Technická zpráva

ÚTIA Akademie věd České republiky
Ústav teorie informace a automatizace AV ČR, v.v.i.

Matlab ADSL Toolbox ver. 11

Ing. Tomáš Mazanec, Ing. Antonín Heřmánek, Ph.D.
{*mazanec,hermanek*}*@utia.cas.cz*

Obsah

1 Úvod **1**

2 Top level skript - do_tx_rx_rand11_adslsim.m **1**

3 Vysílací část **2**
3.1 txDMT2.m 2
3.2 txNQAMfce3.m 3

4 Přijímací část **3**
4.1 rxDMT2.m 3
4.2 rx_QAM_detection.m 5

5 Bitload algoritmus - waterfill.m **5**

6 Algoritmy ekvalizérů TEQ **5**

7 Obsah CD-ROM **6**

8 Výpisy m-skriptů **7**
8.1 txDMT2.m 7
8.2 txNQAMfce3.m 9
8.3 rxDMT2.m 22
8.4 rx_QAM_detection.m 23
8.5 waterfill.m 24

Revize

Revize	Datum	Autor	Popis změn v dokumentu
0	23.2.2007	Mazanec	Vytvoření dokumentu

1 Úvod

Tento dokument popisuje zdrojové kódy *Simulátoru fyzické vrstvy ADSL modemu* a je společně s [3] a [4] dokumentací k ADSL toolboxu pro Matlab vyvíjeného v rámci projektu *Specifikace kvalitativních kritérií a optimalizace prostředků pro vysokorychlostní přístupové sítě* (Grantová agentura Akademie věd České republiky, číslo 1ET300750402). Popisovaná **verze simulátoru** je **11**.

Vlastní ADSL toolbox se sestává z několika skupin funkcí:

- funkce pro vysílací část – tyto funkce mají prefix ***tx****Name*
- funkce pro modelování přenosového prostředí – tyto funkce mají prefix ***tx_rx_****Name*
- funkce pro přijímací část – tyto funkce mají prefix ***rx****Name*
- skupina funkcí realizujících ekvalizaci přijímače ADSL – tyto funkce jsou pojmenovány dle názvu (zkratky) algoritmu

ADSL toolbox vyžaduje instalaci *Signal Processing*, *Optimization* a *Communication toolboxu* pro Matlab verze 14 a vyšší.

2 Top level skript - do_tx_rx_rand11_adslsim.m

Skript do_tx_rx_rand11_adslsim.m obsahuje příklad použití funkcí ADSL toolboxu. Tento skript modeluje inicializační část, přenos bloku dat a ekvalizér dle definovaných vstupních parametrů.

```
[br_Mbps, BER, MMSE, params] = do_tx_rx_rand11_adslsim( SNR ,...
                                                        eq_type ,...
                                                        Ns ,...
                                                        loop_num ,...
                                                        noisemodel ,...
                                                        lower_bitload)
```

Vstup:

SNR	[dB], poměr signálu ku přidanému šumu, druh šumu dle `noisemodel`
eq_type	['string'], algoritmus ekvalizéru: UEC, UTC, MinISI, MBR, MDS, CNA, MSSNR nebo MGSNR
Ns	počet přenesených datových symbolů
loop_num	(1...6), typ referenčního ADSL kanálu (CSA # loop)
noisemodel	(0 nebo 1), 0...AWGN, 1...Model č.1
lower_bitload	(0 nebo 1), 0...Ne, 1...Ano, Korekce bitload algoritmu snížením navrženého bitloadu

Výstup:

Defaultní hodnoty: br_Mbps = 0; BER = NaN; MMSE = NaN;

br_Mbps	datový tok dosažený při simulaci
BER	chybovost (Bit Error Ratio) přenosu datových symbolů při simulaci
MMSE	pro MSE algoritmy - orientační údaj o dosažení optima algoritmu

Pevné a interní parametry ve výstupní struktuře `params`

params.Ntused	počet použitých tónů (subnosných), určuje bitload algoritmus dle zvoleného modelu kanálu
params.cplen	40, délka cyklického prefixu datových symbolů
params.Gam	Gamma [dB], interní konstanta, souvisí s potřebným SNR pro danou N-stavovou QAM
params.Gamgap	Γ [dB], dtto
params.Codgain	Coding gain [dB], dtto
params.Margin	Margin [dB], dtto
params.power	TX Power [dBm], interní konstanta pro reálné škálování signálů v simulaci
params.Nb	délka cílové odezvy (TIR) pro některé algoritmy ekvalizace
params.Nw	délka vlastního ekvalizéru (řád filtru)
params.Ntu	maska použitých tónů
params.bn	vyžití jednotlivých tónů pro přenos, n-bitů na daný tón
params.delay	optimální systémové zpoždění
params.bDMT0	odhad kapacity kanálu: [bits/symbol], teoretický odhad kapacity kanálu
params.RDMT0	odhad kapacity kanálu: [Mbps], teoretický odhad max. přenosové rychlosti kanálu
params.SNRgeo	odhad kapacity kanálu: [dB], poměrná hodnota daná geometrický průměrem odezvy kanálu vůči úrovni šumu

3 Vysílací část

3.1 txDMT2.m

- Funkce která provádí DMT modulaci.
- Volá funkci txNQAMfce3.m.

