بسم الله الرحمن الرحيم

الحلول المختارة لطلاب الهندسة والعمارة

ملحق التفاضل العددي

تبصرة الطلاب ببرمجة الحساب

إعداد

سلمان محمد الناصر

الأستاذ المشارك بكلية الهندسة بجامعة أم القرى

الإصدار الثاني ١٤١٦هـ

تقديم

الحمد لله وحده والصلاة والسلام على من لا نبي بعده سيدنا محمد وعلى آله وصحبه

وبعد،

فإن استخدام الحساب، أيا كان نوعه، بكفاءة عالية من أبرز مميزات العصر الناجح نظرا لكونه ركنا وطيدا في بناء التقنية الهندسية.
ونظرا للحاجة إلى إجراء عمليات التحليل العددي وثيقة الصلة بالحساب نظرا للحاجة إلى إجراء عمليات حسابية بخط معين عدة مراحل قد تبلغ العشرة في بعضها، لذا فإن الحساب لا ينفكاك عنه مهم للوقت ويجب توفيره.

وقد وجدت في إلقاء تدريس لمادة التحليل العددي في فصول عديدة أن حساب (hp 28S) ذو جدوى متفردة في هذا المجال فهو يجمع تعليمات كثيرة وسرعة مقبولة جدا وذاكرة تخدم معظم أغراض المادة فأحببت أن أضع بين أيدي إخواني الطلاب بعضا من البرامج التي تصلح لاختزال التكرار في العمليات اللازمة لكل بعض مسائل المقرر المذكور شارحا في الهامش مؤدى كل خطوة أو مجموعة خطوات في البرنامج مع المعنى ونوعية المسائل التي يصلح لها ولابد لفهم ذلك من القراءة في كتيبات الحساب أولا للتعرف على منطقة في الحساب.
والله أسأل أن يكون هذا العمل مشجعا ومدربا للفكر المناظر في البرمجة ليس بحساب (hp 28S) وحسب بل في كل أنواع الحاسبات.

سلمان ك
١٤٢٦/١٠/٢٩

Conversion to Decimal

Given: First, the no. in list form with 'POINT' separating integer & fractional parts, in stack;
Second, the base in stack &
Third, digit values stored to used names of at most A to W.

Program	Action
<<	
'Z' STO	stores base in 'Z'
DUP POINT POS 'Y' STO	stores the position of 'POINT' in 'Y'
LIST→ 'X' STO	stores the no. of digits + 1 in 'X'
X Y - 1 + ROLL DROP	removes 'POINT'
X 1 - 1 →LIST →ARRY	converts the digits to array
Z Y ^ *	gives power offset
2 X FOR Y Z Y ^ INV NEXT	prepares power digits
X 1 - 1 →LIST →ARRY	converts them to array
DOT >>	gives the decimal eg.

Store this in "TO1ϕ" and press it to get answer.

Conversion from Decimal to any other Base

Given: First, the no. of fractional-part digits;
Second, the decimal no. &
Third, the new base.

Program	Action
`<< 'Z' STO DUP IP`	{ stores base in 'Z' and gets the integer part of the no.
`1 'Y' STO DO`	stores 1 in 'Y' and enters a loop
`DUP Z ÷ IP SWAP`	{ takes the integer part and
`OVER Z * - SWAP`	{ leaves lower residue & higher integer part
`Y 1 + 'Y' STO`	updates 'Y'
`DUP UNTIL Z < END`	{ checks higher integer part and stops if less than 'Z'
`0 Y 2 - FOR X Y X - ROLLD NEXT`	{ Reorders the integer new base digits
`Y →LIST {POINT} +`	{ keeps the new base integer digits in an ordered list
`SWAP FP ROT 'Y' STO`	{ gets the fractional part of the decimal no. and stores the required fractional digits in 'Y'
`1 Y FOR X Z * DUP IP SWAP FP NEXT`	{ takes the fractional part and leaves a higher integer part & lower residue
`IF .5 ≥ THEN 1+ END`	Rounds the remainder (2)
`Y →LIST + >>`	{ keeps the new base fractional digits in an ordered list → Gives the converted no. as a list.

