Szybka transformata Fouriera (FFT)
i jej zastosowania
Metody Numeryczne II A

19.04.2001 r.

1 Szybka transformata Fouriera (FFT)

1.1 Właściwości transformaty Fouriera

Fizyczny proces (sygnał) może być opisany albo w *dziedzinie czasowej*, jako (być może zespolona) funkcja czasu $h(t)$, albo w *dziedzinie częstości*, jako zespolona (by opisać fazę) funkcja częstości $H(f)$. Przejścia między tymi dwoma reprezentacjami można dokonać za pomocą transformaty Fouriera i odwrotnej transformaty Fouriera:

$$H(f) = \int_{-\infty}^{\infty} h(t) e^{2\pi\imath ft} dt \tag{1a}$$

$$h(t) = \int_{-\infty}^{\infty} H(f) e^{-2\pi\imath ft} df \tag{1b}$$

(1c)

Z równań (1a) i (1b) widać, że transformacja Fouriera jest operacją liniową, t.j. transformata Fouriera kombinacji liniowej $\alpha f + \beta h = \alpha F + \beta H$, gdzie F jest transformatą Fouriera funcji f, a H – funkcji h, zaś $\alpha, \beta \in \mathbb{C}$.

Funkcja $h(t)$ może mieć pewne szczególne właściwości: może być czysto rzeczywista lub czysto urojona, może być parzysta bądź nieparzysta *Uwaga:* w przypadku dyskretnej skończonej transformaty Fouriera (patrz rozdział 1.2) badana funkcja jest funkcją okresową o okresie $N\Delta$, tak więc $h(-x_k) = h(x_{N-k})$. Właściwości funkcji $h(t)$ przekładają się na odpowiednie właściwości jej transformaty Fouriera $H(f)$ — patrz tabela 1.

funkcja $h(t)$	transformata $H(f)$
$h(t) = h(t)^*$ rzeczywista	$H(f) = H(-f)^*$
$h(t) = -h(t)^*$ urojona	$H(f) = -H(f)^*$
$h(t) = h(-t)$ parzysta	$H(f) = H(-f)$ parzysta
$h(t) = -h(-t)$ nieparzysta	$H(f) = -H(-f)$ nieparzysta
$h(t)$ rzeczywista i parzysta	$H(f)$ rzeczywista i parzysta
$h(t)$ rzeczywista i nieparzysta	$H(f)$ urojona i nieparzysta
$h(t)$ urojona i parzysta	$H(f)$ urojona i parzysta
$h(t)$ urojona i nieparzysta	$H(f)$ rzeczywista i nieparzysta

Tabela 1: Symetrie $h(t)$ a symetrie $H(f)$

Zachodzą ponadto następujące własności

$$h(at) \Longleftrightarrow \frac{1}{|a|}H(\frac{f}{a}) \qquad \text{skalowanie czasu} \tag{2a}$$

$$\frac{1}{|b|}h(\frac{t}{b}) \Longleftrightarrow H(bf) \qquad \text{skalowanie częstości} \tag{2b}$$

$$h(t-t_0) \Longleftrightarrow H(f)e^{2\pi\imath f t_0} \qquad \text{przesunięcie w czasie} \tag{2c}$$

$$h(t)e^{-2\pi\imath f_0 t} \Longleftrightarrow H(f-f_0) \qquad \text{przesunięcie w częstościach} \tag{2d}$$

$$\frac{d}{dt}h(t) \Longleftrightarrow -2\pi\imath f H(f) \qquad \text{różniczkowanie po czasie} \tag{2e}$$