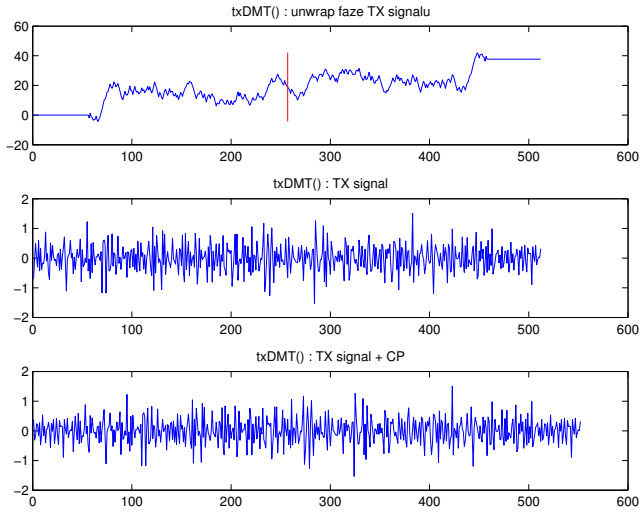
```
[txsigcp, txsig, txbits, txbbits, txsymb] = txDMT2(bindata ,...
                                                   b ,...
                                                   Nd ,...
                                                   Ntused ,...
                                                   cplen ,...
                                                   plot_tx)
```

  - bindata: vektor vstupních dat délky $\geq$ txbits, pole typu logical.

- b: vektor přiřazení počtu bitů na jednotlivé tóny, (délka: Nt, hodnoty: 1 až bmax pro použitý tón nebo záporné pro nepoužitý tón).
- Nd: počet bodů FFT, (dle doporučení 512).
- Ntused: počet použitých tónů, ($\leq$ Nt $\leq$ Nd/2 $-$ 1).
- cplen: délka cyklického prefixu, (dle ITU-T: cplen=40).
- plot_tx: zapíná vykreslování signálů TX bloku, (0 nebo 1).
- txsigcp: vektor výstupního signálu s cyklickým prefixem, (délka: Nd+cplen).
- txsig: vektor výstupního signálu, (délka: Nd).
- txbits: počet vysílaných bitů.
- txbbits: počet bitů na tón vs. počet vyslaných bitů, (pole 2$\times$bmax).
- txsymb: vysílaný DMT symbol (pole komplexních N-QAM)

3.2 txNQAMfce3.m

- Funkce která provádí N-stavovou amplitudově-fázovou modulaci (N-QAM).
- `block_qam = txQAMfce3(data, b)`
  - data: vstupní hodnota - binární číslo - typu pole znaků.
  - b: skalár - počet bitů na daný tón, (1 až bmax).
  - block_qam: komplexní číslo reprezentující N-QAM stav daný hodnotou vstupu.



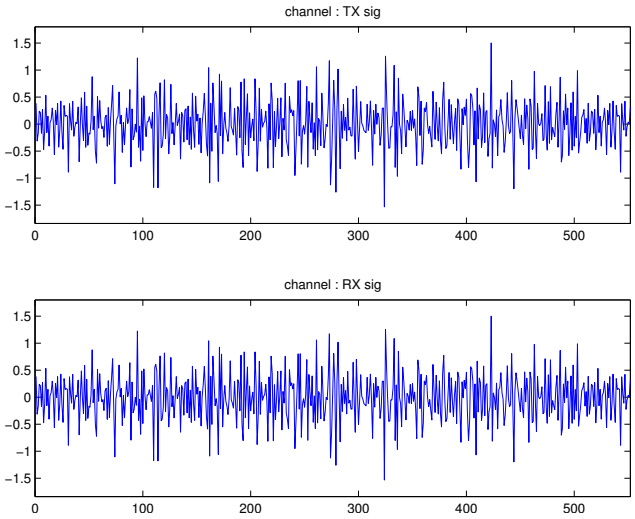
Obrázek 1: Vysílaný DMT symbol: fáze, bez CP a s doplněným CP

4 Přijímací část

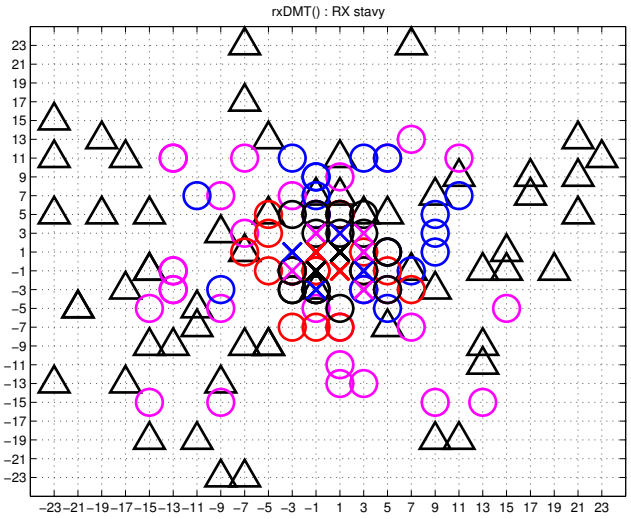
4.1 rxDMT2.m

- Funkce která provádí DMT demodulaci a korekci symbolů ve frekvenční ekvalizaci (FEQ).
- Volá funkci rx_QAM_detection.m.

- `[rxbindata] = rxDMT2(rxsig, b, txbits, Nd, Ntused, cplen, feq, plot_rx)`
  - rxsig: vektor - modulovaný signál na vstupu přijímače.
  - b: vektor přiřazení počtu bitů na jednotlivé tóny, (délka: Nt, hodnoty: 1 až bmax pro použitý tón nebo záporné pro nepoužitý tón).
  - txbits: počet vyslaných bitů.
  - Nd: počet bodů FFT, (dle dop.: 512).
  - Ntused: počet použitých tónů, ($\leq$ Nt $\leq$ Nd$/2-1$).
  - cplen: délka cyklického prefixu, (dle ITU-T: 40).
  - feq: vektor koeficientů FEQ (frekv. ekvalizér řádu M=1).
  - plot_rx: zapíná vykreslování signálů RX bloku, (0 nebo 1).
  - rxbindata: vektor - přijatý blok dat po demodulaci, (pole typu logical).



Obrázek 2: Přijatý DMT symbol odpovídající vysílanému (časový průběh)



Obrázek 3: Detekované přijaté N-QAM stavy

4.2 rx_QAM_detection.m

- Funkce která provádí detekci symbolů N-stavové QAM hledáním min. vzdálenosti.
- `[DMT_symbols, index_QAM, dmin] = rx_QAM_detection(qamin, b, Qs, rx_plot)`
  - qamin: přijatý DMT symbol (pole N-QAM stavů).
  - b: vektor přiřazení počtu bitů na jednotlivé tóny (délka: Nt, 1 až bmax).
  - Qs: matice všech QAM konstelací pro $N = 2^1 \cdots 2^{bmax}$.
  - rx_plot: zapíná vykreslování symbolů a min. vzdáleností, (0 nebo 1).
  - DMT_symbol: pole detekovaných N-QAM stavů (komplexní).
  - index_QAM: pole indexů matice Qs detekovaných N-QAM stavů (hodnoty numericky odpovídají kódovaným hodnotám).
  - dmin: pole zjištěných minimálních vzdáleností.

5 Bitload algoritmus - waterfill.m

- Funkce konfiguruje bitload ADSL kanálu metodou Rate-Adaptive Waterfilling.
- `[Enlv, bn] = waterfill(Sh, Sn, InputPower)`
  - Sh: přenosová odezva kanálu (ve smyslu PSD).
  - Sn: výkonová spektrální hustota šumu (tj. PSD).
  - InputPower: interní konstanta pro reálné škálování signálů (např.: 20dBm)
  - Enlv: vektor výsledných energií, hodnota na každý tón.
  - bn: vektor přiřazení bitů, n-bitů na každý tón.

6 Algoritmy ekvalizérů TEQ

Soubor funkcí pro návrh ekvalizačních filtrů.

- UTC: Unit Tap Constraint na MSE algoritmus, `utc.m`,
  závisí na: `correlations.m`
  `[bopt, wopt, d, MSE, iopt, Dv]=utc(trainingSignal, RxTraining, Nb, Nw, Dmin,... Dmax, 0)`
- UEC: Unit Energy Constraint na MSE algoritmus, `uec.m`,
  závisí na: `correlations.m , eigen.m`
  `[bopt, wopt, d, MSE, Dv]=uec(trainingSignal, RxTraining, Nb, Nw, Dmin, Dmax, 0)`
- MSSNR: Max Shortening SNR, `mssnr.m`,
  `[wopt, d, Dv] = mssnr(h, cplen, Nw, Dmin, Dmax, 0)`
- MinISI: Min Intersymbol Interference, `minisi.m`,
  závisí na: `obje.m , geosnr.m , maxeig.m`
  `[wopt, d, Dv, retval] = minisi(Sx, Sn, Sh, h, Nd, Nb, Nw, Dmin, Dmax,... usedChannels, 0)`

- MBR: Maximizing BitRate, `mbr_adv.m`,
  závisí na: `obje.m , geosnr.m`
  ```
  [wopt, d, Dv, retval] = mbr_adv(Sx, Sn, usedChannels, h, Nd, Nb, Nw, Dmin,...
  Dmax, Wsub, gamgap, Codgain, Margin, numIter, 0)
  ```
- MGSNR: Max Geometric SNR, `geo.m`,
  závisí na: `correlations.m , objective.m , objectiveconfun.m`
  ```
  [bopt, wopt, d, MSE, Dv, retval] = geo(trainingSignal, RxTraining, Nb, Nw, Nd,...
  Dmin, Dmax, MSEmax, Binit, 0, usedChannels)
  ```
- MDS: Min Delay Spread, `mds.m`,
  závisí na: `mdsobj.m`
  ```
  [wopt retval] = mds( winit, h, Nd, Nw, iter)
  ```
- CNA: Carrier Nulling Algorithm, `cna.m`,
  závisí na: `cnaobj.m`
  ```
  [wopt retval] = cna(winit, RxTraining, Nd, Nw, Ntu, iter)
  ```

7 Obsah CD-ROM

Adresářová struktura přiloženého CD je následující:

```
CD-ROM - doc -- Dokumentace
       - src -- Zdrojové kódy pro Matlab a datové soubory s modely ADSL kanálů
```

8 Výpisy m-skriptů

8.1 txDMT2.m

```
function    [txsigcp, txsig, txbits, txbbits, txsymb] = txDMT2(bindata,...
                                                b, Nd, Ntused, cplen, plot_tx)
%
% txsigcp .. signal to trasmit with cp
% txsig ... signal to transmit without cp
% txbits ... number of transmitted bits
% txbbits ... transmitted bits vs. bit per tone
% txsymb ... QAM syblols to transmit in complex coordinates
% bindata ... input vector of length >= txbits, array of type logical
% b ... bit-constelation vector, values from 1 to bmax (1 to 15)
%       negative value means unused tone
% Nd ... ifft length
% Ntused ... no. used tones
% cplen ... CP length
% plot_tx ... 0 or 1
%

if nargin ~= 6
    error('need all 6 inputs')
    return;
end

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Nt = Nd/2 -1;

block_qam = zeros(Nt, 1);

cnt=0;
ind1=0; ind2=0;
for ii=1:Nt
    if b(ii) > 0
        ind1 = ind2 + 1;
        ind2 = ind1 + b(ii) - 1;
        bindataseg = num2str( bindata(ind1:ind2)' , '%d');
        block_qam(ii) = txNQAMfce3(bindataseg, b(ii) );
    else
        block_qam(ii) = 0;
        cnt = cnt+1;
    end
end

if cnt ~= (Nt - Ntused)
    warning('txDMT: Bad control sum of unused tones.');
    cnt
end
```

```
bmax = max(b);

% count b-bits
txbbits = zeros(bmax,2);
for ii=1:bmax
    txbbits(ii,1) = ii;
    txbbits(ii,2) = sum(b==ii);
end

txbits = ind2;

% ifft() QAM fo DMT
block_ifft = zeros(Nd,1);
block_ifft(1) = 0;
block_ifft(2:Nd/2) = block_qam;
block_ifft(Nd/2+1) = 0;
block_ifft(Nd/2+2:Nd) = conj(block_qam(end:-1:1));

txsymb=block_ifft;

txsig = real((1/sqrt(Nd))*ifft(block_ifft));
txsigph = unwrap(angle(block_ifft));

cp = txsig( (end-cplen+1): end);
txsigcp = [cp ; txsig];

if plot_tx == 1

    meantxsig = mean(txsig)
    figure;
        subplot(311);
        plot(txsigph,'b');
        hold on;
        plot([Nd/2+1 Nd/2+1],[min(txsigph) max(txsigph)],'r');
        hold off;
        title('txDMT() : unwrap faze TX signalu');
        subplot(312)
        plot(txsig)
        title('txDMT() : TX signal')
        subplot(313);
        plot(txsigcp);
        title('txDMT() : TX signal + CP')
end

return
```

8.2 txNQAMfce3.m

```
function    block_qam = txNQAMfce3(data, b)

% input can be either vector or matrix

if nargin ~= 2
    error('need both 2 inputs')
    return;
end

% number of matrix rows
Nn = length(data(:,1));

% %%%%%%%%%%%%%%%%%%%%
switch (b)

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 2-stav
    case 1

X = bin2dec(data(:,1)) * -2 + 1;
Y = X;

block_qam = X + Y.*i;

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 4-stav = QAM
    case 2

X = bin2dec(data(:,1)) * -2 + 1;
Y = bin2dec(data(:,2)) * -2 + 1;

block_qam = X + Y.*i;

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 8-QAM

    case 3

X = bin2dec(data(:,2)) * -2 + 1;
Y = bin2dec(data(:,3)) * -2 + 1;
% %% blokove
% 4
X = X - 4 .* ((data(:,1)=='1') & (data(:,2)=='0') & (data(:,3)=='0'));
% 5
Y = Y + 4 .* ((data(:,1)=='1') & (data(:,2)=='0') & (data(:,3)=='1'));
% 6
Y = Y - 4 .* ((data(:,1)=='1') & (data(:,2)=='1') & (data(:,3)=='0'));
% 7
```

```
X = X + 4 .* ((data(:,1)=='1') & (data(:,2)=='1') & (data(:,3)=='1'));

block_qam = X + Y.*i;

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 16-QAM

    case 4

sx = bin2dec( data(:,1) );
sy = bin2dec( data(:,2) );
vx = bin2dec( data(:,3) );
vy = bin2dec( data(:,4) );

vxun = bitxor(sx, vx);
vyun = bitxor(sy, vy);

vxun = bitor( bitshift(vxun,1), ones(Nn,1) );
vyun = bitor( bitshift(vyun,1), ones(Nn,1) );

sx = (sx .* -2 + 1);
sy = (sy .* -2 + 1);
X = vxun .* sx;
Y = vyun .* sy;

block_qam = X + Y.*i;

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 32-QAM

    case 5

sx = bin2dec( data(:,2) );
sy = bin2dec( data(:,3) );
vx = bin2dec( data(:,4) );
vy = bin2dec( data(:,5) );

vxun = bitxor(sx, vx);
vyun = bitxor(sy, vy);

vxun = bitor( bitshift(vxun,1), ones(Nn,1) );
vyun = bitor( bitshift(vyun,1), ones(Nn,1) );

sx = (sx .* -2 + 1);
sy = (sy .* -2 + 1);
X = vxun .* sx;
Y = vyun .* sy;

% %% blokove
%
```

```
% kvadrant 0; X=X+4; if(X mod 6 ~= X), X=X-6
idx = find((data(:,1)=='1') & (data(:,2)=='0') & (data(:,3)=='0'));
for ii=1:length(idx)
    X(idx(ii)) = X(idx(ii)) + 4;
    rX = rem(X(idx(ii)),6);
    if ( rX ~= X(idx(ii)) )
        X(idx(ii)) = rX - 6;
    end
end

% kvadrant 3; X=X-4; if(X mod 6 ~= X): X=X+6
idx = find((data(:,1)=='1') & (data(:,2)=='1') & (data(:,3)=='1'));
for ii=1:length(idx)
    X(idx(ii)) = X(idx(ii)) - 4;
    rX = rem(X(idx(ii)),6);
    if ( rX ~= X(idx(ii)) )
        X(idx(ii)) = rX + 6;
    end
end


% kvadrant 1; Y=Y-4; if(Y mod 6 ~= Y): Y=Y+6
idx = find((data(:,1)=='1') & (data(:,2)=='0') & (data(:,3)=='1'));
for ii=1:length(idx)
    Y(idx(ii)) = Y(idx(ii)) - 4;
    rY = rem(Y(idx(ii)),6);
    if ( rY ~= Y(idx(ii)) )
        Y(idx(ii)) = rY + 6;
    end
end

% kvadrant 2; Y=Y+4; if(Y mod 6 ~= Y): Y=Y-6
idx = find((data(:,1)=='1') & (data(:,2)=='1') & (data(:,3)=='0'));
for ii=1:length(idx)
    Y(idx(ii)) = Y(idx(ii)) + 4;
    rY = rem(Y(idx(ii)),6);
    if ( rY ~= Y(idx(ii)) )
        Y(idx(ii)) = rY - 6;
    end
end

block_qam = X + Y.*i;

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 64-QAM

    case 6

Nv = (b-2) / 2;
```

```
sx = bin2dec( data(:,1) );
sy = bin2dec( data(:,2) );
vx = bin2dec( data(:,3:2:end) );
vy = bin2dec( data(:,4:2:end) );

sxx=bin2dec(num2str(sx * ones(1,Nv), '%d'));
syy=bin2dec(num2str(sy * ones(1,Nv), '%d'));

vxun = bitxor(sxx, vx);
vyun = bitxor(syy, vy);

vxun = bitor( bitshift(vxun,1), ones(Nn,1) );
vyun = bitor( bitshift(vyun,1), ones(Nn,1) );

sx = (sx .* -2 + 1);
sy = (sy .* -2 + 1);
X = vxun .* sx;
Y = vyun .* sy;

block_qam = X + Y.*i;

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 128-QAM

    case 7

Nv = (b-3) / 2;

sx = bin2dec( data(:,2) );
sy = bin2dec( data(:,3) );
vx = bin2dec( data(:,4:2:end) );
vy = bin2dec( data(:,5:2:end) );

sxx=bin2dec(num2str(sx * ones(1,Nv), '%d'));
syy=bin2dec(num2str(sy * ones(1,Nv), '%d'));

vxun = bitxor(sxx, vx);
vyun = bitxor(syy, vy);

vxun = bitor( bitshift(vxun,1), ones(Nn,1) );
vyun = bitor( bitshift(vyun,1), ones(Nn,1) );

sx = (sx .* -2 + 1);
sy = (sy .* -2 + 1);
X = vxun .* sx;
Y = vyun .* sy;

% %% blokove
%
% kvadrant 0; X=X+8; if(X mod 12 ~= X), X=X-12
```

```
idx = find((data(:,1)=='1') & (data(:,2)=='0') & (data(:,3)=='0'));
for ii=1:length(idx)
    X(idx(ii)) = X(idx(ii)) + 8;
    rX = rem(X(idx(ii)),12);
    if ( rX ~= X(idx(ii)) )
        X(idx(ii)) = rX - 12;
    end
end

% kvadrant 3; X=X-8; if(X mod 12 ~= X): X=X+12
idx = find((data(:,1)=='1') & (data(:,2)=='1') & (data(:,3)=='1'));
for ii=1:length(idx)
    X(idx(ii)) = X(idx(ii)) - 8;
    rX = rem(X(idx(ii)),12);
    if ( rX ~= X(idx(ii)) )
        X(idx(ii)) = rX + 12;
    end
end


% kvadrant 1; Y=Y-8; if(Y mod 12 ~= Y): Y=Y+12
idx = find((data(:,1)=='1') & (data(:,2)=='0') & (data(:,3)=='1'));
for ii=1:length(idx)
    Y(idx(ii)) = Y(idx(ii)) - 8;
    rY = rem(Y(idx(ii)),12);
    if ( rY ~= Y(idx(ii)) )
        Y(idx(ii)) = rY + 12;
    end
end

% kvadrant 2; Y=Y+8; if(Y mod 12 ~= Y): Y=Y-12
idx = find((data(:,1)=='1') & (data(:,2)=='1') & (data(:,3)=='0'));
for ii=1:length(idx)
    Y(idx(ii)) = Y(idx(ii)) + 8;
    rY = rem(Y(idx(ii)),12);
    if ( rY ~= Y(idx(ii)) )
        Y(idx(ii)) = rY - 12;
    end
end

block_qam = X + Y.*i;

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 256-QAM

    case 8

Nv = (b-2) / 2;

sx = bin2dec( data(:,1) );
```

```
sy = bin2dec( data(:,2) );
vx = bin2dec( data(:,3:2:end) );
vy = bin2dec( data(:,4:2:end) );

sxx=bin2dec(num2str(sx * ones(1,Nv), '%d'));
syy=bin2dec(num2str(sy * ones(1,Nv), '%d'));

vxun = bitxor(sxx, vx);
vyun = bitxor(syy, vy);

vxun = bitor( bitshift(vxun,1), ones(Nn,1) );
vyun = bitor( bitshift(vyun,1), ones(Nn,1) );

sx = (sx .* -2 + 1);
sy = (sy .* -2 + 1);
X = vxun .* sx;
Y = vyun .* sy;

block_qam = X + Y.*i;

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 512-QAM

    case 9

Nv = (b-3) / 2;

sx = bin2dec( data(:,2) );
sy = bin2dec( data(:,3) );
vx = bin2dec( data(:,4:2:end) );
vy = bin2dec( data(:,5:2:end) );

sxx=bin2dec(num2str(sx * ones(1,Nv), '%d'));
syy=bin2dec(num2str(sy * ones(1,Nv), '%d'));

vxun = bitxor(sxx, vx);
vyun = bitxor(syy, vy);

vxun = bitor( bitshift(vxun,1), ones(Nn,1) );
vyun = bitor( bitshift(vyun,1), ones(Nn,1) );

sx = (sx .* -2 + 1);
sy = (sy .* -2 + 1);
X = vxun .* sx;
Y = vyun .* sy;

% %% blokove
%
% kvadrant 0; X=X+16; if(X mod 24 ~= X), X=X-24
idx = find((data(:,1)=='1') & (data(:,2)=='0') & (data(:,3)=='0'));
```

```
for ii=1:length(idx)
    X(idx(ii)) = X(idx(ii)) + 16;
    rX = rem(X(idx(ii)),24);
    if ( rX ~= X(idx(ii)) )
        X(idx(ii)) = rX - 24;
    end
end

% kvadrant 3; X=X-16; if(X mod 24 ~= X): X=X+24
idx = find((data(:,1)=='1') & (data(:,2)=='1') & (data(:,3)=='1'));
for ii=1:length(idx)
    X(idx(ii)) = X(idx(ii)) - 16;
    rX = rem(X(idx(ii)),24);
    if ( rX ~= X(idx(ii)) )
        X(idx(ii)) = rX + 24;
    end
end


% kvadrant 1; Y=Y-16; if(Y mod 24 ~= Y): Y=Y+24
idx = find((data(:,1)=='1') & (data(:,2)=='0') & (data(:,3)=='1'));
for ii=1:length(idx)
    Y(idx(ii)) = Y(idx(ii)) - 16;
    rY = rem(Y(idx(ii)),24);
    if ( rY ~= Y(idx(ii)) )
        Y(idx(ii)) = rY + 24;
    end
end

% kvadrant 2; Y=Y+16; if(Y mod 24 ~= Y): Y=Y-24
idx = find((data(:,1)=='1') & (data(:,2)=='1') & (data(:,3)=='0'));
for ii=1:length(idx)
    Y(idx(ii)) = Y(idx(ii)) + 16;
    rY = rem(Y(idx(ii)),24);
    if ( rY ~= Y(idx(ii)) )
        Y(idx(ii)) = rY - 24;
    end
end

block_qam = X + Y.*i;
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 1k-QAM

    case 10

Nv = (b-2) / 2;

sx = bin2dec( data(:,1) );
sy = bin2dec( data(:,2) );
vx = bin2dec( data(:,3:2:end) );
```

```
vy = bin2dec( data(:,4:2:end) );

sxx=bin2dec(num2str(sx * ones(1,Nv), '%d'));
syy=bin2dec(num2str(sy * ones(1,Nv), '%d'));

vxun = bitxor(sxx, vx);
vyun = bitxor(syy, vy);

vxun = bitor( bitshift(vxun,1), ones(Nn,1) );
vyun = bitor( bitshift(vyun,1), ones(Nn,1) );

sx = (sx .* -2 + 1);
sy = (sy .* -2 + 1);
X = vxun .* sx;
Y = vyun .* sy;

block_qam = X + Y.*i;
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 2k-QAM

    case 11

Nv = (b-3) / 2;

sx = bin2dec( data(:,2) );
sy = bin2dec( data(:,3) );
vx = bin2dec( data(:,4:2:end) );
vy = bin2dec( data(:,5:2:end) );

sxx=bin2dec(num2str(sx * ones(1,Nv), '%d'));
syy=bin2dec(num2str(sy * ones(1,Nv), '%d'));

vxun = bitxor(sxx, vx);
vyun = bitxor(syy, vy);

vxun = bitor( bitshift(vxun,1), ones(Nn,1) );
vyun = bitor( bitshift(vyun,1), ones(Nn,1) );

sx = (sx .* -2 + 1);
sy = (sy .* -2 + 1);
X = vxun .* sx;
Y = vyun .* sy;

% %% blokove
%
% kvadrant 0; X=X+32; if(X mod 48 ~= X), X=X-48
idx = find((data(:,1)=='1') & (data(:,2)=='0') & (data(:,3)=='0'));
for ii=1:length(idx)
    X(idx(ii)) = X(idx(ii)) + 32;
    rX = rem(X(idx(ii)),48);
```

```
    if ( rX ~= X(idx(ii)) )
        X(idx(ii)) = rX - 48;
    end
end

% kvadrant 3; X=X-32; if(X mod 48 ~= X): X=X+48
idx = find((data(:,1)=='1') & (data(:,2)=='1') & (data(:,3)=='1'));
for ii=1:length(idx)
    X(idx(ii)) = X(idx(ii)) - 32;
    rX = rem(X(idx(ii)),48);
    if ( rX ~= X(idx(ii)) )
        X(idx(ii)) = rX + 48;
    end
end


% kvadrant 1; Y=Y-32; if(Y mod 48 ~= Y): Y=Y+48
idx = find((data(:,1)=='1') & (data(:,2)=='0') & (data(:,3)=='1'));
for ii=1:length(idx)
    Y(idx(ii)) = Y(idx(ii)) - 32;
    rY = rem(Y(idx(ii)),48);
    if ( rY ~= Y(idx(ii)) )
        Y(idx(ii)) = rY + 48;
    end
end

% kvadrant 2; Y=Y+32; if(Y mod 48 ~= Y): Y=Y-48
idx = find((data(:,1)=='1') & (data(:,2)=='1') & (data(:,3)=='0'));
for ii=1:length(idx)
    Y(idx(ii)) = Y(idx(ii)) + 32;
    rY = rem(Y(idx(ii)),48);
    if ( rY ~= Y(idx(ii)) )
        Y(idx(ii)) = rY - 48;
    end
end

block_qam = X + Y.*i;
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 4k-QAM

    case 12

Nv = (b-2) / 2;

sx = bin2dec( data(:,1) );
sy = bin2dec( data(:,2) );
vx = bin2dec( data(:,3:2:end) );
vy = bin2dec( data(:,4:2:end) );

sxx=bin2dec(num2str(sx * ones(1,Nv), '%d'));
```

```
syy=bin2dec(num2str(sy * ones(1,Nv), '%d'));

vxun = bitxor(sxx, vx);
vyun = bitxor(syy, vy);

vxun = bitor( bitshift(vxun,1), ones(Nn,1) );
vyun = bitor( bitshift(vyun,1), ones(Nn,1) );

sx = (sx .* -2 + 1);
sy = (sy .* -2 + 1);
X = vxun .* sx;
Y = vyun .* sy;

block_qam = X + Y.*i;
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 8k-QAM

    case 13

Nv = (b-3) / 2;

sx = bin2dec( data(:,2) );
sy = bin2dec( data(:,3) );
vx = bin2dec( data(:,4:2:end) );
vy = bin2dec( data(:,5:2:end) );

sxx=bin2dec(num2str(sx * ones(1,Nv), '%d'));
syy=bin2dec(num2str(sy * ones(1,Nv), '%d'));

vxun = bitxor(sxx, vx);
vyun = bitxor(syy, vy);

vxun = bitor( bitshift(vxun,1), ones(Nn,1) );
vyun = bitor( bitshift(vyun,1), ones(Nn,1) );

sx = (sx .* -2 + 1);
sy = (sy .* -2 + 1);
X = vxun .* sx;
Y = vyun .* sy;

% %% blokove
%
% b=13% kvadrant 0; X=X+64;  if(X mod 96 ~= X),  X=X-96
idx = find((data(:,1)=='1') & (data(:,2)=='0') & (data(:,3)=='0'));
for ii=1:length(idx)
    X(idx(ii)) = X(idx(ii)) + 64;
    rX = rem(X(idx(ii)),96);
    if ( rX ~= X(idx(ii)) )
        X(idx(ii)) = rX - 96;
    end
```

```
end

% kvadrant 3; X=X-64; if(X mod 96 ~= X): X=X+96
idx = find((data(:,1)=='1') & (data(:,2)=='1') & (data(:,3)=='1'));
for ii=1:length(idx)
    X(idx(ii)) = X(idx(ii)) - 64;
    rX = rem(X(idx(ii)),96);
    if ( rX ~= X(idx(ii)) )
        X(idx(ii)) = rX + 96;
    end
end


% kvadrant 1; Y=Y-64; if(Y mod 96 ~= Y): Y=Y+96
idx = find((data(:,1)=='1') & (data(:,2)=='0') & (data(:,3)=='1'));
for ii=1:length(idx)
    Y(idx(ii)) = Y(idx(ii)) - 64;
    rY = rem(Y(idx(ii)),96);
    if ( rY ~= Y(idx(ii)) )
        Y(idx(ii)) = rY + 96;
    end
end

% kvadrant 2; Y=Y+64; if(Y mod 96 ~= Y): Y=Y-96
idx = find((data(:,1)=='1') & (data(:,2)=='1') & (data(:,3)=='0'));
for ii=1:length(idx)
    Y(idx(ii)) = Y(idx(ii)) + 64;
    rY = rem(Y(idx(ii)),96);
    if ( rY ~= Y(idx(ii)) )
        Y(idx(ii)) = rY - 96;
    end
end

block_qam = X + Y.*i;
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 16k-QAM

    case 14

Nv = (b-2) / 2;

sx = bin2dec( data(:,1) );
sy = bin2dec( data(:,2) );
vx = bin2dec( data(:,3:2:end) );
vy = bin2dec( data(:,4:2:end) );

sxx=bin2dec(num2str(sx * ones(1,Nv), '%d'));
syy=bin2dec(num2str(sy * ones(1,Nv), '%d'));

vxun = bitxor(sxx, vx);
```

```
vyun = bitxor(syy, vy);

vxun = bitor( bitshift(vxun,1), ones(Nn,1) );
vyun = bitor( bitshift(vyun,1), ones(Nn,1) );

sx = (sx .* -2 + 1);
sy = (sy .* -2 + 1);
X = vxun .* sx;
Y = vyun .* sy;

block_qam = X + Y.*i;
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 32k-QAM

    case 15

Nv = (b-3) / 2;

sx = bin2dec( data(:,2) );
sy = bin2dec( data(:,3) );
vx = bin2dec( data(:,4:2:end) );
vy = bin2dec( data(:,5:2:end) );

sxx=bin2dec(num2str(sx * ones(1,Nv), '%d'));
syy=bin2dec(num2str(sy * ones(1,Nv), '%d'));

vxun = bitxor(sxx, vx);
vyun = bitxor(syy, vy);

vxun = bitor( bitshift(vxun,1), ones(Nn,1) );
vyun = bitor( bitshift(vyun,1), ones(Nn,1) );

sx = (sx .* -2 + 1);
sy = (sy .* -2 + 1);
X = vxun .* sx;
Y = vyun .* sy;

% %% blokove
%
% b=15% kvadrant 0; X=X+128; if(X mod 192 ~= X), X=X-192
idx = find((data(:,1)=='1') & (data(:,2)=='0') & (data(:,3)=='0'));
for ii=1:length(idx)
    X(idx(ii)) = X(idx(ii)) + 128;
    rX = rem(X(idx(ii)),192);
    if ( rX ~= X(idx(ii)) )
        X(idx(ii)) = rX - 192;
    end
end

% kvadrant 3; X=X-128; if(X mod 192 ~= X): X=X+192
```

```
idx = find((data(:,1)=='1') & (data(:,2)=='1') & (data(:,3)=='1'));
for ii=1:length(idx)
    X(idx(ii)) = X(idx(ii)) - 128;
    rX = rem(X(idx(ii)),192);
    if ( rX ~= X(idx(ii)) )
        X(idx(ii)) = rX + 192;
    end
end


% kvadrant 1; Y=Y-128; if(Y mod 192 ~= Y): Y=Y+192
idx = find((data(:,1)=='1') & (data(:,2)=='0') & (data(:,3)=='1'));
for ii=1:length(idx)
    Y(idx(ii)) = Y(idx(ii)) - 128;
    rY = rem(Y(idx(ii)),192);
    if ( rY ~= Y(idx(ii)) )
        Y(idx(ii)) = rY + 192;
    end
end

% kvadrant 2; Y=Y+128; if(Y mod 192 ~= Y): Y=Y-192
idx = find((data(:,1)=='1') & (data(:,2)=='1') & (data(:,3)=='0'));
for ii=1:length(idx)
    Y(idx(ii)) = Y(idx(ii)) + 128;
    rY = rem(Y(idx(ii)),192);
    if ( rY ~= Y(idx(ii)) )
        Y(idx(ii)) = rY - 192;
    end
end

block_qam = X + Y.*i;
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    otherwise
        error('wrong Ns')
end

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

return;
```

8.3 rxDMT2.m

```
function    [rxbindata] = rxDMT2(rxsig, b, txbits, Nd, Ntused, cplen, feq, plot_rx)

%
% rxbindata ... recieved and demodulated binary stream, array of type logical
% rxsig ... recieved signal
% b ... bit-constelation vector, values from 1 to bmax (1 to 15)
%       negative value means unused tone
% txbits ... no. of RX/TX bits
% Nd ... ifft length
% Ntused ... no. of used tones
% cplen ... CP length
% feq ... 1 tap FEQ equalizer vector
% plot_rx ... 0 or 1
%

if nargin ~= 8
    error('need all 8 inputs')
    return;
end

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Nt = Nd/2 - 1;

% cutoff CP
rxsigcp = rxsig;
rxsig = rxsigcp(cplen+1:end);

% fft() rxsignal for DMT -> QAM
block_fft = fft(rxsig);
% 1 tap FEQ correction
block_qam = feq(1:end).*block_fft(1:Nd/2);
rxbindata=block_qam;
if plot_rx == 1

  figure;
  plot(block_qam,'r.');
  title('rxDMT() : RX stavy');
  if onMatlab
    set(gca,'XTick',(-23:2:23));
    set(gca,'YTick',(-23:2:23));
  end
  grid on;
end

return
```

