フェルミ・ディラック積分の数値積分

2016.7.1　　鈴木　実

1　はじめに

フェルミ・ディラック分布関数を含む関数の積分は特別な場合を除いて解析的に求めることができない．フェルミ・エネルギーの絶対値が十分大きい場合には収束の速い近似式が利用できるが，十分大きくない場合，数値計算が必要である．この数値計算の精度を高くするために多くの論文が書かれている．その中で，Cloutman [1] は 11 桁の精度をもつフェルミ・ディラック積分の数値計算法を報告している．ここでは，の数値計算法をまとめ，そこで示された FORTRAN のプログラムを C に変換しておこう．

2　フェルミ・ディラック積分

3 次元フェルミ気体の濃度 n を計算すると，状態密度が $\mathscr{E}^{1/2}$ に比例するので，フェルミ・ディラック分布関数を $f(\mathscr{E})$ とすると，

$$\begin{aligned} n &= \frac{1}{2\pi^2}\left(\frac{2m}{\hbar^2}\right)^{3/2}\int_0^\infty \mathscr{E}^{1/2} f(\mathscr{E})\mathrm{d}\mathscr{E} \\ &= \frac{1}{2\pi^2}\left(\frac{2m}{\hbar^2}\right)^{3/2}\int_0^\infty \frac{\mathscr{E}^{1/2}}{1+\mathrm{e}^{(\mathscr{E}-\mu)/k_\mathrm{B}T}}\mathrm{d}\mathscr{E} \qquad (1) \\ &= \frac{1}{2\pi^2}\left(\frac{2mk_\mathrm{B}T}{\hbar^2}\right)^{3/2}\int_0^\infty \frac{x^{1/2}}{1+\mathrm{e}^{x-\eta}}\mathrm{d}x \qquad (2) \end{aligned}$$

となる．ただし，$x = \mathscr{E}/k_\mathrm{B}T$，$\eta = \mu/k_\mathrm{B}T$ である．

フェルミ気体のエネルギーを計算する場合には式 (1) で $\mathscr{E}^{1/2}$ を $\mathscr{E}^{3/2}$ に変えればよい．このように，フェルミ・ディラック分布関数を含む計算には次のような積分が含まれる．

$$F_n(\eta) = \int_0^\infty \frac{x^n}{1+\mathrm{e}^{x-\eta}}\mathrm{d}x \qquad (3)$$

この積分を n 次のフェルミ・ディラック積分 (the Fermi-Dirac integral of order n) という[1]．フェルミ・ディラック積分は，η が負で絶対値が十分大きい場合，フェルミ・ディラック分布関数は指数関数に漸近するためにガンマ関数に漸近する．η が正で絶対値が十分大きい場合には，ゾンマーフェルトの展開公式が使用できる．中間の場合には以下に述べるような数値計算が必要になる．Cloutman [1] は 11 桁の精度を達成するために，変数変換による特異点解消，B 変換および G 変換，Aitken 外挿などの手法を用いている．

Cloutman の数値計算プログラムは，フェルミ・ディラック積分の数表を作成し，数表にない η の値に関してはエルミート補間法 [3, 4] を用いている．エルミート補間では各点における関数値と微分関数値が必要となるが，フェルミ・ディラック積分では，微分は次数の異なるフェルミ・ディラック積分で表される．実際，式 (3) において，η で微分した後で，x に関して部分積分することにより，

$$F_n'(\eta) = nF_{n-1}(\eta) \qquad (4)$$

という漸化式が成立つ．

[1] フェルミ・ディラック積分の定義にはこの他に

$$\mathscr{F}_n = \frac{1}{\Gamma(n+1)}\int_0^\infty \frac{x^n}{1+\mathrm{e}^{x-\eta}}\mathrm{d}x$$