1.2 Dyskretna transformata Fouriera

W najczęściej spotykanych sytuacjach (zwłaszcza przy obróbce sygnału) funkcja $h(t)$ jest zadana przez wartości w równoodległych chwilach czasu:

$$h_k = h(k\Delta) \quad n = \ldots, -3, -2, -1, 0, 1, 2, 3, \ldots \tag{3}$$

Dla każdej wielkości przedziału Δ istnieje szczególna częstość f_c zwana **krytyczną częstością Nyquista**, dana wzorem

$$f_c \equiv \frac{1}{2\Delta} \tag{4}$$

Funkcja o częstości większej od częstości Nyquista ma te same wartości w punktach pomiaru co pewna funkcja o częstości mniejszej od częstości Nyquista. To zjawisko nazywa się **aliasingiem**. Aby zatem nie dostać błędnych wyników częstość próbkowania musi być dwa razy większa od najwyższej znaczącej częstości w sygnale. Oznacza to także, że dla funkcji zadanej przez wartości w dyskretnych punktach możemy określić jednoznacznie transformatę Fouriera tylko dla $-f_c < f < f_c$.

W obliczeniach komputerowych będziemy wyliczać transformatę Fouriera na podstawie wielkości sygnału w skończonej ilości równoodległych punktów próbkowania. Załóżmy, że mamy N (dla uproszczenia niech N będzie parzyste) takich punktów

$$h_k \equiv h(t_k), \quad t \equiv k\Delta, \quad k = 0, 1, 2, \ldots, N-1 \tag{5}$$

Możemy wówczas określić (jednoznacznie) transformatę Fouriera tylko w N punktach:

$$f_n \equiv \frac{n}{N\Delta}, \quad n = -\frac{N}{2}, \ldots, \frac{N}{2} \tag{6}$$

(uwaga: $f_{-N/2} = f_{N/2}$).

Transformata funkcji zadanej na skończonej liczbie punktów równej N jest identyczna z dyskretną transformatą Fouriera funkcji okresowej o okresie $N\Delta$ i wartościach dla jednego okresu takich jak zadana funkcja. *Uwaga:* zatem dla dyskretnej transformaty Fouriera funkcja parzysta to taka, że $h_k = h_{N-k} = h_{-k}$.

Przy obliczaniu transformaty Fouriera funkcji należy pamiętać, że wartości ujemne argumentu odpowiadają odpowiednim indekson $N-k$ z końca tablicy. Jeśli zatem liczymy transformatę Fouriera funkcji skoncentrowanej wokół zera (np. funkcja Gaussa), to aby uniknąć mnożenia transformaty przez $e^{2\pi\imath f \text{przesunięcie}}$ (patrz wzór (2c)) należy rozmieścić wartości funkcji na „zawiniętej" tablicy, t.j. h[k] $= h(k\Delta)$ dla $k < N/2$, zaś h[N−k] $= h(-k\Delta)$ dla $k < N/2$.

Dyskretna transformata Fouriera i odwrotna dyskretna transformata Fouriera zadane są wzorami

$$H_n = \sum_{k=0}^{N-1} h_k e^{2\pi\imath kn/N} \tag{7a}$$

$$h_k = \frac{1}{N}\sum_{n=0}^{N-1} H_n e^{-2\pi\imath kn/N} \tag{7b}$$

Skorzystaliśmy tutaj z tego, że (jak łatwo zauważyć) $H_{-n} = H_{N-n}$, tak więc zamiast zadawać H_n dla $n = -N/2, \ldots, N/2$ możemy używać $n = 0, \ldots, N-1$. *Uwaga:* różne procedury (biblioteki) stosują różną konwencję znaków w transformacie i odwrotnej transformacie Fouriera. Podajemy tutaj konwencję za „Numerical Recipies" [1].

Związek między dyskretną a zwykłą transformatą Fouriera zadany jest wzorem

$$H(f_n) \approx \Delta H_n \tag{8}$$

1.3 Algorytm szybkiej transformaty Fouriera

Najprostsza wersja algorytmu obliczania dyskretnej transformaty Fouriera (wprost ze wzorów (7)) wymaga ilości obliczeń proporcjonalnej do N^2 (gdzie N jest liczbą punktów, w których zadana jest nasza funkcja). Algorytmy szybkiej transformaty Fouriera korzystają z rozkładu N na czynniki: $N = r_1 r_2 \ldots r_p$ i mają koszt (ilość obliczeń) proporcjonalną do $N(r_1 + r_2 + \cdots + r_p$. Najprostszą wersję algorytmu uzyskujemy dla ilości punktów będącej potęga dwójki: $N = 2^p$.

