

```

1  ! =====
2  ! Stabilized Finite Element Analysis for
3  ! 1-Dimensional Advection Equation
4  !
5  ! * Stabilized F.E.M. ( SUPG )
6  ! * Crank-Nicolson Method ( Selectable )
7  ! * E-by-E Bi-CGSTABSolver
8  ! 2001.08.21. coded by Seizo Tanaka
9  !
10 ! ***** Transport +++++
11 ! * E-by-E GP-BiCG Solver
12 ! * Fortran90 Ver.
13 ! 2005.11.03. coded by Seizo Tanaka
14 ! =====
15 ! -- MODULES --
16 module datacm
17   integer :: node, nelm
18   double precision :: dt, alpha
19   double precision, allocatable :: nc(:,:)
20   double precision, allocatable :: xx(:), elength(:)
21   double precision, allocatable :: uu(:), phi(:), dphi(:), bphi(:)
22   integer :: iphbc
23   integer, allocatable :: nphbc(:)
24   double precision, allocatable :: fphbc(:)
25   integer :: isfem
26 end module datacm
27 ! -- MODULES --
28 !
29 ! -- MAIN PROGRAM --
30 !
31 program advection_ID
32   use datacm, only: dt
33   implicit none
34   integer :: istep, iend, iout
35   double precision :: time
36 !
37 !   call datain( iend, iout )
38 !
39 !   do istep = 1, iend
40 !     call solphi ( istep )
41 !     if ( istep / iout * iout == istep ) then
42 !       time = dt * dble(istep)
43 !       call output(time)
44 !     endif
45 !   enddo
46 end program advection_ID
47 ! -- MAIN PROGRAM --
48 !
49 ! -- SUBROUTINES --
50 !
51 !   subroutine datain ( iend, iout )
52 !
53 !
54 !
55 !   use datacm
56 !   implicit none
57 !   integer, intent(out) :: iend, iout
58 !   integer :: noiv
59 !   double precision :: dlength, ela, el, uin
60 !   integer :: i, j, n, m
61 !   double precision :: xdum
62 !
63 !   character(50) :: inpf, mesfile, bcfile, inifile, outfile
64 !
65 !   open(9, file = 'file.dat', status = 'unknown' )
66 !   read(9, '(a)') inpf
67 !   read(9, '(a)') mesfile
68 !   read(9, '(a)') bcfile
69 !
70 !   read(9, '(a)') outfile
71 !   close(9)
72 !
73 !   open(10, file = inpf, status = 'unknown')
74 !   open(11, file = mesfile, status = 'unknown')
75 !   open(12, file = bcfile, status = 'unknown')
76 !   open(13, file = inifile, status = 'unknown')
77 !   open(50, file = outfile, status = 'unknown')
78 !
79 ! READ INPUT DATA
80 !
81 !   read(10,*) iend, iout
82 !   read(10,*) dt, alpha
83 !   read(10,*) uin
84 !   read(10,*) isfem
85 !   close(10)
86 !
87 ! READ MESH
88 !
89 !   read(11,*) node, nelm
90 !   allocate( xx(node) )
91 !   allocate( nc(2,nelm), elength(nelm) )
92 !
93 !   do n = 1, node
94 !     read(11,*) i, xx(i)
95 !   enddo
96 !
97 !   do m = 1, nelm
98 !     read(11,*) i, (nc(j,i), j=1,2)
99 !   enddo
100 !   close(11)
101 !
102 !   do m = 1, nelm
103 !     read(11,*) i, (nc(j,i), j=1,2)
104 !   enddo
105 !   close(11)
106 !
107 !   do m = 1, nelm
108 !     length(m) = xx(nc(2,m)) - xx(nc(1,m))
109 !   enddo
110 !
111 ! Dynamic Memory Allocation
112 !
113 !   allocate( uu(node), phi(node), dphi(node), bphi(node) )
114 !   allocate( ea(2,2,nelm), diag(node) )
115 !
116 !   uu(1:node) = uin
117 !
118 ! DEFINE BOUNDARY CONDITION
119 !
120 !   read(12,*) iphbc
121 !   allocate( nphbc(iphbc), fphbc(iphbc) )
122 !   do i = 1, iphbc
123 !     read(12,*) j, nphbc(i), fphbc(i)
124 !   enddo
125 !   close(12)
126 !
127 ! DEFINE INITIAL CONDITION
128 !
129 !   do n = 1, node
130 !     read(13,*) i, xdum, phi(n)
131 !   enddo
132 !   close(13)
133 !
134 ! Output Initial Condition
135 !
136 !   write(50, '(a)') '# Time=0.0'
137 !   write(50, 500) (n, xx(n), phi(n), n = 1, node)
138 !   format(17,2e15.6)
139 !
140 ! end subroutine datain
141 !
142 !
