Implementace metod frekvenční analýzy v elektrických pohonech

Pracoviště: RICE

Číslo dokumentu: 22190-017-2024

Typ zprávy: Výzkumná zpráva

Řešitelé: Ing. Antonín Glac

Hlavní řešitel: Prof. Ing. Zdeněk Peroutka, Ph.D.

Počet stran: 19

Datum vydání: 24.11.2024

Oborové zařazení: 2.2 Electrical engineering, Electronic engineering, Information engineering - Electrical and electronic engineering

Zadavatel / zákazník:

Zpracovatel / dodavatel:

Západočeská univerzita v Plzni
Research and Innovation centre for Electrical Engineering
Univerzitní 8
306 14 Plzeň

Kontaktní osoba:

Ing. Antonín Glac
tel. 377634108
glac@fel.zcu.cz

Tato zpráva vznikla s podporou projektu SGS-2024-017

Anotace

Tato výzkumná zpráva popisuje praktické zkušenosti s metodami frekvenční analýzy pro použití při řízení elektrických pohonů. Popisuje metody pro výpočet amplitudy a fáze zvolené harmonické složky z měření v reálném čase. V další části se věnuje volbě a implementaci filtrů.

Klíčová slova

Diskrétní Fourierova transformace, SOGI, dolní propust, pásmová propust, IIR

Název zprávy v anglickém jazyce / Report title

Frequency Domain Methods Implementation for Electrical Drives

Anotace v anglickém jazyce / Abstract

This research report describes practical experience with frequency analysis methods for use in the control of electric drives. It describes methods for calculating the amplitude and phase of a selected harmonic component from real-time measurements. The next section discusses the selection and implementation of filters.

Klíčová slova v anglickém jazyce / Keywords

Discrete Fourier Transform, SOGI, Lowpass filter, Bandpass filter, IIR

Obsah

1 Úvod

Tato zpráva popisuje praktické zkušenosti s metodami frekvenční analýzy pro použití při řízení elektrických pohonů. Popisuje metody pro výpočet amplitudy a fáze zvolené harmonické složky z měření v reálném čase.

2 DFT

Diskrétní Fourierova transformace převádí diskrétní průběh v časové oblasti do oblasti frekvenční. Použití DFT pro výpočet celého frekvenčního spektra je výpočetně náročné, pro použití v reálném čase jsou vhodné její modifikace

2.1 FFT

Fast Fourier Transform - rychlá Fourierova transformace. Počet vzorků musí být roven mocnině 2 (nebo doplněn nulami do nejbližší mocniny 2). Výpočet celého spektra je pak méně výpočetně náročný.

2.2 Goertzelův algoritmus

Goertzelův algoritmus je aplikace DFT na vybranou konkrétní složku frekvenčního spektra. Díky tomu je vhodný pro výpočet v reálném čase na mikroprocesoru.

Zdrojový kód algoritmu je popsán: Pomocí velikosti okna (BUFFER_LENGTH) se vybere základní analyzovaná frekvence (=frekvence volání funkce / velikost okna). Pomocí pole konstant frequencies si zvolíme násobky základní frekvence, které jsou vypočítávány. V tomto případě při frekvenci volání funkce 10000Hz a délce okna 1000 vzorků získáme základní frekvenci 10Hz, díky násobkům pak algoritmus vrací amplitudu a fázi pro frekvence 40, 80 a 120Hz.