Bisection Algorithm (pp. 29 of text)

Given: $[a, b]$ & $f(x)$

Prerequisite: store a in 'A' and b in 'B' & $f(x)$ in 'F'

Program	action
<<	
A B + 2 /	puts $\frac{a+b}{2}$ on stack
A 'X' STO F →NUM	puts $f(a)$ on stack
SWAP 'X' STO F →NUM	puts $f\left(\frac{a+b}{2}\right)$ on stack
*	puts $f(a) \cdot f\left(\frac{a+b}{2}\right)$ on stack
0 IF >	Compares product with 0
THEN X 'A' STO	stores $\frac{a+b}{2}$ in 'A' (new a_n) if applicable
ELSE X 'B' STO END	stores $\frac{a+b}{2}$ in 'B' (new b_n) if applicable
X >>	puts new root on stack

Note: You store the above program in 'BSCTN' or any other name you like within 5 letters. Then to solve the root of $f(x)$ in $[a,b]$ all you need to do is to press 'BSCTN' in USER menu, till the desired accuracy is obtained.

Newton - Raphson Algorithm (PP. 42 of text)

Given: P_0, $f(x)$ & $f'(x)$

Prerequists: store $f(x)$ in 'F' & $f'(x)$ in 'FD' and leave P_0 on the stack.

Program	Action
<< DUP 'X' STO	duplicates the stack and stores one value of the two in 'X'
F →NUM	finds the value of $f(x)$
FD →NUM	" " " " $f'(x)$
÷	finds f/f'
- >>	finds $x' = x - \frac{f}{f'}$

Note: You store the above program in 'NEWTN'. Then to solve the problem of finding root of $f(x)=0$, keep pressing 'NEWTN' in User Menu, till the desired accuracy is obtained.

4

V

Modified Newton-Raphson Algorithm.

Given: P_0, $f(x)$, $f'(x)$ & $f''(x)$.

Prerequists: store $f(x)$ in 'F', $f'(x)$ in 'FD', $f''(x)$ in 'FDD' and leave P_0 on stack.

Program	Action
<< DUP 'X' STO F→NUM	puts $x, f(x), f'(x)$ on stack
FD →NUM	
OVER OVER	leaves x, f, f', f, f' on stack
÷	leaves $x, f, f', \frac{f}{f'}$ on stack
FDD →NUM	puts f'' on stack
*	forms ff''/f'
−	gives $f' - (ff''/f')$
÷	gives $\frac{f}{f' - \frac{ff''}{f'}} = \frac{ff'}{f'^2 - ff''}$
− >>	gives $x' = x - \frac{ff'}{f'^2 - ff''}$

Note: Store this in 'MNWTN' and keep pressing it till the desired accuracy is obtained.

5

Secant Algorithm (pp. 47 of text)

Given: P_0, P_1 & $f(x)$

Prerequists: P_0 in 'A', $f(x)$ in 'F', $f(P_0)$ in B & P_1 on stack.

Program	Action
`<<`	
`'X' STO`	Stores P_1 in 'X'
`B F→NUM DUP 'B' STO ÷`	puts $f(P_1)$ in 'B' & $\frac{f(P_0)}{f(P_1)}$ on stack
`1 - INV X A - * X +`	forms $P_2 = P_1 - \frac{f(P_1)(P_1-P_0)}{f(P_1)-f(P_0)}$ $= P_1 + \left[\frac{1}{\frac{f(P_0)}{f(P_1)}-1}\right](P_1-P_0)$
`X 'A' STO ≫`	puts P_1 in 'A' for next iteration

Note: Store this in 'SCNT' and press to get the new approximation. Keep pressing till OK.

9

Aitken's Algorithm

Given: P_0, $g(x)$

Prerequists: Store P_0 in 'A' and $g(x)$ in 'G'

Program	Action
<< A 'X' STO G → NUM 'B' STO	puts in B $g(A)$
B 'X' STO G → NUM 'C' STO	" " C $g(B)$
A C * B B * −	forms $AC - B^2$
A C + B B + −	forms $A + C - 2B$
÷	forms $\hat{A}$
B 'A' STO >>	puts old B as new A for the next iteration

Note: Store this program in 'ATKN', and press to get new approximation, till satisfied.

Steffensen Algorithm (pp. 64 of text)

Given: P_0, $g(x)$

Prerequists: Store P_0 in 'A' and $g(x)$ in 'G'

Program	Action
<<	
A 'X' STO G →NUM 'B' STO	puts $g(A)$ in B
B 'X' STO G →NUM 'C' STO	puts $g(B)$ in C
A C * B B * −	forms $AC-B^2$
A C + B B + −	$A+C-2B$
÷	forms $\hat{P}=\frac{AC-B^2}{A+C-2B}$
'A' STO	puts $\hat{P}$ in A
A 'X' STO G →NUM 'B' STO	puts $g(\,)$ in B
B 'X' STO G →NUM 'C' STO	puts $g(B)$ in C
A B C >>	leaves A, B & C on stack

Note: Store program in 'STFN' and press to get three new approximations of steffensen, in order.

Muller Algorithm (PP. 72 of text)

Given: P_0, P_1, P_2 & $f(x)$

Prerequist: Store P_0 in 'A', P_1 in 'B', $f(x)$ in 'F' and leave P_2 on stack

Program	Action
<< 'C' STO	Stores P_2 in 'C'
A 'X' STO F → NUM	puts $f(P_0)$ on stack
B 'X' STO F → NUM	" $f(P_1)$ " "
C 'X' STO F → NUM	" $f(P_2)$ " "
{3} → ARRY	forms the vector $\langle f_0, f_1, f_2 \rangle$
A C -	forms $P_0 - P_2$ on stack
DUP DUP * SWAP 1	puts $(P_0-P_2)^2$, (P_0-P_2), 1 "
B C -	forms $P_1 - P_2$ on stack
DUP DUP * SWAP 1	puts $(P_1-P_2)^2$, (P_1-P_2), 1 "
0 0 1 {3 3} → ARRY	forms the matrix $\begin{bmatrix} (P_0-P_2)^2 & (P_0-P_2) & 1 \\ (P_1-P_2)^2 & (P_1-P_2) & 1 \\ 0 & 0 & 1 \end{bmatrix}$
÷	brings the vector of parabola coefficients $\langle a, b, c \rangle$ of $y = a(x-P_2)^2 + b(x-P_2) + c$
1 OVER ARRY→ DROP	puts $\langle a,b,c \rangle$ 1, a, b, c on stack
OVER ÷ ÷ ÷ 4 * - √	forms $\sqrt{1 - \frac{4ac}{b^2}}$ on stack
1 - SWAP	forms $\sqrt{1 - \frac{4ac}{b^2}} - 1, \langle a,b,c \rangle$ "
ARRY→ DROP2	forms $\sqrt{1-4ac/b^2} - 1$, a, b on stack
÷ ÷ 2 ÷ C +	forms $x''' = P_3 = P_2 + \frac{b}{2a}\left[\sqrt{1-4ac/b^2} - 1\right]$
B 'A' STO C 'B' STO >>	puts P_1 as new P_0 & P_2 as new P_1 for next iteration

Note: Store this in 'MLLR' and press to get new approximation.

10

Newton System

Given: Three initial values for three unknowns (x,y,z) in three fns.

Prerequisit: Store 0 in 'N' put initial value of first variable on stack then of second then of third then first fn then second then third. If you have a system of two by two, put 0 for third initial value and 'Z' for third function.

Program	Action
<< IF N 1 < THEN 'C' STO 'B' STO 'A' STO 'X' PURGE 'Y' PURGE 'Z' PURGE A X ∂ 'AX' STO A Y ∂ 'AY' STO A Z ∂ 'AZ' STO B X ∂ 'BX' STO B Y ∂ 'BY' STO B Z ∂ 'BZ' STO C X ∂ 'CX' STO C Y ∂ 'CY' STO C Z ∂ 'CZ' STO 1 'N' STO+ Z STO Y STO X STO END	Executes for one time only at start: storing the third fn in 'C', second in 'B', first in 'A', A_x in 'AX', A_y in 'AY', A_z in 'AZ', B_x in 'BX', B_y in 'BY', B_z in 'BZ', C_x in 'CX', C_y in 'CY', C_z in 'CZ', initial value of third variable in 'Z', second in 'Y' & last in 'X'
A →NUM B →NUM C →NUM {3} →ARRY	forms $\langle A_i\ B_i\ C_i \rangle$
AX →NUM AY →NUM AZ →NUM BX →NUM BY →NUM BZ →NUM CX →NUM CY →NUM CZ →NUM {3,3} →ARRY ÷ DUP ABS SWAP	leaves on stack error then $J_{ij}^{-1} F_i$
ARRY→ DROP 'Z' SWAP STO- 'Y' SWAP STO- 'X' SWAP STO- Z Y X >>	replaces $J_{ij}^{-1} F_i$ with z_1 then y_1 then x_1

Note: Store this in 'NSTM' and press ∧ to get at surface first x_1 then y_1 then z_1 then error$_1$. If you have 2x2, ignore the zero of z_1.

Least Square Approximation

Given: $(x_1, y_1), (x_2, y_2) \cdots\cdots (x_n, y_n)$ to fit

$$y = a_m x^m + a_{m-1} x^{m-1} + \cdots\cdots + a_1 x + a_0$$

Pre-requisit: Put the data: $[x_1, \cdots, x_n]$ then $[y_1, \cdots\cdots, y_n]$ then m on stack..

Program	Action.
`<< 'M' STO 'Y' STO 'X' STO`	→ Stores data
`X SIZE LIST→ DROP 'N' STO`	→ store no. of points in 'N'

`0 M FOR K X ARRY→ DROP`
`1 N FOR J M K - ^ N`
`ROLL NEXT NEXT {M 1+N}`
`→ ARRY 'P' STO`
	→ puts $\begin{bmatrix} x^m \\ x^{m-1} \\ x^{m-2} \\ \vdots \\ x \\ 1 \;\; 1 \;\; 1 \;\; 1 \;\; 1 \end{bmatrix}_{(m+1)\times n}$ in 'P'

`P DUP TRN *`	→ puts on stack the (m+1)x(m+1) array of coef. matrix $\begin{bmatrix} \Sigma x^{2m} & \Sigma x^{2m-1} & \cdots \\ \Sigma x^{2m-1} & \Sigma x^{2m-2} & \cdots \\ \Sigma x^{2m-2} & \cdots & \cdots \\ \Sigma x^m & \cdots & n \end{bmatrix}$
`DUP P Y *` `DUP ROT ÷`	→ leaves on stack: # the (m+1) vector of RHS and # the a's $\langle a_m, \cdots a_0 \rangle$ $\begin{bmatrix} \Sigma x^m y \\ \Sigma x^{m-1} y \\ \vdots \\ \Sigma xy \\ \Sigma y \end{bmatrix}_{(m+1)\times 1}$
`DUP P TRN SWAP * Y - DUP DOT ≫`	→ puts LSE on stack

Note: store this program in 'LSA' and press to get: first the least square error, then the a's, then RHS vector, then coef. matrix. You can get your numbers, then.

Neville Method

Given: x-coordinates in 'X', y-coordinates in 'Y' & finally on stack the value, at which Lagrange is to be evaluated.

Program	Action
NVL = << X SIZE	gets Size of 'X'
DUP LIST→ DROP 'L' STO SWAP CON X - 'X' STO	stores in 'L' the no. of points and in 'X' (a-x)'s
1 L 1 - FOR M	Starts column loops
1 L M - FOR K	Starts row loops
X {K M +} GET	gets on stack $a-x_{k+m}$
X {K} GET -1 *	gets on stack $-(a-x_k)$
{2} →ARRY DUP	forms $\left[a-x_{k+m}, -(a-x_k)\right]$ twice on stack
Y {K} GETI ROT ROT GET {2} →ARRY	forms $[y_k, y_{k+1}]$ on stack
DOT SWAP ARRY→ DROP + ÷	finds the element $(M,K)^{c\,r}$ of Neville
NEXT {L M -} →ARRY 'Y' STO Y	Leaves column 'M' elements on stack
NEXT >>	Leaves all columns on stack.

Press NVL and you get top pyramid column in an arry form then next-to-top column and so on down to bottom-most column.

Newton Divided Difference Table

[to get Newton Difference Table remove (:)]

Given: x-coordinates in 'X' & y-coordinates in 'Y'

Program	Action
NDD = << Y SIZE LIST→ DROP 'L' STO	stores in 'L' no. of points
1 L 1 - FOR M	starts column loops
1 L M - FOR K	starts row loops
Y {K} GETI ROTROT GET SWAP -	gets on stack $y_{k+1} - y_k$
X {K M+} GET X {K} GET -	gets on stack $x_{k+m} - x_k$
÷	gets element ('M', 'K') of M. (c r)
NEXT {L M -} →ARRY 'Y' STO Y	Leaves column 'M' elements on stack
NEXT >>	Leaves all column's on stack.

PRESS NDD and you get top pyramid column values in an array then next column down to base.

Hermite Table

Given: x's in 'X', y's in 'Y' & $\dot{y}$'s in 'YD'

Program	Action
HRMT=≪ X SIZE LIST→ DROP 'L' STO	Stores in 'L' no. of points
1 L 1 - FOR N YD {N} GET Y {N} GETI ROT ROT GET - X {N} GETI ROT ROT GET - ÷ NEXT YD {L} GET {L 2 * 1 -} →ARRY 'Y' STO Y	obtains the first DD column elements on stack in matrix form
1 L FOR N X {N} GET DUP NEXT L 2 * 'L' STO {L} →ARRY 'X' STO	Up-grades 'X' & 'L'
2 L 1 - FOR M	starts loop for other columns
1 L M - FOR K	starts loop for column-elements
Y {K} GETI ROT ROT GET -	gets on stack $y_k - y_{k+1}$
X {K} GET X{K M +} GET -	gets on stack $x_k - x_{k+m}$
÷ ⟶	gets element $(c^{r}_{m,k})$ of H.
NEXT {L M -} →ARRY 'Y' STO Y	leaves column 'm' elements on stack
NEXT ≫	leaves all columns

PRESS HRMT and you get his table elements top to base.

IV

Euler Algorithm (pp. 206 of text)

Given: t_0, y_0, h & $y' = f(t, y)$

Prerequisits: Store h in 'H', f in 'F' and leave y_0, t_0 on stack.

Program	Action
<< 'T' STO 'Y' STO	Stores t_0, y_0 in T, Y
Y H F →NUM * +	Forms $y + hf$ = new y
T H + >>	Forms $t + h$ = new t

Note: Store this in 'EULR' and press to get the new values of y & t respectively.

5.

Modified Euler Algorithm

Given: $t_0, y_0, h, y' = f(t, y)$

Prerequisite: Store h in 'H', y' in 'F' and leave y_0, t_0 on stack

Program	Action
<< 'T' STO 'Y' STO	stores t_0, y_0 in T, Y
Y F → NUM DUP H * Y+	forms $y_0 + hf(t_0, y_0)$
'Y' STO T H + 'T' STO	forms $t_0 + h$ in T
F → NUM + H * 2 ÷ +	forms $y_0 + \frac{h}{2}(f(t_0, y_0) + f(t_0+h, y_0+hf))$
T >>	puts new T on stack

Note: Store this in 'MELR' and press to get new values of y & t respectively.

Fourth Order Taylor Algorithm

Given: $t_0, y_0, h, y' = f(t,y), y'', y''' \& y''''$

Prerequists: Store h in 'H', y' in 'F1', y'' in 'F2', y''' in 'F3', y'''' in 'F4' and leave y_0 & t_0 on stack.

