19970831– 服部哲弥

計算機演習 3（第1回）

UBASIC　(1997.09.25)

UBASIC の起動

1. いつものように WINDOWS95 を立ち上げておく.
2. UBASIC の入った FD (floppy disk) を FDD (FD drive) に入れる.
3. マイコンピュータというアイコン (小さい絵) を探しダブルクリック (ダブルクリックは常に左クリック). マイコンピュータの窓が開くのでメニューから V(表示) を左クリック. 開いた窓から O(オプション) を左クリックし，窓が開いたら「表示」をクリック. 中ほど下の方の欄のに「登録されている拡張子は表示しない」のような文にチェックがついているときはチェック欄を左クリックして空欄にしておき，窓の下の方の「適用」を左クリックし，その横の「ＯＫ」を左クリック. (機械によって表示が違うかもしれない. うまく行かないときは申し出ること.)
4. マイコンピュータの窓に戻ったら，(A:) をダブルクリックして FD の窓を開き，ubv32 をダブルクリック. 背景の黒い窓が開いて，窓の左上付近に OK と出る. この窓の中が UBASIC が動いている状態.

注意. ubv32 が見つからないか，最後の黒い窓で

ファイルがみつかりません.
`UBCONST7.dat` がありません.

などの表示が出た場合は，操作を間違えていないか先ず確認. 問題なければ FD が正しくない可能性があるので申し出てください.

窓. 前期に演習したように，WINDOWS には「窓」という概念がある. UBASIC を使いながら他の画面 (メモ帳など) を使うことができる. 他の窓やアイコンが隠れて使いにくいときは UBASIC の窓の右上の三つのボタンの一番左を左クリック. このとき窓は題名だけ (最小化) になって下のメニューバーにある. 再び窓を開くときはこの題名を左クリック.

UBASIC の終了. UBASIC の窓の中で `system` と打ち込んで ENTER キーを押す. (各自試してみてから，UBASIC の窓を開き直すこと.)

プログラミングの例

自然対数の底 $e = 2.718281828459\cdots$ の計算.

1. BASIC の窓を active（画面下のメニューバーの ubv32 を左クリックするか，BASIC の窓のどこかを左クリック）にする．（最初は何もしなくても active になっているはず．）キーボードカーソル（点滅する四角または下線）が行の左端にあることを確認した上で，以下のプログラム（10 で始まる行から 120 で始まる行まで）を先頭の数字（文番号）も含めて打ち込む．行ごとに ENTER キーを最後に打て．**BASIC ではENTERキーまで込みで一つの行が命令として完成する．**このテキストではENTERキーは以後省略する．

```
 10   'e0
 20   input "point=";p
 30   point p
 40   input "n=";n
 50   if n<0 then end
 60   e=1
 70   wrk=1
 80   for i=1 to n
 85   wrk=wrk/i
 90   e=e+wrk
100   next
110   print "n=";n,"e=";e
120   goto 40
```

   訂正するときは左右上下の矢印でキーボードカーソルを動かす．上書きと挿入は INS キーがトグルになっている（同じボタンで機能を切り替えられることを，ボタンが「トグル」になっている，と言う）．WINDOWS のメモ帳などと違って最初は上書き．前の行も修正できるが，修正が終わったら修正した行で ENTER キーを忘れないこと．

2. 打ち終わったら，全部打ち込まれたか確認するために，

   ```
   list
   ```

   （これも最後に ENTER，以下同じ）と打つ．文番号つきで打ち込んだ全ての行が出てくるはず．確認する．

3. `asave "a:e0"`

   で，今打ち込んだプログラム (10 から 120 まで）を FD にファイルとして保存する．（`a:` は DOS/V 機では FD を表す．）`asave` は，テキストファイルとして保存する，という命令．`save` のほうが見慣れている人も多いと思うが，私は `asave` を推奨する．

4. `run`

   と打つ．これはプログラムを実行するという命令．

5. 間違っていなければ画面に `point=?` と出るので，2 と打ち込む．`n=?` と聞いてくるので，0 以上の整数をいろいろ入れて，`e=` の値がどう変わるか調べよ．やめるときは， `n=?` に対して負の数を入れる．

6. 再び `run` と打って，point の値を変えてみて何が変わるか確かめよ．

プログラム・ファイルの操作

`list`, `run`, `asave` は既出．セーブしたファイルの一覧は `files`, `files"a:"` などで見ることができる．例えば `a:` に保存してあるファイル `e0` を呼び出して走らせるには，`load"a:e0"`（プログラムを打ち込んだ状態が再現される）．既にプログラムを打ち込んでいた場合は置き換わる．

WINDOWS のエディタによる編集．プログラムは UBASIC の窓の中で作らずに，WINDOWS のメモ帳（notepad，前期の演習参照）などで，同様に（文番号を含み，メモ帳の一行をプログラムの一行に対応して）打ち込んだファイルを作って，FD に `e0.ub` などの名前で保存しても良い．メモ帳は画面左下スタートボタンの中のプログラム – アクセサリ – メモ帳，を順に左クリック．メモ帳で書いたものを保存するのはメモ帳のメニューの F–A を順にクリックし,「保存場所」の欄の右端の下黒矢印を左クリックして FD (A:) を選び，ファイル名欄に `e0.ub` などを書込んで `OK` を左クリック．このプログラムを読込むには UBASIC の窓を active にして，`load "e0"` などで読み込むことができる．

メモ帳で編集したときのファイル名．設定にもよるが，メモ帳で編集するとファイル名の最後，ピリオドの後（拡張子）が`txt` になる恐れがある．UBASIC のプログラムは拡張子が `ub` ．名前を確かめたり変更したりするには,「マイコンピュータ」のアイコンをダブルクリックし，さらに FD (A:) をダブルクリックして，ファイルの一覧を出す．例えば，`e0.ub` のつもりが `e0.ub.txt` となっていたら，そのファイルを左クリックして，メニューの「ファイル(F)　－　名前の変更(M)」を選び， `e0.ub` に直す．慣れていれば exploler （前期演習参照）を用いるのも簡単．
壊れても構わないファイルしかない最初のうちに，これらのプログラムやファイルの操作を試してみること．

課題

以下を電子メールで`enshu@rkmath.rikkyo.ac.jp` に送れ．（念のため送ったことを口頭で報告せよ．）

1. 例としてあげたプログラムを用いて，`point` の種々の値に対して，初めて結果 (`e`) が変わらなくなる`n` の値を求めて，`point` と `n` の対応を表にせよ．両

者にはどのような関係があるか推測せよ（既に UBASIC を知っている人は推測でなく証明でも構わない）.

2. `point` の値の役割を推測せよ（既に UBASIC を知っている人は，知っていることに基づいて答えて構わない）.

3. 時間が余ったら，木田先生の WWW ページに行って，UBASIC 関連で参照する可能性がありそうな情報をみつけよ.

UBASIC

後期は UBASIC を用いてプログラミングを中心に演習（午前）を行う.

BASIC は命令を一行ごとに解釈して実行する（インタープリタ）言語なのでプログラムを確かめやすく電卓に近い感覚で使え，また，絵を描く命令が簡単である．名前の通り初心者向きの言語で，簡単なプログラムを作るのに向いている．他方，プログラムの構造化（サブプログラムなど）が難しいため複雑なプログラムを作るのに不向きであり，インタープリタ言語なので遅い．高校や自分で勉強した人には，新鮮味に欠けるかもしれない．この演習は，上級向きの欠点よりも初心者向けの利点を重視する.

UBASIC は**立教大学理学部数学科の木田裕司教授**が BASIC をもとに多倍長計算用に作った言語．BASIC の一つの方言と思えるが，プログラム例に出た `point` という命令は BASIC になく，整数論への応用などに便利な UBASIC の特徴を象徴する．UBASIC を用いる主な理由は,

1. UBASIC を構成するファイルが小さくて FD に入れて使える（計算機センターはいろんな人が計算機を使うので，HD を自由に使えない）,

2. WINDOWS の DOS 窓など，MSDOS が使える主だった計算機と OS で使える（ので計算機環境が多少悪くても使える可能性が大）,

3. 無料,

4. 木田先生のホームページから実行ファイルをダウンロードできる,

5. 立教大学数学科の特徴である.（言語を自作した先生のいる大学は日本にはほとんどない．私は他に知らない.）

参考書等

木田裕司，UBASIC86 — 多倍長計算用 BASIC ユーザーズマニュアル，日本評論社．UBASIC の全ての命令の詳細を演習で紹介することは不可能なので，各自で本を買って調べることを勧める．木田先生のホームページから UBASIC のページを見ることも勧める．ubh というオンラインヘルプ用の実行ファイルなどもあり，頻繁に UBASIC を使いたい人はいろいろダウンロードしておくと便利.

計算機演習 3（第 2 回）

UBASIC 言語　(1997.10.09)

キー操作

高校や自分で BASIC を勉強した諸君が大半と思うが，復習．

UBASIC は UBASIC の命令文をキーボードから打ち込む．マウスは使えない．打ち込まれるのはキーボードカーソルの場所．キーボードカーソルは四角または下線（挿入モードと上書きモード）の点滅．以後 UBASIC の窓の中ではカーソルとはキーボードカーソルを指す．

カーソルを動かすキーには以下のようなものがある．

カーソルキー	矢印キー（上下左右）．カーソルを動かす．
DEL	delete と書いてあるかも知れないキー．カーソル位置の文字を消す．
BS	BackSpace と書いてある (?) キー．カーソルの左側の文字を消す．
INS	Insert と書いてある (?) キー．挿入と上書きのモード切り替え（トグル）．

問 A． UBASIC の窓を開き，適当なプログラムを load し (`load "e0"`) てリストを出 (`list`) しておく（第 1 回参照）．カーソルキーを用いてリストの上にカーソルを移動し，上記のキーを試してみよ．

ダイレクトモード

演算記号． 足し算 (+) と引き算 (-) 以外に以下の記号が自由に使える．

/	割り算 ($6/2 = 3.0$)（整数を代入しても実数で結果を得る）
//	割り算 （整数同士のとき有理数になる．UBASIC の特徴の一つ,）
*	掛け算 ($2 * 5 = 10$)
^	べき乗 ($2^3 = 8$)

複数の演算を一つの式で行うとき，括弧 () で演算の順序を指定できる．このほか，整数同士のみに用いる演算記号に，¥(商: `7¥3=2`) と @ (剰余: `7@3=1`) がある．

画面表示命令．

print （? も同じ）	結果の表示
cls	画面を全部消したいとき

問 B． 以下の各命令を実行してみよ．(命令の実行は行の最後に ENTER キーを打つ.)