Pro zajištění numerické stability je výpočet prováděn paralelně ve dvojici, hodnoty jsou střádavě resetovány.

```
#define FREQ_INJ_MAX   (3)
#define FREQ_SETS_MAX  (1)
#define BUFFER_LENGTH  (1000)  //  10000/1000 = base harmonics (10
    Hz)
```

```
typedef struct
{
        real32_t i_ref_req;      /*!< Amplitude of
           current required for injection. */
        real32_t i_ref_theta;    /*!< Phase shift (theta
           ) of current required  for injection. */
        real32_t i_ref_freq;     /*!< Frequency of
           current component. */
        real32_t i_ref;              /*!< Real value of
            current amplitude used for injection. */
        real32_t RotMatCos;             /*!< Real part
           of rotation matrix */
        real32_t RotMatSin;             /*!< Imaginary
           part of rotation matrix */
}input_data_t;

typedef struct
{
        real32_t Real;     /*!< Real component of
           current from DFT. */
        real32_t Imag;     /*!< Imaginary component of
           current from DFT. */
}spectra_RI_t;

typedef struct
{
        real32_t amplitude;       /*!< Value of current
           component from DFT based on measurement. */
        real32_t phase;                 /*!< Value of
           phase shift (theta) of current based on
           measurement. */
}spectra_AP_t;

typedef struct
```

```
        {
                volatile real32_t Voltage[BUFFER_LENGTH];
                   /*!< Buffer of values of voltage for DFT
                   calculation. */
                volatile real32_t Current[BUFFER_LENGTH];
                   /*!< Buffer of values of current for DFT
                   calculation. */
                volatile int16_t p_Buffer;
                             /*!< Pointer to actual position in
                    cyclic buffer */
            volatile int16_t Number_call;
                       /*!< Number of DFT calls to prevent DFT
               instability */
            const int16_t Max_call;                          /*!<
               Maximal number of DFT calls to prevent DFT
               instability */
            const real32_t DFT_Const;
        }Buffer_t;

static const uint16_t frequencies[FREQ_SETS_MAX][FREQ_INJ_MAX] =
    {
    {4,8,12}
};

Buffer_t Buffer = { { 0.0f }, { 0.0f }, ( BUFFER_LENGTH - 1 ),
   0, 1 * BUFFER_LENGTH, 2.0f / BUFFER_LENGTH };
input_data_t input_data[FREQ_INJ_MAX];
spectra_RI_t spectra_current1[FREQ_INJ_MAX];
spectra_RI_t spectra_current2[FREQ_INJ_MAX];
spectra_RI_t spectra_voltage1[FREQ_INJ_MAX];
spectra_RI_t spectra_voltage2[FREQ_INJ_MAX];
spectra_AP_t spectra_currentAP[FREQ_INJ_MAX];
spectra_AP_t spectra_voltageAP[FREQ_INJ_MAX];
```

```
void DftInit(int16_t iFreq)
{
    Buffer.p_Buffer=BUFFER_LENGTH-1;
    input_data[iFreq].RotMatCos=cos(TWO_PI*input_data[iFreq].
       i_ref_freq / ((real32_t) (BUFFER_LENGTH)));
       input_data[iFreq].RotMatSin=sin(TWO_PI*input_data[iFreq].
       i_ref_freq / ((real32_t) (BUFFER_LENGTH)));
}

void Dft(real32_t actual_value, real32_t last_value,
   input_data_t* p_input_data_t, spectra_RI_t* p_spectra_t)
{
    volatile real32_t Real, Imag;
    //Spectra calculation
    Real = p_input_data_t->RotMatCos * p_spectra_t->Real -
       p_input_data_t->RotMatSin * p_spectra_t->Imag +
       actual_value * Buffer.DFT_Const - last_value;
    Imag = p_input_data_t->RotMatSin * p_spectra_t->Real +
       p_input_data_t->RotMatCos * p_spectra_t->Imag;
    p_spectra_t->Real=Real;
    p_spectra_t->Imag=Imag;
}

void DftComp(real32_t Voltage, real32_t Current)
{
        int iFreq;
        for (iFreq = 0; iFreq < FREQ_INJ_MAX; iFreq++)
        {
                DftCall(Voltage, Current,iFreq);
                if (Buffer.Number_call == 2 * Buffer.Max_call -
                   1)  //reseting
                {
```

```
                    spectra_current2[iFreq].Real = 0.0f;
                    spectra_current2[iFreq].Imag = 0.0f;
                    spectra_voltage2[iFreq].Real = 0.0f;
                    spectra_voltage2[iFreq].Imag = 0.0f;
                }
                if (Buffer.Number_call == Buffer.Max_call - 1)
                   //reseting
                {
                    spectra_current1[iFreq].Real = 0.0f;
                    spectra_current1[iFreq].Imag = 0.0f;
                    spectra_voltage1[iFreq].Real = 0.0f;
                    spectra_voltage1[iFreq].Imag = 0.0f;
                }
        }
        DftStep(Voltage, Current);
    for (iFreq = 0; iFreq < FREQ_INJ_MAX; iFreq++)
    {
        SpectraCalc(iFreq);                 //prevod na
           amplitudu a fazi
    }
}

void DftCall(real32_t Voltage, real32_t Current, int16_t iFreq)
{
        if ( Buffer.Number_call < BUFFER_LENGTH )
        {
                Dft( Current, Buffer.Current[Buffer.p_Buffer] ,
                   &input_data[iFreq], &spectra_current1[iFreq]
                   );
                Dft( Current, 0.0f , &input_data[iFreq] , &
                   spectra_current2[iFreq] );
                Dft( Voltage, Buffer.Voltage[Buffer.p_Buffer] ,
                   &input_data[iFreq], &spectra_voltage1[iFreq]
                   );
```

