DAFTAR PUSTAKA

Aydin, A., & Sisman, A. (2016). Discrete density of states. In *Physics Letters, Section A: General, Atomic and Solid State Physics* (Vol. 380, Issue 13). Elsevier B.V. https://doi.org/10.1016/j.physleta.2016.01.034

Bogatyreva, N. S., Osypov, A. A., & Ivankov, D. N. (2009). KineticDB: A database of protein folding kinetics. *Nucleic Acids Research*, *37*(SUPPL. 1), 342–346. https://doi.org/10.1093/nar/gkn696

Dill, K. A. (1985). Theory for the Folding and Stability of Globular Proteins. *Biochemistry*, *24*(6), 1501–1509. https://doi.org/10.1021/bi00327a032

Farris, A. C. K., Seaton, D. T., & Landau, D. P. (2021). Effects of lattice constraints in coarse-grained protein models. *Journal of Chemical Physics*, *154*(8). https://doi.org/10.1063/5.0038184

Habeck, M. (2007). Bayesian reconstruction of the density of states. *Physical Review Letters*, *98*(20), 1–4. https://doi.org/10.1103/PhysRevLett.98.200601

Kumaunang, M. (2010). KARAKTERISASI PRODUK GEN PENGKODE Chaperone Bacillus sp. RP1. *Chemistry Progress*, *3*(1), 52–56.

Lau, K. F., & Dill, K. A. (1989). A Lattice Statistical Mechanics Model of the Conformational and Sequence Spaces of Proteins. *Macromolecules*, *22*(10), 3986–3997. https://doi.org/10.1021/ma00200a030

Montecinos, A., Loyola, C., Peralta, J., & Davis, S. (2021). Microcanonical potential energy fluctuations and configurational density of states for nanoscale systems. *Physica A: Statistical Mechanics and Its Applications*, *562*, 125279. https://doi.org/10.1016/j.physa.2020.125279

Musdalifah, Safrullah, & Surungan, T. (2023). Studi perubahan sifat struktur dan termodinamik pelipatan protein model H0P menggunakan simulasi monte carlo dengan algoritma wang-landau. In prosiding seminar nasional fisika, 2(1), 568-576

Pattanasiri, B., Li, Y. W., Landau, D. P., Wüst, T., & Triampo, W. (2013). Thermodynamics and structural properties of a confined HP protein determined by Wang-Landau simulation. *Journal of Physics: Conference Series*, *454*(1). https://doi.org/10.1088/1742-6596/454/1/012071

Shi, G., 2016. Replica-Exchange Wang–Landau Simulations of Lattice Proteins for The Understanding of The Protein Folding Problem. University of Georgia

Shi, G., Farris, A. C. K., Wüst, T., & Landau, D. P. (2016). Folding in a semi-flexible lattice model for Crambin. *Journal of Physics: Conference Series*, *686*(1), 0–7. https://doi.org/10.1088/1742-6596/686/1/012001

Shi, G., Wüst, T., & David, P. L. (2017). Replica Exchange Wang - Landau Simulation of Lattice Protein Folding Funnels. *Journal of Physics: Conference Series*, *905*(1). https://doi.org/10.1088/1742-6596/905/1/012016

Shi, G., Wüst, T., & Landau, D. P. (2018). Elucidating thermal behavior, native contacts, and folding funnels of simple lattice proteins using replica exchange Wang-Landau sampling. *Journal of Chemical Physics*, *149*(16). https://doi.org/10.1063/1.5026256

Shi, G., Wüst, T., Li, Y. W., & Landau, D. P. (2015). Protein folding of the HOP model: A parallel Wang - Landau study. *Journal of Physics: Conference Series*, *640*(1), 0–6. https://doi.org/10.1088/1742-6596/640/1/012017

Shimizu, H. (2004). Estimation of the density of states by multicanonical molecular dynamics simulation. *Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics*, *70*(5), 7. https://doi.org/10.1103/PhysRevE.70.056704

Vogel, T., Li, Y. W., Wüst, T., & Landau, D. P. (2013). Generic, hierarchical framework for massively parallel Wang-Landau sampling. *Physical Review Letters*, *110*(21), 1–5. https://doi.org/10.1103/PhysRevLett.110.210603

Wang, F., & Landau, D. P. (2001). Efficient, multiple-range random walk algorithm to calculate the density of states. *Physical Review Letters*, *86*(10), 2050–2053. https://doi.org/10.1103/PhysRevLett.86.2050

LAMPIRAN

A. Program Simulasi Pelipatan Protein

```
// This is a program to calculate exact DOS of the 3D HP model
// note : yang diubah adalah jumlah monomer N, urutan asam
amino, for-for, rentang energi

#include <stdio.h>

void update(int v1[][3], int ind, int arrow);

int main() {
    int N = 18; // number of monomers
    int amino_sequence[] =
{0,1,0,1,0,0,1,0,1,2,0,0,2,2,2,1,0,0}; // amino acid sequence of
the protein. H = 0, P =1, and 0 = 2
    // 0,0,1,2,2
    int dim = 3; // dimension of the protein lattice
    int deg = 6;

    //define 2d-array to store the coords of the direction of
fold of monomers in the lattice
    int dir[N-1][dim];
    // int savedir[N-1][dim];

    //set the initial x and z coords of the directions
    for (int i = 0; i < N-1; i++) {
        dir[i][0] = 0;
        dir[i][1] = 1;
        dir[i][2] = 0;
    }

    // array for unit vector s and o that point out the
direction of the folding
    int cf[3] = {0,0,0};

    // variables to calculate the energy of the configuration
    int eHH = 4;
    int nHH = 0;
    int eH0 = 2;
```

```
    int nH0 = 0;
    int etheta = 1;
    int ntheta = 0;
    int totalEnergy = 0;
    int flagoverlap = 0; // if the configuration overlap we do
not need to calculate the energy

    // dos variable to record the energy
    int energy_interval[] =  {0, 100, 1}; //the range of energy
from 0 to 20 with interval 1
    int range = (energy_interval[1] - energy_interval[0]) *
energy_interval[2];
    long long int dos[range];
    for (int t = 0; t < range; t++) {
        dos[t] = 0;
    }
    int doscount = 0;
    int index = N-2;
    int flag = 0;
    int a[N-1];
    a[0] = -1;
    int tmpa[N-2];
    for (int i = 0; i < N-2; i++) {
        tmpa[i] = 0;
    }

    // variables to count the numbers overlap and valid
configurations
    long long int countoverlap = 0;
    long long int countvalid = 0;

    // for every possible pattern of the protein to fold
    for (a[1] = -3; a[1] <= 3; a[1] ++) {
        if(a[1] == 0 || a[1] == -a[0]) continue;
        for (a[2] = -3; a[2] <= 3; a[2] ++) {
            if(a[2] == 0 || a[2] == -a[1]) continue;
        for (a[3] = -3; a[3] <= 3; a[3] ++) {
            if(a[3] == 0 || a[3] == -a[2]) continue;
        for (a[4] = -3; a[4] <= 3; a[4] ++) {
            if(a[4] == 0 || a[4] == -a[3]) continue;
```

```
            for (a[5] = -3; a[5] <= 3; a[5] ++) {
                if(a[5] == 0 || a[5] == -a[4]) continue;
            for (a[6] = -3; a[6] <= 3; a[6] ++) {
                if(a[6] == 0 || a[6] == -a[5]) continue;
            for (a[7] = -3; a[7] <= 3; a[7] ++) {
                if(a[7] == 0 || a[7] == -a[6]) continue;
            for (a[8] = -3; a[8] <= 3; a[8] ++) {
                if(a[8] == 0 || a[8] == -a[7]) continue;
            for (a[9] = -3; a[9] <= 3; a[9] ++) {
                if(a[9] == 0 || a[9] == -a[8]) continue;
            for (a[10] = -3; a[10] <= 3; a[10] ++) {
                if(a[10] == 0 || a[10] == -a[9]) continue;
            for (a[11] = -3; a[11] <= 3; a[11] ++) {
                if(a[11] == 0 || a[11] == -a[10]) continue;
            for (a[12] = -3; a[12] <= 3; a[12] ++) {
                if(a[12] == 0 || a[12] == -a[11]) continue;
            for (a[13] = -3; a[13] <= 3; a[13] ++) {
                if(a[13] == 0 || a[13] == -a[12]) continue;
            for (a[14] = -3; a[14] <= 3; a[14] ++) {
                if(a[14] == 0 || a[14] == -a[13]) continue;
            for (a[15] = -3; a[15] <= 3; a[15] ++) {
                if(a[15] == 0 || a[15] == -a[14]) continue;
            for (a[16] = -3; a[16] <= 3; a[16] ++) {
                if(a[16] == 0 || a[16] == -a[15]) continue;

                totalEnergy = 0;
                nHH = 0;
                nH0 = 0;
                ntheta = 0;
                flagoverlap = 0;

                for (int i = 1; i < N-1; i++) {
                    update(dir, i, a[i]);
                }

                // cek overlap
                for (int i = 0; i < N-1; i++) {
                    if(i != N-2 && (dir[i][0] == -dir[i+1][0]) &&
(dir[i][1] == -dir[i+1][1]) && (dir[i][2] == -dir[i+1][2])) {
                        flagoverlap = 1;
```

```
                    countoverlap++;
                    // printf("!!!Konfigurasi overlap!!! \n");
                    break;
                }
                for (int j = i + 3; j < N-1; j+=2 ) {
                    for (int k = i; k <= j; k++) {
                        cf[0] += dir[k][0];
                        cf[1] += dir[k][1];
                        cf[2] += dir[k][2];
                    }
                    if ((cf[0] == 0) && (cf[1] == 0) &&  (cf[2]
== 0)) {
                        flagoverlap = 1;
                        break;
                    }
                    cf[0] = 0; cf[1] = 0; cf[2] = 0;
                }
                if (flagoverlap) {
                    countoverlap++;
                    // printf("!!!Konfigurasi overlap!!! \n");
                    break;
                }
            }

            if (flagoverlap) continue;

            // calculate energy
            for (int i = 0; i < N-3; i++) {
                if (amino_sequence [i] == 0) {
                    for (int j = i+3; j < N ; j +=2) {
                        // komponen H-H
                        if (amino_sequence[j] == 0) {
                            for (int k = i; k <= j-1; k++) {
                                cf[0] += dir[k][0];
                                cf[1] += dir[k][1];
                                cf[2] += dir[k][2];
                            }
                            if ((cf[0]*cf[0] + cf[1]*cf[1] +
cf[2]*cf[2]) == 1 ) {
                                nHH++;
```

```
                        }
                        cf[0] = 0; cf[1] = 0;cf[2] = 0;
                    }

                    //komponen H - 0
                    if (amino_sequence[j] == 2) {
                        for (int k = i; k <= j-1; k++) {
                            cf[0] += dir[k][0];
                            cf[1] += dir[k][1];
                            cf[2] += dir[k][2];
                        }
                        if ((cf[0]*cf[0] + cf[1]*cf[1] +
cf[2]*cf[2]) == 1 ) {
                            nH0++;
                        }
                        cf[0] = 0; cf[1] = 0;cf[2] = 0;
                    }
                }
            }

            //komponen 0 - H
            if (amino_sequence [i] == 2) {
                for (int j = i+3; j < N ; j +=2) {
                    if (amino_sequence[j] == 0) {
                        for (int k = i; k <= j-1; k++) {
                            cf[0] += dir[k][0];
                            cf[1] += dir[k][1];
                            cf[2] += dir[k][2];
                        }
                        if ((cf[0]*cf[0] + cf[1]*cf[1] +
cf[2]*cf[2]) == 1 ) {
                            nH0++;
                        }
                        cf[0] = 0; cf[1] = 0;cf[2] = 0;
                    }
                }
            }
        }

        // calculate ntheta
```

```
            for (int i = 0; i < N-2; i++) {
                if (a[i] != a[i+1]) ntheta++ ;
            }



            countvalid++;


            totalEnergy = eHH*nHH + eH0*nH0s + etheta*ntheta;

            dos[totalEnergy] += 1;

        }}}}}}}}}}}}}}}
    }

    int sumdos = 0;
    for (int i = 0; i < range; i++) {
        if (dos[i] != 0) sumdos += dos[i];
    }

    double normdos[range];

    for(int j = 0; j < range; j++) {
        if (dos[j] != 0) normdos[j] = (double) (dos[j]) /
(double) (sumdos);
    }

    //print dos ke file output
    FILE *f;
    f = fopen("result1.out", "w+");
    for (int i = 0; i < range; i++) {
        if (dos[i] != 0) {
            printf("%d      %.8e      %lli \n", i, normdos[i],
dos[i]);
            fprintf(f, "%d      %.8e      %lli \n", i,
normdos[i], dos[i]);

        }
    }
    fprintf(f,"valid configuration = %lli  \n", countvalid);
```

```
    fprintf(f,"overlap configuration = %lli  \n", countoverlap);
    printf("valid configuration = %lli  \n", countvalid);
    printf("overlap configuration = %lli  \n", countoverlap);

    fclose(f);
}

void update(int v1[][3], int ind, int arrow) {
    switch (arrow) {

    // kanan
    case -1:
        v1[ind][0] = 0;
        v1[ind][1] = 1;
        v1[ind][2] = 0;
        break;

    case 1:
        v1[ind][0] = 0;
        v1[ind][1] = -1;
        v1[ind][2] = 0;
        break;

    // bawah
    case -2:
        v1[ind][0] = 1;
        v1[ind][1] = 0;
        v1[ind][2] = 0;
        break;

    case 2:
        v1[ind][0] = -1;
        v1[ind][1] = 0;
        v1[ind][2] = 0;
        break;

    // depan
    case -3:
        v1[ind][0] = 0;
        v1[ind][1] = 0;
```

```
            v1[ind][2] = 1;
            break;

        case 3:
            v1[ind][0] = 0;
            v1[ind][1] = 0;
            v1[ind][2] = -1;
            break;

        }

    }
```