תוכניות דוגמא

fderiv1.c,
fderiv2a.c fd2a.asm,
fderiv2b.c fd2b.asm,
fderiv2c.c fd2c.asm.

התפקיד העיקרי של התוכניות הללו הינו להמחיש את השימוש בפוינטר לפונקציה כפרמטר, דבר נפוץ במיוחד בחישובים נומריים.

בהנתן פונקציה נתונה נניח ע"י איזה קוד חישובי, אנחנו רוצים לעשות קירוב נומרי של הנגזרת שלה בנקודה. זהו תסריט שמתרחש כאשר יודעים לחשב פונקציה מסוימת אבל לא בהכרח יודעים איך היא נראית אנליטית, למשל כאשר היא פתרון של משוואה שמחושבת באופן נומרי או חישוב נומרי אחר.

חישוב הקירוב הנומרי מבוסס על כך שהנגזרת הוא הגבול של

$$f'(x) = \lim_{h\to 0} (f(x+h) - f(x-h))/(2h)$$

כלומר אם נחשב את הקירוב לנגדזרת ע"י הנוסחה (*):

$$f'(x) = (f(x+h) - f(x-h))/(2h) \quad (*)$$

עבור h מספיק קטן, נקבל (בתאוריה לפחות) קירוב לנגזרת. השאלה היא איזה h לבחור. ככל ש-h קטן יותר החישוב יהיה יותר נכון מבחינה מתמטית אולם אם נבחר h קטן מדי יגרום לבעיות של אובדן דיוק כתוצאה משגיאות עיגול. חוץ מזה, h חייב להיות קטן יחסית ל-x ולאו דווקא קטן במושגים מוחלטים. פתרון אפשרי הוא להתחיל אם $h = |x/2.0|$ וכל הזמן להקטין את h ע"י חלוקה ב-2.0 עד שנגיע לכך שהחישובים מתחילים להתכנס, כלומר 2 חישובים עוקבים של הנוסחה (*) ההפרש ביניהם בערכם המוחלט קטן מאפסילון נבחר. בתוכניות הללו נבחר אפסילון כערך המוחלט של f(x) חלקי 8192.0. אילו החישובים היו בדיוק יותר גבוה מ-float היינו מחלקים ביותר.

השיקולים העומדים מאחרי האלגוריתם הזה קשורים לנושאים של דיוק חישובי שלא כאן המקום לפרטם.