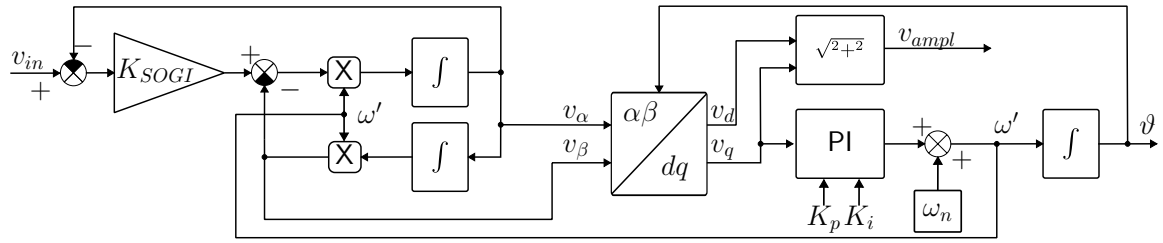
```
                Dft( Voltage, 0.0f , &input_data[iFreq], &
                    spectra_voltage2[iFreq] );
        }
        else if((Buffer.Number_call >= Buffer.Max_call) && (
            Buffer.Number_call<(BUFFER_LENGTH + Buffer.Max_call))
            )
        {
                Dft( Current, 0.0f , &input_data[iFreq], &
                    spectra_current1[iFreq] );
                Dft( Current, Buffer.Current[Buffer.p_Buffer] ,
                    &input_data[iFreq] , &spectra_current2[iFreq]
                     );
                Dft( Voltage, 0.0f , &input_data[iFreq], &
                    spectra_voltage1[iFreq] );
                Dft( Voltage, Buffer.Voltage[Buffer.p_Buffer] ,
                    &input_data[iFreq], &spectra_voltage2[iFreq]
                    );
        }
        else
        {
                Dft(Current, Buffer.Current[Buffer.p_Buffer], &
                    input_data[iFreq], &spectra_current1[iFreq]);
                Dft(Current, Buffer.Current[Buffer.p_Buffer], &
                    input_data[iFreq], &spectra_current2[iFreq]);
                Dft(Voltage, Buffer.Voltage[Buffer.p_Buffer], &
                    input_data[iFreq], &spectra_voltage1[iFreq]);
                Dft(Voltage, Buffer.Voltage[Buffer.p_Buffer], &
                    input_data[iFreq], &spectra_voltage2[iFreq]);
        }
}

void DftStep(real32_t Voltage, real32_t Current)
{
        Buffer.Voltage[Buffer.p_Buffer]=Voltage*Buffer.DFT_Const
```

```
            ;
        Buffer.Current[Buffer.p_Buffer]=Current*Buffer.DFT_Const
            ;
        Buffer.p_Buffer--;
        if (Buffer.p_Buffer<0)
        {
                Buffer.p_Buffer = BUFFER_LENGTH - 1;
        }
        Buffer.Number_call++;
        if (Buffer.Number_call>2*Buffer.Max_call-1)
        {
                Buffer.Number_call=0;
        }
}

void SpectraCalc(int16_t iFreq)
{
        if ( Buffer.Number_call < Buffer.Max_call )
        {
                RealImagToAmpPhase(&spectra_current1[iFreq],&
                    spectra_currentAP[iFreq]);
                RealImagToAmpPhase(&spectra_voltage1[iFreq],&
                    spectra_voltageAP[iFreq]);
        }
        else
        {
                RealImagToAmpPhase(&spectra_current2[iFreq],&
                    spectra_currentAP[iFreq]);
                RealImagToAmpPhase(&spectra_voltage2[iFreq],&
                    spectra_voltageAP[iFreq]);
        }
}

void RealImagToAmpPhase(spectra_RI_t* p_spectraRI,spectra_AP_t*
```



Obr. 3.1: Schéma SOGI fázového závěsu

```
    p_spectraAP)
{
        p_spectraAP->amplitude = sqrtf(p_spectraRI->Real*
            p_spectraRI->Real + p_spectraRI->Imag*p_spectraRI->
            Imag);
        p_spectraAP->phase=atan2f(p_spectraRI->Imag, p_spectraRI
            ->Real);
}
```

3 SOGI

SOGI - Second Order Generalized Integrator [1] - je typ fázového závěsu. Umožňuje na základě (harmonického) časového průběhu veličiny určit jeho amplitudu a fázový posuv. Schéma je uvedeno na Obr. 3.1. Úhlová rychlost ω_n je nastavena jako hledaná frekvence, v tomto konkrétním případě jako frekvence drážkové harmonické.

