Глава восьмая

ПРИМЕРЫ ПРОГРАММ ПМКС

8.1. Моделирование процесса инееобразования

```
//    tsnowm.cpp                    //
//    Test for subprogram snowm.cpp //
#include <iostream.h>
#include <stdio.h>
#include "pscs.hpp"
void main()
   {
      FILE *put ;
      int i = 2;
      double a[6], b[5];
      int jn[2];
      double tb=20., tm=18.5, p=1.;
      put = fopen("ressnow", "w");
      a[0] = fix(tb, tm, p, i);
      a[1] = 293.;
      a[2] = 200.;
      a[3] = 1.;
      a[4] = 100.;
      a[5] = 6.;
      jn[0] = 1;
      jn[1] = 1;
      snowm(put, a, jn, b);
      i = fclose(put);
      cout << "i = " << i;
   }
```

Подпрограмма snowm предназначена для расчета процесса образования инея в составе программ моделирования. Вычисляются: толщина инея, образовавшегося на поверхности испарителя, его теплофизические характеристики, распределение температур по толщине слоя и коэффициенты влаговыпадения.

Входные параметры

FILE *put - указатель на файл для печати результатов;

double a[6]-массив исходных данных, элементы которого:

0 - влагосодержание воздуха, кг/кг;
1 - температура воздуха, К;
2 - температура поверхности испарителя, К;
3 - время работы испарителя, час;

4 - коэффициент теплоотдачи к наружной поверхности, свободной от инея, Вт/(м^{2}К);

5 - скорость движения воздуха, м/с.

Управляющие параметры

int jn[2] - массив управляющих параметров, где:

0 - признак для печати массивов a и b;

1 - признак для печати распределения: плотности, теплопроводности и темпе-ратуры по толщине слоя инея.

Выходные параметры

double b[5] - массив результатов вычислений, где:

0 - общая толщина слоя инея, м;

1 - средняя плотность инея, кг/(дм^3);

2 - средняя теплопроводность инея, Вт/(м К);

3 - температура на поверхности инея, К;

4 - коэффициент влаговыпадения.