התוכנית fderivl.c היא מימוש האלגוריתם בשפת C. התוכנית המשולבת fderiv2a.c ו-fd2a.asm, fderiv2b.c ו-fd2b.asm ו-fderiv2ca.c ו-fd2ca.asm ממשות את האלגוריתם הנומרי באסמבלי הנקרא מ-C עבור מספרים ממשיים 32, 64 ו-80 ביט בהתאמה.

```
/* fderiv1.c - approximate derivative function */

#include <stdio.h>
#include <math.h>

float approx_fderiv(float (*f)(float), float x)
{
  float h, fd0, fd1, eps;

  h = x/2.0;
  fd1 = ((*f)(x+h) - (*f)(x-h))/(2*h);
  eps = fabs((*f)(x)/8192.0);

  do {
      fd0 = fd1;
      h = h/2.0;
      fd1 = ((*f)(x+h) - (*f)(x-h))/(2*h);
     } while(fabs(fd0 - fd1) > eps );

  return fd1;

} /* approx_deriv */


float f(float x)
{
  return x*x*x - 2.0*x*x + 3.0*x - 8.0;
} /* f */

float real_fderiv(float x)
{
  return 3.0*x*x - 4.0*x + 3.0;
} /* f */


int main()
{
  printf("approx_deriv(5.0) = %f\n", approx_fderiv(f, 5.0));
  printf("real_fderiv(5.0) = %f\n", real_fderiv(5.0));

} /* main */
```

```
E:\>tcc -v fderiv1.c
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
fderiv1.c:
Turbo Link  Version 5.0 Copyright (c) 1992 Borland International

        Available memory 4103660

E:\>FDERIV1.EXE
approx_deriv(5.0) =  58.001526
real_fderiv(5.0) = 58.000000

E:\>
```

```
/* fderiv2a.c - approximate derivative function */

#include <stdio.h>
#include <math.h>


extern float approx_fderiv(float (*f)(float), float x);


float f(float x)
{
  return x*x*x - 2.0*x*x + 3.0*x - 8.0;
} /* f */

float real_fderiv(float x)
{
  return 3.0*x*x - 4.0*x + 3.0;
} /* real_fderiv */


int main()
{
  printf("approx_deriv(5.0) = %f\n", approx_fderiv(f, 5.0));
  printf("real_fderiv(5.0) = %f\n", real_fderiv(5.0));

  return 0;
} /* main */
```

```
C:\>tcc fderiv2a.c fd2a.asm
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
fderiv2a.c:
fd2a.asm:
Turbo Assembler  Version 3.1  Copyright (c) 1988, 1992 Borland
International

Assembling file:   fd2a.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  397k

Turbo Link  Version 5.0 Copyright (c) 1992 Borland International

        Available memory 4108576

C:\>fderiv2a.exe
approx_deriv(5.0) = 58.001526
real_fderiv(5.0) = 58.000000

C:\>
```

```
;
; fd2a.asm - implement numerical differentiation
;
;
  ASSUME CS:_TEXT, DS:_DATA
;
  ; Static Variables
;
_DATA SEGMENT DWORD PUBLIC 'DATA'
h  DD ?
two DD 2.0
fd0 DD 0.0
eps_const DD 16384.0
eps  DD 0.0
temp DD 0.0
;
_DATA  ENDS
;
_TEXT SEGMENT BYTE PUBLIC 'CODE'
    ;
    ;     float approx_fderiv (float (*f)(float), float x)
    ;                                [BP+4]           [BP+6]
    ;
        .386
        .387
       PUBLIC _approx_fderiv
_approx_fderiv  PROC    NEAR
        PUSH    BP
        MOV     BP,SP
        FLD DWORD PTR [BP+6]
        FDIV two
        FABS
        FSTP h
    ;
    ;       compute (*f)(x+h) -(*f)(x-h) )/ (2*h);
    ;
        FLD     DWORD PTR [BP+6]
        FADD    h
        FSTP    temp
        PUSH    temp                    ; Push x+h
        CALL    WORD PTR [BP+4]     ; ST(0) = f(x+h)
        ADD     SP,4    ;  Free Parameter
        FLD     DWORD PTR [BP+6]
        FSUB    h
        FSTP    temp
        PUSH    temp                    ; Push x-h
        CALL    WORD PTR [BP+4]     ; ST(0) = f(x-h), ST(1) = f(x+h)