143 !
144 !
145 !
146 !
147 !
148 !
149 !
150 !
151 !
152 !
153 !
154 !
155 !
156 !
157 !
158 !
159 !
160 !
161 !
162 !
163 !
164 !
165 !
166 !
167 !
168 !
169 !
170 !
171 !
172 !
173 !
174 !
175 !
176 !
177 !
178 !
179 !
180 !
181 !
182 !
183 !
184 !
185 !
186 !
187 !
188 !
189 !
190 !
191 !
192 !
193 !
194 !
195 !
196 !
197 !
198 !
199 !
200 !
201 !
202 !
203 !
204 !
205 !
206 !
207 !
208 !
209 !
210 !
211 !
212 !
213 !
214 !
215 !
216 !
217 !
218 !
219 !
220 !
221 !
222 !
223 !
224 !
225 !
226 !
227 !
228 !
229 !
230 !
231 !
232 !
233 !
234 !
235 !
236 !
237 !
238 !
239 !
240 !
241 !
242 !
243 !
244 !
245 !
246 !
247 !
248 !
249 !
250 !
251 !
252 !
253 !
254 !
255 !
256 !
257 !
258 !
259 !
260 !
261 !
262 !
263 !
264 !
265 !
266 !
267 !
268 !
269 !
270 !
271 !
272 !
273 !
274 !
275 !
276 !
277 !
278 !
279 !
280 !
281 !
282 !
283 !
284 !
285 !
286 !
287 !
288 !
289 !
290 !
291 !
292 !
293 !
294 !
295 !
296 !
297 !
298 !
299 !
300 !
301 !
302 !
303 !
304 !
305 !
306 !
307 !
308 !
309 !
310 !
311 !
312 !
313 !
314 !
315 !
316 !
317 !
318 !
319 !
320 !
321 !
322 !
323 !
324 !
325 !
326 !
327 !
328 !
329 !
330 !
331 !
332 !
333 !
334 !
335 !
336 !
337 !
338 !
339 !
340 !
341 !
342 !
343 !
344 !
345 !
346 !
347 !
348 !
349 !
350 !
351 !
352 !
353 !
354 !
355 !
356 !
357 !
358 !
359 !
360 !
361 !
362 !
363 !
364 !
365 !
366 !
367 !
368 !
369 !
370 !
371 !
372 !
373 !
374 !
375 !
376 !
377 !
378 !
379 !
380 !
381 !
382 !
383 !
384 !
385 !
386 !
387 !
388 !
389 !
390 !
391 !
392 !
393 !
394 !
395 !
396 !
397 !
398 !
399 !
400 !
401 !
402 !
403 !
404 !
405 !
406 !
407 !
408 !
409 !
410 !
411 !
412 !
413 !
414 !
415 !
416 !
417 !
418 !
419 !
420 !
421 !
422 !
423 !
424 !
425 !
426 !
427 !
428 !
429 !
430 !
431 !
432 !
433 !
434 !
435 !
436 !
437 !
438 !
439 !
440 !
441 !
442 !
443 !
444 !
445 !
446 !
447 !
448 !
449 !
450 !
451 !
452 !
453 !
454 !
455 !
456 !
457 !
458 !
459 !
460 !
461 !
462 !
463 !
464 !
465 !
466 !
467 !
468 !
469 !
470 !
471 !
472 !
473 !
474 !
475 !
476 !
477 !
478 !
479 !
480 !
481 !
482 !
483 !
484 !
485 !
486 !
487 !
488 !
489 !
490 !
491 !
492 !
493 !
494 !
495 !
496 !
497 !
498 !
499 !
500 !
501 !
502 !
503 !
504 !
505 !
506 !
507 !
508 !
509 !
510 !
511 !
512 !
513 !
514 !
515 !
516 !
517 !
518 !
519 !
520 !
521 !
522 !
523 !
524 !
525 !
526 !
527 !
528 !
529 !
530 !
531 !
532 !
533 !
534 !
535 !
536 !
537 !
538 !
539 !
540 !
541 !
542 !
543 !
544 !
545 !
546 !
547 !
548 !
549 !
550 !
551 !
552 !
553 !
554 !
555 !
556 !
557 !
558 !
559 !
560 !
561 !
562 !
563 !
564 !
565 !
566 !
567 !
568 !
569 !
570 !
571 !
572 !
573 !
574 !
575 !
576 !
577 !
578 !
579 !
580 !
581 !
582 !
583 !
584 !
585 !
586 !
587 !
588 !
589 !
590 !
591 !
592 !
593 !
594 !
595 !
596 !
597 !
598 !
599 !
600 !
601 !
602 !
603 !
604 !
605 !
606 !
607 !
608 !
609 !
610 !
611 !
612 !
613 !
614 !
615 !
616 !
617 !
618 !
