Alexander Neuwirth, uebung 7, 23.6.18, Dr. Marcus Boeckmann

1)

a) 'mfunktion.f90' ist ein Modul, dass die Verwendung von mathematische Funktionen ermoeglicht. 'base_operations.inc' und 'dx_operations.inc' sind grundlegende Sammlungen von functions, die in das Modul 'mfunktion' eingebunden sind. 'rechnung.f90' ist das Main-Program. Für h=0.01 zu 0.001 nimmt der Fehler ab, da der zu h proportionale Naeherungsfehler mehr abnimmt als der antiproportionale Rundungsfehler steigt. Bei h=0.0001 tritt der inverse Fall auf. Folglich liegt das optimierte h vermutlich zwischen 0.001 und 0.0001.

b) Das optimierte h ist gegeben durch h_opt=sqrt(2 em/f''(x)). em ist die Maschinenungenauigkeit. Im Fall f=x^2 und em=10^-7 betraegt f''(2)=2. Es folgt ein h_opt von 10^-3.5. Im Program rechnung wird einmal das theoretisch berechnete h_opt_calc mit h_opt_bi verglichen, welches durch das Bisektions-Verfahren bestimmt wurde. Es zeigt sich, dass der h Wert des Bisektions-Verfahren einen kleineren Fehler liefert, dennoch ist die Groeßenordung der h-Werte gleich. Dies liegt daran, dass in der Vorlesung lediglich der Fehler fuer den Worst-Case optimiert wurde. Außerdem treten moeglicherweise zusaetzliche Rundungsfehler bei den Zwischenschritten im Modul 'mfunktion' auf. (Des Weiteren kann die Maschinengenauigkeit em abweichen)

2)

a) er=sqrt(N)*em+(b-a)^5*f''''/f/N^4 ist die Formel fuer den gesamten Fehler. Es kann nach e nach N abgeleitet und gleich Null gesetzt werden und nach N umgeformt werden, um das optimale N zu bestimmen. f''''/f betraegt for f(x)=e^x immer 1. Also ist der/dN = em/sqrt(N)/2 - (b-a)^5/N^5*4 = 0 => N^4.5 = (b-a)^5/em*8. a=0 b=0.1 => N=4.417=~5.

b) Der exakte Wert des Integrals ist e^0.1-1 = 0.105170918. Folglich ist die Abweichung 5e-9.

c) Die Formel fuer er ist in 2)a) gegeben. Mit N=5 erhaelt man er=2.4e-7.

d) Analog zu 2)a) mit er = sqrt(N)*em+(b-a)^3*f''/f/N^2. Es ergibt sich ein N=69.3144=~70

e) Einsetzten von N=70 in die Formel von 2)d) liefert er=1.041e-6

f) Das Simpson-Verfahren ist schneller, da weniger Stuetzpunkte verwendet werden und somit die Funktion weniger oft aufgerufen/ausgerechnet werden muss. Zusaetzlich zeigte sich bei den theoretischen Fehlerabschaetzungen, dass das Trapez-Verfahren ungenauer ist.

Die auf meinem PC erhaltenen Ausgaben sind in output.txt gespeichert.

Zum Modul 'mfunktion':

Es wird ein type(funktion) definiert, welcher function-pointer hat, welche auf functions verweisen, die ausgefuehrt werden sollen durch Operatoren +,-,*,/,**. (function meint im folgenden eine Fortran function und funktion eine type(funktion) objekt)
Es wird drei funktionen d,x,dx zur Verfuegung gestellt. Diese lassen sich durch Operatoren mit anderen funktionen oder Reals verknuepfen.
Eine Funktion kann mittels %get(x) and der Stelle x ausgewertet werden.
Eine Division durch dx fuehrt dazu, dass das Vorwaertsdifferenz-Verfahren auf den Zaehler angewendet wird.
Eine Multiplikation mit dx fuehrt dazu, dass das Simpson-Verfahren auf die funktion vor dx andgewendet wird. %get(x) fuehrt nun zu einem Error wenn man einen real uebergibt, stattdessen wird eine real array uebergeben, welches zuerst die obere und dann die untere Grenze des Integrals definiert.
Die Ausfuehrungen functionen der funktion dx sind in dx_operations.inc enthalten.
Es lassen sich beliebige functions als funktion definieren, insofern diese dem interface func genuegen. Dann kann mittels create_funktion(f) eine funktion erzeugt werden.

Fuer die jeweiligen function pointer wurden abstracte interface angelegt, die definieren wie eine solche function ausssieht.
Die Parameter der Verfahren N,eps,h sind als Variablen des Moduls gegeben.

Da ein funktion pointer array allocated wird sollte zum Schluss die Subroutine dealloc aufgerufen werden, welche die funktionen in dem pointer array recursiv deallocated.

Um auch funktionen zu haben die nicht von x abhaengen gibt es functions die dem interface func genuegen muessen, jedoch die x Variable nicht benutzten. Dies fuehrt zur Compiler Warnung unused-dummy-argument.