        ADD     SP,4    ;  Free Parameter
        FSUB                    ; ST(0) =  f(x+h)-  f(x-h), ST(1) = Empty
        FLD     h       ; ST(0) = h, ST(1) = f(x+h)-  f(x-h)
        FMUL    two     ; ST(0) = 2h, ST(1) = f(x+h)-  f(x-h)
        FDIV            ; ST(0) = (f(x+h)-  f(x-h))/2h
        FSTP fd0
        PUSH DWORD PTR [BP+6]
        CALL WORD PTR [BP+4]
        ADD SP,4
        FDIV eps_const
        FABS
        FSTP eps
 Do1:
```

```
        FLD h
        FDIV two
        FSTP h
    ;
    ;       compute (*f)(x+h) -(*f)(x-h) )/ (2*h);
    ;
        FLD     DWORD PTR [BP+6]
        FADD    h
        FSTP    temp
        PUSH    temp                ; Push x+h
        CALL    WORD PTR [BP+4]     ; ST(0) = f(x+h)
        ADD     SP,4    ;  Free Parameter
        FLD     DWORD PTR [BP+6]
        FSUB    h
        FSTP    temp
        PUSH    temp                ; Push x-h
        CALL    WORD PTR [BP+4]     ; ST(0) = f(x-h), ST(1) = f(x+h)
        ADD     SP,4    ;  Free Parameter
        FSUB                        ; ST(0) =  f(x+h)-  f(x-h), ST(1) = Empty
        FLD     h       ; ST(0) = h, ST(1) = f(x+h)-  f(x-h)
        FMUL    two     ; ST(0) = 2h, ST(1) = f(x+h)-  f(x-h)
        FDIV            ; ST(0) = (f(x+h)-  f(x-h))/2h
        FLD ST  ; ST(0) = (f(x+h)-  f(x-h))/2h, ST(1) = (f(x+h)- f(x-h))/2h
        FLD fd0
        FSUB    ; ST(0) = current - fd0
        FABS    ;  ST(0) = | current - fd0 |
        FCOMP eps
        FSTSW AX
        SAHF
        FSTP  fd0
        JAE Do1
    ;
        FLD fd0
        MOV     SP,BP
        POP     BP
        RET
_approx_fderiv  ENDP
;
_TEXT ENDS
;
        END
```

```
/* fderiv2b.c - approximate derivative function */

#include <stdio.h>
#include <math.h>


extern double approx_fderiv(double (*f)(double), double x);


double f(double x)
{
  return x*x*x - 2.0*x*x + 3.0*x - 8.0;
} /* f */

double real_fderiv(double x)
{
  return 3.0*x*x - 4.0*x + 3.0;
} /* real_fderiv */


int main()
{
  printf("approx_deriv(5.0) = %lf\n", approx_fderiv(f, 5.0));
  printf("real_fderiv(5.0) = %lf\n", real_fderiv(5.0));

  return 0;
} /* main */
```

```
C:\>tcc fderiv2b.c fd2b.asm
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
fderiv2b.c:
fd2b.asm:
Turbo Assembler  Version 3.1  Copyright (c) 1988, 1992 Borland
International

Assembling file:   fd2b.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  397k

Turbo Link  Version 5.0 Copyright (c) 1992 Borland International

        Available memory 4108576

C:\>fderiv2b.exe
approx_deriv(5.0) = 58.000095
real_fderiv(5.0) = 58.000000

C:\>
```

```
;
; fd2b.asm - implement numerical differentiation
;
;
  ASSUME CS:_TEXT, DS:_DATA
;
  ; Static Variables
;
_DATA SEGMENT DWORD PUBLIC 'DATA'
;
h  DQ ?
two DD 2.0
fd0 DQ 0.0
eps_const DQ 131072.0
eps  DQ 0.0
temp DQ 0.0
;
_DATA  ENDS
;
_TEXT SEGMENT BYTE PUBLIC 'CODE'
    ;
    ;     double approx_fderiv (double (*f)(double), double x)
    ;                                [BP+4]              [BP+6]
    ;
        .386
        .387
       PUBLIC _approx_fderiv
_approx_fderiv  PROC    NEAR
        PUSH    BP
        MOV     BP,SP
        FLD QWORD PTR [BP+6]
        FDIV two
        FABS
        FSTP h
    ;