Используемые внешние подпрограммы fp2, fh2, fix.

```
//    snowm.cpp                     //
//    Accumulated frost simulation //
#include <math.h>
#include <stdio.h>
#include "pscs.hpp"
void snowm(FILE *put, double a[ ], int jn[ ], double b[ ])
   {
   int i, j, jj, in, x2 = 2, j1;
   double h, db, cp, pw, ti, al, xx, p21, p22,
       e[50], w[50], te[50], rr[50], bb[50], c[50],
       g[50], d[50], l[50], kk[50], m[50], q[50],
       dm[50], t[50], dt[50], r[50], pt[50],
       n1 = 0.6596, n2 = -1.0526, n3 = -0.474114, f1, f2, f3;
   for (j = 0; j < 50; j++)
       dm[j] = t[j] = dt[j] = r[j] = pt[j] = 0e0;
   if (a[3] < 1e-6 || a[2] >= 273.)
      {
      for (i = 0; i < 5; i++)
       b[i] = 0e0;
      b[3] = a[2];
      goto fin;
      }
   if (jn[0] == 1)
      {
      fprintf(put, "    Simulation with snowm:\n");
      fprintf(put,
```

```
      "  x = %e  t = %e  tw = %e  tau = %e  alpha = %e  w = %e\n",
          a[0], a[1], a[2], a[3], a[4], a[5]);
      fprintf(put, "\n");
      }
   a[2] -= 273.;
   a[1] -= 273.;
   pw = 1.03;
   ti = 0e0;
   for (i = 0; i < 5; i++)
       b[i] = 0e0;
   h = 2e-4 * (1. + 0.2 * a[3]) * (1. - a[2] / 25);
   db = a[0];
   t[0] = a[2];
   cp = 1.006 + 1.887 * db;
   al = a[4];
   dt[0] = fp2(a[2]);
   for (i = 1; i <= 50; i++)
       {
       for (j = i; j >= 1; j--)
        if (j == i)
           {
           j1 = j - 1;
           f1 = fabs(t[j1]);
           f2 = pow(f1, n1);
           f3 = pow(a[5], n2);
           xx = fix(t[j1], t[j1], pw, x2);
          e[j] = 1. + (db - xx) * (2822. - 2.09 * t[j1])
          / (a[1] - t[j1]) / cp;
           rr[j] = 0.615059 / (1. + 0.98082 * f2 * f3);
           f1 = rr[j] - 0.02;
           f2 = sqrt(f1);
           bb[j] = 1.5 - f2 / 1.64;
           if (i == 1)
              bb[j] = 1.;
           kk[j] = al * bb[j] * e[j];
           c[j] = kk[j] / cp / 1e3;
           g[j] = h * rr[j] * 1e3;
           te[j] = c[j] * (db - xx) * 3600.;
           te[j] = g[j] / te[j];
           ti += te[j];
           if (j > 1)
           {
           f1 = t[j1]+273.;
           f2 = pow(rr[j1], n3);
           f3 = sqrt(f1);
           p21 = fp2(t[j1]);
           p22 = fp2(t[j-2]);
           m[j1] = p21 * 212.39 / f1 * 1.5966 * f2 * f3;
           dt[3] = m[j1] / d[j1] * (p21 - p22) * te[j];
           if (dt[3] < 0e0)
              dt[3] = 0e0;
           if (dt[3] < g[j])
              g[j] -= dt[3];
           }
           d[j] = g[j] / rr[j] / 1e3;
           l[j] = 0.7 * rr[j];
```

```
        r[0] = 1. / kk[j];
        for (jj = 1; jj <= i; jj++)
         r[jj] = r[jj - 1] + d[jj] / l[jj];
        q[i] = (a[1] - t[0]) / r[i];
        t[i] = q[i] * d[i] / l[i] + t[i - 1];
        t[1] = q[i] * d[1] / l[1] + t[0];
        dt[1] = fp2(t[1]);
        if (i > 2)
        for (jj = 2; jj <= i - 1; jj++)
            t[jj] = q[i] * d[jj] / l[jj] + t[jj - 1];
        t[i] = q[i] * d[i] / l[i] + t[i-1];
        }
if (t[i] >= a[1])
{
in = i - 1;
goto z;
}
else
{
if (j == 1 && i > 1)
   {
   f1 = t[1]+273.;
   f2 = pow(rr[1], n3);
   f3 = sqrt(f1);
   m[1] = dt[1] * 212.39 / f1 * 1.5966 * f2 * f3;
   w[1] = m[1] / d[1] * (dt[1] - dt[0]) * te[1];
   rr[1] = (rr[1] * d[1] + w[1] / 1e3) / d[1];
   l[1] = 0.7 * rr[1];
   }
if (j > 1 && j < i)
   {
   f1=t[j]+273.;
   f2 = pow(rr[j], n3);
   f3 = sqrt(f1);
   p22 = fp2(t[j]);
   m[j] = p22 * 212.39 / f1 * 1.5966 * f2 * f3;
   dt[2] = p21;
   f1 = t[j1] + 273.;
   f2 = pow(rr[j1], n3);
   f3 = sqrt(f1);
   m[j1] = dt[2] * 212.39 / f1 * 1.5966 * f2 * f3;
   w[j] = (m[j] / d[j] * (p22 - dt[2]) - m[j1] / d[j1]) *
   (dt[2] - fp2(t[j - 2])) * te[j];
   if (w[j] < 0e0)
        w[j] = 0e0;
   rr[j] = (rr[j] * d[j] + w[j] / 1e3) / d[j];
   l[j] = 0.7 * rr[j];
   }
}
in = i;
if (t[i-1] > 0e0)
{
in = i - 1;
goto z;
   }
if (ti > a[3] || t[i] > 0e0)
```

```
       goto z;
       }
 z:
   if (in == 0)
      {
      b[0] = d[1];
      b[1] = rr[1];
      b[2] = l[1];
      b[3] = t[1] + 273.;
      b[4] = e[1];
      goto lm3;
      }
   for (i = 1; i <= in; i++)
       {
       b[0] += d[i];
       b[1] += rr[i];
       b[2] += l[i];
       b[4] += e[i];
       if (jn[1] == 1)
       fprintf(put, "  i = %i  d = %e  rr = %e  l = %e  t = %e\n",
i, d[i], rr[i], l[i], t[i]);
       }
   b[3] = t[in];
   h=(double) in;
   b[1] /= h;
   b[2] /= h;
   b[4] /= h;
   b[3] += 273.;
 lm3:
   if (jn[0] == 1)
      fprintf(put, "\n  delta = %e  ro = %e  lambda = %e  ti = %e
ksi = %e\n",b[0], b[1], b[2], b[3], b[4]);
   a[1] += 273.;
   a[2] += 273.;
 fin:;
   }
```

Подпрограмма fp2 предназначена для вычисления парциального давления насыщенного влажного воздуха при температуре *t*. Здесь *t* - °C.

```
//    fp2.cpp               //
//    Water vapour pressure //
#include <math.h>
#include "pscs.hpp"
double fp2(double t)
  {
  double f, q, tk, tm, f1;
  tk = t + 273.;
  tm = t + 273.15;
  if (t < 0.)
     {
     f = -24.4867 - 5631.12 / tk + 8.2312 * log10(tk) -
     0.0386144 * tk + 2.77494e-5 * tk * tk;
     f1 = 10.2 * exp(f);
     }
```

```
  else
     {
     q = 373.16 / tm;
     f = 2.302585 * (0.0141966 - 3142.305 * (1. / tm - 0.00268) +
     8.2 / 2.302585 * log10(q) - 0.0024804 * (100.01 - t));
     f1=exp(f);
     }
  return(f1);
  }
```

Подпрограмма fh2 предназначена для вычисления энтальпии насыщенного влажного воздуха при температуре t и давлении p. Здесь t - °C, p - кгс/см^2. Используемая внешняя подпрограмма fp2.

```
//    fh2.cpp            //
//    Humid air enthalpy //
#include <math.h>
#include "pscs.hpp"
double fh2(double t, double p)
  {
  double fp2(double);
  double p2, f1;
  p2 = fp2(t);
  f1 = 0.2405 * t + (371.52 + 0.26901 * t) * p2 / (p - p2);
  return(f1);
  }
```