という定義がある．n が負の整数の場合でも有限の値をもち，微分した場合の漸近関係が簡単になるなどの特徴がある．

3　フェルミ・ディラック積分の数値計算

3.1　変数変換

ここでは $n > -1$ の場合を考える．$n = -1/2$ の場合，被積分関数は $x = 0$ が特異点となる．数値積分では，$x = z^2$ とおけばよい．すなわち，

$$F_n(\eta) = 2\int_0^\infty \frac{z^{2n+1}}{1+\mathrm{e}^{z^2-\eta}}\mathrm{d}z \tag{5}$$

である．

3.2　G 変換と B 変換

上限が ∞ の場合の定積分の数値計算を精度よく求める手法として，G 変換 (G transform) と B 変換 (B transform) がある (Gray and Atchison) [2]．いま，

$$S(\eta) = \int_a^\infty f(\eta, x)\mathrm{d}x \tag{6}$$

$$S_t(\eta) = \int_a^t f(\eta, x)\mathrm{d}x \tag{7}$$

とすると，

$$S(\eta) = \lim_{t\to\infty} S_t(\eta) \tag{8}$$

である．

このとき，G 変換および B 変換は次のように定義される．

$$G[S(\eta); t, k] = \frac{S_{t+k}(\eta) - R_t(\eta, k)S_t(\eta)}{1 - R_t(\eta, k)}, \quad R_t \neq 1 \tag{9}$$

$$R_t(\eta, k) = \frac{f(\eta, t+k)}{f(\eta, t)} \quad k > 0 \tag{10}$$

$$B[S(\eta); t, k] = \frac{S_{tk}(\eta) - \rho_t(\eta, k)S_t(\eta)}{1 - \rho_t(\eta, k)}, \quad \rho_t \neq 1 \tag{11}$$

$$\rho_t(\eta, k) = \frac{kf(\eta, kt)}{f(\eta, t)} \quad k > 1 \tag{12}$$

ここで，

$$\lim_{t\to\infty} G[S(\eta); t, k] = \lim_{t\to\infty} B[S(\eta); t, k] = S(\eta) \tag{13}$$

が成り立つ．

ここで Gray と Atchison が示したことは，$\lim_{t\to\infty} R_t(\eta, k) \neq 0, 1$ ならば，G は $S_{t+k}(\eta)$ よりも速く収束する，ということである．

B 変換についても同様のことが成り立つ．

さらに，$f = \mathrm{e}^{-x}$ のとき，S_t が正しく求められれば，$G[S(\eta); t, k] = S$ である．$f = x^{-s}$ ならば，$B[S(\eta); t, k] = S$ である．

3.3 Aitken 外挿

積分範囲が有限で，分割数を μ，2μ，4μ，分割幅を H，$H/2$，$H/4$，それぞれの数値積分結果を I_μ，$I_{2\mu}$，$I_{4\mu}$，I を正しい積分値とする．そうすると，

$$I = I_\mu + cH^p \tag{14}$$

$$I = I_{2\mu} + c(H/2)^p \tag{15}$$

$$I = I_{4\mu} + c(H/4)^p \tag{16}$$

が成り立つ．これから，

$$I = I_{4\mu} - \frac{(I_{4\mu} - I_{2\mu})^2}{I_{4\mu} - 2I_{2\mu} + I_\mu} \tag{17}$$

$$p = \frac{1}{\log_{10}(2)} \log_{10}\left[\frac{I - I_\mu}{I - I_{2\mu}}\right] \tag{18}$$

$$c = \frac{I - I_\mu}{H^p} \tag{19}$$

という関係式が得られる．

3.4 $\eta < 0$ の場合

$\eta \leq 0$ の場合，次の近似式が成立つ (Cox and Giuli, 1968)．

$$F_n(\eta) = \Gamma(n+1)\mathrm{e}^\eta \sum_{r=0}^{\infty} (-1)^r \frac{\mathrm{e}^{r\eta}}{(r+1)^{n+1}}, \quad n > -1 \tag{20}$$

