10. Prilozi

10.1 Popis korištenih oznaka

$\mathbf{u}(t)$ – vektor upravljanja
$\mathbf{x}(t)$ – vektor stanja sustava
t – vrijeme
t_0 – početni vremenski trenutak
t_f – konačni vremenski trenutak
J – kriterij optimalnosti, funkcija cilja
H – Hamiltonian
T – konstantni period diskretizacije
$\tau^{(k)}$ – varijabilni period diskretizacije
η – konstantni koeficijent konvergencije
N – broj diskretnih vremenskih podintervala
M – broj iteracija gradijentnog algoritma
n – dimenzija vektora stanja
m – dimenzija vektora upravljanja
$\mathbf{X}(j)$ – Jacobijeva matrica po vektoru stanja
$\mathbf{U}(j)$ – Jacobijeva matrica po vektoru upravljanja
G_B – kaznena funkcija za ograničenja tipa rubnih uvjeta
G_V – kaznena funkcija za ograničenja tipa nejednakosti
G_E – kaznena funkcija za ograničenja tipa jednakosti
K_B – koeficijent kaznene funkcije G_B
K_V – koeficijent kaznene funkcije G_V
K_E – koeficijent kaznene funkcije G_E
J_G – ukupna suma kaznenih funkcija
$\mathbf{q}(t)$, $\dot{\mathbf{q}}(t)$ – vektori generaliziranih koordinata i brzina robota
$\mathbf{W}$ – matrica težinskih koeficijenata, matrica koeficijenata pojačanja
S_j – funkcija zasićenja, aktivacijska funkcija
D_j – derivacija funkcije zasićenja
$\mathbf{I}$ – jedinična matrica
$\mathbf{0}$ – nul-matrica

Ostale oznake, kao i navedene oznake s drugim značenjem, objašnjene su u tekstu.

10.2 Program za BPTT algoritam

```
// OptimalControl.c
// BPTT algoritam za rjesavanje problema vremenski optimalnog upravljanja
// dinamikom robota s dva stupnja slobode (rotaciona i translaciona).
// Koristi se metoda konjugiranog gradijenta.

#include <stdio.h>
#include <math.h>
#include "Nrutil.h"          // procedure za alokaciju memorije
#include "Dat1.h"            // procedure za rad s datotekama
#include "matrix.h"          // procedure za matricne operacije
#include "Dim1Min.h"         // procedure za 1-D  minimizaciju

#define Ndat 10              // broj datoteka
#define N 1000               // broj vremenskih intervala
#define Noff 2000            // broj iteracija za OFF-line
#define NMOD 100             // broj iteracija za resetiranje
#define NDIM (N-1)           // broj varijabli fun. cilja
#define Nn 4                 // broj komponenti vektora stanja
#define Nm 2                 // broj komponenti vektora upravljanja
#define T ((tf-to)/N)        // period diskretizacije
#define to 0.0               // pocetno vrijeme
#define tf 1.5               // konacno vrijeme
#define x01 0.0              // pocetna vrijednost x1(0)=x01
#define x02 0.0              // pocetna vrijednost x2(0)=x02
#define x03 0.0              // pocetna vrijednost x3(0)=x03
#define x04 0.0              // pocetna vrijednost x4(0)=x04
#define x1f 1.571            // konacna vrijednost x1(tf)=x1f
#define x2f 0.0              // konacna vrijednost x2(tf)=x2f
#define x3f 1.0              // konacna vrijednost x3(tf)=x3f
#define x4f 0.0              // konacna vrijednost x4(tf)=x4f
#define U1g 600.0            // granicna vrijednost upravljacke varijable  'u1'
#define U2g 500.0            // granicna vrijednost upravljacke varijable  'u2'

#define TOL 1.0e-4           // tolerancija rjesenja
#define uo 100.0             // pocetna vrijednost u(0)=uo
#define uf 200.0             // konacna vrijednost u(tf)=uf
#define S 800.0              // koeficijent uz kaznenu funkciju (g=0)
#define K 0.04               // koeficijent uz kaznenu funkciju (g>=0)
#define eps 0.01             //minimalna vrijednost funkcije cilja

#define J12 193.0            // ukupni moment inercije
#define M1 5.0               // masa 'M'
#define M2 97.0              // masa 'm'
#define Aa 1.1               // duljina ruke
#define A1 (J12+M1*Aa*Aa)    // konstanta dinamickog modela
#define A2 (2*M1*Aa)         // konstanta dinamickog modela
#define A3 (M1+M2)           // konstanta dinamickog modela


double **Uu, **Ju, dt;
```

```
void Vrijeme(double *t, double h, int n)        // vrijeme
{
    int i;

    for(i=0;i<=n;i++) t[i]=i*h;
}

void Init_u0(double **u, double *t, int n)                    // pocetna vrijednost  u
{
    int i, j;

    for(j=1;j<=Nm;j++)
        for(i=0;i<=n;i++)
            u[j][i]=(uf-uo)*((double)(i)/n)+uo;
}

void func(double *f, double **x, double **u, int i)            //  x' = f(x, u)
{
    double x1, x2, x3, x4, u1, u2, naz;

    x1 = x[1][i]; x2 = x[2][i];x3 = x[3][i]; x4 = x[4][i];
    u1 = u[1][i]; u2 = u[2][i];
    naz = (A1 + A2*x3 + A3*x3*x3);

    f[1] = x2;
    f[2] = (-2*A3*x2*x3*x4 - A2*x2*x4 + u1)/naz;
    f[3] = x4;
    f[4] = (A3*x3*x2*x2 - A2*x2*x2/2 + u2)/A3;
}

void vec_x(double **x, double **u, double h, int n)          // Eulerov metod
{
    int i, j;
    double *f;

    f=dvector(1, Nn);
    x[1][0]=x01; x[2][0]=x02; x[3][0]=x03; x[4][0]=x04;
    for(i=0;i<=n;i++){
        for(j=1;j<=Nn;j++){
            func(f, x, u, i);
            x[j][i+1]=x[j][i]+h*f[j];
        }
    }
    free_dvector(f, 1);
}

double sn4(double x)
{
    double r1;

    if (x >= 0.0) r1=0.0;
    if (x < 0.0) r1=1.0;
    return(r1);
}
```

```
double Fu(double u1, double ug)            // parcijalna derivacija F po u
{
    double w1, w2, ret;

    w1 = u1 + ug;
    w2 = ug - u1;

    ret = 2*K*(w1*sn4(w1) - w2*sn4(w2));
    return(ret);
}

void JaccobX(double **X, double **x, double **u, double h, int i)        // Jacobijan po x
{
    double x1, x2, x3, x4, u1, u2, naz;

    x1 = x[1][i]; x2 = x[2][i];x3 = x[3][i]; x4 = x[4][i];
    u1 = u[1][i]; u2 = u[2][i];
    naz = (A1 + A2*x3 + A3*x3*x3);


    X[1][1] = 1.0; X[1][2] = 0.0; X[1][3] = 0.0; X[1][4] = 0.0;

    X[2][1] = h; X[2][2] = 1.0 + h*(-2*A3*x3*x4 - A2*x4)/naz; X[2][3] = 0.0;
    X[2][4] = h*(2*A3*x3*x2 - A2*x2*x2)/A3;

    X[3][1] = 0.0;
    X[3][2] = h*(-2*A3*x2*x4/naz - (A2+2*A3*x3)*(-2*A3*x2*x3*x4 - A2*x2*x4 + u1)/naz/naz);
    X[3][3] = 1.0; X[3][4] = h*x2*x2;

    X[4][1] = 0.0; X[4][2] = h*(-2*A3*x2*x3 - A2*x2)/naz; X[4][3] = h; X[4][4] = 1.0;
}

void JaccobU(double **U, double **x, double **u, double h, int i)        // Jacobijan po  u
{
    double x3, naz;

    x3 = x[3][i];
    naz = (A1 + A2*x3 + A3*x3*x3);

    U[1][1] = 0.0; U[1][2] = h/naz; U[1][3] = 0.0; U[1][4] = 0.0;
    U[2][1] = 0.0; U[2][2] = 0.0; U[2][3] = 0.0; U[2][4] = h/A3;
}

void vecFx(double *vFx, double   **x, double **u, int j)                // parcijalna derivacija F po x
{
    vFx[1] = 0.0;
    vFx[2] = 0.0;
    vFx[3] = 0.0;
    vFx[4] = 0.0;
}

void vecFu(double *vFu, double   **x, double **u, int j)                // parcijalna derivacija F po u
{
    vFu[1] = Fu(u[1][j], U1g);
    vFu[2] = Fu(u[2][j], U2g);
}
```

```
void vecGx(double *Gx, double **x)                    // parcijalna derivacija G_B po x
{
    Gx[1] = 2*S*(x[1][N]-x1f);
    Gx[2] = 2*S*(x[2][N]-x2f);
    Gx[3] = 2*S*(x[3][N]-x3f);
    Gx[4] = 2*S*(x[4][N]-x4f);
}

double J(double **u)                          // ukpna funkcija cilja
{
    int i;
    double r1, r2=0.0, ret, w1, w2, w3, w4, **x;
    double s1, s2, s3, s4;

    x=dmatrix(1, Nn, 0, N);

    vec_x(x, u, dt, NDIM);
    s1 = (x[1][N]-x1f)*(x[1][N]-x1f);
    s2 = (x[2][N]-x2f)*(x[2][N]-x2f);
    s3 = (x[3][N]-x3f)*(x[3][N]-x3f);
    s4 = (x[4][N]-x4f)*(x[4][N]-x4f);
    r1 = S*(s1+s2+s3+s4);
     for(i=0;i<=NDIM;i++){
        w1 = (u[1][i] + U1g); w2 = (U1g - u[1][i]);
        w3 = (u[2][i] + U2g); w4 = (U2g - u[2][i]);
        r2 = r2 + dt*K*(w1*w1*sn4(w1)+w2*w2*sn4(w2)+w3*w3*sn4(w3)+w4*w4*sn4(w4));
    }
    ret = r1 + r2;
    free_dmatrix(x, 1, Nn, 0, N);
    return(ret);
}

double Jfun(double z)                         // metoda najbrzeg spusta
{
    double r1, **y;
    int i, j;

    y=dmatrix(1, Nm, 0, NDIM);

    for(i=1;i<=Nm;i++)
        for(j=0;j<=NDIM;j++)
            y[i][j] = Uu[i][j] - z*Ju[i][j];
    r1 = J(y);
    free_dmatrix(y, 1, Nm, 0, NDIM);
    return(r1);
}

double J0(double **x)                         // kaznena funkcija za rubne uvjete
{
    double r1, s1, s2, s3, s4;

    s1 = (x[1][N]-x1f)*(x[1][N]-x1f);
    s2 = (x[2][N]-x2f)*(x[2][N]-x2f);
    s3 = (x[3][N]-x3f)*(x[3][N]-x3f);
    s4 = (x[4][N]-x4f)*(x[4][N]-x4f);
    r1 = S*(s1+s2+s3+s4);
    return(r1);
}
```

```
double J1(double **u, int n)                    // kaznena funkcija za ograničenja vektora u
{
    int i;
    double r2=0.0, w1, w2, w3, w4;

    for(i=0;i<=n;i++){
        w1 = (u[1][i] + U1g); w2 = (U1g - u[1][i]);
        w3 = (u[2][i] + U2g); w4 = (U2g - u[2][i]);
        r2 = r2 + K*(w1*w1*sn4(w1)+w2*w2*sn4(w2)+w3*w3*sn4(w3)+w4*w4*sn4(w4));
    }
    return(r2);
}

void Grad(double **Ju, double **x, double **u, double h)      // gradijent funkcije cilja
{
    double **X, **U, *P, *P1, *Fq, *Fw, *S1, *Gx, *Gu, *J0;
    double **Xt, **Ut, **Dj, **D1, **Xq, **Xqt;
    int j;

    X = dmatrix(1,Nn,1,Nn);
    U = dmatrix(1,Nm,1,Nn);
    P = dvector(1,Nn);
    S1 = dvector(1,Nm);
    P1 = dvector(1,Nn);
    Fq = dvector(1,Nn);
    Fw = dvector(1,Nm);
    Gx = dvector(1,Nn);
    J0 = dvector(1,Nm);
    Gu = dvector(1,Nm);
    Xt = dmatrix(1,Nn,1,Nn);
    Ut = dmatrix(1,Nn,1,Nm);
    Dj = dmatrix(1,Nn,1,Nn);
    D1 = dmatrix(1,Nn,1,Nn);
    Xq = dmatrix(1,Nn,1,Nm);
    Xqt = dmatrix(1,Nm,1,Nn);

    IdentMat(Dj, Nn);
    NullVector(P, Nn);

    vecGx(Gx, x);
    vecFu(Fw, x, u, N-1);
    JaccobU(U, x, u, h, N-1);
    MultVec(Gu, U, Gx, Nn, Nm);
    Storage(Ju, Fw, Gu, h, N-1);

    for(j=N-2;j>=0;j--){

        JaccobX(X, x, u, h, j+1);
        JaccobU(U, x, u, h, j);
        vecFx(Fq, x, u, j+1);
        vecFu(Fw, x, u, j);

        MultVec(P1, X, P, Nn, Nn);
        SummVec(P, Fq, P1, Nn);
        MultVec(S1, U, P, Nn, Nm);
        SummVec(J0, Fw, S1, Nm);

        TranMat(Xt, X, Nn, Nn);
        TranMat(Ut, U, Nm, Nn);
```

```
        MultMat(D1, Dj, Xt, Nn, Nn, Nn);
        MoveMat(Dj, D1, Nn, Nn);
        MultMat(Xq, Dj, Ut, Nn, Nn, Nm);
        TranMat(Xqt, Xq, Nn, Nm);
        MultVec(Gu, Xqt, Gx, Nn, Nm);

        Storage(Ju, J0, Gu, h, j);
    }
    free_dmatrix(X,1,Nn,1,Nn);
    free_dmatrix(U,1,Nm,1,Nn);
    free_dvector(P, 1);
    free_dvector(P1, 1);
    free_dvector(S1, 1);
    free_dvector(Fq, 1);
    free_dvector(Fw, 1);
    free_dvector(Gx, 1);
    free_dvector(Gu, 1);
    free_dvector(J0, 1);
    free_dmatrix(Xt,1,Nn,1,Nn);
    free_dmatrix(Ut,1,Nn,1,Nm);
    free_dmatrix(Dj,1,Nn,1,Nn);
    free_dmatrix(D1,1,Nn,1,Nn);
    free_dmatrix(Xq,1,Nn,1,Nm);
    free_dmatrix(Xqt,1,Nm,1,Nn);
}

void Opt_Upr(double **u, double *t)          // gradijentni algoritam + vremenski optimalno upravljanje
{
    double **x, **D, fnc, qmin, bren, br, naz, alf, fc0, fc1, fc, Dt;
    int i, j, k;

    x=dmatrix(1, Nn, 0, N);
    Ju=dmatrix(1, Nm, 0, N-1);
    D=dmatrix(1, Nm, 0, N-1);

    dt=T;

    for(k=0;k<=Noff;k++){

        if (k>0) naz = AbsVel(Ju, NDIM);
        vec_x(x, u, dt, N-1);
        Grad(Ju, x, u, dt);

        fc0 = J0(x);
        fc1 = J1(u, NDIM);
        fc = fc0 + fc1;
        Dt = (fc - eps);

        if (Dt<0.0) dt = dt - 0.00001;

        naz = AbsVel(Ju, NDIM);
        if (k%NMOD==0){
            for(j=1;j<=Nm;j++)
                for(i=1;i<=NDIM;i++)
                    D[j][i] = -Ju[j][i];
        }
        bren=Brent1(Jfun, TOL, &qmin);
        for(j=1;j<=Nm;j++)
            for(i=0;i<=N-1;i++)
```

```
                u[j][i]=u[j][i]-qmin*Ju[j][i];
        br = AbsVel(Ju, NDIM);
        if (k==0) alf = 0.0;
        else alf = br/naz;

        for(j=1;j<=Nm;j++)
            for(i=1;i<=NDIM;i++)
                D[j][i] = -Ju[j][i] + alf*D[j][i];
        naz = br;

        if ((k%100)==0){
            fnc = J(u);
            printf("%d J=%lf fc=%lf tf=%lf\n", k, fnc, fc, N*dt);
            puni_toc((double)(k), fc0, 7);
            puni_toc((double)(k), fc1, 8);
            puni_toc((double)(k), fc, 9);
            puni_toc((double)(k), fnc, 10);
        }
    }
    free_dmatrix(x, 1, Nn, 0, N);
    free_dmatrix(Ju, 1, Nm, 0, N-1);
    free_dmatrix(D, 1, Nm, 0, N-1);
}

void main(void)                     // glavni program
{
    double *t, **x;

    t=dvector(0, N);
    x=dmatrix(1, Nn, 0, N);
    Uu=dmatrix(1, Nm, 0, NDIM);

    datot(Ndat);                    // otvaranje datoteka
    Vrijeme(t, T, N);               //  t  - vrijeme
    Init_u0(Uu, t, N-1);            // Uu  - pocetni vektor upravljanja
    Opt_Upr(Uu, t);                 // Uu  - optimalni vektor upravljanja
    Vrijeme(t, dt, N);
    vec_x(x, Uu, dt, N-1);          //  x  - optimalni vektor stanja

    puni_dat1(t, x[1], 0, N, 1);
    puni_dat1(t, x[2], 0, N, 2);
    puni_dat1(t, x[3], 0, N, 3);
    puni_dat1(t, x[4], 0, N, 4);
    puni_dat1(t, Uu[1], 0, N-1, 5);
    puni_dat1(t, Uu[2], 0, N-1, 6);

    free_dvector(t, 0);
    free_dmatrix(x, 1, Nn, 0, N);
    free_dmatrix(Uu, 1, Nm, 0, NDIM);

    datzt(Ndat);                    // zatvaranje datoteka
}
```

```
//  matrix.h

void Greska(double **z, double **x, double **y, int n)            // razlika 2 vektora
{
     int i, j;

     for(j=1;j<=Nn;j++)
          for(i=0;i<=n;i++)
               z[j][i]=x[j][i]-y[j][i];
}

void MultMat(double **z, double **x, double **y, int n, int m, int l)            // množenje matrica
// Z=X*Y       X-n*m, Y-m*l,  Z-n*l
{
  int i, j, k;

  for(i=1; i<=n; i++){
     for(j=1; j<=l; j++){
       z[i][j] = 0.0;
       for(k=1; k<=m; k++)
          z[i][j] += x[i][k]*y[k][j];
     }
  }
}

void MultVec(double *z, double **x, double *y, int n, int m)           // množenje matrice s vektorom
//  z=X*y    z-m*1   X-m*n   y-n*1
{
  int i, j;

  for(i=1; i<=m; i++){
     z[i] = 0.0;
     for(j=1; j<=n; j++)
       z[i] += x[i][j]*y[j];
  }
}

void SummVec(double *z, double *x, double *y, int n)            // zbroj dva vektora
{
  int i;

  for(i=1; i<=n; i++) z[i] = x[i] + y[i];
}

void TranMat(double **y, double **x, int n, int m)                // transponiranje matrice
//  x-n*m   y-m*n
{
  int i, j;

  for(i=1; i<=n; i++)
     for(j=1; j<=m; j++)
       y[j][i] = x[i][j];
}
```

```
void MoveMat(double **y, double **x, int n, int m)              // prebacivanje jedne matrice u drugu
{
  int i, j;

  for(i=1; i<=n; i++)
      for(j=1; j<=m; j++)
        y[i][j] = x[i][j];
}

void NullVector(double *x, int n)                   // nul vektor
{
      int i;

    for(i=1; i<=n; i++)       x[i] = 0.0;
}

void IdentMat(double **x, int n)                    // jedinicna matrica
{
      int i, j;

    for(i=1; i<=n; i++)
            for(j=1; j<=n; j++){
                  if (i==j) x[i][j] = 1.0;
                  else x[i][j] = 0.0;
            }
}

void Storage(double **Ju, double *J0, double *Gu, double h, int j)
{
      int i;

      for(i=1; i<=Nm; i++)
            Ju[i][j] = h*J0[i] + Gu[i];
}

double AbsVel(double **d, int n)              // skalarni produkt dva vektora
{
      int i, j;
      double r=0.0;

      for(i=1; i<=Nm; i++)
            for(j=1; j<=n; j++)
                  r = r + d[i][j]*d[i][j];

      return(r);
}
```

// **Dim1Min.h**

```
#define GOLD 1.618034
#define GLIMIT 100.0
#define TINY 1.0e-20
#define MAX(a,b) ((a) > (b) ? (a) : (b))
#define SIGN(a,b) ((b) > 0.0 ? fabs(a) : -fabs(a))
#define SHFT(a,b,c,d) (a)=(b);(b)=(c);(c)=(d);

void mnbrak(double *ax, double *bx, double *cx, double *fa, double *fb, double *fc, double
(*func)(double))
{
    double ulim,u,r,q,fu,dum;

    *fa=(*func)(*ax);
    *fb=(*func)(*bx);
    if (*fb > *fa) {
        SHFT(dum,*ax,*bx,dum)
        SHFT(dum,*fb,*fa,dum)
    }
    *cx=(*bx)+GOLD*(*bx-*ax);
    *fc=(*func)(*cx);
    while (*fb > *fc) {
        r=(*bx-*ax)*(*fb-*fc);
        q=(*bx-*cx)*(*fb-*fa);
        u=(*bx)-((*bx-*cx)*q-(*bx-*ax)*r)/
            (2.0*SIGN(MAX(fabs(q-r),TINY),q-r));
        ulim=(*bx)+GLIMIT*(*cx-*bx);
        if ((*bx-u)*(u-*cx) > 0.0) {
            fu=(*func)(u);
            if (fu < *fc) {
                *ax=(*bx);
                *bx=u;
                *fa=(*fb);
                *fb=fu;
                return;
            } else if (fu > *fb) {
                *cx=u;
                *fc=fu;
                return;
            }
            u=(*cx)+GOLD*(*cx-*bx);
            fu=(*func)(u);
        } else if ((*cx-u)*(u-ulim) > 0.0) {
            fu=(*func)(u);
            if (fu < *fc) {
                SHFT(*bx,*cx,u,*cx+GOLD*(*cx-*bx))
                SHFT(*fb,*fc,fu,(*func)(u))
            }
        } else if ((u-ulim)*(ulim-*cx) >= 0.0) {
            u=ulim;
            fu=(*func)(u);
        } else {
            u=(*cx)+GOLD*(*cx-*bx);
            fu=(*func)(u);
        }
        SHFT(*ax,*bx,*cx,u)
        SHFT(*fa,*fb,*fc,fu)
    }
```

```
}

#undef GOLD
#undef GLIMIT
#undef TINY
#undef MAX

#define ITMAX 100
#define CGOLD 0.3819660
#define ZEPS 1.0e-10

double brent(double ax, double bx, double cx, double (*f)(double), double tol, double *xmin)
{
	int iter;
	double a,b,d,etemp,fu,fv,fw,fx,p,q,r,tol1,tol2,u,v,w,x,xm;
	double e=0.0;
	void nrerror();

	a=((ax < cx) ? ax : cx);
	b=((ax > cx) ? ax : cx);
	x=w=v=bx;
	fw=fv=fx=(*f)(x);
	for (iter=1;iter<=ITMAX;iter++) {
//		printf("iter=%d\n",iter);
		xm=0.5*(a+b);
		tol2=2.0*(tol1=tol*fabs(x)+ZEPS);
		if (fabs(x-xm) <= (tol2-0.5*(b-a))) {
			*xmin=x;
			return fx;
		}
		if (fabs(e) > tol1) {
			r=(x-w)*(fx-fv);
			q=(x-v)*(fx-fw);
			p=(x-v)*q-(x-w)*r;
			q=2.0*(q-r);
			if (q > 0.0) p = -p;
			q=fabs(q);
			etemp=e;
			e=d;
			if (fabs(p) >= fabs(0.5*q*etemp) || p <= q*(a-x) || p >= q*(b-x))
				d=CGOLD*(e=(x >= xm ? a-x : b-x));
			else {
				d=p/q;
				u=x+d;
				if (u-a < tol2 || b-u < tol2)
					d=SIGN(tol1,xm-x);
			}
		} else {
			d=CGOLD*(e=(x >= xm ? a-x : b-x));
		}
		u=(fabs(d) >= tol1 ? x+d : x+SIGN(tol1,d));
		fu=(*f)(u);
		if (fu <= fx) {
			if (u >= x) a=x; else b=x;
			SHFT(v,w,x,u)
			SHFT(fv,fw,fx,fu)
		} else {
			if (u < x) a=u; else b=u;
			if (fu <= fw || w == x) {
```

```
                v=w;
                w=u;
                fv=fw;
                fw=fu;
            } else if (fu <= fv || v == x || v == w) {
                v=u;
                fv=fu;
            }
        }
    }
    nrerror("Too many iterations in BRENT");
    *xmin=x;
    return fx;
}

#undef ITMAX
#undef CGOLD
#undef ZEPS
#undef SIGN

double Brent1(double (*func)(double), double tol, double *xm)
{
    double ax,bx,cx,fa,fb,fc,br,xmin;

    ax=1.0; bx=2.0; cx=3.0;
    mnbrak(&ax,&bx,&cx,&fa,&fb,&fc,func);
    br=brent(ax,bx,cx,func,tol,&xmin);
    *xm=xmin;
    return(br);
}
```

// **Dat1.h**

```
//#include <stdlib.h>
//#include <string.h>

FILE *fp[28];

void datot(int N_dat)                    // otvaranje N_dat datoteka
{
    int ibr;
    char init_name[8], file_name[12], ibr_str[5];

    printf ("\nUnesite inicijalno ime datoteke: ");
    scanf ("%s", init_name);
    for (ibr=1; ibr<=N_dat; ibr++){
        strcpy (file_name, init_name);
        if (ibr < 10)
            sprintf (ibr_str, "0%d", ibr);
        else if (ibr < 100)
            sprintf (ibr_str, "%d", ibr);
        else
            sprintf (ibr_str, "%d", ibr);
        strcat (file_name, ibr_str);
        strcat (file_name, ".dat");
        fp[ibr] = fopen (file_name, "w");
    }
}

void datzt(int how)                      // zatvaranje N_dat datoteka
{
    int ibr;

    for (ibr=1; ibr<=how; ibr++){
        fclose(fp[ibr]);
        printf ("Datoteka %d je zatvorena\n", ibr);
    }
}

void puni_dat1(double *x, double *y, int n1, int n2, int j)      // ponjenje datoteke vektorima x[i], y[i]
{
    int i;

    for(i=n1;i<=n2;i++)
        fprintf (fp[j],"%10.6lf  %10.6lf\n", x[i], y[i]);
}

void puni_toc(double x, double y, int j)                         // ponjenje datoteke skalarima x, y
{
    fprintf (fp[j], "%12.10lf %12.10lf\n", x, y);
}
```

// **Nrutil1.h**

```
#include <malloc.h>
#include <stddef.h>
#include <stdlib.h>

void nrerror(char error_text[])
{
	void exit();

	fprintf(stderr,"Numerical Recipes run-time error...\n");
	fprintf(stderr,"%s\n",error_text);
	fprintf(stderr,"...now exiting to system...\n");
	exit(0);
}

double *dvector(int nl,int nh)
{
	double *v;

	v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
	if (!v) nrerror("allocation failure in dvector()");
	return v-nl;
}

double **dmatrix(int nrl,int nrh,int ncl,int nch)
{
	int i;
	double **m;

	m=(double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
	if (!m) nrerror("allocation failure 1 in dmatrix()");
	m -= nrl;

	for(i=nrl;i<=nrh;i++) {
		m[i]=(double *) malloc((unsigned) (nch-ncl+1)*sizeof(double));
		if (!m[i]) nrerror("allocation failure 2 in dmatrix()");
		m[i] -= ncl;
	}
	return m;
}

void free_dvector(double *v,int nl)
{
	free((char*) (v+nl));
}


void free_dmatrix(double **m, int nrl, int nrh, int ncl, int nch)
{
	int i;

	for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
	free((char*) (m+nrl));
}
```