Algorytm obliczania szybkiej transformaty Fouriera dla $N = 2^k$ nosi nazwę **metody Cooleya–Tukeya**. Opera się ona na zastosowaniu tzw. metody *dziel i rządź*: dzielimy problem na dwa *takie same* problemy o dwukrotnie mniejszym rozmiarze, a następnie składamy je do rozwiązania pełnego problemu.

Dyskretna transformata Fouriera dla danych długości N może być zapisana jako kombinacja liniowa dwu transformat Fouriera, każdej o długości $N/2$; jednej złożonej z punktów o parzystych, a drugiej z punktów o nieparzystych indeksach w oryginalnej transformacie:

$$\begin{aligned}
H_n &= \sum_{k=0}^{N-1} e^{2\pi\imath kn/N} h_k = \\
&= \sum_{\beta=0}^{N/2-1} e^{2\pi\imath(2\beta)n/N} h_{2\beta} + \sum_{\beta=0}^{N/2-1} e^{2\pi\imath(2\beta+1)n/N} h_{2\beta+1} = \\
&= \sum_{\beta=0}^{N/2-1} e^{2\pi\imath\beta n/(N/2)} h_{2\beta} + \left(e^{2\pi\imath/N}\right)^n \sum_{\beta=0}^{N/2-1} e^{2\pi\imath\beta n/(N/2)} h_{2\beta+1} = \\
&= H^{|}_{n \bmod \frac{N}{2}} + W^n H^{\nmid}_{n \bmod \frac{N}{2}}
\end{aligned} \tag{9}$$

Wprowadziliśmy oznaczenie $W = e^{2\pi\imath/N}$. Łatwo zauważyć, że w obu sumach wartość dla $n = N/2 + n_1$ (gdzie $k \geq 0$) jest taka sama jak wartość dla n_1:

$$(W^2)^{n\beta} = (W^2)^{\left(\frac{N}{2}+k_1\right)\beta} = (W^2)^{\frac{N\beta}{2}} (W^2)^{k_1\beta} = (W^N)^\beta (W^2)^{k_1\beta} = (W^2)^{k_1\beta},$$

gdyż $W^N = 1$.

Dla każdej z części możemy zastosować to samo rozumowanie, za każdym razem zmniejszając rozmiar zagadnienia dwa razy. Po $\log_2 N$ krokach dojdziemy do transformaty Fouriera o długości 1. Transformata Fouriera dla $N = 1$ jest funkcją identycznościową. Mamy zatem

$$H_n^{|\nmid\nmid|\cdots|} = h_l \quad \text{dla pewnego } l$$

(symbol $|$ odpowiada braniu częsci złożonej z parzystych indeksów w kolejnym kroku $(2 \mid k)$, symbol $\nmid$ z nieparzystych $(2 \nmid k)$).

Pozostaje zatem znaleźć indeks l odpowiadający danej sekwencji wyborów indeksów parzystych/nieparzystych. Widzimy, że przy pierwszym kroku w sumie oznaczonej symbolem $|$ pojawiają się tylko elementy h_k o indeksach parzystych, czyli ostatni bit indeksu l dla sekwencji rozpoczynającej się od $|$ wynosi 0. Następny podział zachodzi ze względu na przedostatni bit indeksu tablicy h. Zatem indeks l odpowiadający sekwencji symboli $|\nmid\nmid|\cdots|$ (określającej kolejne wybory indeksów parzystych/nieparzystych w kolejnej części) otrzymujemy, biorąc liczbę binarną uzyskaną przez odwrócenie sekwencji i zamianę $|$ na 0, a $\nmid$ na 1.