    ;       compute (*f)(x+h) -(*f)(x-h) )/ (2*h);
    ;
        FLD     QWORD PTR [BP+6]
        FADD    h
        FSTP    temp
        PUSH    DWORD PTR temp+4    ; Push x+h
        PUSH    DWORD PTR temp
        CALL    WORD PTR [BP+4]     ; ST(0) = f(x+h)
        ADD     SP,8     ;  Free Parameter
        FLD     QWORD PTR [BP+6]
        FSUB    h
        FSTP    temp
        PUSH    DWORD PTR temp+4; Push x-h
        PUSH    DWORD PTR temp
        CALL    WORD PTR [BP+4]     ; ST(0) = f(x-h), ST(1) = f(x+h)
        ADD     SP,8    ;  Free Parameter
        FSUB                    ; ST(0) =  f(x+h)-  f(x-h), ST(1) = Empty
        FLD     h       ; ST(0) = h, ST(1) = f(x+h)-  f(x-h)
        FMUL    two     ; ST(0) = 2h, ST(1) = f(x+h)-  f(x-h)
        FDIV            ; ST(0) = (f(x+h)-  f(x-h))/2h
        FSTP fd0
```

```
        PUSH DWORD PTR [BP+10]
        PUSH DWORD PTR [BP+6]
        CALL WORD PTR [BP+4]
        ADD SP,8
        FDIV eps_const
        FABS
        FSTP eps
Do1:

        FLD h
        FDIV two
        FSTP h
   ;
   ;       compute (*f)(x+h) -(*f)(x-h) )/ (2*h);
   ;
        FLD     QWORD PTR [BP+6]
        FADD    h
        FSTP    temp
        PUSH    DWORD PTR temp+4    ; Push x+h
        PUSH    DWORD PTR temp
        CALL    WORD PTR [BP+4]     ; ST(0) = f(x+h)
        ADD     SP,8     ;  Free Parameter
        FLD     QWORD PTR [BP+6]
        FSUB    h
        FSTP    temp
        PUSH    DWORD PTR temp+4; Push x-h
        PUSH    DWORD PTR temp
        CALL    WORD PTR [BP+4]     ; ST(0) = f(x-h), ST(1) = f(x+h)
        ADD     SP,8    ;  Free Parameter
        FSUB                    ; ST(0) =  f(x+h)-  f(x-h), ST(1) = Empty
        FLD     h       ; ST(0) = h, ST(1) = f(x+h)-  f(x-h)
        FMUL    two     ; ST(0) = 2h, ST(1) = f(x+h)-  f(x-h)
        FDIV            ; ST(0) = (f(x+h)-  f(x-h))/2h
        FLD ST  ; ST(0) = (f(x+h)-  f(x-h))/2h, ST(1) = (f(x+h)- f(x-h))/2h
        FLD fd0
        FSUB    ; ST(0) = current - fd0
        FABS    ;  ST(0) = | current - fd0 |
        FCOMP eps
        FSTSW AX
        SAHF
        FSTP  fd0
        JAE Do1
   ;
        FLD fd0
        MOV     SP,BP
        POP     BP
        RET
_approx_fderiv  ENDP
;
_TEXT ENDS
;
         END
```

```
/* fderiv2c.c - approximate derivative function */

#include <stdio.h>
#include <math.h>


extern long double approx_fderiv(long double (*f)(long double), long double x);


long double f(long double x)
{
  return x*x*x - 2.0*x*x + 3.0*x - 8.0;
} /* f */

long double real_fderiv(long double x)
{
  return 3.0*x*x - 4.0*x + 3.0;
} /* real_fderiv */


int main()
{
  printf("approx_deriv(5.0) = %Lf\n", approx_fderiv(f, 5.0));
  printf("real_fderiv(5.0) = %Lf\n", real_fderiv(5.0));

  return 0;
} /* main */
```

```
C:\>tcc fderiv2c.c fd2c.asm
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
fderiv2c.c:
fd2c.asm:
Turbo Assembler  Version 3.1  Copyright (c) 1988, 1992 Borland
International

Assembling file:   fd2c.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  397k

Turbo Link  Version 5.0 Copyright (c) 1992 Borland International

        Available memory 4108576

C:\>fderiv2c.exe
approx_deriv(5.0) = 58.000006
real_fderiv(5.0) = 58.000000

C:\>