Подпрограмма fix предназначена для вычисления при int i = 1 относительной влажности, при i = 2 - влагосодержания влажного воздуха по температурам по сухому tb и мокрому tm термометрам при давлении p. Здесь t - °C, p - кгс/см^2. Используемые внешние подпрограммы fp2, fh2.

```
//    fix.cpp                                              //
//    Humid air relative humidity and moisture content //
#include <math.h>
#include "pscs.hpp"
double fix(double tb, double tm, double p, int i)
  {
  double fi, p1, h, f1;
  double fp2(double), fh2(double, double);
  if (tm >= tb)
     fi = 1.;
  else
     {
     p1 = fp2(tb);
     h = fh2(tm, p);
     fi = (p * h - 0.2405 * tb) / (371.52 + 0.26901 * tb + h) /
p1;
     }
  if (i == 1)
     f1 = fi;
  else
     {
     p1 = fp2(tm);
     f1 = 0.622 * fi * p1 / (p - fi * p1);
```

```
    }
  return(f1);
  }
```

8.2. Расчет разгонных характеристик теплообменников

```
//    tsfd.cpp                                          //
//    Test for subprograms rxdyn.cpp and sfd.cpp  //
#include <stdio.h>
#include <math.h>
#include "pscs.hpp"
void main()
  {
  static double ba[6] =
      { 33.1501, 0.02782, 56.67119, 0.02050, 0.04722, 0. };
  double brx[8], arx[6];
  int i, j, ie, er, it;
  FILE *stream;
  stream = fopen("ressfd","w");
  fprintf(stream, "\n  Test for RXDYN & SFD\n");
  for (j = 0; j < 5; j++)
      arx[j] = ba[j];
  prarf(stream, "arx", "e", arx, 5);
  ie = 0;
  arx[5] = arx[0] * arx[3];
  for (j = 0; j < 8; j++)
      brx[j] = 0e0;
  it = 0;
  while (ie == 0)
        {
        ie = 1;
     it++;
        rxdyn(arx, brx, &er);
        fprintf(stream, " %i   %f   %f   %f   %f   %f   %f   %f   %f
%f\n",
          it,  arx[5],  brx[0],  brx[1],  brx[2],  brx[3],  brx[4],
brx[5], brx[6], brx[7]);
        for (j = 0; j < 8; j++)
            if (brx[j] < 0.999)
               ie = 0;
        arx[5] += arx[2];
        }
  i = fclose(stream);
  }
```

Расчет временных (разгонных) характеристик теплообменников осуществляется с помощью подпрограммы rxdyn.

Входные параметры

double a[6] - массив аргументов для расчета временных характеристик, где:

0 - ξ^*;

1 - ε^*;

2 - T_w*;
3 - T_h*;
4 - L_c;
5 - τ, с.

Выходные параметры
double b[8] - массив временных характеристик, в котором:

0 - h_{TT1};
1 - $h_{\upsilon T1}$;
2 - $h_{T\upsilon 1}$;
3 - $h_{\upsilon\upsilon 1}$;
4 - h_{TGh};
5 - $h_{\upsilon Gh}$;
6 - h_{TGc};
7 - $h_{\upsilon Gc}$.

int *er - код возврата из подпрограммы. Равен 0 при нормальном окончании работы и 1, если время a[5] меньше времени транспорта a[0] * a[3].

Используемая внешняя подпрограмма sfd.

```
//    rxdyn.cpp                                    //
// Dynamic characteristics calculation of    //
// two-flow heat exchanger                   //
#include <math.h>
#include "pscs.hpp"
void rxdyn(double a[ ], double b[ ], int *er)
{
  double v[25], pi = 3.14159, nv = 0.8, nn = 0.6, as[7], bs[6],
         v2, v1, v0, vk, d2, nv1, sq1, sq;
  int i;
  *er = 0;
  if (a[5] <= a[0] * a[3])
     {
     for (i = 0; i < 8; i++)
         b[i] = 0e0;
     *er = 1;
     goto fin;
     }
  v[2] = a[0];
  v[0] = a[1];
  v[8] = a[2];
  v[18] = a[3];
  v[10] = a[4];
  v[19] = (a[5] - v[2] * v[18]) / v[8];
  v[20] = a[5] / v[8];
  v[1] = v[8] / v[18];
```

```
  v0 = 1. + v[0];
  v1 = 1. + v[1];
  v2 = 1. + v[1] * v0;
  vk = v2 * v2;
  sq = vk - 4. * v[1] * v[0];
  if (sq > 0e0)
     sq1 = sqrt(sq);
  else
     sq1 = 0.;
  d2 = 1. / (2. * v[8]);
  v[6] = d2 * (-v2 + sq1);
  v[7] = d2 * (-v2 - sq1);
  v[4] = 1. + v[8] * v[6];
  v[5] = 1. + v[8] * v[7];
  nv1 = 1. + (1. - nv) * v[0];
  v[9] = nv / nv1;
  v[12] = v0 * v[8];
  v[3] = (1. - nv) * v[12] / nv1;
  v[11] = (1. - nn) * v0 * v[10] / (v[0] * (nn * v[10] + 1.));
  v[21] = 1. / v0;
  v[22] = 1. / v1;
  v[17] = v[0] * v[2];
  v[24] = 1. / exp(v[17]);
  as[0] = v[2];
  as[1] = v[19];
  as[2] = v[4];
  as[3] = v[5];
  as[4] = v[20];
  as[5] = v[17];
  as[6] = v[1];
  sfd(as, bs);
  v[23] = bs[0] - bs[1];
  b[0] = bs[0];
  b[1] = v[23];
  b[2] = (v[7] * bs[2] - v[6] * bs[3]) / (v[7] - v[6]) / (1. -
v[24]);
  b[3] = (1. / (v[10] + 1. - v[21] * v[24])) * (v[10] + ((v[5] -
         v[21]) * bs[2] - (v[4] - v[21]) * bs[3]) / (v[5] - v[4])
+ v[0] * v[21] * v[24] * v[23]);
  b[4] = v[1] * bs[5] / (v[2] * v1);
  b[5] = 1. / (v[11] + v[2] + 1.) * (v[11] + v[1] * v[22] * bs[5]
+ v[22] * bs[4] + v[23]);
  b[6] = v[1] * (bs[5] + v[3] * bs[4] / v[8]) / (v[2] * v1);
  b[7] = 1. / (v[2] + v[9]) *(v[1] * v[22] * bs[5] + v[9] * v[22]
*
        bs[4] + v[9] * v[23]);
  for (i = 0; i < 8; i++)
      if (b[i] >= 0.999)
         b[i] = 1e0;
fin:;
  }
```