8.4 rx_QAM_detection.m

```
function [DMT_symbols, index_QAM, dmin] = rx_QAM_detection(qamin, b, Qs, rx_plot)

%
% [DMT_symbols, index_QAM, dmin] = rx_QAM_detection(qamin, b, Qs, rx_plot)
%
% Description:
%    Function calculates minimum distance between received QAM signal and
%    the orriginal QAM constalation for DMT modulation.
%
% Inputs:
%   qamin - is the received DMT vector (in QAM space)
%   b     - is a vector defining the number of states ($m$) of QAM for
%           corresponding subcarrirer in qamin
%   Qs    - is a matrix containing all QAM signal constalation for m=1..15
%           This matrix can be generated using genNQAM
%   rx_plot - if '1' plots the graph of received symbols and minimum
%           distance values for all subcariers of DMT
% Outputs:
%   DMT_symbol - signal point form m-ary QAM signal constalation having the
%                smallest distance w.r.t. qamin
%  index_QAM   - index of selected signal point in QAM signal constalation
%  dmin        - value of minimum distances for all subcarriers

bmask=b>0;
ii_vals=find(b>0);
dmin=zeros(size(b));
DMT_symbols=zeros(size(b));
index_QAM=zeros(size(b));

for jj=1:length(ii_vals)
    ii=ii_vals(jj);
    [dmin(ii), index_QAM(ii)]=min(abs(Qs(b(ii),1:2^b(ii))-qamin(ii+1)));
    DMT_symbols(ii)=Qs(b(ii),index_QAM(ii));
end

if rx_plot
    figure,plot(DMT_symbols,'*'), title('States of the received DMT symbol') ;
    figure, plot(dmin), hold on, plot(ones(size(dmin))*sqrt(2),'r');
    hold off; title('Minimum distances vs. dmin/2');
end
DMT_symbols=[0; DMT_symbols];

index_QAM=index_QAM -1;
```

8.5 waterfill.m

```
function [Enlv,bn] = waterfill(rimpch,noise,InputPower)

%   waterfill : Water Filling Algorithm.
%   Enlv : Energies per channel Vector.
%   bn : Bits per channel
%   Sh : Channel Impulsive response vector.
%   Sn : Noise per channel vector.
%   InputPower : total available power.
%
%   If the information available is the normalized signal to noise per channel
%   vector it should be use as rimpch and assing the value 1 to noise.
%
%   Based on John Cioffi development.

gunsortinv=noise./rimpch;
j=length(rimpch);
[d,index]=sort(gunsortinv);
i=1;
gsortinv=zeros(1,j+1);
gsortinv(1:j)=d(1:j);
gsortinv(j+1)=totalenergy;
ka=totalenergy+gsortinv(i);
k=ka;
while (k-(gsortinv(i))>0)
    i=i+1;
    ka=ka+gsortinv(i);
    k=ka/i;
end
ka=ka-gsortinv(i);
k=ka/(i-1);
energy=zeros(1,j);
for m=1:(i-1)
    energy(m)=k-gsortinv(m);
    m=m+1;
end
energy2=zeros(1,j);
for m=1:j
    energy2(index(m))=energy(m);
    m=m+1;
end
b=0.50.*(1/log(2)).*log(1+(energy2./gunsortinv));
y=energy2;
```

Reference

[1] Bingham, J. A. C. *ADSL, VDSL and Multicarrier Modulation*. A Wiley-Interscience Publication, John Wiley & Sons, Inc., New York, USA, 2000.

[2] Cioffi, J. M., Al-Dhahir, N. M. W. Efficiently Computed Reduced-Parameter Input-Aided MMSE Equalizers for ML Detection: A Unified Approach. *IEEE Trans. on Information Theory*, 42(3):903–915, May 1996.

[3] Mazanec, T., Heřmánek, A. ADSL - ekvalizační techniky: Rešeršní práce. Výzkumná zpráva č. 2184, ÚTIA AV ČR, únor 2007.

[4] Mazanec, T., Heřmánek, A. Simulace ADSL downstream přenosu: Webová aplikace, program. ÚTIA AV ČR, únor 2007.

[5] Van Acker, K. Equalization and Echo Cancellation for DMT Modems. SISTA-ESAT K.U. Leuven, Belgium, January 2001.

[6] Ysebaert, G. Equalization and echo Cancellation in DMT-based Systems. SISTA-ESAT K.U. Leuven, Belgium, April 2004.