Aby uniknąć kłopotliwego wyliczania indeksów po których sumujemy w kolejnych podziałach (lub ich zapamiętywanie w dodatkowej tablicy), najprościej jest ułożyć (przestawić) elementy tablicy z danymi (naszym sygnałem) h[k] (czyli h_k) tak, aby sumowanie rozciągało się zawsze po kolejnych elementach tablicy. Przykładowa procedura rekurencyjna (używająca dadatkowego pomocniczego wektora) mogła by wyglądać następująco:

```
void fft_rec(fcomplex data[], float tmp[], int n, int isign)
{
  int i, imod;
  int n2 = n/2;
  fcomplex wn = { 0, 0 };

  if(n == 1) /* przypadek graniczny */
    return;

  /* przestawiamy elementy, by wywołania rekurencyjne */
  /* miały je ułożone po kolei */
  for (i = 0; i < n2; i++) {
    /* elementy o indeksach parzystych */
    tmp[i].re = data[2*i].re;
    tmp[i].im = data[2*i].im;

    /* elementy o indeksach nieparzystych */
    tmp[i + n2].re = data[2*i+1].re;
    tmp[i + n2].im = data[2*i+1].im;
  }

  /* wywołujemy rekurencyjnie funkcję dla */
  /* "indeksów parzystych" i "nieparzystych */
  fft_rec(tmp,    data, n2, isign);
  fft_rec(tmp+n2, data, n2, isign);

  /* wyliczamy kolejne elementy transformaty fouriera */
  for (i = 0; i < n; i++) {
    imod = i % n2;
    wn.re = cos(isign*2*M_PI*i/N);
    wn.im = sin(isign*2*M_PI*i/N);

    /* H_n = H^o_{n mod N/2} + W^n H^e_{n mod N/2} */
    data[i].re = tmp[imod].re + wn.re*tmp[n2 + imod].re - wn.re*tmp[n2 + imod].re;
    data[i].re = tmp[imod].im + wn.re*tmp[n2 + imod].im + wn.im*tmp[n2 + imod].re;
  }
}
```

(Jawnie zapisujemy dodatkowy wektor pomocniczy dla uproszczenia [zapisu] procedury.)

Stosunkowo łatwo zauważyć, że metoda ta prowadzić będzie do przestawiania niektórych elementów aż $\log N$ razy (tyle ile głębokość rekurencji). Gdyby udało się tego uniknąć i ustawić elementy od razu w żądanej kolejności to rekurencję można by zastąpić iteracją. Składamy rozwiązanie dla transformaty Fouriera o długości 2 z transformat Fouriera o długości 1:

$$H_k^{\cdot\cdot} = H_k^{\cdot\cdot|} + W^k H^{\cdot\cdot\dagger},$$

następnie dla długości 2 i tak dalej; prosta iteracja. Jeśli ułożymy elementy tablicy h w ten sposób, by elementy odpowiadające $H^{\cdot\cdot|}$ i $H^{\cdot\cdot\dagger}$ leżały obok siebie, obliczanie transformaty Fouriera będzie proste. Należy zatem najpierw posortować (ułożyć) elementy w kolejności składania transformat, czyli w kolejności odwróconych bitów. Ponieważ relacja odwrócenia bitowego jest przemienna, zatem przy odpowiedniej organizacji programu wystarczy zrobić co najwyżej jedno przemieszczenie na element. Można to zrobić „w miejscu".

Algorytm można zatem złożyć z dwu części. Pierwsza to takie uporządkowanie elementów, by h_i odpowiadające elementom z dwu najbardziej wewnętrznych podziałów leżały obok siebie itd., tzn. według kolejności l-ów. Druga to kolejne składania cząstkowych transformat w jedną całość.

Bardziej szczegółowe informacje n.t. algorytmu Cooleya–Tukeya można znaleźć w „Numerical Recipies in C” [1].

1.4 Opis procedury `four1` z NRC

Procedura `four1` (zamieszczona na stronie WWW) oblicza dyskretną transformatę Fouriera tablicy data[1..2∗nn], zastępując nią oryginalne dane, dla isign = 1. Dla isign = -1 oblicza odwrotną transformatę Fouriera (pomnożoną przez nn). Argument nn oznacza ilość liczb zespolonych. Procedurę tę można zastosować bez zmian do tablic indeksowanych od 0 (jak w C), tzn. dla data[0..2∗nn−1] wywołując ją w następujący sposób

```
four1(data - 1, N, 1);
```

Dane rozmieszczone są jako kolejno część rzeczywista i urojona kolejnych liczb zespolonych. data[1] jest częścią rzeczywistą pierwszej wartości h, zaś data[2] jest jej częścią urojoną. Na macierz data można patrzeć albo jako na tablicę nn liczb zespolonych fcomplex, albo jako na dwa razy wiekszą tablicę liczb rzeczywistych.

Uwaga: W macierzy wynikowej w elementach data[1] i data[2] znajduje się częstość zerowa, zaś w elementach data[2∗nn−1] i data[2∗nn] częstość Nyquista. Rozmiar tablicy (liczony w liczbach zespolonych) **musi** być całkowitą potęgą 2.

1.5 Analiza procedury `four1` z NRC

```
include <math.h>

#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr
```

Powyższa makrodefinicja zamienia dwie zmienne wartościami. Uwaga: nie da się jej użyć w wyrażeniu, musi być zakończona średnikiem. Podwójnie szacuje swoje argumenty (czyli nie należy jej np. wywoływać z argumentami SWAP(∗src++, ∗dest++)).

```

void four1(float data [], int nn,int isign )
```

Zastępuje data[1..2∗nn] przez jej dyskretną transformatę Fouriera, jeśli isign jest równe 1; lub zastępuje data[1..2∗nn] przez nn razy jej odwrotna transformata Fouriera (nieznormalizowaną odwrotną transformatę Fouriera), jeśli isign jest równe −1. data jest tablicą liczb zespolonych o długości nn, lub patrząc inaczej tablicą liczb rzeczywistych o długości 2∗nn. Rozmiar tablicy nn **MUSI** być całkowitą potęgą 2 (procedura tego nie sprawdza!).

```
{
  unsigned long n, mmax, m, j, istep, i;
  double wtemp, wr, wpr, wpi, wi, theta; /* podwójna precyzja do
                                            rekurencji trygonometrycznej */
  float tempr, tempi;

  n = nn << 1; /* n = 2*nn */
  j = 1;

  /*** Odwracanie bitowe ***/
  for (i = 1; i < n; i += 2) {
    if (j > i) {
      /* zamienamy miejscami dwie liczby zespolone */
      SWAP(data[j], data[i]);
      SWAP(data[j+1], data[i+1]);
    }
    m = n >> 1;
    while (m >= 2 && j > m) {
      j -= m;
```

```
      m »= 1;
    }
    j += m;
  }
```

Najpierw sortujemy elementy w kolejności odwróconych bitów, tak by kolejne kawałki, które składamy ze sobą zgodnie ze wzorem Danielsona–Lanczosa (9), zajmowały ciągłe kawałki tablicy i leżały obok siebie.

Algorytm użyty w procedurze four1 z NRC jest trudny do przeanalizowania. Prostszy algorytm, chociaż nie tak szybki, mógłby korzystać z makrodefincji:

```
#define SET_BIT(Value, Bit)   \
  ((Value) |= ((unsigned long)1 « (Bit)))

#define CLEAR_BIT(Value, Bit) \
  ((Value) &= ~((unsigned long)1 « (Bit)))

#define IS_BIT_SET(Value, Bit) \
  (((Value) & ((unsigned long)1 « (Bit))) != 0)

/* poniższa definicja nie jest najlepsza */
#define NBITS(Value) \
  ((int)(log((double)Value)/log(2.0)))
```