- `print "hello, world"`
- `print 123*456` （または `? 123*456`）
- `? 13//9*3` と `? 13.0//9*3` (どう違うか？)
- `+-/^¥@` も使った式 (? から始めないと表示しない，注意).

関数． (他にもいろいろある．詳しくは木田先生のマニュアルを参照．)

abs(x)	絶対値	int(x)	x 以下の最大の整数	sin(x)	sinh(x)		
sgn(x)	符号	round(x)	x に一番近い整数	cos(x)	cosh(x)		
sqrt(x)	平方根	!(x)	x の階乗	tan(x)	exp(x)		
log(x)	自然対数			atan(x)			

不等号と if 文． =<> は命題（真偽の決まる文）を作る．If文や繰り返し文 (repeat 文，while文) などで条件を表すのに用いるのが普通．条件を表すのに `and` や `or` を使える．If文（に限らないが）の文法は通常の BAISC と UBASIC で微妙に違うので注意．下記問Cのようなやさしい例題で試すように．

変数． = は変数への代入というもう一つの重要な意味がある．例えば `a` は変数である．UBASIC では 16 文字以内の英数字で最初の一字は英字ならば，変数の資格がある．ただし，命令と同じ単語は使えないので，命令に見える単語は避けるのが無難．UBASIC ではさらに fn で始まる英数字列は関数とみなされるので変数名にはなれない．

問C．

1. `?sin(3.141592/4)` と `?1/sqrt(2)` を比べよ．
2. `?3>3` と `?3>=3` を比べよ．
3. `?(4=4)+(4>3)` を実行するとどうなるか？結果を予想してから確かめよ．
4. `new` または `clr a` を実行してから `?a=3` を実行せよ．次に，`a=3` を実行せよ．その上で `?a=3` を実行せよ．
5. `a=4: b=3: if a<b then c=1 else c=0 endif: ? c` としてみよ．カーソルキーでこの行に戻って 4 を 3 に，3 を 4 にして (`ENTER` して) みよ．
6. `a=3:b=4:c=5: if and{a<b, b<c} then d=1 else d=0 endif:?d` としてみよ．上の問と同様に a, b, c の値の組み合わせを８通り変えて結果を確かめよ．最初の `and` を `or` に変えるとどうなるか．

問C4で `a=3` が３カ所出てくるが，真ん中のものは = が代入命令を表す（ほかの二つは ? があるので表示命令文となり，= は条件式）．問C5で，: は二つの文を一行に書くときの切れ目．次の文と同値．

```
a=4
b=3
if a<b then c=1 else c=0 endif
? c
```

問D． 上記４行を打ち込んで確かめよ．

大きな整数，大きな桁数の実数

UBASIC の特徴の一つに，桁数の大きな数を（特別な命令やプログラムなしに）使える点がある．

問E． `?!(4)` や `?!(5)` などを実行して ! が階乗を計算していることを確認せよ．次に `?!(100)` や `?!(1000)` などを実行してどれくらい大きな整数まで計算できる

か調べてみよ.（たいていのプログラミング言語は，大きな整数を扱うには特別なプログラムを必要とする.）

プログラム

```
 5 a=1
10 ?a
20 end
```

などのように，数字（文番号）から始まる行のこと．このように書いて ENTER を押しても何も起こらないようにみえるが，例えば文番号 10 の行は「a=1 という命令をメモリの 10 番という記憶場所に登録せよ.」という命令である．目には見えないが，主記憶装置の 10 番に相当する場所に命令の文字列 a=1 が書き込まれる．

list	主記憶装置に書き込まれたプログラム文を文番号順に見る
list 30-60	文番号 30 以上 60 以下の部分を見る
run	主記憶装置のプログラム文の中身の命令を番号順に実行する
renum	文番号を $10n$, $n = 1, 2, 3, \cdots$ につけ直す 上の例では文番号が順に 10, 20, 30 になる． 間の文番号はプログラム訂正のとき行をはさむのに利用する．
new	主記憶装置に書き込まれたプログラム文を全て消し， 変数も初期化 (0 に) する（新規作成用）.
clr a	変数 a を初期化する．clr wrk など.

問 F． 第 1 回の e0 というプログラムでは一箇所 85 が入っている．これを読み込んで (load) renum を実行してみよ．list をとって，goto 文の中の文番号がどう変わったか比べてみよ.

一組の命令を何度も繰り返して実行したいときには毎回ダイレクトモードで同じ命令を打ち込むより，一度プログラム文として打ち込んでおいて，run を繰り返すほうが手間が圧倒的に少ない．いくつかのプログラム文を文番号順に並べて全体で目的の仕事をさせるようにしたものをプログラムと呼ぶ．**普通はダイレクトモードではなく，プログラムを作って実行する．ダイレクトモードで用いた種々の演算や条件式は全てプログラム文にそのまま書くことができる．** 初回の演習で既に自然対数の底を計算するプログラム (e0.ub というファイルに入っているプログラム) を作って実行したことは言うまでもない.

プログラムの暴走． プログラムの間違いなどのためにプログラムを実行し (run) ていていつまでたっても終わらないことを言う．CTRL-c（CTRL キーと c を同時に押す）でプログラムの実行を中断できる．この場合， cont という命令で続きを再開できる．暴走したのか時間がかかっているだけなのか分からないときに CTRL-c をかけて，ダイレクトモードでいろんな変数の値を調べて，問題なければ cont で再開する，というのが BASIC 的な使い方.

入出力文

`input` データを入力（読み込み）する．
`print` データを出力（書き込み）する．
`open` ファイルから（へ）の入出力をするための準備をする．
`close` ファイルへの入出力を終了して内容を確保する．
`kill` ファイルを削除する．(`kill"es1out.txt"`)
`eof` （入力）ファイルの終了を検出する関数．`#3` に対するファイルにデータが残っていれば`eof(3)=0`，なければ`=1`．
`val` 文字列の表す数値や数式を与える関数．

例：

```
 10 'es1
 20 input "input a number"; num
 30 num2=num*2
 40 print "The number is";num;".  Double it and you get";num2;""
 50 open "es1in.txt" for input as #1
 60 kill"es1out.txt": open "es1out.txt" for output as #2
 70 while not eof(1)
 80   input #1,a : print a : print #2 val(a)*2
100 wend
110 close #1,#2
120 print "End of job." : end
```

問 G． `new` としてから，プログラム es1 を打ち込め．別に，数値を数行にわたって打ち込んだファイルをメモ帳(notepad)で作成し，FD に`es1in.txt` という名前で保存せよ（Input 文は一つの行からデータを読み込むと，次回は次の行になる）．マイコンピュータなどでファイル名を確認しておく（第 1 回参照）．`run` で動くことを確認せよ．メモ帳で`es1out.txt` を開いて見よ．結果はどうなっているか？

課題

以下を電子メールで`enshu@rkmath.rikkyo.ac.jp` に送りなさい．（念のため送ったことを口頭で報告しなさい．）

1. 問 A–G を実行し，質問に答え，問題点があれば内容を報告せよ．
2. アンケートにも協力下さい．
   (a) 立教大学の講義演習以外で BASIC でプログラムを書いたことがあるか (yes か no)？Yes の場合，当てはまるのは？(i) 高校の授業で習った（科目名）．(ii) 自分で勉強した．(iii) それ以外（具体的に）．

(b) 他のプログラミング言語でプログラムを書いたことがある場合その言語名.

計算機演習 3（第3回）

例題（区分求積法）　(1997.10.16)

区分求積法

最も簡単な数値積分のアルゴリズムだが，周期関数に対しては非常に効率的で，DFT (Discrete Fourier Transform: フーリエ積分の離散化) に使われ，FFT (Fast Fourier Transform) の伏線ともなる．

有限区間の積分は積分変数の相似変換 $x' = ax+b$ で 0 から 1 までの積分に置き換えることができる．数値積分では積分を有限和で近似する (離散化)．単純に区分求積法で離散化すれば，$\int_0^1 f(x)\,dx \approx \frac{1}{n}\sum_{i=0}^{n-1} \mathrm{file1}(i)$ となる．ここで $\mathrm{file1}(i) = f(\frac{i}{n})$, $i = 0, 1, 2, \cdots, n-1$, と置いた．これを UBASIC の関数副プログラムに直したのが次の例である．

```
10 'ekubun
20 'Integration of f(x); 0<x<1. file1(i)= f(i/n), 0 <= i < n.
30 fnkubun(n)
40 local a,i
50 a=file1(0)
60 for i=1 to n-1
70   a=a+(file1(i)-a)/(i+1)
80 next
90 return(a)
```

作業 (第１回参照)．以上９行をメモ帳 (notepad) を開いて文番号つきで打ち込み，`ekubun.ub` という名前で FD に保存せよ (ゼロとオー，一とエルを間違えやすい．注意．例えば`file1` という文字列の３文字目はエル最後は一)．ファイル名の確認と訂正は，マイコンピュータの FD を左ダブルクリックで選ぶ．完成したら，ubv32 の窓を開いてプログラムを主記憶装置に読み込め (`load"ekubun"`)．

文法解説

注釈文 (行 10, 20)．`rem` または ’ で始まる文．(計算機ではなく) 人のための覚書．計算機は何もしないで飛ばすので，ないのと同じだが，人にとっては重要．

繰り返し (loop) 文 (行 60, 80)．ひとかたまりのプログラム部分を繰り返し用いる命令．短いプログラムで長い計算をさせることができる．ekubun では，先ず $i = 1$ として行 70 の計算を行い，次に，(行 90 に行かずに) $i = 2$ として行 70 を繰り返す．以下同様に $i = n-1$ まで繰り返したのち，行 90 に進む．行 80 の`next` は「次の i(で`for` と`next` の間の作業を繰り返す)」という意味．`for`‥`next` の構文は第１回のプログラム例 e0 でも使った．

配列 (行 50, 70). 引数 (数学の添字に相当) で区別する (大量の) 変数. 繰り返し文と組み合わせると, 短いプログラムで大量のデータを処理できる. UBASIC には, FD に書き込むファイル (ランダムファイル) を配列として使う機能があり, 3つの変数名`file1`, `file2`, `file3` がこのために使える. 後ろの括弧の中に引数 (添字) を入れる. 引数は変数や式でもよいが, 非負整数でないといけない. 行50, 70 はランダムファイル`file1`にデータが入っているときに, それを利用 (参照) する文例. ランダムファイルにデータを入れる (代入) 文は後述.

