j)
765                 ay(n) = 1.0
766                 by(n) = 0.0
767                 cy(n) = 0.0
768                 dy(n) = mat_dash(n,j)
769                 ay(p) = 1.0
770                 by(p) = 0.0
771                 cy(p) = 0.0
772                 dy(p) = mat_dash(p,q)
773                 laney = mat_dash(:,j)
774                 !computing coefficients for inner points
775                 do i = 2,n-1
776                     if ((i==p).and.(j==q)) cycle
777                     ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
778                     aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
779                     an = gman(i,j)*Del_x(i,j)/delyn(i,j)
780                     as = gmas(i,j)*Del_x(i,j)/delys(i,j)
781                     delv = Del_x(i,j)*Del_y(i,j)
782                     b = sc(i,j)*delv
783                     ap = ae+aw+an+as-(sp(i,j)*delv)
784                     ay(i) = ap
785                     by(i) = an
786                     cy(i) = as
787                     dy(i) = ae*mat_dash(i,j+1)+aw*mat_dash(i,j-1)+b
788                 end do
789                 call inmatsolve(n,ay,by,cy,dy,laney)
790                 mat_star(:,j) = laney
791             end do
792             !boundaries
793             do i = 1,m
794                 mat_star(1,i) = (ques+mat(2,i)*(gmas(2,i)/delys(2,i)))/(gmas(2,i)/delys(2,i))
795                 mat_star(n,i) = (quen+mat(n-1,i)*(gman(n-1,i)/delyn(n-1,i)))/(gman(n-1,i)/delyn(n-1,i))
796             end do
797             do i = 1,n
798                 mat_star(i,1) = (quew+mat(i,2)*(gmaw(i,2)/delxw(i,2)))/(gmaw(i,2)/delxw(i,2))
```

```
799                 mat_star(i,m) = (quee+mat(i,m-1)*(gmae(i,m-1)/delxe(i,m-1)))/(gmae(i,m-1)/delxe(i,m-1))
800             end do
801             !corner points
802             mat_star(1,1) = (mat_star(1,2)+mat_star(2,1))*0.5
803             mat_star(1,n) = (mat_star(1,n-1)+mat_star(2,n))*0.5
804             mat_star(n,1) = (mat_star(1,2)+mat_star(n-1,1))*0.5
805             mat_star(n,n) = (mat_star(n-1,n)+mat_star(n,n-1))*0.5
806             !relaxation
807             mat = omega*mat_star+(1-omega)*mat_dash
808             !error using eucleadian distance
809             eps = sqrt(sum((mat-mat_dash)**2))
810             if (eps<1e-10) exit
811         end do
812         end if
813 
814         if(sweep == '-x') then
815         do while (.true.)
816             c = c + 1
817             print*,c
818             eps = 0.0
819             mat(p,q) = phipq
820             !backup matrix
821             mat_dash = mat
822             mat_star = mat
823             do j = m-1,2,-1
824                 !fixing coefficients for boundary points and a fixed point
825                 ay(1) = 1.0
826                 by(1) = 0.0
827                 cy(1) = 0.0
828                 dy(1) = mat_dash(1,j)
829                 ay(n) = 1.0
830                 by(n) = 0.0
831                 cy(n) = 0.0
832                 dy(n) = mat_dash(n,j)
833                 ay(p) = 1.0
```

```
834                 by(p) = 0.0
835                 cy(p) = 0.0
836                 dy(p) = mat_dash(p,q)
837                 laney = mat_dash(:,j)
838                 !computing coefficients for inner points
839                 do i = 2,n-1
840                     if ((i==p).and.(j==q)) cycle
841                     ae = gmae(i,j)*Del_y(i,j)/delxe(i,j)
842                     aw = gmaw(i,j)*Del_y(i,j)/delxw(i,j)
843                     an = gman(i,j)*Del_x(i,j)/delyn(i,j)
844                     as = gmas(i,j)*Del_x(i,j)/delys(i,j)
845                     delv = Del_x(i,j)*Del_y(i,j)
846                     b = sc(i,j)*delv
847                     ap = ae+aw+an+as-(sp(i,j)*delv)
848                     ay(i) = ap
849                     by(i) = an
850                     cy(i) = as
851                     dy(i) = ae*mat_dash(i,j+1)+aw*mat_dash(i,j-1)+b
852                 end do
853                 if ((i==2).and.(c==1)) then
854                     !print*,ax
855                     !print*,bx
856                     !print*,cx
857                     !print*,dx
858                 end if
859                 call inmatsolve(n,ay,by,cy,dy,laney)
860                 mat_star(:,j) = laney
861             end do
862             do i = 1,m
863                 mat_star(1,i) = (ques+mat(2,i)*(gmas(2,i)/delys(2,i)))/(gmas(2,i)/delys(2,i))
864                 mat_star(n,i) = (quen+mat(n-1,i)*(gman(n-1,i)/delyn(n-1,i)))/(gman(n-1,i)/delyn(n-1,i))
865             end do
866             do i = 1,n
867                 mat_star(i,1) = (quew+mat(i,2)*(gmaw(i,2)/delxw(i,2)))/(gmaw(i,2)/delxw(i,2))
868                 mat_star(i,m) = (quee+mat(i,m-1)*(gmae(i,m-1)/delxe(i,m-1)))/(gmae(i,m-1)/delxe(i,m-1))
```

```
869             end do
870             !corner points
871             mat_star(1,1) = (mat_star(1,2)+mat_star(2,1))*0.5
872             mat_star(1,n) = (mat_star(1,n-1)+mat_star(2,n))*0.5
873             mat_star(n,1) = (mat_star(1,2)+mat_star(n-1,1))*0.5
874             mat_star(n,n) = (mat_star(n-1,n)+mat_star(n,n-1))*0.5
875             !relaxation
876             mat = omega*mat_star+(1-omega)*mat_dash
877             !error using eucleadian distance
878             eps = sqrt(sum((mat-mat_dash)**2))
879             if (eps<1e-10) exit
880         end do
881         end if
882     end subroutine newtdma2dinmat
883
884 end module
885
886
887
```