Специальные функции, необходимые для расчета разгонных характеристик , вычисляются с помощью подпрограммы sfd.

Входные параметры

double a[7] - массив исходных данных для расчета спецфункций, где аргументы:

0 - ξ;
1 - η;
комплексы физических параметров:
2 - c_1;
3 - c_2;
4 - τ/a;
5 - α;
6 - μ.

Выходные параметры

double b[6] - массив значений спецфункций, элементы которого:

0 - v_1;
1 - $v_{1.0}$;
2 - $\varphi_{1.1*}$;
3 - $\varphi_{1.2*}$;
4 - $\varphi_{1.\mu}$;
5 - $\varphi_{2.\mu}$.

Используемые внешние подпрограммы besmod, udyn.

```
//    sfd.cpp                                          //
//  Special functions for dynamic characteristics //
//  calculation of heat exchanger                  //
#include <math.h>
#include "pscs.hpp"
// static function prototype
static double pex(double);
void sfd(double a[ ], double b[ ])
{
  double e1, e2, e3, e4, p, bm[2], ep, u, eps,
     d, e, v11, v12, f1n, f11zn, f12zn,
        f2n, f1mn, f1zn, ea, v1m, v2, v20, pea, pa;
  pa = a[0] * a[1];
  p = 2. * sqrt(pa);
  ep = pex(a[0]);
  ea = pex(a[1]);
  pea = ep * ea;
  besmod(p, bm);
  b[1] = bm[0] / pea;
  eps = 1e-4;
  udyn(a[0], a[1], eps, &u);
  if (u < 0e0)
```

```
     b[0] = 1e0;
  else
     b[0] = u / pea;
  pa = a[1] / a[0];
  v20 = bm[1] / ep * sqrt(pa) / ea;
  v2 = (a[1] - a[0]) * b[0] + a[0] * (b[1] + v20);
  d = a[0] / a[2];
  e = a[1] * a[2];
  udyn(d, e, eps, &v11);
  if (v11 < 0e0)
     v11 = 1e0;
  else
     v11 /= pea;
  d = a[0] / a[3];
  e = a[1] * a[3];
  udyn(d, e, eps, &v12);
  if (v12 < 0e0)
      v12 = 1e0;
  else
      v12 /=  pea;
  e1 = pex((1. + a[6]) * a[4]);
  e2 = pex(a[5]);
  e3 = pex((1. - a[2]) * a[4]);
  e4 = pex((1. - a[3]) * a[4]);
  f11zn = 1. / e3 - v11 / e2;
  f12zn = 1. / e4 - v12 / e2;
  f2n = a[4] - v2;
  f1n = 1. - b[0];
  d = -a[0] / a[6];
  e = -a[1] * a[6];
  udyn(d, e, eps, &v1m);
  if (v1m < 0e0)
     v1m = 1e0 ;
  else
     v1m /=  pea;
  f1mn = 1. / e1 - v1m;
  f1zn = 1. - b[0] / e2;
  b[2] = f1zn - f11zn;
  b[3] = f1zn - f12zn;
  b[4] = f1n - f1mn;
  b[5] = f2n - b[4] / (1. + a[6]);
  }

static double pex(double a)
  {
  if (a < 87.)
     return (exp(a));
  else
    return (1e50);
  }

//    besmod.cpp                    //
//    Modified Bessel-functions     //
#include <math.h>
#include "pscs.hpp"
void besmod(double x, double b[ ])
```

```
  {
  double t, y, s, z, xm, signx;
  if (x > 72.)
     xm = 72.;
  else
     xm = x;
  signx = (x < 0.) ? -1. : 1.;
  y = fabs(xm);
  if (y <= 10.)
     {
     t = x * x / 100;
     b[0] = ((((((((((0.668832 * t + 0.143995) * t + 9.66646) *
     t+26.2422) * t + 95.902) * t + 239.239) * t + 471.312) * t +
     678.85) * t + 678.181) * t+ 434.027) * t + 156.25) * t +
     24.9999) * t + 1.;
     b[1] = xm* ((((((((((0.0241117 * t + 0.0160446) * t +
     0.411871)* t + 1.35567) * t + 5.2838) * t + 4.9822) * t+
     33.6516) * t + 56.5112) * t + 67.8173) * t + 54.2534) * t +
     26.0417) * t + 6.25) * t + 0.5);
     }
  else
     {
     t = 10. / y;
     z = sqrt(y);
     s = exp(y);
     b[0] = s * ((((((0.107871e-5 * t - 0.72753e-6) * t +
     0.580397e-5)*t + 0.286977e-4) * t + 0.280601e-3) * t +
     0.498644e-2) * t + 0.398942) / z;
     b[1] = s * ((((((-0.123545e-5 * t + 0.74632e-6)* t +
     0.726227e-5* t -0.403144e-4) * t - 0.467618e-3) * t –