Należy pamiętać, że odwracamy bity do NBITS(nn) wyłącznie (czyli do NBITS(nn) − 1 = NBITS(n/2) włącznie.

```

  /** Algorytm Danielsona−Lanczosa **/
  mmax = 2;
  while (n > mmax) { /* zewnętrzna pętla wykonywana log_2(*|lstinline!nn!) razy */
    istep = 2*mmax;

    /* inicjujemy rekurencję trygonometryczną */
    theta = 6.28318530717959/(isign*mmax);
    wtemp = sin(0.5*theta);
    wpr = −2.0*wtemp*wtemp;
    wpi = sin(theta);
    wr = 1.0;
    wi = 0.0;
```

Korzystamy tutaj ze wzoru na funkcje trygonometryczne kąta podwojonego: $\cos 2\theta = 1 - 2\sin^2\theta$. wp traktowane jako zmienna zespolona jest równe $wp = (cos\theta - 1) + i\sin\theta$, zaś $w = e^{2\pi\imath(n-1)}$. Kąt θ jest tak dobrany, że $wp + 1 = e^{2\pi\imath/N}$.

```

    for (m = 1; m < mmax; m += 2) { /* dwie wewnętrzne pętle */
      for (i = m;i <= n;i += istep) {
```

Pierwsza pętla, po m przebiega po kolejnych elementach (liczbach zespolonych) $H_k^{\cdots}$ i $H^{\cdots}$. Druga pętla przebiega po kolejnych częściach (po kolejnych węzłach tego samego poziomu w drzewie rekurencji). Indeks i odpowiada $|$, zaś $j - \nmid$.

```
        j = i + mmax;

        /* to jest wzór Danielsona−Lanczosa */
        tempr = wr*data[j]   − wi*data[j+1];
        tempi = wr*data[j+1] + wi*data[j];
        data[j ]   = data[i ]   − tempr;
        data[j+1]= data[i+1] − tempi;
```

```
            data[i ]   += tempr;
            data[i+1] += tempi;
```

Obliczamy tutaj $H_k^{\cdots} = H_k^{\cdots 1} + W^k H^{\cdots 1}$, gdzie W jest równe wr + $\imath$wi. Korzystamy z tego, że $e^{\alpha i + 2\pi\imath N/2} = -e^{\alpha i}$.

```
        }

        /* rekurencja trygonometryczna */
        wr = (wtemp=wr)*wpr - wi*wpi + wr;
        wi = wi*wpr + wtemp*wpi + wi;
```

Tutaj wyliczamy nową potęgę W za pomocą odpowiedniej rekurencji trygonometrycznej $w_{\mathrm{nowe}} = we^{2\pi\imath/N} = w(1 + wp)$.

```
      }
      mmax = istep;
   }
}

#undef SWAP
```

2 Zastosowania dyskretnej transformaty Fouriera

2.1 Znajdowanie dominującej częstości

Często jest tak, że w doświadczeniach dostajemy sygnał w reprezentacji „czasowej”, a wynik odczytujemy z dominującej częstości. Jeśli nie możemy bezpośrednio mierzyć częstości, jednym z rozwiązań jest wprowadzenie sygnału do komputera (za pomocą przetwornika analogowo-cyfrowego) a następnie potraktowanie go dyskretną (zazwyczaj rzeczywistą) transformatą Fouriera.

Przykładem takiego doświadczenia może być pomiar prędkości cząstek rozpraszających światło w ośrodku za pomocą efektu Dopplera: krzyżujemy w punkcie pomiaru dwie wiązki laserowe z tego samego lasera (rozszczepione poprzednio) i odczytujemy prędkość z częstości dudnień. (Tak naprawdę to pomiar ten wyznacza składową prędkości; przy użyciu dodatkowych wiązek możemy wyznaczyć cały wektor prędkości.)