619 !
620 !
621 !
622 !
623 !
624 !
625 !
626 !
627 !
628 !
629 !
630 !
631 !
632 !
633 !
634 !
635 !
636 !
637 !
638 !
639 !
640 !
641 !
642 !
643 !
644 !
645 !
646 !
647 !
648 !
649 !
650 !
651 !
652 !
653 !
654 !
655 !
656 !
657 !
658 !
659 !
660 !
661 !
662 !
663 !
664 !
665 !
666 !
667 !
668 !
669 !
670 !
671 !
672 !
673 !
674 !
675 !
676 !
677 !
678 !
679 !
680 !
681 !
682 !
683 !
684 !
685 !
686 !
687 !
688 !
689 !
690 !
691 !
692 !
693 !
694 !
695 !
696 !
697 !
698 !
699 !
700 !
701 !
702 !
703 !
704 !
705 !
706 !
707 !
708 !
709 !
710 !
711 !
712 !
713 !
714 !
715 !
716 !
717 !
718 !
719 !
720 !
721 !
722 !
723 !
724 !
725 !
726 !
727 !
728 !
729 !
730 !
731 !
732 !
733 !
734 !
735 !
736 !
737 !
738 !
739 !
740 !
741 !
742 !
743 !
744 !
745 !
746 !
747 !
748 !
749 !
750 !
751 !
752 !
753 !
754 !
755 !
756 !
757 !
758 !
759 !
760 !
761 !
762 !
763 !
764 !
765 !
766 !
767 !
768 !
769 !
770 !
771 !
772 !
773 !
774 !
775 !
776 !
777 !
778 !
779 !
780 !
781 !
782 !
783 !
784 !
785 !
786 !
787 !
788 !
789 !
790 !
791 !
792 !
793 !
794 !
795 !
796 !
797 !
798 !
799 !
800 !
801 !
802 !
803 !
804 !
805 !
806 !
807 !
808 !
809 !
810 !
811 !
812 !
813 !
814 !
815 !
816 !
817 !
818 !
819 !
820 !
821 !
822 !
823 !
824 !
825 !
826 !
827 !
828 !
829 !
830 !
831 !
832 !
833 !
834 !
835 !
836 !
837 !
838 !
839 !
840 !
841 !
842 !
843 !
844 !
845 !
846 !
847 !
848 !
849 !
850 !
851 !
852 !
853 !
854 !
855 !
856 !
857 !
858 !
859 !
860 !
861 !
862 !
863 !
864 !
865 !
866 !
867 !
868 !
869 !
870 !
871 !
872 !
873 !
874 !
875 !
876 !
877 !
878 !
879 !
880 !
881 !
882 !
883 !
884 !
885 !
886 !
887 !
888 !
889 !
890 !
891 !
892 !
893 !
894 !
895 !
896 !
897 !
898 !
899 !
900 !
901 !
902 !
903 !
904 !
905 !
906 !
907 !
908 !
909 !
910 !
911 !
912 !
913 !
914 !
915 !
916 !
917 !
918 !
919 !
920 !
921 !
922 !
923 !
924 !
925 !
926 !
927 !
928 !
929 !
930 !
931 !
932 !
933 !
934 !
935 !
936 !
937 !
938 !
939 !
940 !
941 !
942 !
943 !
944 !
945 !
946 !
947 !
948 !
949 !
950 !
951 !
952 !
953 !
954 !
955 !
956 !
957 !
958 !
959 !
960 !
961 !
962 !
963 !
964 !
965 !
966 !
967 !
968 !
969 !
970 !
971 !
972 !
973 !
974 !
975 !
976 !
977 !
978 !
979 !
980 !
981 !
982 !
983 !
984 !
985 !
986 !
987 !
988 !
989 !
990 !
991 !
992 !
993 !
994 !
995 !
996 !
997 !
998 !
999 !
1000 !

```

```

135 !