     0.0149603) * t + 0.398942) / z * signx;
     }
  }

//    udyn.cpp                                  //
//    Calculation of U(ksi, eta) function    //
#include <math.h>
#include "pscs.hpp"
//   static function prototype
static double fct(int);
//   static variable
static int in;
static double dzf,etf;
void udyn(double dz, double et, double eps, double *sum)
  {
  int tim = 1, k, n = 0, t = 0;
  double m[16], mb[2], d, mn, aet, adz, d1, d2, d3, m7;
  aet = fabs(et);
  adz = fabs(dz);
  if (aet < 1e-3)
     {
     *sum = 1e0;
     goto fin;
     }
  if (adz < 1e-3)
     {
```

```
     *sum = exp(aet);
     goto fin;
     }
  if ((aet + adz) > 20.)
     {
     d1 = dz * et;
     d2 = et / dz;
     d3 = dz + et;
     d = 2. * sqrt(d1);
     m7 = sqrt(adz) - sqrt(aet);
     mn = erf(m7);
     d1 = 6.3662e-1 / d;
     m[0] = sqrt(d1);
     besmod (d, mb);
     m[1] = mb[0];
     m[2] = mb[1];
     m[3] = m[1] - 2. * m[2] / d;
     m7 = sqrt(d2);
     m[7] = sqrt(m7);
     m[8] = 1. + m[7];
     m[9] = m[7] / (m[8] * m[8]);
     m[10] = m[9] * m[9];
     m[11] = m[10] * m[9];
     m[12] = m[11] * m[9];
     m[13] = hsin(d);
     m[14] = hcos(d);
     m[15] = m[13] + m[14];
     m[4] = m[0] * m[15];
     m[5] = m[0] * (m[15] - m[15] / d);
     d = 1. - mn;
     if (d > 2.)
        d = 2.;
     *sum = 5e-1 * d * exp(d3) + m[1] / m[8] + 0.5 * (1. - m[7]) /
            m[8] * (m[9] * (2. * m[1] - m[4]) + m[10] * (6. * m[1]
          -* m[4] + 2. * m[2]) + m[11] * (20. * m[1] - 15. * m[4]
          + 12. * m[2] - m[5]) + m[12] * (70. * m[1] - 56. * m[4]
             + 56. * m[2] - 8. * m[5] + 2. * m[3]));
     goto fin;
    }
  in = 0;
  dzf = dz;
  etf = et;
  euler(fct, eps, tim, &m[0]);
  *sum = m[0] / 2.;
iter:
  in++;
  euler(fct, eps, tim, &mn);
  for (k = 0; k <= n; k++)
      {
      d = (mn + m[k]) / 2.;
      m[k] = mn;
      mn = d;
      }
  if (fabs(mn) < fabs(m[n]) && n < 15)
     {
     d = mn / 2;
```

```
     n++;
     m[n] = mn;
     }
  else
     d = mn;
  *sum += d;
  if (fabs(d) < eps)
     t++ ;
  else
     t = 0;
  if (t < tim)
     goto iter;
fin:;
  }
static double fct(int i)
   {
   double f1 = 1., f2 = 1., pd, pe, pi, rv;
   int j;
   if (i == 0 && in == 0)
      return(1e0);
   if (i == 1 && in == 0)
      {
      rv = dzf * etf;
      return (rv);
      }
   if (i > 1 && in == 0)
      {
      for (j = 2; j <= i; j++)
          f1 *= (double) j;
      pi = (double) i;
      pd = pow(dzf, pi);
      pe = pow(etf, pi);
      rv = pd * pe / (f1 * f1);
      if (rv > 1e30)
        rv = 1e30;
      if (rv < 1e-9)
         rv = 1e-9;
      return(rv);
      }
   if (i == 0 && in == 1)
      {
      rv = etf;
      return(rv);
      }
   if (i == 0 && in > 1)
      {
      for (j = 2; j <= in; j++)
          f2 *= (double) j;
      pe = pow(etf, (double) in);
      rv = pe / f2;
      if (rv > 1e30)
      rv = 1e30;
      if (rv < 1e-9)
      rv = 1e-9;
      return(rv);
      }
```

```
   for (j = 2; j <= i; j++)
       f1 *= (double) j;
   for (j = 2; j <= i + in; j++)
       f2 *= (double) j;
   pi = (double) i;
   pd = pow(dzf, pi);
   pi = (double) (i + in);
   pe = pow(etf, pi);
   rv = pd * pe / (f1 * f2);
   if (rv > 1e30)
      rv = 1e30;
   if (rv <1e-9)
      rv = 1e-9;
   return(rv);
   }
```

8.3. Безусловная и условная минимизация комплексным методом

```
//    tkomplex.cpp                        //
//    Test for subprogram komplex.cpp     //
#include <stdio.h>
#include <math.h>
#include "pscs.hpp"
int i;
static void func(double[ ], double *);
void main()
   {
   static double um[3] = {3e0, 1e-10, 0e0},
       bl[4] = {-3., -1., -3., -1.};
   double fl;
   static int im[5] = {1, 4, 500, 0, 50};
   FILE *stream;
   stream=fopen("reskomp", "w");
   i = 0;
   komplex(stream, im, um, func, bl, &fl);
   fprintf(stream, "\n   i=%i\n", i);
   i = fclose(stream);
   }

static void func(double y[ ], double *ss)
   {
   double p2 = 2., p3 = 3.;
   i++;
   *ss = 100. * pow(y[1] - pow(y[0], p2), p2) + pow( 1. –
   y[0], p2) +90. * pow(y[3] - pow(y[2], p2), p2) +
   pow(1. - y[2], p3) + 10.1 * (pow(y[1] - 1. ,p2) +
   pow(y[3] - 1., p2)) + 19.8 * (y[1] - 1.) * (y[3] - 1.);
   }
```

Подпрограмма komplex предназначена для решения задачи безусловной минимизации функции n переменных модифицированным методом комп-

лексного поиска. В ней поиск минимума ведется по многогранникам с числом вершин 2 * n , где n - размерность пространства.

Оператор начала подпрограммы имеет унифицированный формат:

void komplex(put, pr1, pr2, f, bl, fl).

Здесь FILE *put - файл для печати;
int pr1[5] - информационный массив, где:

0 - код подробности выдаваемой информации, 0 или 1;
1 - размерность пространства n;
2 - максимум итераций;
3 - число сделанных итераций;
4 - число шагов для печати;

double pr2[3] - управляющий массив, где:

0 - начальный размер стороны многогранника;
1 - малое число для окончания поиска;
2 - нижняя граница значения функции;

void (*f)(double [ ], double *) - подпрограмма, вычисляющая минимизируемую функцию;

double bl[n] - на входе в подпрограмму вектор начальных приближений, на выходе - найденная точка минимума;

double *fl - значение функции в точке минимума.

Используемая внешняя подпрограмма random.

```
//    komplex.cpp                                          //
//    Unconstreined optimization by complex method     //
#include <stdio.h>
#include <math.h>
#include "pscs.hpp"
#define b(i, j) b[i * m + j]
void komplex(FILE *put, int pr1[ ], double pr2[ ],
     void (*f)(double [ ], double *), double bl[ ], double *fl)
   {
   double f1, f2, fr, fu, p, sr, a1, gc = 0.5, ge = 2., gr = 1.3,
       w0 = 0e0, w1 = 1e0, *b, *c, *fs, *d, *bu, *b1, *b2;
   int bit, i, j, k, l, m, m1, r = 0, r2 = 0, n;
   long x0;
   m = pr1[1];
   n = 2 * m - 2;
```

```
   m1 = n + 1;
   b = new double[m1 * m];
   c = new double[m];
   fs = new double[m1];
   d = new double[m];
   bu = new double[m];
   b1 = new double[m];
   b2 = new double[m];
   if (pr1[0] == 1)
      fprintf(put, "   Function minimization by complex
      method\n");
   p = 2. * pr2[0];
   x0 = 123274659;
   b(0, 0) = bl[0] - pr2[0] + random(w0, w1, x0) * p;
   x0 = 0;
   for(j = 1 ; j < m; j++)
      b(0, j) = bl[j] - pr2[0] + random(w0,w1,x0) * p;
   for(i = 1; i < m1; i++)
      {
      for(j = 0;j < m; j++)
      b(i, j) = bl[j] - pr2[0] + random(w0, w1, x0) * p;
      }
   for(i = 0; i < m1; i++)
      {
      for(j = 0; j < m; j++)
      d[j] = b(i, j);
      (*f)(d, &fs[i]);
      }
 l1:
   r++;
   fu =- 1e30;
   *fl = 1e30;
   for(i = 0; i < m1; i++)
      {
      if (fs[i] > fu)
         {
         fu = fs[i];
         k = i;
         }
      if (fs[i] < *fl)
         {
         *fl = fs[i];
         l = i;
         }
      }
   for(i = 0; i < m; i++)
      {
      bu[i] = b(k, i);
      bl[i] = b(l, i);
      }
   r2++;
   if (r2 == pr1[4] && pr1[0] == 1)
      {
      r2 = 0;
      fprintf(put, "\n   r = %d fl = %e\n", r, *fl);
      prarf(put, "bl", "e", bl, m);
```

```
      }
   if (r > pr1[2])
      goto l3;
   if (*fl <= pr2[2])
      goto l3;
   fr = 0e0;
   for(i = 0; i < m1; i++)
      fr += fs[i];
   fr /= m1;
   sr = 0e0;
   for(i = 0; i < m1; i++)
      {
      a1 = fs[i] - fr;
      sr += a1 * a1;
      }
   sr /= m;
   if (sqrt(sr) <= pr2[1])
      goto l3;
   for(i = 0; i < m; i++)
      c[i] = 0e0;
   for(i = 0; i < m; i++)
      {
      for(j = 0; j < m1; j++)
         c[i] += b(j, i);
      }
   for(i = 0; i < m; i++)
      c[i] = (c[i] - bu[i]) / n;
   for(i = 0; i < m; i++)
      b1[i] =(1. + gr) * c[i] - gr * bu[i];
   (*f)(b1, &f1);
   if (f1 <* fl)
      {
      for(i = 0; i < m; i++)
         b2[i] = b1[i] * ge + c[i] * (1. - ge);
      (*f)(b2, &f2);
      if (f2 <* fl)
      {
      for(i = 0; i < m; i++)
         bu[i] = b(k, i) = b2[i];
         fs[k] = f2;
            goto l1;
         }
       else
         {
         for(i = 0; i < m; i++)
         bu[i] = b(k, i) = b1[i];
         fs[k] = f1;
         goto l1;
         }
      }
   else
      {
      bit = 0;
      for(i = 0; i < m1; i++)
         {
         if (i != k)
```

```
            {
            if (f1 < fs[i])
               bit++;
         }
      }
      if (bit == n)
      {
         for(i = 0; i < m; i++)
         bu[i] = b(k, i) = b1[i];
         fs[k] = f1;
         goto l1;
         }
      else
      {
      if (f1 < fu)
         {
         for(i = 0; i < m; i++)
          bu[i] = b(k, i) = b1[i];
         fs[k] = f1;
         goto l1;
         }
      }
      }
   for(i = 0; i < m; i++)
      b2[i] = gc * bu[i] + (1. - gc) * c[i];
   (*f)(b2, &f2);
   if (f2 > fu)
      {
      for(i = 0; i < m1; i++)
         {
         for(j = 0; j < m; j++)
         {
         b(i, j) = 0.5 * (b(i, j) + bl[j]);
         d[j] = b(i, j);
         }
         (*f)(d, &fs[i]);
         }
      goto l1;
      }
   else
      {
      for(i = 0; i < m; i++)
      bu[i] = b(k, i) = b2[i];
         fs[k] = f2;
         goto l1;
      }
l3:
   pr1[3] = r;
   if (pr1[0] == 1)
      {
      fprintf(put, "\n   r = %d fl = %e\n", r, *fl);
      prarf(put, "bl", "e", bl, m);
      }
   delete [ ]b;
   delete [ ]c;
   delete [ ]fs;
```

```
   delete [ ]d;
   delete [ ]bu;
   delete [ ]b1;
   delete [ ]b2;
   }
```

Подпрограмма random генерирует случайные числа, равномерно распределенные на интервале [a,b]. При первом обращении к подпрограмме random значение x0 должно быть нечетным и находиться в интервале 10000000000 < x0 < 34359738368. При последующих обращениях должно быть x0 = 0.

```
//    random.cpp              //
//    Random number generator //
#include "pscs.hpp"
double random(double a, double b, long x0)
  {
  static long x, m27 = 134217728, m28 = 268435456, m29 =
536870912;
  double c, d, e;
  if (x0 != 0)
     x = x0;
  x *= 5;
  if (x >= m29)
     x -= m29;
  if (x >= m28)
     x -= m28;
  if (x >= m27)
     x -= m27;
  c = (double) x;
  d = (double) m27;
  e = (b - a) * c / d + a;
  return(e);
  }


//    tpfm.cpp                                        //
//    Test for subprograms pfm.cpp, komplex.cpp   //
#include <iostream.h>
#include <stdio.h>
#include <math.h>
#include "pscs.hpp"
static void funres(double [ ], double [ ]);
int i;
void main()
   {
   static double um[4] = {1e0, 1e-8, 1e-8, 1e-8},
     bl[2] ={2., 2.};
   double fl = 1.;
   static int im[8] ={1, 2, 1, 2, 20, 100, 0, 0};
   FILE *stream;
   stream = fopen("respfm", "w");
   i = 0;
   cout << " PFM + KOMPLEX started\n";
```

```
   pfm(stream, im, um, funres, bl, &fl, komplex);
   fprintf(stream,"\n i = %i\n", i);
   cout << " PFM ended\n";
   cout << " i = " << i << "\n";
   i = fclose(stream);
   }

static void funres(double x[ ], double s[ ])
   {
   double a, b;
   i++;
   a = x[0] - 2.;
   b = x[1] - 1.;
   s[0] = a * a + b * b;
   s[1] = x[0] - 2. * x[1] + 1.;
   s[2] = -x[0] * x[0] / 4. - x[1] * x[1] + 1.;
   }
```

Подпрограмма pfm предназначена для решения общей задачи нелинейного программирования методом параметризации целевой функции Моррисона. Параметры подпрограммы имеют следующий смысл:

FILE *put - указатель на файл для печати;

int pr1[8] - информационный массив, где:

0 - код подробности печатаемой информации, 0 | 1;
1 - размерность пространства n;
2 - число ограничений типа равенств m;
3 - общее число ограничений p;
4 - максимум итераций условной минимизации;
5 - максимум итераций безусловной минимизации;
6 - число сделанных итераций it;
7 - код возврата из подпрограммы rc:

rc = 1 если |f| < pr2[1]; rc = 2 если |f(i)-f(i-1)| < pr2[2]; rc = 3 если ||x(i)-x(i-1)|| < pr2[3]; rc = 4 если it > pr1[4].

double pr2[4] - управляющий массив, где:

0 - начальный размер шага;
1 - критерий прерывания по значению f;
2 - критерий прерывания по значению |f(i)-f(i-1)|;
3 - критерий прерывания по значению ||x(i)-x(i-1)||.

void (*fres)(double [ ], double [ ]) - подпрограмма вычисления минимизируемой функции и ограничений;

double bl[n] - на входе в подпрограмму вектор начальных приближений, на выходе - найденная точка минимума;

double *fl - на входе в подпрограмму нижняя граница целевой функции, на выходе - значение штрафной функции в точке минимума.

(*extr)(FILE *, int *, double *, void (*)(double [ ], double *),
double *, double *) - унифицированная подпрограмма безусловной минимизации.

При pr1[0] = 1 на выходе из подпрограммы pfm печатаются значения штрафной функции, переменных, целевой функции, ограничений в точке минимума и кода возврата.

```
//    pfm.cpp                                                    //
//    Constreined optimization by parametrization method    //
#include <stdio.h>
#include <math.h>
#include "pscs.hpp"
//     static function prototypes
static void hg(double *),
         func(double [ ], double *),
         (*fres1)(double [ ], double [ ]);
//      static variables
static int m, p;
static double *rx;
static double r = 1e-4, k1 = 0.333333e4,
     k2=- 1e-4, k3 = 0.333333e-8, eta;
void pfm(FILE *put, int pr1[ ], double pr2[ ],
        void (*fres)(double [ ], double[ ]), double bl[ ],
        double *fl,
        void (*extr)(FILE *, int[ ], double[ ],
        void (*)(double[ ], double *), double[ ], double *))
   {
    double f1, xnorm, q2[3], del, *b1;
    int i, iter, q1[5], n;
    fres1 = fres;
    n = pr1[1];
    m = pr1[2];
    p = pr1[3];
    rx = new double [p + 1];
    b1=  new double [n];
    if (pr1[0] == 1)
       {
       fprintf(put,
     "  Constreined optimization by parametrization method\n");
       fprintf(put, "  n = %i  m = %i  p = %i\n", n, m, p);
       }
    q1[0] = pr1[0];
    q1[1] = pr1[1];
    q1[2] = pr1[5];
    q1[4] = 50;
```

```
    q2[0] = pr2[0];
    q2[1] = 1e-6;
    q2[2] =- 1e-30;
    eta =* fl;
    (*fres)(bl, rx);
    f1 = rx[0];
    for (i = 0; i < n; i++)
        b1[i] = bl[i];
    for (iter = 1; iter <= pr1[4]; iter++)
        {
        (*extr)(put, q1, q2, func, bl, fl);
        if (*fl <= pr2[1])
           {
           pr1[7] = 1;
           goto lab;
           }
        del =* fl - f1;
        if (fabs(del) <= pr2[2])
           {
           pr1[7] = 2;
           goto lab;
           }
        xnorm = 0e0;
        for (i = 0; i < n; i++)
            {
            del = bl[i] - b1[i];
            xnorm += del * del;
            }
        if (xnorm <= pr2[3])
           {
           pr1[7] = 3;
           goto lab;
           }
        eta += sqrt(*fl);
        for (i = 0; i < n; i++)
            b1[i] = bl[i];
        f1 =* fl;
        }
    pr1[7] = 4;
lab:
    pr1[6] = iter;
    if (pr1[0] == 1)
       {
       (*fres)(bl, rx);
       fprintf(put, "\n  Penalty function fl = %e\n", *fl);
       fprintf(put, "  variable\n");
       prarf(put, "bl", "e", bl, n);
       fprintf(put, "  function & restrictions\n");
       prarf(put,"rx", "e", rx, p + 1);
       fprintf(put, " return code  %i\n", pr1[7]);
       }
    delete [ ]rx;
    delete [ ]b1;
    }

static void hg(double *g)
```

```
    {
    int i;
    double y, sum = 0e0;
    for (i = 1; i <= m; i++)
        rx[i] *= rx[i];
    for (i = m + 1; i <= p; i++)
        {
        y =- rx[i];
        if (y <= 0e0)
           rx[i] = 0e0;
        if ( 0e0 < y && y <= r)
           rx[i] = k1 * y * y * y;
        if (r < y)
           rx[i] = y * y + k2 * y + k3;
        }
    y = rx[0] - eta;
    for (i = 1; i <= p; i++)
        sum += rx[i];
    *g = y * y + sum;
    }

static void func(double x[ ], double *g)
    {
    (*fres1)(x, rx);
    hg(g);
    }
```