Innym przykładem jest badanie zjawiska tzw. rezonansu stochastycznego w układach z detektorami progowymi. W najprostszym sformułowaniu mamy dwa detektory progowe o progach A i $-A$. Na początku sygnał wyjściowy jest dla uproszczenia zerowy. Jeśli na wejściu pojawi się sygnał większy niż A to na wyjściu z układu pojawia się sygnał 1, aż do momentu gdy na wejściu pojawi się sygnał mniejszy niż $-A$ (próg detektora „dolnego”). Wtedy na wyjściu pojawia się sygnał -1, aż do sygnału na wejściu większego niż próg A itd. Na wejściu mamy sygnał okresowy (o znanej częstości) o amplitudzie [dużo] mniejszej od wartości progu detektora, w towarzystwie szumu. Jeśli szum jest dostatecznie duży wśród wartości wejściuwych pojawią się wartości ponadprogowe. Wykonujemy analizę fourierowską sygnału wyjściowego i badamy kształt widma wokół częstości podprogowego sygnału okresowego. Powinniśmy zaobserwować, że wraz ze wzrostem szumu następuje coraz lepsze wykrywanie częstości sygnału, aż szum zacznie dominować. Ponieważ do badania potrzebujemy tylko fragment widma, zadanie to można wygodnie rozwiązywać za pomocą zwykłego (powolnego) algorytmu transformaty Fouriera.

2.2 Filtry pasmowe

Najprostszym zastosowaniem transformaty Fouriera jest filtrowanie przez wycinanie (bądź tłumienie) nieporządanych częstości. Możemy np. za pomocą filtru górnoprzepustowego próbować usunąć szum z sygnału, lub za pomocą filtru dolnoprzepustowego wydobyć sygnał wolnozmienny (trend) z wysokozmiennego sygnału.

Gdy przesyłamy dane za pomocą modyfikacji sygnału o danej częstości, czy to za pomocą modulacji amplitudy, czy to za pomocą modulacji fazy, możemy poprawić stosunek sygnału do szumu traktując sygnał przed „zdekodowaniem” filtrem pasmowym przenoszącym częstości wokół częstości nośnej. Oczywiście nie możemy dać filtru zbyt wąskiego, inaczej możemy zakłócić przesyłane dane.

Jeśli badamy sygnał wolnozmienny (n.p. zależność przyspieszenia grawitacyjnego od pływów Księżyca) możemy odfiltrować część szumu za pomocą filtru dolnoprzepustowego. Jeśli wiemy, że częstość sygnału jest większa od jakiejś wartości możemy pozbyć się wolnozmiennego dryfu aparaturowego za pomocą filtru górnoprzepustowego.

2.3 Obliczanie splotu za pomocą transformaty Fouriera

Splotem dwu funkcji $f(t)$ i $g(t)$ nazywamy funkcję

$$(f \otimes g)(t) := \int_{-\infty}^{\infty} d\tau f(\tau) g(t-\tau) \tag{10}$$

Jeśli znamy transformaty Fouriera obu funkcji to splot można łatwo wyliczyć korzystając z tego, że transformata Fouriera splotu jest iloczynem transformat czynników:

$$\mathcal{FT}[f \otimes g](f) = \mathcal{FT}[f](t) \mathcal{FT}[g](t) \tag{11}$$

Zazwyczaj w zastosowaniach te dwie funkcje mają inne znaczenie. Jedna z nich reprezentuje „prawdziwy” sygnał i jest funkcją przynajmniej teoretycznie o nieograniczonej domenie, a druga jest funkcją odpowiedzi aparatury i jest wypikowana wokół zera.

Ponieważ dla dyskretnej (skończonej) transformaty Fouriera sygnały są traktowane jak okresowe, więc gdy prawdziwy sygnał nie jest okresowy to aby uniknąć efektu splatania końca danych z początkiem należy dane dopełnić na końcu zerami na długości około długości funkcji splatanej.