Program	Action
<< 'T' STO 'Y' STO	Stores t_0, y_0 in T, Y
F4 → NUM H * 4 ÷	forms $hy''''/4$
F3 → NUM + H * 3 ÷	forms $\frac{hy'''}{3} + \frac{h^2y''''}{3*4}$
F2 → NUM + H * 2 ÷	forms $\frac{hy''}{2} + \frac{h^2y'''}{2*3} + \frac{h^3y''''}{2*3*4}$
F1 → NUM + H *	forms $hy' + \frac{h^2y''}{2!} + \frac{h^3y'''}{3!} + \frac{h^4y''''}{4!}$
Y +	forms new $y = y + hy' + \frac{h^2y''}{2!} + \frac{h^3y'''}{3!} + \frac{h^4y''''}{4!}$
T H + >>	forms new $t = t + h$

Note: Store this in 'TYLR' and press to get new value of y & t respectively.

10.

Midpoint Algorithm

Given: $t_0, y_0, h, y' = f(t,y)$

Prerequisits: Store h in 'H', y' in 'F' and leave y_0, t_0 on stack

Program	Action
<< 'T' STO 'Y' STO	stores t_0, y_0 in T, Y
Y F →NUM H * 2 ÷ Y + 'Y' STO	forms $\frac{hf(t_0,y_0)}{2} + y_0$ in Y
T H 2 ÷ + 'T' STO	forms $t_0 + \frac{h}{2}$ in T
F →NUM H * +	forms $y_0 + hf(t_0+\frac{h}{2}, y_0+\frac{hf}{2})$ (new y)
T H 2 ÷ + >>	forms new t

Note: Store this in 'MDPT' and press to get new y & t respectively.

C1

Heun Algorithm

Given: t_0, y_0, h & y'

Prerequisits: Store h in 'H', y' in 'F' and leave y_0, t_0 on stack

Program	Action
<< 'T' STO 'Y' STO	Stores t_0, y_0 in T, Y
Y F → NUM DUP	puts y_0, f_0, f_0 on stack
H * 2 * 3 ÷ Y + 'Y' STO	forms $y = y_0 + \frac{2}{3} h f_0$
T H 2 * 3 ÷ + 'T' STO	forms $t = t_0 + \frac{2h}{3}$
F → NUM 3 * + H * 4 ÷ +	forms new y $= y_0 + \frac{h}{4}\left(f_0 + 3f\left(t_0 + \frac{2h}{3}, y_0 + \frac{2h}{3} f_0\right)\right)$
T H 3 ÷ +>>	forms new t

Note: Store this in 'HEUN' and press to get new y & t.

Runge-Katta Algorithm (pp. 225 of text)

Given: t_0, y_0, h & y'

Prerequists: Store h in 'H', y' in 'F' & leave y_0, t_0 on stack

Program	Action
<< 'T' STO 'Y' STO	stores t_0, y_0 on T, Y
Y F → NUM DUP	puts Y, f_0 & f_0 on stack
H * 2 ÷ Y + 'Y' STO	puts $y = y_0 + \frac{h}{2} f_0$ on Y
T H 2 ÷ + 'T' STO	puts $t = t_0 + \frac{h}{2}$ on T
OVER F → NUM OVER OVER	puts $y_0, f(t,y), y_0 + f(t,y)$ on stack
H * 2 ÷ + 'Y' STO 2 *	puts $y = y_0 + \frac{h}{2} f(t,y)$ on Y
OVER F → NUM OVER OVER	puts $y_0, f(t,y), y_0 + f(t,y)$ on stack
H * + 'Y' STO 2 *	puts $y = y_0 + h f(t,y)$ on Y
T H 2 ÷ + 'T' STO	puts $t = t_0 + h$ on T
F → NUM + SWAP DROP + SWAP DROP + H *	forms $k_1 + 2k_2 + 2k_3 + k_4$
6 ÷ + T >>	forms new y & t

Note: Store this in 'RNKT' and press to get new y & t.