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201

```

```

call mkmtvc( node, nelm, nc, elength, uu, dt, alpha, &
phi, dphi, ea, diag, bphi, iphbc, nphbc, fphbc, isfem )
call gpbcg &
( k, node, nelm, nc, dphi, ea, diag, bphi, iphbc, nphbc )
phi = phi + dphi
write(6,*) ' ===== Now Computing ===== '
write(6,*) ' STEP =', istep
write(6,*) ' TIME =', dt * dble(istep)
write(6,*) 'GPBCG(k):', k
end subroutine solphi
-----
subroutine mkmtvc &
( node, nelm, nc, el, uu, dt, alpha, &
phi, dphi, ea, d, b, ibc, nbc, fbc, isfem )
implicit none
integer, intent(in) :: node, nelm, nc(2,nelm), ibc, nbc(IBC), isfem
double precision, intent(in) :: el(nelm), fbc(IBC), uu(node), dt, alpha
double precision, intent(inout) :: phi(node)
double precision, intent(inout) :: ea(2,2,nelm), d(node)
double precision, intent(out) :: b(node), dphi(node)
integer :: m, i, n1, n2
double precision :: tau, u1, u2, dl, dl06, dl03
double precision :: emu1, emu2
double precision :: etg11, etg12, etg21, etg22
double precision :: ets11, ets12, ets21, ets22
double precision :: eag11, eag12, eag21, eag22, eag1, eag2
double precision :: eas11, eas12, eas21, eas22, eas1, eas2, aau
do i = 1, ibc
phi(nbc(i)) = fbc(i)
enddo
b = 0.0d0
d = 0.0d0
do m = 1, nelm
n1 = nc(1,m)
n2 = nc(2,m)
u1 = uu(n1)
u2 = uu(n2)
dl = el(m)
tau = dl / ( u1 + u2 ) * dble(isfem)
dl06 = dl / 6.0d0
dl03 = dl06 * 2.0d0
emu1 = ( 2.0d0 * u1 + u2 ) * dl06
emu2 = ( u1 + 2.0d0 * u2 ) * dl06
etg11 = dl03 / dt
etg12 = dl06 / dt
etg21 = dl06 / dt
etg22 = dl03 / dt
ets11 = - emu1 / ( dt*d1 )
ets12 = - emu2 / ( dt*d1 )
ets21 = emu1 / ( dt*d1 )
ets22 = emu2 / ( dt*d1 )
eag11 = - emu1 / dl
eag12 = emu1 / dl

```

```

202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268

```

```

eag21 = - emu2 / dl
eag22 = emu2 / dl
aau = u1 * emu1 + u2 * emu2
eas11 = aau * ( - 1.0d0 / dl ) * ( - 1.0d0 / dl )
eas12 = aau * ( - 1.0d0 / dl ) * ( - 1.0d0 / dl )
eas21 = aau * ( - 1.0d0 / dl ) * ( - 1.0d0 / dl )
eas22 = aau * ( - 1.0d0 / dl ) * ( - 1.0d0 / dl )
ea(1,1,m) = ( etg11 + ets11*tau ) + ( eag11 + eas11*tau ) * alpha
ea(1,2,m) = ( etg12 + ets12*tau ) + ( eag12 + eas12*tau ) * alpha
ea(2,1,m) = ( etg21 + ets21*tau ) + ( eag21 + eas21*tau ) * alpha
ea(2,2,m) = ( etg22 + ets22*tau ) + ( eag22 + eas22*tau ) * alpha
d(n1) = d(n1) + ea(1,1,m)
d(n2) = d(n2) + ea(2,2,m)
eag1 = eag11 * phi(n1) + eag12 * phi(n2)
eag2 = eag21 * phi(n1) + eag22 * phi(n2)
eas1 = eas11 * phi(n1) + eas12 * phi(n2)
eas2 = eas21 * phi(n1) + eas22 * phi(n2)
b(n1) = b(n1) - ( eag1 + eas1*tau )
b(n2) = b(n2) - ( eag2 + eas2*tau )
enddo
d = 1.0d0 / dsqrt(dabs(d))
do m = 1, nelm
ea(1,1,m) = ea(1,1,m) * ( d(nc(1,m)) * d(nc(1,m)))
ea(1,2,m) = ea(1,2,m) * ( d(nc(1,m)) * d(nc(2,m)))
ea(2,1,m) = ea(2,1,m) * ( d(nc(2,m)) * d(nc(1,m)))
ea(2,2,m) = ea(2,2,m) * ( d(nc(2,m)) * d(nc(2,m)))
enddo
dphi = 0.0d0
end subroutine mkmtvc
-----
subroutine gpbcg &
( k, node, nelm, nc, xx, ea, d, rr, ibc, nbc )
implicit none
integer, intent(in) :: node, nelm, nc(2,nelm), ibc, nbc(IBC)
integer, intent(out) :: k
double precision, intent(in) :: ea(2,2,nelm), d(node)
double precision, intent(inout) :: xx(node), rr(node)
double precision, allocatable :: pp(:), tt(:), rs(:), Ap(:), At(:)
double precision :: yy(:), uu(:), zz(:), ww(:)
double precision :: rtr, rsr, rsr0, Att, Aty, At2, ydy, ydt
integer :: kmax
double precision :: eps
data kmax / 1000 /
data eps / 1.0d-06 /
allocate( pp(node), tt(node), rs(node), Ap(node), At(node) )
allocate( yy(node), uu(node), zz(node), ww(node) )
rr = rr * d
! Define Initial Guess
call matvec ( node, nelm, nc, ibc, nbc, ea, xx, Ap )
rs = rr - Ap
rr = rr
pp = rr

```

