

```

1:
2: *****
3: ./code/odemod.f90
4:
5: MODULE odemod
6: *****
7: ***** Module file to hold various numerical methods for
8: ***** solving ordinary differential equations.
9: *****
10:
11: IMPLICIT NONE
12:
13:
14: CONTAINS
15:
16: *****
17: *****
18:
19: SUBROUTINE rk4(y,x,h)
20: !**** Does step of Runge-Kutta 4th Method ****
21:
22: ! *** Dummy declarations ***
23: REAL, INTENT(IN) :: h
24: REAL, INTENT(INOUT) :: y,x
25:
26: ! *** Local declarations ***
27: REAL :: k1,k2,k3,k4
28:
29: !**** Calculate k values ****
30: k1=h*ode(x,y)
31: k2=h*ode(x+h/2,y+k1/2)
32: k3=h*ode(x+h/2,y+k2/2)
33: k4=h*ode(x+h,y+k3)
34:
35: !**** Calculate y(x) ****
36: y=y+(k1+2*k2+2*k3+k4)/6
37:
38: x=x+h
39:
40: END SUBROUTINE rk4
41:
42: *****
43: *****
44:
45:
46: SUBROUTINE adam2(y,x,h)
47: !**** Does one step of adams 2 step Method ****
48:
49: ! *** Dummy declarations ***
50: REAL, INTENT(IN) :: h
51: REAL, INTENT(INOUT) :: y,x
52:
53: ! *** Local declarations ***
54: REAL :: y_in
55: INTEGER, SAVE :: count=1
56: REAL, SAVE :: oy1 !*** Store previous value
57:
58:
59: y_in=y !** Make a copy of incoming Y_(n)
60:
61: If (count < 2) THEN !** If not enough starting values
62: CALL rk4(y,x,h) !** Do Runge-Kutta 4th order
63: count=count+1 !** Increment count
64: ELSE !** Do the adams 2 step method
65: y=y+h/2*(3*ode(x,y)-ode(x-h,oy1))
66: x=x+h !** increment x
67:

```

```

68:
69: oy1=y_in !** Update Y_{n-1}
70:
71: END SUBROUTINE adam2
72:
73: *****
74:
75: SUBROUTINE adam3(y,x,h)
76: !**** Does one step of adams 3 step Method ****
77:
78: ! *** Dummy declarations ***
79: REAL, INTENT(IN) :: h
80: REAL, INTENT(INOUT) :: y,x
81:
82: ! *** Local declarations ***
83: REAL :: y_in
84: INTEGER, SAVE :: count=1
85: REAL, SAVE :: oy1,oy2 !*** Store previous values
86:
87: y_in=y !** Make a copy of incoming Y_(n)
88:
89: If (count < 3) THEN !** If not enough starting values
90: CALL rk4(y,x,h) !** Do Runge-Kutta 4th order
91: count=count+1 !** Increment count
92: ELSE !** Do the adams 3 step method
93: y=y+h/12*(23*ode(x,y)-16*ode(x-h,oy1)+5*ode(x-2*h,oy2))
94: x=x+h !** increment x
95: ENDIF
96:
97: oy2=oy1 !** Update Y_{n-2}
98: oy1=y_in !** Update Y_{n-1}
99:
100: END SUBROUTINE adam3
101: *****
102:
103: SUBROUTINE adam4(y,x,h)
104: !**** Does one step of adams 4 step Method ****
105:
106: ! *** Dummy declarations ***
107: REAL, INTENT(IN) :: h
108: REAL, INTENT(INOUT) :: y,x
109:
110: ! *** Local declarations ***
111: REAL :: y_in
112: INTEGER, SAVE :: count=1
113: REAL, SAVE :: oy1,oy2,oy3 !*** Store previous values
114:
115: y_in=y !** Make a copy of incoming Y_(n)
116:
117: If (count < 4) THEN !** If not enough starting values
118: CALL rk4(y,x,h) !** Do Runge-Kutta 4th order
119: count=count+1 !** Increment count
120: ELSE !** Do the adams 4 step method
121: y=y+h/24*(55*ode(x,y)-59*ode(x-h,oy1)+37*ode(x-2*h,oy2)-&
122: 9*ode(x-3*h,oy3))
123: x=x+h !** increment x
124: ENDIF
125:
126: oy3=oy2 !** Update Y_{n-3}
127: oy2=oy1 !** Update Y_{n-2}
128: oy1=y_in !** Update Y_{n-1}
129:
130: END SUBROUTINE adam4
131:
132: *****
133: *****
134:

```

```

135: SUBROUTINE modhamming(y,x,h)
136: !***** Does one step of Hamming's Method *****
137:
138: ! *** Dummy declarations ***
139: REAL, INTENT(IN) :: y,x
140: REAL, INTENT(OUT) :: y,x
141:
142: ! *** Local declarations ***
143: REAL :: mpy,pv,y_in
144: INTEGER, SAVE :: count=1
145: REAL, SAVE :: oy1,oy2,oy3,opy1
146:
147: y_in=y !** Make a copy of y_{n}
148:
149: If (count < 4) THEN !** If not enough starting values
150: CALL rk4(y,x,h) !** Do Runge-Kutta 4th order
151: count=count+1 !** Increment count
152: ELSE IF (count==4) THEN !** Do the Hamming's method
153: py=oy3+4.0*h*(2*ode(x,y)-ode(x-h,oy1)+2*ode(x-2*h,oy2))/3
154: y=(9*y-oy2)/8+3*h*(ode(x+h,py)+2*ode(x,y)-ode(x-h,oy1))/8
155: x=x+h !** increment x
156: count=count+1 !** Increment count
157: ELSE !** Do the Modified Hamming's method
158: py=oy3+4.0*h*(2*ode(x,y)-ode(x-h,oy1)+2*ode(x-2*h,oy2))/3
159: mpy=py+112.0/121*(y-opy1)
160: y=(9*y-oy2)/8+3*h*(ode(x+h,mpy)+2*ode(x,y)-ode(x-h,oy1))/8
161: x=x+h !** increment x
162: ENDF
163:
164: oy3=oy2 !** Update y_{n-3}
165: oy2=oy1 !** Update y_{n-2}
166: oy1=y_in !** Update y_{n-1}
167:
168: opy1=py !** Update py_{n-1}
169:
170: END SUBROUTINE modhamming
171:
172: !*****
173:
174: FUNCTION ode(x,y)
175: !** Function to return the value of the differential
176: !** equation for a given x and y
177:
178: !** Dummy declarations
179: REAL, INTENT(IN) :: x,y
180:
181: !** Function declaration ***
182: REAL :: ode
183:
184: ode=-y+x+2
185:
186: END FUNCTION ode
187:
188: !*****
189:
190: END MODULE odemod
191: *****
192: *****
193: *****
194: *****
195: ./code/main.f90
196:
197: PROGRAM main
198: !** Driver program for ode module
199: !**
200: !**
201:

```

```

202: USE odemod, ONLY : rk4,adam2,adam3,adam4,modhamming
203:
204: IMPLICIT NONE
205:
206: REAL, PARAMETER :: h=0.1
207:
208: REAL :: x,& !** Hold current "x" value of ode
209: y,& !** Hold current "y" value of ode
210: ytrue,& !** Hold current true solution
211: err=0 !** Hold the current error "ABS(y-ytrue)"
212:
213: INTEGER :: i
214:
215: x=0 / y=2 !** Initial Values
216:
217: ytrue=EXP(-x)+x+1
218: PRINT '(a4,a12,a12,a12)', "x", "yapprox", "y-True", "Error"
219: PRINT '(F4.2,F12.6,F12.6,F12.6)', x,y,ytrue,err
220:
221: DO i=1,10
222: !CALL rk4(y,x,h)
223: !CALL adam2(y,x,h)
224: !CALL adam3(y,x,h)
225: !CALL adam4(y,x,h)
226: CALL modhamming(y,x,h)
227: ytrue=EXP(-x)+x+1
228: err=ABS(y-ytrue)
229: PRINT '(F4.2,F12.6,F12.6,F12.6)', x,y,ytrue,err
230: ENDDO
231:
232: END PROGRAM main
233: *****
234:
235:
236: *****
237: ./code/results.f90
238:
239: =====
240: RK4
241: =====
242: x yapprox Y-True Error
243: 0.00 2.000000 2.000000 0.000000
244: 0.10 2.004838 2.004838 0.000000
245: 0.20 2.018731 2.018731 0.000000
246: 0.30 2.040818 2.040818 0.000000
247: 0.40 2.070320 2.070320 0.000000
248: 0.50 2.106531 2.106531 0.000000
249: 0.60 2.148812 2.148812 0.000000
250: 0.70 2.196585 2.196585 0.000000
251: 0.80 2.249329 2.249329 0.000000
252: 0.90 2.306570 2.306570 0.000000
253: 1.00 2.367880 2.367879 0.000000
254:
255: =====
256: ADAM2
257: =====
258:
259: x yapprox Y-True Error
260: 0.00 2.000000 2.000000 0.000000
261: 0.10 2.004838 2.004838 0.000000
262: 0.20 2.019112 2.018731 0.000381
263: 0.30 2.041487 2.040818 0.000669
264: 0.40 2.071219 2.070320 0.000899
265: 0.50 2.107611 2.106531 0.001080
266: 0.60 2.150030 2.148812 0.001219
267: 0.70 2.197906 2.196585 0.001321
268: 0.80 2.250722 2.249329 0.001393

```

solutions.txt Tue Dec 06 13:04:11 2011

269: 0.90 2.308009 2.306570 0.001439
270: 1.00 2.369344 2.367879 0.001464

271:
272:
273: ADAM3
274:
275:

276: x yapprox Y-True Error
277: 0.00 2.000000 2.000000 0.000000
278: 0.10 2.004838 2.004838 0.000000
279: 0.20 2.018731 2.018731 0.000000
280: 0.30 2.040786 2.040818 0.000032
281: 0.40 2.070264 2.070320 0.000056
282: 0.50 2.106455 2.106531 0.000076
283: 0.60 2.148720 2.148812 0.000092
284: 0.70 2.196482 2.196585 0.000103
285: 0.80 2.249217 2.249329 0.000113
286: 0.90 2.306451 2.306570 0.000119
287: 1.00 2.367757 2.367879 0.000123
288:

289:
290: ADAM4
291:
292:

293: x yapprox Y-True Error
294: 0.00 2.000000 2.000000 0.000000
295: 0.10 2.004838 2.004838 0.000000
296: 0.20 2.018731 2.018731 0.000000
297: 0.30 2.040818 2.040818 0.000000
298: 0.40 2.070323 2.070320 0.000003
299: 0.50 2.106536 2.106531 0.000005
300: 0.60 2.148818 2.148812 0.000007
301: 0.70 2.196594 2.196585 0.000008
302: 0.80 2.249338 2.249329 0.000009
303: 0.90 2.306580 2.306570 0.000010
304: 1.00 2.367890 2.367879 0.000010
305:

306:
307: MODHAMMING
308:
309:

310: x yapprox Y-True Error
311: 0.00 2.000000 2.000000 0.000000
312: 0.10 2.004838 2.004838 0.000000
313: 0.20 2.018731 2.018731 0.000000
314: 0.30 2.040818 2.040818 0.000000
315: 0.40 2.070320 2.070320 0.000000
316: 0.50 2.106531 2.106531 0.000000
317: 0.60 2.148811 2.148812 0.000000
318: 0.70 2.196585 2.196585 0.000000
319: 0.80 2.249328 2.249329 0.000001
320: 0.90 2.306569 2.306570 0.000000
321: 1.00 2.367879 2.367879 0.000001
322: *****
323: