Cvičení 2 – Optimalizace pomocí genetického algoritmu

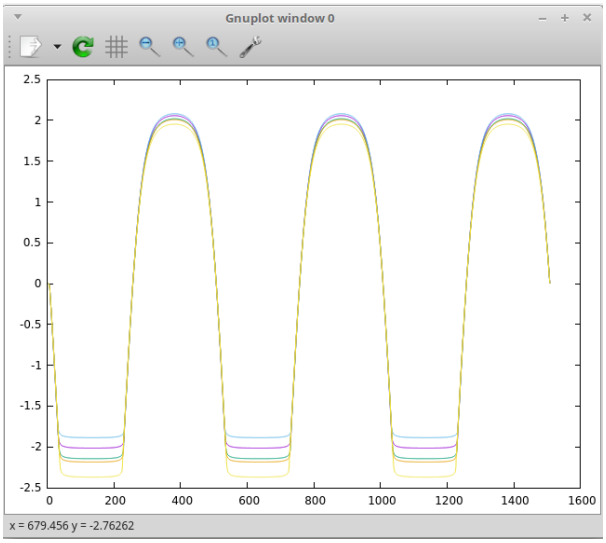
Jméno: Kateřina Fořtová (xforto00)

Charakteristika experimentu: Je zadán elektrický obvod, máme za úkol nastavit hodnoty součástek tak, aby bylo dosaženo určité chování. Můžeme měnit hodnoty rezistorů R1, R2, Rc, Re a kondenzátorů Cin, Cout, Ce. Jedná se o zesilovač s jedním bipolárním tranzistorem, vstupem je signál s malou amplitudou, na výstupu chceme sinusový signál s amplitudou 2 V. Každého jedince ohodnotíme pomocí simulace v NGSPICE. Výsledkem simulace je diskrétní řada vzorků (3 periody) fce[x]. Chceme vyhodnotit, jak moc se blíží ideálnímu průběhu -2sin(x).

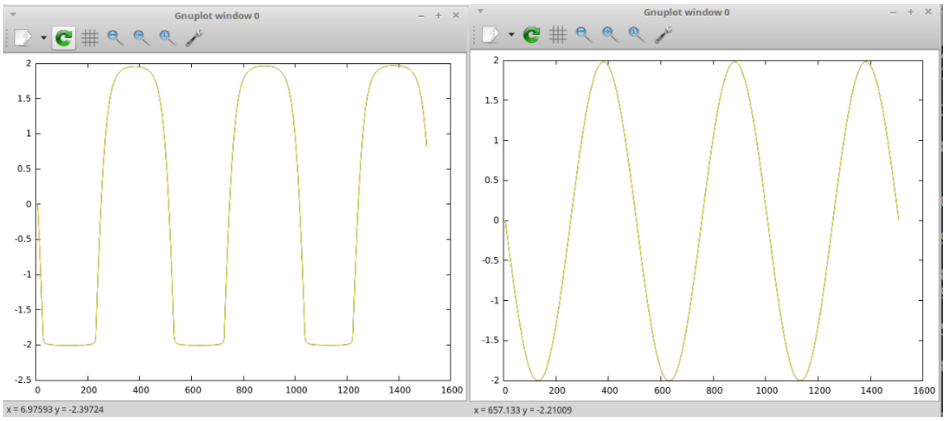
Experimentování s fitness funkcí a parametry:

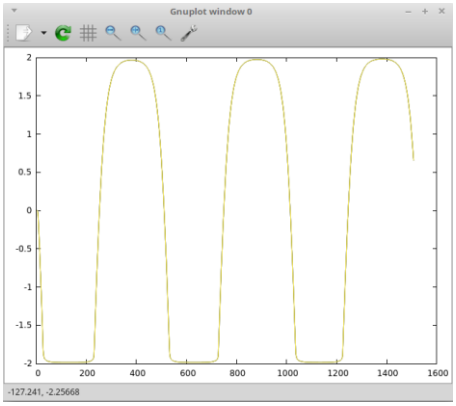
```
double expxxx(double x) {return exp(x)*x-x;} // http://fooplot.com/exp(x)*x-x

double fitness (std::vector<double> sim_res) {
    int samples = sim_res.size();
    const double gain = 2;

    return expxxx(sim_res[samples*9/12] - -gain) + expxxx(-sim_res[samples*11/12] +gain);
}
```

```
const int NDIM = 7; // pocet dimenzi (velikost problemu)
const int PROBLEM_MIN = 1;       // dostupne hodnoty soucastek
const int PROBLEM_MAX = 1000000; // rezistory 1 Ohm -- 1000 kOhm, kapacity 1 nF -- 1000 u

const int POPSIZE = 20; // velikost populace
const int SELECTION = 3; // velikost vybrane casti populace
const int ELITISM = 1; // kolik nejlepsich jedincu nechat beze zmeny
const int MAXGEN = 100; // maximalni pocet generaci
const double P_CROSSOVER = 0.0; // pravdepodobnost krizeni
const double P_MUTATION_I = 0.99; // pravdepodobnost mutace jedince
const double P_MUTATION_G = 0.4; // pravdepodobnost mutace genu v jedinci
```

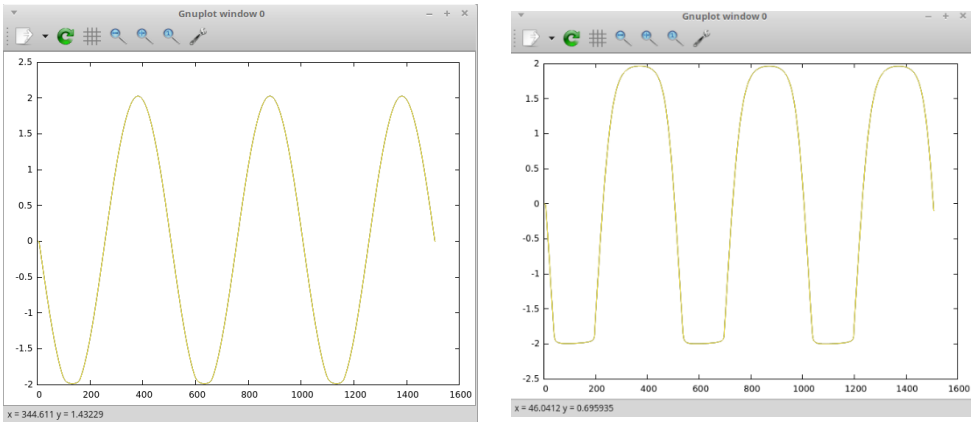



```
const int NDIM = 7; // pocet dimenzi (velikost problemu)
const int PROBLEM_MIN = 1;       // dostupne hodnoty soucastek
const int PROBLEM_MAX = 1000000; // rezistory 1 Ohm -- 1000 kOhm, kapacity 1 nF -- 1000 u

const int POPSIZE = 100; // velikost populace
const int SELECTION = 10; // velikost vybrane casti populace
const int ELITISM = 1; // kolik nejlepsich jedincu nechat beze zmeny
const int MAXGEN = 100; // maximalni pocet generaci
const double P_CROSSOVER = 0.4; // pravdepodobnost krizeni
const double P_MUTATION_I = 0.7; // pravdepodobnost mutace jedince
const double P_MUTATION_G = 0.7; // pravdepodobnost mutace genu v jedinci
```



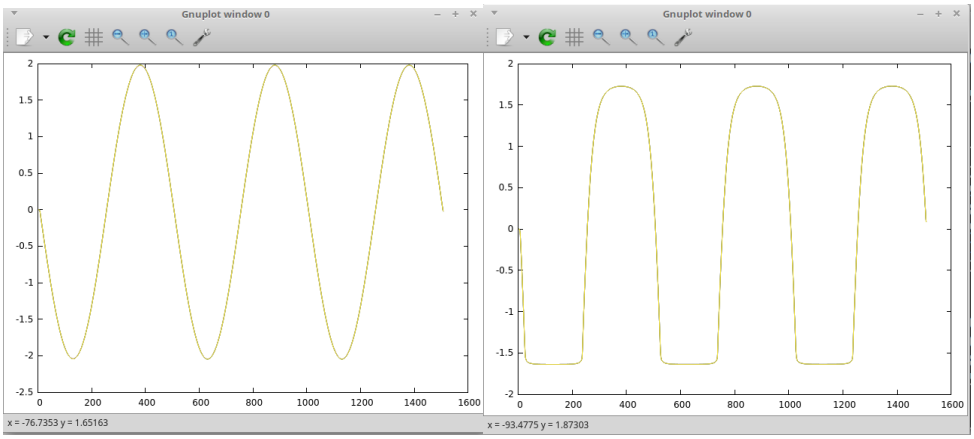



```
const int NDIM = 7; // pocet dimenzi (velikost problemu)
const int PROBLEM_MIN = 1;        // dostupne hodnoty soucastek
const int PROBLEM_MAX = 1000000; // rezistory 1 Ohm -- 1000 kOhm, kapacity 1 nF -- 1000 u

const int POPSIZE = 100; // velikost populace
const int SELECTION = 10; // velikost vybrane casti populace
const int ELITISM = 3; // kolik nejlepsich jedincu nechat beze zmeny
const int MAXGEN = 80; // maximalni pocet generaci
const double P_CROSSOVER = 0.4; // pravdepodobnost krizeni
const double P_MUTATION_I = 0.7; // pravdepodobnost mutace jedince
const double P_MUTATION_G = 0.7; // pravdepodobnost mutace genu v jedinci
```



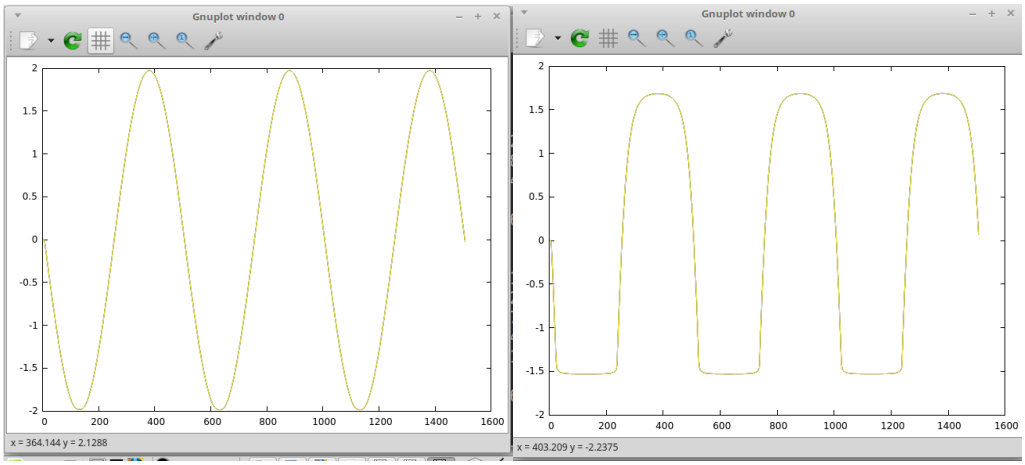

```
double expxxx(double x) {return exp(x)*x-x;} // http://fooplot.com/exp(x)*x-x

double fitness (std::vector<double> sim_res) {
    int samples = sim_res.size();
    const double gain = 2;

    //return expxxx(-fabs(sim_res[samples*11/12]) + gain) + expxxx(-fabs(sim_res[samples*9/12]) + gain);

    double sum_err = 0;
    for (int s=0; s<samples; s++) {
        //sum_err += pow( sim_res[s] - gain * -sin(2*3.14159265358979323846264338327950288 4 * s/(samples/3)) ,2); // minimalizace souctu ctvercu
        //sum_err += exp(fabs( sim_res[s] - gain * -sin(2*3.14159265358979323846264338327950288 4 * s/(samples/3)) ))-1; // exponencialni chyby
        sum_err += expxxx( -fabs( sim_res[s] ) +gain*fabs( -sin(2*3.14159265358979323846264338327950288 4 * s/(samples/3)) )); // nesymetricke odc
    }
    return sum_err;

}
```

```
const int NDIM = 7; // pocet dimenzi (velikost problemu)
const int PROBLEM_MIN = 1;       // dostupne hodnoty soucastek
const int PROBLEM_MAX = 1000000; // rezistory 1 Ohm -- 1000 kOhm, kapacity 1 nF -- 1000 uF

const int POPSIZE = 100; // velikost populace
const int SELECTION = 10; // velikost vybrane casti populace
const int ELITISM = 3; // kolik nejlepsich jedincu nechat beze zmeny
const int MAXGEN = 80; // maximalni pocet generaci
const double P_CROSSOVER = 0.4; // pravdepodobnost krizeni
const double P_MUTATION_I = 0.7; // pravdepodobnost mutace jedince
const double P_MUTATION_G = 0.7; // pravdepodobnost mutace genu v jedinci
```




```
double expxxx(double x) {return exp(x)*x-x;} // http://fooplot.com/exp(x)*x-x

double fitness (std::vector<double> sim_res) {
    int samples = sim_res.size();
    const double gain = 2;

    //return expxxx(-fabs(sim_res[samples*11/12]) + gain) + expxxx(-fabs(sim_res[samples*9/12]) + gain);

    double sum_err = 0;
    for (int s=0; s<samples; s++) {
        //sum_err += pow( sim_res[s] - gain * -sin(2*3.14159265358979323846264338327950288 4 * s/(samples/3)) ,2); // minimalizace souctu ctvercu
        sum_err += exp(fabs( sim_res[s] - gain * -sin(2*3.14159265358979323846264338327950288 4 * s/(samples/3)) ))-1; // exponencialni chyby
        //sum_err += expxxx( -fabs( sim_res[s] ) +gain*fabs( -sin(2*3.14159265358979323846264338327950288 4 * s/(samples/3)) )); // nesymetricke c
    }
    return sum_err;
}
```

```
const int NDIM = 7; // pocet dimenzi (velikost problemu)
const int PROBLEM_MIN = 1;       // dostupne hodnoty soucastek
const int PROBLEM_MAX = 1000000; // rezistory 1 Ohm -- 1000 kOhm, kapacity 1 nF -- 1000 uF

const int POPSIZE = 50; // velikost populace
const int SELECTION = 3; // velikost vybrane casti populace
const int ELITISM = 1; // kolik nejlepsich jedincu nechat beze zmeny
const int MAXGEN = 100; // maximalni pocet generaci
const double P_CROSSOVER = 0.0; // pravdepodobnost krizeni
const double P_MUTATION_I = 0.7; // pravdepodobnost mutace jedince
const double P_MUTATION_G = 0.7; // pravdepodobnost mutace genu v jedinci
```



Závěr: Nejlepšího průběhu opakovaně dosahovala první fitness funkce s populací 100 jedinců, pravděpodobností křížení 40 % a mutace 70 %, elitismem u jednoho nebo tří jedinců a počtem 100 nebo 80 generací. Další fitness funkce využívající exponenciální chyby nebo nesymetrické odchylky měli i při experimentování s více nastaveními parametrů nestálý průběh při více bězích.