Dla typowego przypadku splotu sygnału z funkcją odpowiedzi, funkcję odpowiedzi przedstawiamy w następujący sposób: wartości dla dodatnich czasów (opóźnień) na początku tablicy, wartości dla ujemnych opóźnień na końcu tablicy (korzystając z „okresowości”: $h_{N-k} \equiv h_{-k}$). Dla uniknięcia efektów brzegowych sygnał dopełniamy zerami na długości częsci ujemnej domeny funkcji odpowiedzi (t.j. na długości m, gdzie $m = \inf_k\{k\colon h_{-k} \neq 0\}$; plus ewentualne dopełnienie zerami do najbliższej potęgi dwójki przy stosowani FFT (algorytmu szybkiej transformaty Fouriera).

2.4 Algorytm szybkiego mnożenia wielomianów

Niech

$$P(x) = \sum_{k=0}^{N-1} a_k x^k$$

będzie wielomianem stopnia $N-1$. Wielomian ten można opisać jednoznacznie albo przez podanie wartości współczynników a_k, albo przez podanie wartości $P(x_n)$ w N różnych punktach. Gdy dwa wielomiany $P(x)$ i $Q(x)$ zadane są ciągami współczynników (a_k) i (b_k), trudno jest policzyć ich iloczyn $P(x)Q(x)$ w tej reprezentacji. Wiadomo natomiast, że $(PQ)(x_n) = P(x_n)Q(x_n)$. Jeśli znajdziemy wartości iloczynu w $\deg(PQ)+1$ punktach, będziemy mogli znaleźć jego współczynniki. Wystarczy policzyć wartości wielomianów $P(x)$ i $Q(x)$ w tych samych $\deg(PQ)+1$ punktach.

Jeśli jako te punkty wybierzemy kolejne pierwiastki (zespolone) odpowiedniego (N–tego, gdzie N jest ilością wystarczającą do wyznaczenia żądanego wielomianu) stopnia z jedności:

$$x_n = e^{2\pi\imath n/N}, \tag{12}$$

to do obliczenia wartości wielomianu przy danych jego wspólczynnikach można bedzie zastosować dyskretną transformatę Fouriera

$$P(x_n) = P(e^{2\pi\imath n/N}) = \sum_{k=0}^{N-1} a_k e^{2\pi\imath kn/N}, \tag{13}$$

a do obliczenia wartości współczynników wielomianu mając zadane wartości (do odtworzenia wielomianu) w tych punktach odwrotną transformatę Fouriera:

Zakładając, ze oba wielomany $P(x)$ i $Q(x)$ są stopnia $N-1$, wystarczy policzyć wartości w $2(N-1)$ punktach (gdyż takiego co najwyżej stopnia może być ich iloczyn). Rozszerzamy je w tym celu do wielomianów $2N-2$. stopnia (dodatkowe współczynniki przyjmując równe zeru), przy pomocy szybkiej transformaty Fouriera wyliczamy wartości w zespolonych pierwiastkach z jedności, mnożymy „punkt po punkcie" wynik i odtransformujemy z powrotem do reprezentacji za pomocą współczynników.

To zadanie można także traktować jako przykład zastosowania splotu dyskretnego dwu wektorów (w tym przypadku zakończonych zerami). Mnożenie dwu wielomianów w reprezentacji „współczynnikowej" robione jest za pomocą splotu. Transformata Fouriera splotu przekłada się na iloczyn transformat. Jest to to samo zadanie przedstawione w nieco innym języku.

W ten sam sposób można mnożyć przez siebie duże liczby, obliczając na końcu przeniesienia.

Więcej informacji n.t. zastosowań transformaty Fouriera w następnej części

Jakub Narębski, Paweł Jankowski

Bibliografia

[1] „Numerical Recipies in C: the Art of Scientific Computing", `http://www.nr.com`

[2] Z. Fortuna, B. Macukow, J. Wąsowski.: „Metody numeryczne"