```

```
;
; fd2c.asm - implement numerical differentiation
;
;
;
  ASSUME CS:_TEXT, DS:_DATA
;
  ; Static Variables
_DATA SEGMENT DWORD PUBLIC 'DATA'
;
h  DT ?
two DD 2.0
fd0 DT 0.0
eps_const DT 2097152.0
eps  DT 0.0
temp DT 0.0
;
_DATA  ENDS
;
_TEXT SEGMENT BYTE PUBLIC 'CODE'
    ;
    ; long double approx_fderiv (long double (*f)(double), long  double x)
    ;                                    [BP+4]                    [BP+6]
    ;
         .386
         .387
        PUBLIC _approx_fderiv
_approx_fderiv  PROC    NEAR
         PUSH    BP
         MOV     BP,SP
         FLD  TBYTE PTR [BP+6]
         FDIV two
         FABS
         FSTP h
    ;
    ;       compute (*f)(x+h) -(*f)(x-h) )/ (2*h);
    ;
         FLD     TBYTE PTR [BP+6]
         FLD     h
         FADD
         FSTP    temp
         PUSH    DWORD PTR temp+6     ; Push x+h
         PUSH    DWORD PTR temp+2
         PUSH    WORD PTR temp
 ;
         CALL    WORD PTR [BP+4]     ; ST(0) = f(x+h)
         ADD     SP,10     ;  Free Parameter
         FLD     TBYTE PTR [BP+6]
         FLD     h
         FSUB
         FSTP    temp
         PUSH    DWORD PTR temp+6     ; Push x-h
         PUSH    DWORD PTR temp+2
         PUSH    WORD PTR temp
 ;
         CALL    WORD PTR [BP+4]     ; ST(0) = f(x-h), ST(1) = f(x+h)
         ADD     SP,10    ;  Free Parameter
         FSUB                    ; ST(0) =  f(x+h)-  f(x-h), ST(1) = Empty
         FLD     h       ; ST(0) = h, ST(1) = f(x+h)-  f(x-h)
         FMUL    two     ; ST(0) = 2h, ST(1) = f(x+h)-  f(x-h)
         FDIV            ; ST(0) = (f(x+h)-  f(x-h))/2h
         FSTP fd0
```

```
        PUSH DWORD PTR [BP+12]
        PUSH DWORD PTR [BP+8]
        PUSH WORD PTR [BP+6]
        CALL WORD PTR [BP+4]
        ADD SP,10
        FLD eps_const
        FDIV
        FABS
        FSTP eps

Do1:
        FLD h
        FDIV two
        FSTP h
    ;
    ;       compute (*f)(x+h) -(*f)(x-h) )/ (2*h);
    ;
        FLD     TBYTE PTR [BP+6]
        FLD     h
        FADD
        FSTP    temp
        PUSH    DWORD PTR temp+6     ; Push x-h
        PUSH    DWORD PTR temp+2
        PUSH    WORD PTR temp
        CALL    WORD PTR [BP+4]     ; ST(0) = f(x+h)
        ADD     SP,10     ;  Free Parameter
        FLD     TBYTE PTR [BP+6]
        FLD     h
        FSUB
        FSTP    temp
        PUSH    DWORD PTR temp+6     ; Push x-h
        PUSH    DWORD PTR temp+2
        PUSH    WORD PTR temp
        CALL    WORD PTR [BP+4]     ; ST(0) = f(x-h), ST(1) = f(x+h)
        ADD     SP,10    ;  Free Parameter
        FSUB                    ; ST(0) =  f(x+h)-  f(x-h), ST(1) = Empty
        FLD     h       ; ST(0) = h, ST(1) = f(x+h)-  f(x-h)
        FMUL    two     ; ST(0) = 2h, ST(1) = f(x+h)-  f(x-h)
        FDIV            ; ST(0) = (f(x+h)-  f(x-h))/2h
        FLD ST  ; ST(0) = (f(x+h)-  f(x-h))/2h, ST(1) = (f(x+h)- f(x-h))/2h
        FLD fd0
        FSUB    ; ST(0) = current - fd0
        FABS    ;  ST(0) = | current - fd0 |
        FLD   eps
        FCOMPP
        FSTSW AX
        SAHF
        FSTP  fd0
        JNAE Do1  ; Reversed logic
    ;
        FLD fd0
        MOV     SP,BP
        POP     BP
        RET
_approx_fderiv  ENDP
;
_TEXT ENDS
;
        END
```