Pro správné fungování fázového závěsu je třeba naladit zesílení - rezonanční zesílení $K_r = \frac{\omega_n^2 \Delta t}{K_{r0}}$ a proporční a integrační zesílení PI regulátoru K_p a K_i.

Zdrojový kód v C je popsán:

```
#include <math.h>
#include <sogi_variable_speed.h>
#define SOGI_INITOMEGA (M_PI*100.0/30.0*6.0*PP_X)
#define SOGI_KI 2000.0f*SOGI_DT
#define SOGI_KP 120.0f
#define SOGI_KR_DIV 400.0f
#define SOGI_KR_INIT (SOGI_INITOMEGA*SOGI_INITOMEGA/SOGI_KR_DIV*
    SOGI_DT)
```

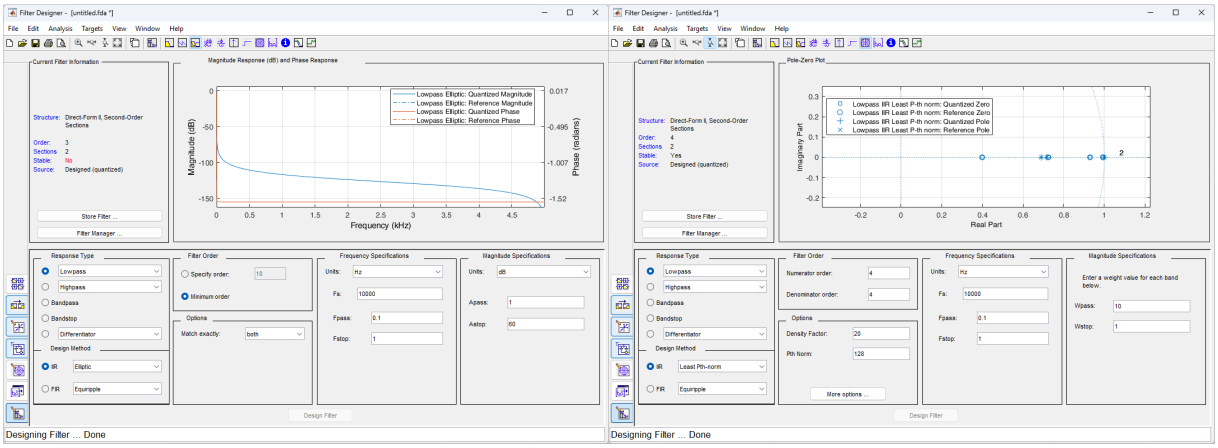
```
Sogi_struct_t sogi_M = {0.0f,0.0f,0.0f,0.0f,0.0f,0.0f,
   SOGI_INITOMEGA,SOGI_KR_INIT,SOGI_KI,SOGI_KP,SOGI_DT,0.0f};
Sogi_struct_t sogi_lsq = {0.0f,0.0f,0.0f,0.0f,0.0f,0.0f,
   SOGI_INITOMEGA,SOGI_KR_INIT,SOGI_KI,SOGI_KP,SOGI_DT,0.0f};
float SOGI_KR = (M_PI*100.0/30.0*6.0*PP_X)*(M_PI*100.0/30.0*6.0*
   PP_X)/SOGI_KR_DIV*SOGI_DT;

void Sogi(Sogi_struct_t* p_sogi_t, float speed_omega_mech)
{
        float cosin, sinus, err;
        SOGI_KR = (speed_omega_mech*6.0*PP_X)*(speed_omega_mech
           *6.0*PP_X)/SOGI_KR_DIV*SOGI_DT;
        p_sogi_t->Kr = SOGI_KR;
        p_sogi_t->InitOmega = speed_omega_mech*6.0*PP_X;
        cosin = cos(p_sogi_t->Theta);
        sinus = sin(p_sogi_t->Theta);
        err = p_sogi_t->Input - cosin * p_sogi_t->Sre - sinus *
           p_sogi_t->Sim;
        p_sogi_t->Sre += p_sogi_t->Kr * err * cosin;
        p_sogi_t->Sim += p_sogi_t->Kr * err * sinus;
        p_sogi_t->dOmega -= p_sogi_t->Ki * p_sogi_t->Sim;
        p_sogi_t->Omega = p_sogi_t->dOmega - p_sogi_t->Kp *
           p_sogi_t->Sim +  p_sogi_t->InitOmega;
        p_sogi_t->Theta += p_sogi_t->Omega * p_sogi_t->dT;
        if (p_sogi_t->Theta > 3.14159265358979)
    {
        p_sogi_t->Theta -= 6.28318530717959;
    }
        if (p_sogi_t->Theta < -3.14159265358979)
    {
        p_sogi_t->Theta += 6.28318530717959;
    }
    p_sogi_t->Sampl = sqrt(p_sogi_t->Sre * p_sogi_t->Sre +
       p_sogi_t->Sim  * p_sogi_t->Sim);
```

```
        if((p_sogi_t->Omega * speed_omega_mech) > 0.0f)  // same
            sign
        {
        p_sogi_t->Theta_out = p_sogi_t->Theta;
    }
    else
    {
        p_sogi_t->Theta_out = -p_sogi_t->Theta;
    }
}
```