$n = 1/2$，$n = 3/2$，および $n = 5/2$ の場合には，

$$F_{1/2}(\eta) = \frac{\pi^{1/2}}{2} \sum_{j=1}^{\infty} (-1)^{j+1} \frac{\mathrm{e}^{j\eta}}{j^{3/2}}, \tag{21}$$

$$F_{3/2}(\eta) = \frac{3\pi^{1/2}}{4} \sum_{j=1}^{\infty} (-1)^{j+1} \frac{\mathrm{e}^{j\eta}}{j^{5/2}} \tag{22}$$

$$F_{5/2}(\eta) = \frac{15\pi^{1/2}}{8} \sum_{j=1}^{\infty} (-1)^{j+1} \frac{\mathrm{e}^{j\eta}}{j^{7/2}} \tag{23}$$

である．

3.5 $\eta > 25$ の場合

この場合はゾンマーフェルト展開が用いられる．

$$I(\eta) = \int_0^\infty \frac{\phi'(u)\mathrm{d}u}{\mathrm{e}^{u-\eta} + 1} \tag{24}$$

に対して，次の近似式が成立つ．

$$I(\eta) \simeq \phi(\eta) + 2\sum_{j=1}^{\infty} C_{2j}\phi^{2j}(\eta) \tag{25}$$

ただし，

$$C_2 = \frac{\pi^2}{12}, \quad C_4 = \frac{7\pi^4}{720}, \quad C_6 = \frac{31\pi^6}{30240}, \quad C_8 = \frac{127\pi^8}{1209600}, \quad C_{10} = \frac{511\pi^{10}}{47900160} \tag{26}$$

である．

式 (3) に応用すると，

$$F_n(\eta) = \frac{\eta^{n+1}}{n+1}\left[1 + \sum_{r=1}^{\infty} 2C_{2r}\left(\prod_{k=n-2r+2}^{n+1} k\right)\eta^{-2r}\right], \quad n > 0, \eta \gg 1 \tag{27}$$

となる．

$n = 1/2$，$n = 3/2$，$n = 5/2$ については，

$$F_{1/2}(\eta) \simeq \frac{2}{3}\eta^{3/2} + \frac{\pi^2}{12}\eta^{-1/2} + \frac{7\pi^4}{960}\eta^{-5/2} + \frac{31\pi^6}{4608}\eta^{-9/2} + \frac{1397\pi^8}{81920}\eta^{-13/2} \tag{28}$$

$$F_{3/2}(\eta) \simeq \frac{2}{5}\eta^{5/2} + \frac{\pi^2}{4}\eta^{1/2} - \frac{7\pi^4}{960}\eta^{-3/2} - \frac{31\pi^6}{10752}\eta^{-7/2} - \frac{381\pi^8}{81920}\eta^{-11/2} \tag{29}$$

$$F_{5/2}(\eta) \simeq \frac{2}{7}\eta^{7/2} + \frac{5\pi^2}{12}\eta^{3/2} + \frac{7\pi^4}{192}\eta^{-1/2} + \frac{31\pi^6}{10752}\eta^{-5/2} + \frac{127\pi^8}{49152}\eta^{-9/2} \tag{30}$$

のようになる．

参考文献

[1] L. D. Cloutman, “Numerical evaluation of the Fermi-Dirac integrals”, Astrophys. J. Suppl. Ser. **71**, 677-699 (1989).

[2] H. L. Gray and T. A. Atchison, “Applications of the G and B transforms to the Laplace transform”, Proceedings of the 1968 23rd ACM national conference, pp.73-77, (1968). （この論文の pdf は有料，サイトは下記）

`http://dl.acm.org/citation.cfm?id=810568&dl=ACM&coll=DL&CFID=808499028&CFTOKEN=95593607`

[3] 「5 次エルミート補間多項式」(2016/6/30 のエントリー).

[4] 「エルミート補間多項式」(2016/6/29 のエントリー).

[5] 「フェルミ・ディラック積分数表を作成するプログラム」(2016/7/4 のエントリー).