関数 (行 30, 40, 90). 行30–90 の一まとまりを, 関数を定義する副プログラムにするための行. 構造化 (いろんなプログラムにそのまま差し込んで使えるように, プログラムの一部分をひとかたまに考えること) の例. `fn` で始まる行が関数名と引数 (変数) を決める. 行30 は, 関数名が `fnkubun` であり, 引数が一つで, 行30–90 ではその引数を n として引用することを表す. 行40 は, `a`や`i` という変数名をこのプログラム部分以外で使っても, ここで使う`a`や`i` とは無関係 (別の変数) であることを宣言する. 関数の引数 (ここでは`n`) は自動的に local な変数になるので local 宣言してはならない. (それ以外の変数は, 同じ名前ならばプログラムのどこで使っても同じ変数.) `return(a)` は関数`fnkubun` の値. 例えばプログラムの別の行で `x=fnkubun(3)` とこの関数を呼ぶ (使う) と, `x` の値は行90 の`a` の値になる.

主プログラムの例

関数副プログラムだけでは何も作業しない (ekubun の9行を作成あるいは読み込んだところで `run` としてみよ). 主プログラムが必要である. 次の ekbnsmp は, 区分求積法の分点数`n` を入力すると, 積分 $\int_0^{\pi/2} \cos x\, dx$ の (近似) 値を計算する.

```
 10 'ekbnsmp
 20 'Integration of cos(x) from x=0 to x=pi/2.
 30 point 3
 40 input "bunten no kazu=";n
 50 if n<0 then end
 60 kill"ekbnin.ubd"
 70 open "ekbnin" as file1(n-1) word 4
 80 for i=0 to n-1
 90   file1(i)=#pi/2*cos(#pi/2*i/n)
100 next
110 s=fnkubun(n)
120 print "int[0,pi/2] cos(x) dx =";s
130 close file1
140 end
```

作業. UBASIC の窓で, 主記憶装置を初期化 (`new`) し, 以上を打ち込み, FD に保存せよ (`asave "ekbnsmp"`). ekubun のときのようにメモ帳で打ち込んで保存しても

よい．この場合はUBASICの窓で読み込む(`load "ekbnsmp"`)．次に，`append"ekubun"`と打ち込んでから，主記憶装置のプログラムを確かめよ(`list`)．ekbnsmpのプログラムの後にekubunのプログラムが続いているはず．確認したら，実行(`run`)し，`n`の値をいろいろ入れて，結果と正確な値との差が`n`とともにどう変わるか調べよ．さらに，行90の2カ所の`#pi/2`を`#pi*2`に変えて，$\int_0^{2\pi} \cos x\, dx$について同様に調べよ(下記課題1参照．行20, 120の`pi/2`も`pi*2`に変えるべき)．

文法解説

append．別々のファイルに保存してある複数の部分プログラムを主記憶装置に読み込んで全体で一つの大きなプログラムにする命令．`append`で読み込んだ部分は既に主記憶装置にある部分より大きな文番号に付け直される．

`input`, `print`, `if`…`then`は第2回参照．`end`は既に何度も出ているが，プログラム実行をそこで終えてダイレクトモードに戻る命令．

`kill"ekbnin.ubd"` **(行60)．**ファイルを削除する命令．なければ何もしない．ここでは何度も試す場合に備えて前に作ったファイルを消す．

ランダムファイル (行70, 90, 130)．行70は`file1`を配列として用いるための宣言．実体はファイルなのでファイル名(ここではekbnin.ubd：拡張子ubdは自動的につく)が必要．ekbnsmpでは許される引数は0, 1, 2, …, $n-1$．例えば行70で`file1(n-1)`のところを`file1(100)`と書けば引数は0, 1, 2, …, 100となる．`word 4`は配列要素一つ当たり用いる記憶容量が2×4B(バイト)という意味．行30の`point`の値(小数点以下の精度を表す)は(整数部の格納のために)`word`の値より小さくとる必要がある．変数`file1`の使用を終えるときに`close file1`とする(ファイルをBASICの管理から切り放すという意味)．行90で`file1`に値を代入している(通常の変数と同様)．

`#pi` **(行90)．**円周率．ちなみに，`#e`は自然対数の底．

ユーザー関数の使用 (行110)．ekubunで定義した関数`fnkubun(n)`の使用．

サイズと時間の概算

ファイルサイズの概算．ファイルekbnin.ubdは配列`file1`を格納する．行70で`file1`は要素一つ当たり4 word ($=2\times 4$B)使うから，FDに空き容量が800KBあるとすると$n=100000$が上限．

時間の概算．演習時間内に計算が終わることも重要(演習でなくても重要)．nの値はこの点でも制限される．ekbnsmpとekubunはnまでの繰り返しを行う繰り返し文がそれぞれ一つずつ入っているから，プログラムの実行時間Tはnの一次式$T=an+b$ (a, bは定数)と推測できる．(細かく言うと，数値によって時間が変わる．また，WINDOWS95は複数のタスクを同時に処理するので他の仕事とのかねあいがある．しかし，おおよそでも作業量を見積もることは極めて重要．)a, bは個々の計算機(hardware)によっても違うので，各自nを変えて時間の見当をつけるべきである．**重要：最初は小さな数で試すこと．**

課題

以下についての報告をメモ帳(notepad)で作成し印刷して提出せよ.

1. 積分範囲が $\pi/2$ までと 2π まで，それぞれについてekbnsmp を実行して，数値計算結果と正確な値との差が n とともにどう変わるか報告せよ．(参考：区分求積法は周期関数に対しては精度がよい．)

2. ekubun では $a_i = a_{i-1} + (\mathrm{file1}(i) - a_{i-1})/(i+1)$, $i = 1, 2, \cdots, n-1$, $a_0 = \mathrm{file1}(0)$, $a = a_n$, という計算をしている．これが $\dfrac{1}{n}\displaystyle\sum_{i=0}^{n-1} \mathrm{file1}(i)$ に一致することを証明せよ（ヒント：n についての数学的帰納法）.

3. 適当な命令を使って`#pi`が円周率(の近似値)であることを確かめ，その確かめ方と結果を報告せよ.

4. もし，ファイルサイズ上許される n の範囲でプログラムの実行時間が数分以上かかるようならば，プログラムの実行時間の式 $T = an + b$ の a と b を実測値から求めよ.(10秒程度から5分以内の n を2,3探して試せばよい．不正確な値でもよく，有効数字一桁で答えれば十分．時間をはかる命令を使えばもう少し正確になる(次回参照).)

5. 無限区間の積分だと $x' = ax + b$ では 0 から 1 までの積分にならない．この場合， ekubun を用いて数値積分するにはどうすればよいか？できれば複数の方法(考え方)を示せ.

6. 余裕のある人は，ekubun を修正して台形公式のプログラムを作れ．(最小限の変更を探して，どの行をどう変えれば良いかを報告せよ．)

参考：実数の数値計算

通常(IEEE規格)は計算機の実数や複素数は，整数部と小数部の桁数の合計(有効精度)が決まっている．数学以外の自然科学や工学，その他統計学などを用いる社会の諸学問ではこのほうが自然である．このとき課題2のような平均の計算を定義通り n 個の項を足してから n で割って計算すると，項数 n が大きいとき近似が非常に悪くなること(情報落ち)がある.

一般に計算機では実数(無限小数)は二進法有限桁で表現するために，近似値しか表現できない．このことから実数を含む数値計算では誤差が必ず生じる．通常，繰り返し文などで誤差が累積して大きくなる現象を**情報落ち**と**桁落ち**の二種類に分類する．UBASICは小数点以下の桁数が`point`で決まっているので，情報落ちは事実上起きないが，これはUBASICの興味深い特徴の一つである．他の言語では**ekubun で用いたアルゴリズム(計算の手順)が標準**なので，覚えておくこと．UBASIC でも桁落ちは起きる(次回参照).

計算機演習 3（第4回）

補足　(1997.10.23)

桁落ち

```
10 'eketa
20 one=1
30 for i=1 to 20 step 2
40   print i,one
50   one=(one-9999/10000)*10000
60 next
70 print i,one
80 end
```

for(行30)． for···nextはloop変数を1ずつ増やしながら作業を繰り返す命令であったが,「step 整数」(行30)とすると，増やす量を1以外に変更できる．eketaでは $i = 1, 3, 5, \cdots, 19$ に対して順に行40–50を実行する．to 20の意味は「20を超えない範囲で」(iを増やしてみて20を越えたら行70に進む)ということ．Loopを終了した(行70に進んだ)時点でのloop変数(i)の値は最後にloop内の作業を実行したときの値より1step大きい.（eketaの実行結果を確認せよ.）

=(行50)． 変数oneに入っていた数値(最初は1)を用いて右辺を計算し，その結果を変数oneに代入する．従ってoneの値は一般に変化する．

変数の型(行20,50)． 変数が整数か実数か複素数かはUBASICが自動的に判断する．変数oneは行20で整数1を代入するので行40では整数として1が表示される．行50の右辺で演算/は整数も実数として割り，結果は実数となる．整数と実数の混在する四則は全て実数になるので，行50で実数がoneに代入され，以後oneは実数になる．

桁落ち．

行50ではone= 1ならば右辺は1に等しく，従って，新しいoneの値は1のままのはずであるが，出力は1から大きくずれる．実数を表すのに有限桁の2進小数を用いるので，9999/10000は正確には0.9999に等しくない．その誤差はUBASICの初期状態(point= 4)では小数点以下20桁である(? pointと打ち込んでみよ)が，行50では**ほぼ等しい二つの量**one**と**9999/10000**の差**をとるため，相対誤差(誤差/(one-9999/10000))が1万倍になり，10000をかけて再び1に近い数字にしたとき，誤差が小数点以下$20 - 4 = 16$桁目に上がってくる．これを繰り返すと$20/4 = 5$回目には整数部に誤差が影響する．一般にほぼ等しい二つの量の差をとると小数の有限桁での打ち切りに由来する相対誤差が大きくなる．計算機科学では，eketaの実行結果をこのように理解して，**桁落ち**と呼ぶ．プログラムを作るとき，桁落ちをなるべく避けなければいけない．

繰り返し文(while...wend)

```
10 'enewton
20 'Newton-hou no reidai ;  x/(x+1)=log((x+1)/2) no kai (x>0).
30 point 5 : x=0 : y=3 : print "shokichi=",y
40 while abs(y-x)>10*#eps
50   x=y : y=x-fnf(x)/fnfp(x) : print "x=",y
60 wend
70 end
80 fnf(x) : f=x/(x+1)-log((x+1)/2) : return(f)
90 fnfp(x) : f=-x/(x+1)^2 : return(f)
```

while 条件式···**wend(行40, 60)．** for 文のようにloopを作る．条件式を調べて，成り立てばwendまでを実行し，条件式に戻って条件を調べる．条件が不成立になるとwendの次の文に実行が移る．while 文は，繰り返し回数が前もって分からないなどの，非定型な繰り返しに向く．

#eps(行40)． 表せる最小の正の数．point の設定で決まる小数点以下の一番下の桁が1という数．そのままだと危険なので行60 では10倍して使っている．

アルゴリズム． 方程式 $\frac{x}{x+1} = \log\frac{x+1}{2}$ $(x > 0)$ は代数的には解けないので，計算機で解の近似値(数値解)を求める意味が特に大きい．ニュートン法は $f(x) = 0$ の解 x_∞ の近似列 (x_∞ に収束する列)$\{x_n\}$ を $x_{n+1} = x_n - f(x_n)/f'(x_n)$, $n = 0, 1, 2, \cdots$, で再帰的に与える ($f' = \frac{df}{dx}$)．プログラムに翻訳するとき以下のような点も問題になる．1. **繰り返しをやめる判定**：実数は有限桁の近似値しか表現できないので有限項で終わる．2. **変数の個数**：再帰式では変数を使い回せば，直前の近似値と新しい近似値2個でよい．変数 x_n 全て用意するのは無駄．3. **結果の出力**：手で計算すると結果がそのまま見えるので，出力命令を軽視しがち．

プログラムを作る前に，計算機を意識したアルゴリズムを書き下し，さらに表や図(フローチャート)で書き直すこと．

enewtonのアルゴリズム

y: 最新の近似値(初期値：解の近似値)
x: 一つ前の近似値(初期値：y と異なる数)
ϵ: 小さな正の定数(許容誤差)
(A) $|y - x| \gtrless \epsilon$
YES $\Longrightarrow$ $x \leftarrow y$
$y \leftarrow x - f(x)/f'(x)$
(A) に戻る
NO $\Longrightarrow$ yを出力 $\Longrightarrow$ 終了

サブルーチン

この example1 の 行 80 が機械の状態によってエラーが出るらしい．うまく行かなかったら省略せよ．

```
 5 'example 1 for gosub
10 i=5: ? i
20 gosub 70
30 ? i
40 gosub *sample
50 ? i
60 end
70 *sample
80 local i
90 i=i+1 : ? i
100 return
```

```
 5 'example 2 for gosub
20 gosub 70
40 gosub *sample
60 end
70 *sample
90 i=i+1 : ? i
100 return
```

`gosub` **行番号 (行 20, 40)．**サブルーチンと呼ばれる副プログラムに飛ぶ命令．指定された行番号 (example 1 では行 70) から (通常は)`return`(行 100) までをサブルーチンと呼ぶ．行 20 まで処理が来ると行 70 に処理が移り，行 80, 90, と進み，行 100 で`return` を見ると，`gosub` で呼び出された行 20 の次 (行 30) に進む．行 40 でも行 70 に飛び (次の項目を参照)，行 100 の`return` で行 40 の次 (行 50) に進む．サブルーチンは，一まとまりの処理を作って独立したファイルに保存 (`asave`) しておいて，`append` して使うのに適している．複数の変数を更新するときなど関数の形に書きにくいときに向いている (次回も参照).

ラベル (行 40, 70)．`gosub 70`, `goto 70` など文番号を参照するとき，その行にラベル (`*`で始まる 23 字以内の文字) をつけて，文番号の代わりにラベルで参照できる．サブルーチンを別のファイルにして`append` で主プログラムにつなぐときは文番号が変化するので，ラベルを使うのが便利 (次回も参照).

`local`**(行 80)．**同じ変数名が副プログラムの外にあっても，中の変数とは別に扱うことを宣言する．`local` の指定があるので，行 90 の`i` は，`gosub 70` で飛んでくる度に，初期化 ($i = 0$) された新しい変数とみなされる．また，サブルーチン以外の i には影響を与えない.

次に，行 80 を消して実行してみよ (プログラム例右側の example 2 も参照)．`80` と打って`ENTER` すれば消える．(文番号だけ打ち込むと「その行を主記憶装置から消せ」という命令．) サブルーチンを複数回呼び出すとき，`local` 指定がないと先のサブルーチンでの作業結果があとのサブルーチンでの変数の値に影響する．**どの言語でも副プログラムと変数の影響の関係は紛らわしいので，マニュアルを読んだ上で簡単な例で試して理解すること．**

時間計測

多数回繰り返し文を実行する前には必要な時間の予測をすべきである (第3回参照). UBASIC には時間を与える変数time がある.

```
10 *starttime
20 'estime
30 clr time: tms=time: tms=right(tms,8)
40 return
```

```
10 *endtime
20 'eetime
30 tme=time: tme=right(tme,8)
40 sec=(asc(mid(tme,8,1))-asc(mid(tms,8,1)))
50 sec=sec+(asc(mid(tme,7,1))-asc(mid(tms,7,1)))*10
60 sec=sec+(asc(mid(tme,5,1))-asc(mid(tms,5,1)))*60
70 sec=sec+(asc(mid(tme,4,1))-asc(mid(tms,4,1)))*600
90 ? "keika jikan (byou)=";sec
100 return
```

使い方. 上の二つのサブルーチンをそれぞれ estime と eetime というファイルに保存する. 課題を処理する元のプログラムを作成またはload して, その時間経過を測りたい部分の最初と最後 (全時間を測りたいときはプログラムの最初と最後) にそれぞれgosub *starttime および gosub *endtime という行を挿入し, estime と eetime をappend して実行する.

```
   ...                          →      ...
 90 'mae no gyou                       90 'mae no gyou
100 'koko kara                         95 gosub *starttime
   ...                                100 'koko kara
200 'kokomade                            ...
210 'tsugi no gyou                     200 'kokomade
   ...                                 215 gosub *endtime
300 end                                210 'tsugi no gyou
                                         ...
```

説明. time は文字がデータとして入っている. 例えば3時間11分24秒は3:11:24 という**文字列**として時間が入っている (正しくは先頭に空白がいくつか入っている). ここの3 は数字ではなく, 3 という形をした文字の扱いなので, time*60 のような計算ができない. eetimeでは1文字ずつ数字に翻訳して分の値を60倍して秒に直す. 文字としての0123456789 はそれぞれ計算機では $48, 49, 50, \cdots, 57$ という数字 (ascii code) で表現される. asc という関数がこの値を与える. 例えばasc(2) とすると50と表示されるなど. right(a,n) は変数a の右側 n 文字を与える文字列操作関数. mid(a,m,n) は変数a の m 文字目から n 文字を与える関数.

描画

BASICは絵を描く命令が最初から用意されているのが嬉しいが，DOS/V機でUBASICの描画命令を行うと画面全体がUBASICに独占(フルスクリーンモード)され，カーソルが画面から消えるのがつらいところ．今期の演習では用いきれなかった．他方，描いた絵や図はWINDOWSのクリップボードにコピーできるので，前期の演習のようにお絵かきソフトに取り込み，印刷したりホームページに張り付けられる．規則的な図，特に関数や実験観測データなどのグラフを描くのが容易なので，応用範囲は広い．UBASICの描画命令をいくつか列挙しておくので，以下を順に試して頂きたい．例はダイレクトモード(第2回参照)で行っているが，他の命令と同様にプログラムにそのまま利用できる．

UBASICの描画命令の例．注意：グラフィック命令を行うと画面全体が黒くなって驚くかもしれないが，ALT-ENTER(この二つのキーを同時に押すこと)で窓表示に戻る．一連のグラフィック命令を終えたところで窓表示に戻るとよい．また，どの状態でもUBASICを終える(systemとうってENTER)と元のWINDOWS95の画面に戻る．(フルスクリーンモードではカーソルが表示されないので，暗闇の中でタイプしているような感じと思うが，タイプした文字は表示されるのでそれを手がかりにする．) 数字は画面の左上を$(0,0)$，右下を$(639,399)$とする座標で指定する．

`circle(300,203),101,5` 中心$(300,203)$半径101の円をパレット5番の色で描く
`circle(500,199.5),96.5,7,#pi/3,11*#pi/6` 円の$\pi/3$ラジアンから$11*\pi/6$ラジアンまでの角度の範囲を描く
`paint(400,200),3,5` 点$(400,200)$を含むパレット5番の色で囲まれた領域を3番の色で塗りつぶす
`for i=1 to 50:pset(200,100+2*i),2:next` `pset(x,y),n`は点(x,y)にn番の色で点をうつ
`line(50,40)-(639,399),1` 点$(50,40)$と点$(639,399)$を結ぶ線分を1番の色で描く

この他，cls 2は画面の絵を全部消す命令．ちなみに，cls 1はテキストを全部消し，cls 3は絵とテキストを消す．また，copy 2は絵をプリンターに出力する命令だが，7号館ではできないようだ．次に述べるクリップボードへの取り込みのほうが汎用性があってよいだろう．(copyと同様に2のところを1,3とすることもできる．)

画像のクリップボードへの取り込み．

UBASICで絵を描き終えたらALT-PRINTSCREENとする．これでUBASICの窓(中に描いた絵)を画像としてWINDOWSのクリップボード(実体は主記憶装置)に取り込んだことになる．ALT-ENTER(二つのキーを同時に押す)で窓表示に戻り，ペイントなどのお絵かきソフトを開き(前期演習参照)，描画画面でCTRL-v(同時に押す)

とするとUBASICで描いた絵が窓ごとペーストされるので，必要な範囲を切り貼りして加工保存印刷などをすればよい．前期にやったように保存したbmpファイルをIrfan viewでgifファイルに変換すれば描いた絵や図をホームページに貼ることもできる．(`ALT-PRINTSCREEN`はWINDOWSの機能なのでUBASIC以外の窓でも使える．)

変数についての補足

大文字． BASICでは基本的に変数や命令の大文字と小文字を区別しない．UBASICの場合全部小文字でプログラムを打ち込んでから`list`をとると，変数の最初の文字などが大文字になっているが，あまり気にしなくてよい．

複素数． UBASICの虚数単位$\sqrt{-1}$は`#i`．`2.5+3.3#i`などと使ったり，変数に代入して`a=2.5+3.3#i`などと使える(このとき変数`a`は自動的に複素数になる)．複素数の四則演算や指数関数(`exp(2*#pi*#i*x)`)なども可能．

loop変数． `for`‥`next`のloop文で繰り返し回数を数えるloop変数の動く範囲(`for i=a to b step c`の`a`,`b`,`c`)は変数や式でも構わないが，整数でないといけない．特に，`n/2`は実数になるので許されないが，`n//2`は`n`が偶数ならば整数になるので`a`,`b`,`c`いずれにも用いられる．

課題

以下についてメモ帳(notepad)で報告書を作成して提出せよ．

1. プログラムeketaの行50で`/`の代わりに`//`を使うとどうなるか？(UBASICの特徴の一つ！)
2. example 1 for gosub と example 2 for gosub の出力結果を報告し，説明文と矛盾がないか確かめよ．
3. estimeとeetimeを用いて区分求積法ekubun(第3回)のnと時間Tの関係を調べよ．(ekbnsmpを`load`して，行105, 115を`105 gosub *starttime`, `115 gosub *endtime`とし，ekubun, estime, eetimeを`append`して実行．)
4. 方程式 $\dfrac{x}{x+1} = \log\dfrac{x+1}{2}$ の $x > 0$ における解を数値計算で求め，答が何桁まで信用できるかも考察して報告せよ．enewtonの最小限の変更で解を小数点以下50桁(以上)求めるにはどうすればよいか？

余裕がある人の追加課題．

1. `#eps`は小さな数なので，`? #eps`としてみても`0.0`という答しか返ってこない．この値の大きさを知るためにはどんな命令を書けばよいか？適切な命令を実行して，`point`の値(`? point`)とともに報告せよ．
2. 解説ではenewtonのプログラムを文書と図示の中間的な形で書いたが，図示(フローチャート)するとさらに分かりやすい．工夫して図示してみよ．(基本的には各行を枠で囲って処理の順序を枠の間の矢印で表現すればよい．)

計算機演習 3（第５，６回）

プログラミングとアルゴリズム (1997.11.06, 11.13)

注意：一まとまりが一回では終わらないことと，予習が望ましい内容を含むことから，２回分をまとめて配る．次回も利用するので間違えないこと．

問題(解きたいこと)を計算機で解くとき，先ずアルゴリズム(課題解決のための処理の手順)を決める．アルゴリズムも数学的な解法という大枠から，入出力等の計算機上の制約やプログラミング言語の文法などの約束まで意識した詳細なアルゴリズムまでいくつかの段階を要する．一番詳細なレベルはプログラムの直訳に等しい．文法さえ間違えなければ(基礎を練習してマニュアルを横に置けば)，アルゴリズムからプログラムまでは難しくない．アルゴリズムを図示した流れ図(フローチャート)は，場合分けなどの見落としを防ぎプログラムの作成を容易にするので推奨される．さらに試験的実行，訂正(debug)・改訂が一連の作業に加われば，プログラミングの全体像となる．

問題に対してアルゴリズムは一つではない．上手に解かないと時間切れで解ききれない点は計算機も他の全ての数学と同様である．アルゴリズムの工夫を重視する理由の一つである．第３回で作った区分求積法のサブルーチン ekubun を少し書き換えて離散フーリエ変換(DFT)のサブルーチン edft を作り，高速フーリエ変換(FFT: より短かい時間で DFT と同じ結果を与えるアルゴリズム)に基づくサブルーチン efft と性能を実測比較する．

離散フーリエ変換(DFT)

フーリエ級数展開 $f(x)=\sum_{k\in\mathbf{Z}} C(k)\exp(\sqrt{-1}kx)$ のフーリエ係数 $C(k)$ は(f が適当に素直な関数なら)，$C(k)=\int_0^{2\pi} f(x)\exp(-\sqrt{-1}kx)\,\frac{dx}{2\pi}$ と書ける．$C(k)$ を区分求積法で計算するには(積分範囲が 2π までであることに注意すれば)，ekubun を少し書き換えれば次のプログラム edft で良いことが分かる．edft が用いる手順を DFT(Discrete Fourier Transform) と呼ぶことがある．行 70 は被積分関数と積分範囲(変数変換)に注意して ekubun を参考にしながら各自で完成せよ．

```
 5 *ftr
10 'edft
20 'file3(k)= Fourier coeff. C(k) of f(x).
30 'file1(i)= f(2 pi i/n), 0 <= i < n.
40 local a,i,k
45 for k=0 to n-1
50   a=file1(0)
60   for i=1 to n-1
70     a=a+(file1(i)*exp(2*#pi*#i*[        ])-a)/(i+1)
```

```
 80   next
 90   file3(k)=a
100 next
110 return
```

作業. UBASIC の窓で ekubun を`load` するか，メモ帳で ekubun.ub を編集して，フーリエ係数を求める関数副プログラム edft を作り，`asave "edft"`で保存 (メモ帳の場合は edft.ub という名前で保存) せよ.

例題. edft を用いて周期 2π の周期関数 $f(x) = 3/(5 + 4\cos\, x)$ のフーリエ係数を求める．区分求積法の分点数 n は，簡単のために自然数 m を手で入力して $n = 2^m$ で与える．行 100 は変数変換に注意して ekbnsmp を参考ながら各自で完成せよ.

```
 10 'eftsmp
 20 'Fourier coeff. C(k) of f(x)=1/(a+b cos x)
 30 point 2
 35 a=5/3: b=4/3
 40 input "log(bunten-suu)/log2=";m
 50 n=2^m
 60 kill"eftin.ubd": kill"eftout.ubd"
 70 open "eftin" as file1(n-1) word 8
 75 open "eftout" as file3(n-1) word 8
 80 i=0
 90 while i<n
100   file1(i) =1/(a+b*cos(          ))
110   i=i+1
120 wend
130 gosub *ftr
135 for k=0 to min(10,n-1)
140   print "C(";k;")=";file3(k)
145 next
150 close file1, file3
160 end
```

f が実数値関数なので， $C(-k)$, $k > 0$, は $C(k)$ の複素共役だから，$k \geq 0$ だけ計算すればよい．この例ではフーリエ係数の最初の 10 項しか表示していないが，サブルーチン (edft) は $C(k)$, $0 \leq k < n$, 全ての項を計算している．(興味があればプログラムの行 135 を書き換えてもっと表示させてみよ．)

作業. edft と同様に主プログラム eftsmp を作って保存せよ.

高速フーリエ変換(FFT)

edft (DFT) は分点数nと関数fの分点における値

$$\texttt{file1(i)} = f(2\pi i/n), \quad i = 0, 1, 2, \cdots, n-1,$$

を与えたとき，フーリエ級数の第k項の係数$C(k)$の近似値

$$\texttt{file3(k)} = C^{(n)}(k) = \frac{1}{n}\sum_{\ell=0}^{n-1} f(2\pi\ell/n)\exp(-2\pi\sqrt{-1}k\ell/n),\ k = 0, 1, 2, \cdots, n-1,$$

を計算する．

同じ量を計算する別のアルゴリズムにFFT(Fast Fourier Transform)がある．FFTは(nの素因数が有界という条件で)分点数nを大きくするときDFTよりはるかに速い．数値計算の古典なので取り上げる．FFTは任意のnに適用可能であるが，簡単のため，以下では$n = 2^m$の場合に限る．

FFTの原理． $n = 2^m$とする．関数fの分点における値

$$C'_0(i, 0) = f(2\pi i/n), \quad 0 \le i < n = 2^m,$$

を与えたとき，漸化式

$$\begin{aligned}
&\begin{pmatrix} C'_j(i,k) \\ C'_j(i, 2^{j-1}+k) \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 \\ 0 & \exp(-2\pi\sqrt{-1}i/2^{m-j+1}) \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} C'_{j-1}(i,k) \\ C'_{j-1}(2^{m-j}+i,k) \end{pmatrix}, \\
&i = 0, 1, 2, \cdots, 2^{m-j}-1,\ k = 0, 1, 2, \cdots, 2^{j-1}-1, \quad j = 1, 2, 3, \cdots, m,
\end{aligned}$$

で $C'_j(i,k),\ i = 0, 1, 2, \cdots, 2^{m-j}-1,\ k = 0, 1, 2, \cdots, 2^j-1,\ j = 1, 2, \cdots, m,$ を定義すると，

$$C^{(n)}(k) = \frac{1}{n}C'_m(0,k), \quad k = 0, 1, 2, \cdots, n-1,$$

が成り立つ．

FFTの原理に基づいて(UBASIC向きに)アルゴリズムを書き直すと，例えば次のようになる．途中の空欄は上記原理を参考にしながら各自で完成せよ．フローチャートの書式は一種類ではないので，一度伝統的な書式を覚えたあとは，自分なりに工夫しても良いかも知れない．(他の人と共同でプログラムを作るときは我流で書いてならない.）標準的なフローチャートをTEXで書くのは難しい上に紙の無駄遣いになるので，以下の書き方は我流のフローチャートをTEX向きに変形した．

efftのアルゴリズム

$n = 2^m$: フーリエ積分の離散化の分点数(主プログラムで入力)

file1$(i) = f(2\pi i/n)$, $0 \le i < n$ (主プログラムで配列として与えられる)

file3$(k) = C^{(n)}(k) = \dfrac{1}{n}\displaystyle\sum_{\ell=0}^{n-1} f(2\pi\ell/n)\exp(-2\pi\sqrt{-1}k\ell/n)$, $0 \le k < n$ (出力)

recursion時作業ファイル(file3は結果表示と兼用):

file2$(i2^{j-1}+k) = C'_{j-1}(i,k)$, $\quad 0 \le i < 2^{m-j+1}$, $0 \le k < 2^{j-1}$

file3$(i2^{j}+k) = C'_{j}(i,k)$, $\quad 0 \le i < 2^{m-j}$, $\quad 0 \le k < 2^{j}$

file2 ⟵ file1 $(i = 0 \sim n-1)$

$j:\ 1 \sim m$

| $i:\ 0 \sim 2^{m-j}-1$

| |

| | $f_1 \longleftarrow$ file2$(i2^{j-1}+k)$

| | $f_2 \longleftarrow$ file2$(2^{m-1}+i2^{j-1}+k)$

| | file3$(i2^{j}+k) \longleftarrow$

| | file3$(i2^{j}+2^{j-1}+k) \longleftarrow \exp(-2\pi\sqrt{-1}i/2^{m-j+1})(f_1-f_2)$

| ———

| ———

| file2 ⟵ file3 $(i = 0 \sim n-1)$

———

file3 ⟵ file2/n $(k = 0 \sim n-1)$

アルゴリズムさえ細かいところまでつめておけば，これをプログラムに翻訳するのはやさしい．次のefftはFFTに基づく離散フーリエ変換のサブルーチンである．途中の空欄は上記efftのアルゴリズムを参考にしながら各自で完成せよ．

作業． edft, eftsmp と同様に efft を作って保存せよ．

```
 10 *ftr
 20 'efft
 30 'file3(k)= Fourier coeff. of f(x) by Fast Fourier Transform.
 40 'file1(i)= f(2 pi i/n), 0 <= i < n, n=2^m.
 50 local i,j,k,nj,nj2,nnj,inj2,dphi,phi,f1,f2
 60 kill "eftwk.ubd": open "eftwk" as file2(n-1) word 8
 70 for i=0 to n-1: file2(i)=file1(i): next
 80 for j=1 to m
 90 'nj=2^j, nj2=2^(j-1), nnj=n/2^j=2^(m-j)
100 'inj2=i*2^(j-1), phi=exp(-2 pi #i i 2^(j-1)/n)
110   nj=2^j: nj2=nj//2: nnj=n//nj: dphi=exp(-#pi*#i/nnj)
```

```
120   inj2=0: phi=1
140   for i=0 to nnj-1
150     for [     ] to nj2-1
160       f1=file2(inj2+k)
170       f2=file2(inj2+n//2+k)
180       file3(inj2*2+k)=[     ]
190       file3(inj2*2+nj2+k)=phi*(f1-f2)
200     next
210     inj2=inj2+nj2: phi=phi*dphi
220   next
230   for i=0 to n-1: file2(i)=file3(i): next
240 next
250 for k=0 to n-1: file3(k)=file2(k)/n: next
260 close file2
270 return
```

第４回で作った時間計測サブルーチン estime, eetime を用いて二つのサブルーチン edft と efft の速さを比較する．

手順．

1. フーリエ級数展開の例題 eftsmp を読み込む (`load "eftsmp"`).
2. 時間計測サブルーチン estime, eetime を用いるため，行 129, 131 を追加する：

```
129 gosub *starttime
131 gosub *endtime
```

3. 時間計測サブルーチン estime, eetime を連結して保存する：

```
append "estime"
append "eetime"
asave "eftsmpt"
```

4. フーリエ積分のサブルーチンとして先ず edft の時間を測る：

```
append "edft"
run
```

入力 m を小さな数から 1 ずつ増やして分点数 $n = 2^m$ と経過時間との関係を表に書き留めておく．最終的なフーリエ係数も記録しておく．

5. 時間計測サブルーチンのついた例題を読み込みなおし，efft の時間を測る：

```
load "eftsmpt"
append "efft"
run
```

edft で測ったのと同じ m について分点数 n と経過時間との関係を表に書き留める．最終的なフーリエ係数も記録する．

6. 先ず，出力結果のフーリエ係数を比べてプログラムに誤りがないことを確認する．両者の結果を比較して速さや n を増やしたときの経過時間の増加について考察する．例えば，それぞれについて時間 T と n の関係をグラフにするなどして，T と n の関係を推測する．概ね $T = an^\alpha + b$ という形 (a, b, α は定数) に近いと仮定して，α が edft と efft でそれぞれいくつになるかを大雑把に見積もる．

注意． 変数の配列への代入の仕方を工夫するアルゴリズムの改善によって，efft においてランダムファイル`file2`を使わないようにできる．アルゴリズムが見えにくくなるので efft のプログラムは最善にはしなかった．

課題

メモ帳 (notepad) で以下についての報告書を作成し，印刷して提出せよ．第５回にできたところまで報告書を提出し，残りを第６回に提出せよ (家で考えてくることを強く推奨する)．第５回に全部できた場合は第５回のときに申し出ること．

1. 今回のプリントの最初の段落の説明 (プログラミング完成までの手順) をフローチャート風に図示してみよ．

2. 離散フーリエ変換 edft の行 70 の空欄を正しく埋めよ．

3. 例題 eftsmp の行 100 の空欄を正しく埋めよ．

4. 高速フーリエ変換 FFT のプログラム efft のアルゴリズムおよびプログラムの空欄を正しく埋めよ．

5. FFT のプログラム efft の行 120, 210 などは FFT のアルゴリズムに陽には出てこない計算をしている．実は，アルゴリズムは見やすくするため，行 120, 210 の操作を単純に書下しているが，プログラム efft のほうは速くするために工夫した．行 210 に出てくる変数 `phi` がアルゴリズムで何と書かれてい

るか指摘し，両者が等しいことを証明し，さらに，なぜプログラムのように書換えたほうが速い(かもしれない)かを考察せよ．

6. 高速フーリエ変換(FFT)のアルゴリズムが離散フーリエ変換(DFT)と同じ値を与えることを証明せよ．
(ヒント:$C'_j(i,k) = \sum_{\ell=0}^{2^j-1} f(2\pi(\frac{i}{n}+\frac{\ell}{2^j}))\exp(-2\pi\sqrt{-1}k(\frac{i}{n}+\frac{\ell}{2^j}))$, $0 \le k < 2^j$, $0 \le i < 2^{m-j}$, が成り立つことを j についての帰納法で証明する．$0 \le k < 2^j$ なる整数 k は $k = p2^{j-1} + q$, $p = 0, 1$, $0 \le q < 2^{j-1}$, と一意的に表され，$0 \le \ell < 2^j$ なる整数 ℓ は $\ell = 2s + t$, $0 \le s < 2^{j-1}$, $t = 0, 1$, と一意的に表されるので，$C'_j(i,k)$ の証明したい式(上記右辺)に代入する．)

7. $C(k)$, $k \in \mathbf{Z}$, を正確に「手で」(「解析的に」)計算し，数値計算の結果との差(数値計算の誤差)と n の関係を表にせよ．また，その傾向について気がついたことを述べよ．

8. n を増やすとともに数値計算結果が $C(k)$ の正しい一般項に近づく様子が一目で分かるように，出力(行140)を工夫せよ（書き直せ）．書き直した行140(複数行を用いた場合は新たに追加した行全て)，および，出力を報告し，誤差と n, k の関係について推測できる規則を報告せよ．求めよ．

9. 他の関数の例を各自自由に選び，eftsmp を書き換え，解説の手順にならってフーリエ係数を求めよ．選んだ関数とフーリエ係数(最初の10項ほどで構わない)を報告せよ．

課題予告

第７，８，９回は，課題を出してプログラムを自作してもらう予定である．しかし，自分のやりたいことのためにプログラムを自作するのが一番の練習なので，自分でプログラミングにとりかかっている(言語不問)人は事前に申し出ること．

申し出がない場合の課題の概要：

$$P_{d+1}(x) = \frac{1}{x}\int_0^x P_d(y)P_d(x-y)\,dy, \quad x \ge 0,\ d = 0, 1, 2, \cdots,$$
$$P_0(x) = \begin{cases} 1, & 0 \le x \le 1, \\ 0 & x > 1\,, \end{cases}$$

で帰納的に定義された関数列 $P_d(x)$, $x \ge 0$, $d = 0, 1, 2, \cdots$, について，d が大きくなるときの P_d の変化の様子が知りたい．適切な量を工夫して数値計算で調べて報告せよ（詳しくは後述）．

計算機演習 3（第７，８，９回）

課題制作　(1997.11.20, 11.27, 12.04)

課題 毎回，以下についての報告を電子メール (enshu@rkmath.rikkyo.ac.jp) またはメモ帳 (notepad) による印刷で提出せよ.

第７，８，９回

$$P_{d+1}(x) = \frac{1}{x}\int_0^x P_d(y)P_d(x-y)\,dy, \quad x \geq 0,\ d = 0, 1, 2, \cdots,$$
$$P_0(x) = \begin{cases} 1, & 0 \leq x \leq 1, \\ 0 & x > 1, \end{cases}$$

で帰納的に定義された関数列 $P_d(x)$, $x \geq 0$, $d = 0, 1, 2, \cdots$, について，d が大きくなるときの P_d の変化の様子が知りたい．予想されるスケール因子 L (後述の説明参照) に注目し，適切な量を選んで数値計算を行え．結果および P_d の変化についての数学的な予想を報告せよ（毎回その時点での結果または途中経過を提出）.

第８，９回 第１０回以降は自由制作とする．各自自分が３回以内で仕上げられる題材を選んで課題の内容の計画を立てよ（第８回に課題名，第９回に大枠のアルゴリズムを，それぞれ提出）．例はプリント最後の課題予告の項目にあげてみた.

第９回 最後に以下のプログラミングの項目を読み返して，それぞれの項目についてどれくらい身にしみたかあるいはぴんと来ないか，感想を述べよ.

プログラミング

プログラム作成の際に心がけることのうちには以下の点があると思う.

1. いきなりプログラムを書き始めると行き詰まるので，解きたい問題を well-defined(数学的に明確) にし，アルゴリズムを決定し，それを図示 (flow chart) することで，人 (プログラマ) にとってするべき仕事 (プログラミング) を明確にすること，
2. 間違いをあとから直す (debug) のは難しいので，間違えないように書くこと，例えば，
   (a) 用いる言語の基本を練習し，リファレンスマニュアルを横に置いて書くこと，
   (b) 微妙な場合には簡単なプログラムで確認すること，
   (c) 1 と l と I，0 と o のような紛らわしいタイプミスにも注意すること，

3. 上の注意を守っても，最初から完璧に仕上げるのは不可能だから，修正や改訂のために，予めコメント行を活用したり，分かりやすいプログラム (構造化) を書くこと，

4. 完成したと思っても，最初は簡単な場合 (少ないデータ，簡単な入力，手で計算して答の分かる場合，特殊な入力) で実行して，プログラムが期待通り動くことを確かめること．

一言で言うと，**人**は数学的な問題を解くには不十分なシステムであり，それを踏まえて問題を解こう，ということ．プログラムの自作を通して以上の注意が理解できることを第７，８，９回の目標とする．

課題について

1. 数値計算 (計算機で近似計算を行うこと) はコンピュータ応用の歴史的原点であり，第２種情報処理技術者試験では基本的な知識が仮定されている．他の話題は立教大学数学科に私より詳しい先生方が大勢いることもあって，数値計算の課題を選んだ．しかし，自分のやりたいことのためにプログラムを自作するのが一番の練習なので，**自分でプログラミング (言語不問) にとりかかっている人は，最初の日 (11.20) までに申し出ること．**課題として適切ならばその完成をもって第７－９回の課題に代える．特に手がけている問題のない場合は，上述の課題とする．

2. UBASIC はランダムファイルなど便利にできている部分もあるが，BASIC はインタープリタ言語なので，数値計算にとっては実行速度が遅いという致命的な欠点がある．**３回分の報告を定期的に提出すれば，他の言語コンパイラで同様の課題を行っても構わない．**

3. 以下は私 (服部) が試作したプログラムに基づいての解説である．課題の報告条件を満たせば，手順は以下のとおりでなくても構わない．

4. 毎回やり残した部分は家で考えてくることを強く推奨する．本を参照したり友人同士で相談することも構わない．第９回終了時点でプログラムが完成していることが期待されるが，途中までしか行かなかった場合でも，できた範囲を報告すること．

P_d の性質．P_d についていくつかの性質が分かっているのでそれを利用する．

- 任意の $d = 0, 1, 2, \cdots$ について，$0 \leq x \leq 1$ ならば $P_d(x) = 1$，$x \geq 2^d$ ならば $P_d(x) = 0$．一般に，$P_d(x)$ は x について広義単調減少．従って，特に，$0 \leq P_d(x) \leq 1$ が任意の $x \geq 0$ に対して成立．

- 各 $x \geq 0$ 毎に，$P_d(x)$ は d について広義単調増加．

- $1 \le x \le 2$ のとき $P_1(x) = \dfrac{2}{x} - 1$.
- 次の性質 (scaling) が予想される：ある $L > 1$ があって，$f_d(x) = P_d(L^d x)$ は $d \to \infty$ で収束するか，少なくとも d を大きく変えてもあまり変化しない．このような L を**スケール因子**と呼ぶことがある．L は $\dfrac{a}{a+1} = \log \dfrac{a+1}{2}$ の $a > 0$ における唯一の解を a_0 とするとき，$L = \left(\dfrac{a_0+1}{2}\right)^{1/a_0} \approx 1.261070486830679$ で与えられるようにみえる．
- f_d は P_d の性質を受け継ぐ．例えば，$x \ge (2/L)^d$ ならば $f_d(x) = 0$，$x \le (1/L)^d$ ならば $f_d(x) = 1$．また，$1/L \le x \le 2/L$ のとき $f_1(x) = \dfrac{2}{Lx} - 1$，および，$f_{d+1}(x) = \displaystyle\int_0^1 f_d(Lx\frac{1-u}{2}) f_d(Lx\frac{1+u}{2})\, dz$.

アルゴリズムの大枠．以上に基づいて次のようなアルゴリズムを提案する．

1. P_d の代わりにあまり変化しないと思われる $f_d(x) = P_d(L^d x)$ について調べる．(必要ならば結果において x を L^d 倍すればよい．) スケール因子 L は上記の値を信じて使ってよいが，方程式をプログラムの最初で数値的に解いてから用いる方法もある (第４回 enewton 参照).
2. n を，離散化するときの分点数とする．$x > (2/L)^d$ ならば $f_d(x) = 0$ なので，関数 f_d は $x_{d,i} = 2^{di/n} L^{-d},\ i = 0, 1, 2, \cdots, n,$ での値で代表させ (離散化)，UBASIC のランダムファイルを用いて `file3(d,i)`$= f_d(x_{d,i}),\ i = 0, 1, 2, \cdots, n,$ とする．$x \le (1/L)^d$ と $x \ge (2/L)^d$ ではそれぞれ $f_d(x) = 1$ と $f_d(x) = 0$ が分かっているので関数値を保存しておく必要はない．積分は関数値の大きいところがきくはずなので，x の小さいところから多くサンプルをとるように選ぶが，i の指数関数に選ぶのは「山勘」であり，他の選び方も考えるべきである．積分の分点数と関数の代表点の数を両方とも n としたが，変えることもできる．
3. Iteration(帰納的定義の繰り返し) の回数 $dmax$ を手で入力することにして，$d = 1, 2, 3, \cdots, dmax - 1$ について次の作業を繰り返す：$\{f_d(x_{d,i}) \mid i = 0, 1, 2, \cdots, n\}$ の計算がすんだとき，$\{f_{d+1}(x_{d+1,k}) \mid k = 0, 1, 2, \cdots, n\}$ を，f_d を用いた積分で求める．
4. 積分は台形公式を用いた関数副プログラム (edaikei) とする．区分求積法 ekubun.ub を１行書き換えれば作れる．
5. $f_d(x)$ は関数副プログラム (enaisou) で計算する．$x \notin \{x_{d,i} \mid i = 0, 1, 2, \cdots, n\}$ ならば内挿しなければならない．$x_{d,i} = 2^{di/n} L^{-d}$ とおいたので，$f_d(x)$ を求

めるのに $z = \dfrac{n \log x}{d \log 2} + n \dfrac{\log L}{\log 2}$ を主プログラムで計算し，$i = [z]$(整数部) と $i+1$ の間で線型内挿すれば関数副プログラムは簡単に書ける．

6. 出力は

   (a) $\{(x_{d,i}, f_d(x_{d,i})) \mid i = 0, 1, 2, \cdots, n\}$ をそのまま数値として出力,

   (b) $(x_{d,i}, f_d(x_{d,i}))$ をグラフとして図示,

   (c) 点 x'_i, $i = 1, 2, \cdots, M$, を決めてそこでの値 $f_d(x'_i)$ を数値として出力,

   (d) 点 $y_i, i = 1, 2, \cdots, M$, を決めて $f_d(x''_i) = y_i$ となる x''_i を数値として出力,

   など，種々の方法があり得る．最後の方法はプログラム上は多分一番高級だが，f_d は値域が $[0, 1]$ $(f_d(0) = 1,\ f_d((2/L)^d) = 0)$ と決まっているので，例えば，$y_i = i/10,\ i = 1, 2, \cdots, 9$ と選べばよいので，前もって f_d の詳細な性質を知らなくても確実に出力を得られる．これに対して f_d の support$(f(x) \neq 0$ となる範囲) は $[0, (2/L)^d]$ と，d によって大きく違うので，x'_i を d によらずに決める決め方が難しい．但し，適切な x の範囲が予想できれば第 3 の方法はたいへん簡単にプログラムできる．$f_d(x_{d,i})$ の数値を列挙するのはプログラムは最も簡単だが，結果が人の目には分かりづらいし紙やディスクの無駄遣いになりがち．グラフにして図示するのは大変良い方法だが，各自の研究に任せる．例えば $f_d(x_{d,i})$ を全てファイルに保存し，あとから作表ソフトなどを使ってグラフ化する方法もある．

プログラミングの大枠

1. $\displaystyle\int_0^1 f(x)\,dx$ を台形公式で計算する関数副プログラム edaikei の作成.

   (a) 区分求積法のプログラム ekubun(第 3 回参照) を`load` して，コメント文は適宜修正し，関数宣言文を`fndaikei(n)` に置き換え，途中もう一カ所 (どこか？) を正しく書き換えれば完成．`asave edaikei` で保存.

   (b) サンプルプログラムを走らせてテストする．区分求積法用のサンプルプログラム ekbnsmp(第 3 回) を`load` して，積分計算の作業で`fnkubun` を呼び出す代わりに`fndaikei` を呼び出すように一行書き換え，上で保存した`edaikei` を`append edaikei` でつないで実行する．積分の近似値が正しく得られるか，分点数 n や関数を代えて試してみる．(途中経過報告の内容例：テストプログラムで選んだ関数，分点数，出力，正しい値との差などを報告して，プログラムが正しいと判断した根拠を述べよ．以下でも同様に報告せよ.)

2. 離散データで表示された関数の任意の点での値を内挿で求める enaisou の作成.

(a) 離散化のデータ数nと変数変換$x = 2^{dz/n}L^{-d}$のdは主プログラムで与えられているとする．関数の離散データは主プログラムでランダムファイル`file3`に代入されているとする．ここで，`file3`$(i) = f_d(x_{d,i})$ $(x_{d,i} = 2^{di/n}L^{-d})$．さらに，勝手な点 $x > 0$ における f_d の値 $f_d(x)$ を求めるために，主プログラムで$z = \dfrac{n\log x}{d\log 2} + n\dfrac{\log L}{\log 2}$ ($x = 2^{dz/n}L^{-d}$をzについて解いた式)を計算してあるものとする．

(b) 関数宣言文はn,d,zを引数としする．即ち，`fnnaisou(n,d,z)`とする．

(c) enaisouを構成する文は以下のものがあるべきである．

i. 適切なコメント文(複数可)

ii. 関数宣言文(`fnnaisou(n,d,z)`)

iii. $z \leq 0$ ならば値1を返し， $z \geq n$ ならば値0を返す．ここで値yを返すとは `return y` という命令であった．

iv. j を z の整数部(`j=int(z)`)とする．

v. (残りの$0 < z < n$の場合は)平面内の2点$(j, $`file3`$(j))$と$(j+1, $`file3`$(j+1))$を$z-j : j+1-z$に内分した点を$(z, y)$とするとき値$y$を返す(内挿)．

(d) 参考になるプログラムを演習していないので，細かいアルゴリズムやフローチャートを書いてからプログラムを書くことを勧める．

(e) サンプルプログラムによるテスト．関数P_1 (の離散データ)をプログラム内で`file3`に与えて，関数P_1の離散データを出力するサンプルプログラムを考え，作成し，enaisouを`append`して実行し，出力を本当の値(例えば$P_1(x) = \frac{2}{x} - 1$ $(0 \leq x \leq 1)$)と比べよ．(本当の値も主プログラムの中で計算させて，本当の値と差を出力するように主プログラムを組んでおくとテストしやすい．) 主プログラムからenaisouを呼び出すとき，変数xから変数変換$z = \dfrac{n\log x}{d\log(2/L)}$で$z$に置き換えないといけないことに注意してプログラムすること．注意深くやればP_2くらいは手で計算できるので，そこまで試せばかなり信頼できるだろう.(上の例にならって途中経過の報告に加える.)

3. 主プログラム erec の作成.

初期化： n, $dmax$ などの入力，ランダムファイルや出力ファイルの初期化，recursionの初期関数(f_1)の定義，など．参考までに下に見本を掲げる．

Recursion: 元の関数f_d(離散点での値)から新しい関数f_{d+1}をつくる．ここが主要部．`file1`(i)に被積分関数$f_d(Lx(1-u)/2)f_d(Lx(1+u)/2)$の$u = i/n$(台形公式の分点)での値を代入$i = 0, 1, 2, \cdots, n$すれば，

`file2(k)=fndaikei(n)` によって積分値 $f_{d+1}(x)$ を得る．但し，$x = x_{d+1,k} = 2^{(d+1)k/n}L^{-d-1}$．これを $k = 0, 1, 2, \cdots, n$ に対して行う ($k = 0$ では1, $k = n$ では0, に注意)．

f_d の値は関数`fnnaisou`で計算するが，対応関係は

$$f_d(x) = \texttt{fnnaisou}(n, d, \frac{n \log x}{d \log 2} + n\frac{\log L}{\log 2})$$

であることに注意．

一旦`file2`に代入した f_{d+1} の離散データは recursion の次の d では被積分関数に使われるので f_d を置き換えることになる．つまり，`file2`の代入が終わったら，

```
350   for k=1 to n-1 : file3(k)=file2(k) : next
360   file3(0)=1 : file3(n)=0
```

を実行しておけばよい．

出力： Recursion(d の loop) の最後 (`next` の直前) に f_{d+1} の値を出力する．いくつかの方法をアルゴリズムの項で説明したので各自の工夫に任せる．下の例の行200–240 は第４の方法を意識している．

主プログラム部分の詳しいことは各自の演習に任せる．

4. できれば f_2 の式を手で計算しておいて，erec で f_1 から求めた結果と比較するテストプログラムを作ればプログラムのチェックになるだろう．

5. 考察．erec を保存 (`asave`) し，edaikei と enaisou を`append`して小さな $dmax$ で走らせて正しく実行するか，時間がどの程度かかりそうか，などを確認する．テストと debug(修正) を繰り返して正しい出力を得られるようになったら，本格的に走らせて，出力，および，冒頭の課題に関して気がついたことを報告する．

主プログラム erec の初期化に関する部分の例．

```
10 'erec
20 'f(d+1;x)=1/(Lx)*int[0,Lx] f(d;y)f(d;Lx-y) dy, f(0;x)=(0<x<1)
21 '         =int[0,1] f(d;Lx(1-u)/2)f(d:Lx(1+u)/2)du
30 point 2
40 input "bunten no kazu=";n
50 if n<0 then end
60 input "zenkashiki kosu=";dmax
70 if dmax<=1 then ? "You need no numerical cals.." : end
```

```
 80 kill "erec1.ubd" : kill "erec2.ubd" : kill"erec3.ubd"
 90 open "erec1" as file1(n) word 3
100 open "erec2" as file2(n) word 3
110 open "erec3" as file3(n) word 3
120 'file1(i)=normalized integrand of recursion.
130 'file2(k)=f_{d+1}(2^((d+1)k/n)/L^(d+1)).
140 'file3(k)=f_{d}(2^(dk/n)/L^d).
150 open "erecout.txt" for append as #2
160 ? #2 "n=";n
170 lg2=log(2) : scale=1.261070486830679 :'scale=L
180 for i=0 to n : file3(i)=2^(1-i/n)-1 : next
190 ddkn=2^(1/n) : dkn=ddkn    :'dkn=2^((d+1)/n) (for output)
200 ?    " d";:for k=9 to 1 step -1:?    using(2,5),k/10;:next
210 ? #2 " d";:for k=9 to 1 step -1:? #2 using(2,5),k/10;:next
220 ? : ? #2
230 for d=1 to dmax-1
240   ? using(3,0),d+1; : ? #2 using(3,0),d+1;
250 ' recursion
```

ここに主要部（漸化式の計算と結果の数値出力）が来る．

```
510   ? : ? #2
520 next
530 close
540 end
```

課題予告の例

第１０回以降の課題の例を思いつくままにあげておく．なお，前期にならって，自由制作の最終回までに，制作内容や結果の要旨を各自ホームページに掲載し，それを相互に鑑賞して人気投票を行う．

1. 第７－９回の課題制作の発展．例えば

   (a) スケール因子Lを代えたらどうなるかを調べる．(この課題例はプログラムを改めて実行しなくても結論が出せるはず.)

   (b) 初期関数を変えて同じ recursion を実行する．例えば$f_1(x) = P_1(Lx)$の代わりに$f(x) = \exp(-x^a)$から出発するとどうなるか？(注：$f(x) = \max\{1 - x^a, 0\}$のほうがよいかもしれない．) aは，今までと同様に，$\dfrac{a}{a+1} = \log\dfrac{a+1}{2}$ の$a > 0$における唯一の解．(この関数$f(x)$はP_dの性質で述べた scaling の予想の根拠となる関数．) この場合，xからzへ

の変数変換するならば，$z < 0$のところも関数の値は1からずれるのでファイルに保存しないといけないことに注意．変数変換しないほうがいい可能性もある．

(c) n を変えた計算に基づいて誤差の蓄積の大きさを評価する．関数を代表点で表す方法，台形公式の分点などを個別に変えること，などの工夫で誤差の縮小をめざす．

(d) 高速化，例えばサブルーチンのインライン化 (主プログラムに書いてしまうことでプログラムの見かけは複雑になるが，サブルーチン呼び出しという仕事をなくして実行を高速化すること) や最内側ループの計算量を減らすこと．

(e) グラフィック命令 (第４回参照) を用いて再帰的に得られる関数を図示．

2. 午後の授業 (計算機演習４) のアルゴリズムの具体化 (乱数，ソート，その他)．例えば，乱数を発生させて大量のデータを集め，それを関数の代表点とみなして，フーリエ変換 (FFT;第５，６回参照) でフーリエ係数を求めてみる．「理想的」な乱数ならばどうなるべきかという理論的考察を行い，結果を比較する．さらに，乱数データをソートしたものを別の関数の代表点とみなして，フーリエ係数を求めて比較してみる，など．他の授業で勉強したものでも，もちろん構わない．

3. グラフィック命令を用いた数学的な複雑な量の図示．例えばランダムウォークの可視化など．

4. プログラミングよりも前期にやったホームページの改良に専念したい場合は，具体的な計画と実現可能性がはっきりしていれば認めるので，第８回に申し出て，第９回に具体的な計画を提出せよ．例えばグラフィック命令で書いたグラフや図形を加工してホームページに掲載するなど (第４回参照)．

計算機演習 3（第１０－最終回）

自由制作　(1997.12.11, 12.18, 1998.01.08)

課題

1. 第８，９回に立てた計画に基づいて自由制作を行い，毎回途中経過または結果を電子メール

   `enshu@rkmath.rikkyo.ac.jp`

   またはメモ帳(notepad)による印刷によって報告する.

2. 最終回に，計算機演習３（午前の部）に対する感想を電子メールによって報告する．(感想には，題材，進む速さ，満足または失望，理解の程度，を含めて下さい．)

3. 制作した課題とその結果についての説明等，計算機演習(後期)の成果を前期演習で作成した各自のホームページの下に新しいページを作って掲載する．(計算機演習3の課題の内容がすぐ分かるように最初に短い要約をつけるとよい．) 自分の課題が終了したら，他の諸君のページの中から後期の演習に関して最も気に入ったものを選び，

   自分の名前と学籍番号
   一番気に入ったページの学生の名
   判定理由

   をメールで報告する．２年生のホームページは，計算機演習の数学科のホームページ

   `http://150.93.96.124/`

   から開けばよい.

演習時間外に勉強していて疑問が生じた場合は
`hattori@rkmath.rikkyo.ac.jp`
あてにメールで問い合わせてもよい.（返事が遅れることもあります.）演習の課題のメール先と問い合わせ先が違うので注意.