4 Návrh digitálního filtru v aplikaci Matlab

Pomocí aplikace Filter Designer je možné naladit koeficienty filtru a zvolit vhodnou formu. Zároveň je možné posoudit stabilitu filtru pro implementaci v single precision float formátu.



Obr. 4.1: Aplikace Filter Designer

4.1 Bandpass filtr

Tato část se zabývá návrhem a implementací filtru typu pásmová propust (Bandpass). Pro implementaci byl zvolen filtr s nekonečnou impulsní odezvou (IIR) eliptického typu.

4.1.1 Eliptický filtr s proměnnou frekvencí propustného pásma

Implementace kódu v C je založena na projektu z https://www.dsp-weimich.com/downloads/, IIR Filter and C Implementation Using Octave GNU Tool.

Struktura filtru je SOS form II (second order sections) řádu 6 (3 sekce SOS). Šířka propustného pásma byla zvolena jako předpokládaná frekvence drážkové harmonické (6x elektrická frekvence) $\pm 10\%$.

funkce ellip() navrhne filtr ve formě přenosové funkce. Funkce tf2sos ho pak převede do formátu SOS. Každá sekce je reprezentována koeficienty $a_{0...n}$,$b_{0...n}$. Koeficient $a_0 = 1$, výstup celého filtru je nutné přenásobit zesílením filtru gn.

V tomto konkrétním případě má řádka matice ss tvar $[b_0, b_1, b_2, 1, a_1, a_2]$, v C kódu má pak matice float iir_coef_float formát $[b_0, b_1, b_2, a_1, a_2]$

Aby mohla být frekvence variabilní, jsou koeficienty filtru navrženy pro různé frekvence a výsledek je interpolován. Pro výstupní zesílení je použita lineární interpolace, pro ostatní koeficienty pak interpolace 2. řádu.

Matlab kód pro návrh a interpolaci koeficentů filtru:

```
Fs = 10000;  % Sampling Frequency
N    = 6;    % Order Fc1 = 72;  % First Cutoff Frequency Fc2 = 88;
     % Second Cutoff Frequency
freq_vect = 0.1:0.01:4;
gn_all = (zeros(length(freq_vect),1)); ss_all = (zeros(size(
   ss200,1),size(ss200,2),length(freq_vect)));
for i=1:length(freq_vect)
[nm,dn] = ellip(N/2,3,40, (freq_vect(i) .* [Fc1, Fc2] /(Fs/2)),"
   bandpass");
[ss,gn] = tf2sos(nm,dn);
gn_all(i) = (gn); ss_all(:,:,i) = (ss);
end

gn_poly1 = polyfit(freq_vect_a(1:200),gn_all(1:200),1);
gn_poly_val1 = polyval(gn_poly1,freq_vect_a);
```

```
 ii=0;
figure(1)
for i=1:size(ss_all,1)
for j=[2,3,5,6]
ii=ii+1;
        figure(1)
  subplot(3,4,ii)
        plot(freq_vect_a, squeeze(ss_all(i,j,:)),'x-')
       % if j > 4
  ss_poly3 = polyfit(freq_vect_a(24:200),squeeze(ss_all(i,j
     ,24:200)),2)
            ss_poly_val = polyval(ss_poly3,freq_vect_a);
            figure(2);
           hold off
       plot(freq_vect_a, squeeze(ss_all(i,j,:)),'x-');
          hold on              plot(freq_vect_a, ss_poly_val,'x
            -');
            keyboard         % end      end end
```

```
%% sos format: b0 b1 b2 1 a1 a2
% output = gn * sosfilt();
```