プログラムソース

以下のプログラムは原論文 [1] では FORTRAN で書かれている．実行させると，$n = 1/2$ の場合のフェルミ・ディラック積分結果が出力される．論文には $n = -1/2, 1/2, 3/2, 5/2$，$-5 \leq \eta \leq 25$ の数表が掲載されているが，論文に掲載されているプログラムは $n = 1/2$ の場合のみの計算である．実際の数表を作成するプログラムはこの次のエントリー [5] で掲載する．

1. フェルミ・ディラック積分の数値計算プログラム

```
/* fermi_dirac_inetgral.c  -o  fermi_dirac_integral                    */
/* translated from FDTAB written in the FORTRAN language to C.        */
/* 2016.3.23 by M. Suzuki                                             */

/* This program is based on the programme by Lawrence D. Cloutman,    */
/* described in his original paper, "Numerical Evaluation of the      */
```

```
/* Fermi-Dirac Integrals",The Astrophysical Journal of Supplement                */
/* Series vol.71, pp.677-699 (1989).  The paper is available at                  */
/* http://adsabs.harvard.edu/abs/1989ApJS...71..677C                             */

/* PROGRAM FDTAB                                                                 */
/* COMPUTE ACCURATE TABLES OF FERMI-DIRAC INTEGRALS                              */
/* USING SIMPSON'S RULE WITH EXTRAPOLATION TECHNIQUES                            */
/*                                                                               */

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

/*                                                                               */
/* Evaluate the integrand at (x, eta)                                            */
double fint(double x, double eta)
{
    double z;
    z=2*x*x/(exp(x*x-eta)+1);
    return z;
}


int main(int argc, char *argv[])
{
    FILE *fp;
    char filenameout[200], dummy[200];
    int m2, m4, mpts, mptsav, n, ne, nfail, ngb, nm, itrap, N;
    int    i, ii, j;
    double a, b, bad, bgt, binc, bk, bkt, bratio, bsave;
    double bxfrm, cextr, de, denom, digits, eta, eta0, extrap;
    double fmax, fsum, gk, gkt, gxfrm, h, pextr;
    double rgb, rob, rog, xmax;
    double relerr, seta, sgxfrm;
    double x[2000], f[2000], value[20];

    /* INITIAL DATA */
    /* (a, b) are the starting values fo the integration limits (0, t)           */
    a=0.0;
    b=1.0;

    /* binc is the increment in b used by the adaptive upper                     */
    /* integration limit routine, bratio is the maximum value                    */
    /*  of f(b)/max(f),where the function f is the integrand.                    */
    binc=0.5;
    bratio=1.0e-6;

    /* Incrementing factors for the B and G transforms                           */
    gk=1.0;
    bk=1.1;

    /* mpts is the number of mesh points to which Simpson's rule                 */
    /* is applied for the coarsest mesh, must be an odd integer                  */
    mpts=201;

    /* eta0 is the first value of degeneracy parameter, de is the                */
```

```
/* increment in eta, and ne is the number of eta values                    */
eta0=-5.0;
de=0.05;
ne=601;

/* nfail is a diagnostic that counts extrapolation failures                */
nfail=0;

/* itrap=1 for trapezoidal rule,                                           */
/* otherwise Simpson's rule integration                                    */
itrap=0;

/* make sure mpts is even for Simpson's rule                               */
if((mpts % 2)!=1 && itrap==0)
{
    printf("%d should be even\n", mpts);
    exit(0);
}

mptsav=mpts;
bsave=b;


/*                                                                         */
/* Outer loop is over all values of eta                                    */
for(n=0;n<ne;n++)
{
    eta=eta0+de*n;
    bad=bsave;
    printf("\neta=%lf\n", eta);

/* This loop does the three integrations required for the                  */
/* B and G transforms, after which the transforms are computed.            */
    for(ngb=0;ngb<3;ngb++)
    {
/*                                                                         */
/* Set appropriate upper integration limit                                 */
        b=bad;
        if(ngb==1) b=bad+gk;
        if(ngb==2) b=bad*bk;

/* This loop increments the number of mesh points to do the                */
/* three integrations required for each Aitken extrapolation.              */
        for(nm=0;nm<3;nm++)
        {
            while(1)
            {
                m2=(mpts-3)/2;
                m4=m2+1;
                h=(b-a)/(mpts-1);

                fmax=-1.0e100;

                for(j=0;j<mpts;j++)
                {
                    x[j]=a+h*j;
                    f[j]=fint(x[j], eta);
```

```
                    if(f[j] >= fmax)
                    {
                        xmax=x[j];
                        fmax=f[j];
                    }
                }
                if((ngb != 0) || (nm != 0) || (f[mpts-1] < bratio*fmax) || (b > 100.0)) break;

                b+=binc;
                bad=b;
            }
            if(nm==0)
            printf("fmax %le at %lf, end point x=%lf f=%le\n", fmax,xmax,x[mpts-1],f[mpts-1]);

            if(itrap==1)
/*                                                                              */
/* Use trapezoidal rule integrations                                            */
            {
                fsum=0.0;
                for(i=0;i<mpts;i++)
                {
                    fsum+=f[i];
                }
                value[nm]=h*(fsum-0.5*(f[0]+f[mpts-1]));
            }
            else
/*                                                                              */
/* Integrate the f array with Simpson's rule                                    */
            {
                fsum=0.0;
                for(ii=0;ii<m2;ii++)
                {
                    i=ii;        // j descending order
                    i=m2-ii+1;    // j ascending order

                    j=m4-i;
                    fsum+=2*f[2*j+1]+f[2*j+2];
                }
                value[nm]=h*(2*fsum+f[0]+f[mpts-1]+4*f[mpts-2])/3;
            }
            printf("Integral=%20.14le for mpts=%d\n", value[nm], mpts);
            printf("%d\t%le\t%le\t%le\n", nm, value[0], value[1], value[2]);

            if(nm < 2)    mpts=2*(mpts-1)+1;
        }

/*                                                                              */
/* Aitken extrapolation                                                         */
/*                                                                              */
        denom=value[2]+value[0]-2*value[1];
        if(denom != 0.0)
        {
            extrap=value[2]-pow((value[2]-value[1]),2)/denom;
        }
        else
        {
```

```
            extrap=value[2];
            nfail++;
            printf("Cannot perform Aitken extrapolation\n");
        }

        if(extrap-value[1] != 0.0)
        {
            denom=(extrap-value[0])/(extrap-value[1]);
        }
        else
        {
            denom=0.0;
        }

        if(denom > 0.0)
        {
            pextr=log10(denom)/log10(2.0);
            h=(b-a)/(mptsav-1);
            cextr=(extrap-value[0])/pow(h, pextr);
        }
        else
        {
            nfail++;
            pextr=-99999999.;
            cextr=pextr;
            printf("Cannot compute Aitken parameters p and c\n ");
            printf("extrap=%le, value[1,2]=%le, %le\n", extrap, value[0], value[1]);
        }

        relerr=fabs((extrap-value[2])/extrap);
        if(relerr > 0.0) digits=-log10(relerr);
        else digits=-30.0;
        printf("Extrapolated integral = %le, ", extrap);
        printf("p=%le, c=%le, h=%le, digits=%lf\n", pextr, cextr, h, digits);

        if(ngb==0)
        {
            bgt=extrap;
            rgb=f[mpts-1];
        }
        if(ngb==1)
        {
            gkt=extrap;
            rog=f[mpts-1];
        }
        if(ngb==2)
        {
            bkt=extrap;
            rob=f[mpts-1];
        }
        mpts=mptsav;
    }
/*                                                                      */
/* Calculate the B and G transforms                                     */
/*                                                                      */
    rog=rog/rgb;
    rob=bk*rob/rgb;
```

```
        printf("%le\t%le\t%20.12le\n", rog, rob, f[mpts-1]);
    /*                                                                          */
    /* Limited error checking                                                   */
        if(rob <= 0.0 || rog <=0.0)
        {
            printf("eta=%le, rob=%le, rog=%le, Cannot do B and G transforms", eta, rob, rog);
            exit(0);
        }

        gxfrm=(gkt-rog*bgt)/(1.0-rog);
        bxfrm=(bkt-rob*bgt)/(1.0-rob);
        printf("G transform=%le, B transform=%le\n", gxfrm, bxfrm);

        relerr=fabs((gxfrm-gkt)/gxfrm);
        if(relerr > 0.0) digits=-log10(relerr);
        else digits=-30;
        seta=eta;
        sgxfrm=gxfrm;
        printf("%lf\t%18.11le\n", seta, sgxfrm);
        printf("%lf\t%le, %lf digits G xfrm\n", seta, sgxfrm, digits);
    }
}
```