C kód interpolace koeficientů filtru (konstanty jsou výstupem předchozí Matlab funkce):

```
    if(freq_in < 20.0f)
    {freq_in = 20.0f;}
    if(freq_in > 160.0f)
    {freq_in = 160.0f;}
```

```
//matlab - polyfit 1st order - C:\glac\Transactions\
   LQ_LC_filter_control\iir_bandpass_test_ellip.m
```

```
output_gain = 1.0e-5 * ((0.345184862502385) * freq +
   0.064150708615190);
//matlab - polyfit 2nd order - C:\glac\Transactions\
   LQ_LC_filter_control\iir_bandpass_test_ellip.m
iir_coef_float[0][0] = 1.0f;  //const
iir_coef_float[0][1] = 0.0f;
iir_coef_float[0][2] = -0.999999994038209;
iir_coef_float[0][3] = (0.000000387249802 * freq * freq +
   (0.000040831678829) * freq + (-2.000004230864189));
iir_coef_float[0][4] = (0.000000000511442 * freq * freq +
   (-0.000040539318601) * freq + 0.999996550715736);
iir_coef_float[1][0] = 1.0f;  //const
iir_coef_float[1][1] = (0.000000609517239 * freq * freq +
   (0.000000243022415) * freq + (-2.000006623960171));
iir_coef_float[1][2] = 0.999999995886100;   //const
iir_coef_float[1][3] = (0.000000467588486 * freq * freq +
   0.000018592262194 * freq + (-2.000004749053868));
iir_coef_float[1][4] = (0.000000000135574 * freq * freq +
   (-0.000018362104035) * freq + 0.999998880375069);
iir_coef_float[2][0] = 1.0f;  //const
iir_coef_float[2][1] = (0.000000249364403 * freq * freq +
   (0.000000043350605) * freq + (-2.000001186715448));
iir_coef_float[2][2] =  1.000000010075695;  //const
iir_coef_float[2][3] = (0.000000323278459 * freq * freq +
   0.000015484439600 * freq + (-2.000007441612358));
iir_coef_float[2][4] = ((0.000000000462108) * freq * freq +
   (-0.000015367555701) * freq + 1.000004569136928);
```

4.2 Lowpass filtr s dlouhou časovou konstantou

Úkolem filtru je odstranit frekvence vyšší než je zvolená zlomová frekvence filtru. V tomto konkrétním případě má filtr velmi dlouhou časovou konstantu - zlomová frekvence je velmi nízká. ($F_s = 10000$ Hz, $F_{pass} = 0.1$ Hz, $F_{stop} = 1$ Hz).

4.2.1 FIR, IIR

Návrh filtru s konečnou impulzní odezvou (FIR) pro dlouhou časovou konstantu vede na extrémně vysoký řád filtru a zároveň nepříznivý průběh amplitudové i fázové charakteristiky.

Vhodnější je použití filtru s nekonečnou impulzní odezvou (IIR). Na výběr je z mnoha forem. Nejčastěji používané typy Butterworth, Chebyshev I a II a Elliptic mají i při nízkém řádu filtru vhodnou charakteristiku, ale při použití single precision float formátu a dlouhé časové konstantě nejsou numericky stabilní.

4.2.2 IIR Least Pth Norm

Filtr splňující požadavky na utlumení nežádoucích frekvencí a zároveň numericky stabilní je typ IIR Least Pth Norm.

5 Závěr

Pro optimální řízení elektrických pohonů je vhodné využití matematických metod, které pracují ve frekvenční oblasti. Pro převod do frekvenční oblasti pak využíváme metody založené na Fourierově transformaci, případně na fázovém závěsu. V některých případech je vhodné časový signál nejprve filtrovat, a pak teprve převádět do frekvenční oblasti.

Reference

[1] Dušan Janík, Jakub Talla, Tomáš Komrska, and Zdeněk Peroutka. Optimalization of sogi pll for single-phase converter control systems: Second order generalized integrator (sogi). In *2013 International Conference on Applied Electronics*, pages 1–4, 2013.

Seznam obrázků

Seznam tabulek

Historie revizí

Rev.	Kapitola	Popis změny	Datum	Jméno
1	Všechny	První revize zprávy	01.07.2021	...