2. フェルミ・ディラック積分数表からエルミート補間法で数値を得るプログラム

```
/* FDSET.c  -o  FDSET                                                       */
/* translated from FDSET written in the FORTRAN language to C.              */
/* 2016.7.2 by M. Suzuki                                                    */

/* This program is based on the programm written by Lawrence D. Cloutman,   */
/* described in his original paper, "Numerical Evaluation of the            */
/* Fermi-Dirac Integrals",The Astrophysical Journal of Supplement           */
/* Series vol.71, pp.677-699 (1989).  The paper is available at             */
/* http://adsabs.harvard.edu/abs/1989ApJS...71..677C                        */

/* SUBROUTINE FDSET                                                         */
/* THIS SUBROUTINE INITIALIZES ARRAYS NEEDED BY FUNCTION FD.                */
/* FDSET IS CALLED ONCE (AND ONLY ONCE) BEFORE CALLING FD.                  */
/*                                                                          */

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

static double fdtab;
static double F[601][4], ahi[5][3], alo[5][3];

/*                                                                          */
/* SET UP F-D INTEGRAL ASSYMPTOTIC EXPANSION COEFFICIENTS                   */
/*                                                                          */
int FDSET(void)
{
    FILE *fp;

    char filenamein[100];
    int i;
```

```
    double a, pi, e0, f0, f1, f2, f3;

    pi=M_PI;
    strcpy(filenamein, "FD_table.txt");

    a=sqrt(pi)/2;
    alo[0][0]=a;
    alo[1][0]=-a/pow(sqrt(2),3);
    alo[2][0]=a/pow(sqrt(3),3);
    alo[3][0]=-a/8;
    alo[4][0]=a/pow(sqrt(5),3);
    a=3*sqrt(pi)/4;
    alo[0][1]=a;
    alo[1][1]=-a/pow(sqrt(2),5);
    alo[2][1]=a/pow(sqrt(3),5);
    alo[3][1]=-a/32;
    alo[4][1]=a/pow(sqrt(5),5);
    a=15*sqrt(pi)/8;
    alo[0][2]=a;
    alo[1][2]=-a/pow(sqrt(2),7);
    alo[2][2]=a/pow(sqrt(3),7);
    alo[3][2]=-a/128;
    alo[4][2]=a/pow(sqrt(5),7);
    ahi[0][0]=2/3;
    ahi[1][0]=pow(pi, 2)/12;
    ahi[2][0]=pow(pi, 4)*7/960;
    ahi[3][0]=pow(pi, 6)*31/4608;
    ahi[4][0]=pow(pi, 8)*1397/81920;
    ahi[0][1]=0.4;
    ahi[1][1]=pow(pi, 2)/4;
    ahi[2][1]=-ahi[2][0];
    ahi[3][1]=-pow(pi, 6)*31/10752;
    ahi[4][1]=-pow(pi, 8)*381/81920;
    ahi[0][2]=2/7;
    ahi[1][2]=pow(pi, 2)*5/12;
    ahi[2][2]=pow(pi, 4)*7/192;
    ahi[3][2]=pow(pi, 6)*31/10752;
    ahi[4][2]=pow(pi, 8)*127/49152;
/*                                                                              */
/* READ IN FERMI-DIRAC INTEGRAL TABLES GIVEN IN TABLE 5                         */
/*                                                                              */
    if((fp=fopen(filenamein, "r"))==0) exit(0);
    i=0;
    while(i<601)
    {
        fscanf(fp, "%lf\t%le\t%le\t%le\t%le\n", &e0,&f0,&f1,&f2,&f3);
        F[i][0]=f0;
        F[i][1]=f1;
        F[i][2]=f2;
        F[i][3]=f3;
        i++;
    }
    fclose(fp);
    return 0;
}
```

```
/*                                                                              */
/* COMPUTES THE FERMI-DIRAC INTEGRAL FD FOR DEGENERACY                          */
/* PARAMETER ETA.  n=1 FOR ORDER 1/2; n=2 FOR ORDER 3/2; n=3 FOR ORDER 3/2.     */
/*                                                                              */
double FD(double eta, int n)
{
    int j, k, kk, l;
    double u, z;
    double x0, x1, x2, y0, y1, y2, s0, s1, s2;
    double HERMITE5();

    if(eta >= -5 && eta <= 25)

/*                                                                              */
/* FIFTH ORDER HERMITE INTERPOLATION FOR INTERMEDIATE VALUES OF ETA.            */
/*                                                                              */
    j=(eta+5)*20;
    if(j < 1) j=1;
    if(j > 599) j=599;
    x0=-5.0+0.05*(j-1);
    x1=x0+0.05;
    x2=x0+0.1;

    y0=F[j-1][n];
    y1=F[j][n];
    y2=F[j+1][n];

    s0=F[j-1][n-1]*(n-0.5);
    s1=F[j][n-1]*(n-0.5);
    s2=F[j+1][n-1]*(n-0.5);

    z=HERMITE5(eta, x0, y0, s0, x1, y1, s1, x2, y2, s2);
    return z;

    if(eta <-5)
    {
        z=0;
        for(k=0;k<5;k++)
        {
            kk=k+1;
            z+=alo[k][n]*exp(k*eta);
        }
        return z;
    }

    if(eta >25)
    {
        u=sqrt(eta);
        l=1+2*n;
        z=0;
        for(k=0;k<5;k++)
        {
            z+=ahi[k][n]*pow(u,l-4*k);
        }
        return z;
    }
}
```

```
/*                                                                          */
/* FIFTH ORDER HERMITE INTERPOLATION                                        */
/*                                                                          */
/* x = value of the independent variable where the function value           */
/* Hermite5 is desired.                                                     */
/* x0, x1, x2 = values of the independent variable at the three             */
/* interpolation nodes.                                                     */
/* p0, p1, p2 = function values at the interpolation nodes.                 */
/* dp0, dp1, dp2 = first derivatives of the function                        */
/* at the interpolation nodes.                                              */
/*                                                                          */

double HERMITE5(double x, double x0, double p0, double dp0, double x1,
                double p1, double dp1, double x2, double p2, double dp2)
{
    double xp, h;
    double a0, a1, a2, a3, a4, a5;
    h=x1-x0;
    xp=x-x1;
    a0=p1;
    a1=dp1;
    a4=(h*(dp2-dp0)-2*(p0+p2-p1-p1));
    a2=((p0+p2-p1-p1)*-a4)/(4*h*h);
    a4*=0.25/pow(h, 4);
    a5=0.25*(h*(dp2+4*dp1+dp0)-3*(p2-p0));
    a3=(0.5*(p2-p0)-h*dp1-a5)/pow(h, 3);
    a5/=pow(h, 5);
    return ((((a5*xp+a4)*xp+a3)*xp+a2)*xp+a1)*xp+a0;
}


int main(int argc, char *argv[])
{
    FILE *fp;
    int i, n;
    double eta, z;

    if(argc<3)
    {
        printf("Usage; a.out eta n\n");
        exit(0);
    }
    i=FDSET();

    eta=atof(argv[1]);
    n=atoi(argv[2]);
    z=FD(eta, n);
    printf("%8.5lf\t%2d\t%19.11le\n", eta, n, z);
}
```