PSI-3431 Processamento Estatístico de Sinais

Experiência 3: Densidade Espectral de Potência e Ruído de Quantização

Crie um sinal de áudio amostrado a $f_a = 40$kHz

$$s(t) = 0{,}5\cos(2\pi.f_0 t) + 0{,}3\cos(4\pi.300t) + 0{,}15\cos(6\pi.f_0 t),$$

com $f_0 = 100\pi$ Hz, com uma duração de 3s (veja que a frequência é 100π Hz mesmo, e a frequência do segundo componente é 600 Hz, para evitar que o sinal de tempo discreto seja periódico).

Se você for resolver o exercício no Matlab, use a função `quantize2.m` para criar um sinal `sq` quantizado com 5 bits. Você pode usar o comando `sound(sq, fa)` para ouvir o sinal. Se for usar Julia, use

```
using FixedPointNumbers
```

e o comando `sq = Fixed{Int16,4}.(s)` (o comando `Fixed{Int16, k}` especifica números em ponto fixo usando k bits à direita da vírgula - então se você quer simular operações com números entre -1 e 1 com 5 bits, 1 bit é para o sinal e 4 para a parte fracionária). Para ouvir em Julia na linha de comando é necessário gravar o sinal de áudio usando o pacote `WAV` e ouvir fora, mas é mais fácil ouvir o som com o comando `yq = SampleBuf(sq, fa)` do pacote `SampledSignals` (aparece uma janelinha para tocar o sinal, tanto em um notebook do Jupyter, quanto em um arquivo no Atom).

A função `fxfilt` em Julia é chamada com o comando `y = fxfilt(b, a, x, precdupla)`, em que `b` são os coeficientes do numerador do filtro, `a` são os coeficientes do denominador (se `a=[1]`, o filtro é FIR), e `x` é o sinal de entrada. A forma como a saída `y` é calculada depende dos sinais de entrada:

- Se `b`, `a` e `x` forem vetores de ponto flutuante, então a saída é calculada em ponto flutuante normalmente (e o resultado deve o mesmo de usar a função `filt` comum).
- Se `b`, `a` forem ponto flutuante e `x` for ponto fixo, então a saída é calculada em ponto fixo, mas usando os valores exatos dos coeficientes. A precisão da saída é sempre a mesma da da entrada, `x`. Se

`precdupla=true` os cálculos intermediários são feitos em precisão dupla, caso contrário, a saída de cada multiplicador é quantizada para a mesma precisão de `x`.

- Se `b`, `a` e `x` forem todos ponto fixo, então a saída é calculada em ponto fixo. Repare que o número de bits usado para os coeficientes pode ser diferente da precisão da entrada. Se `precdupla=true` os cálculos intermediários são feitos em precisão dupla, caso contrário, a saída de cada multiplicador é quantizada para a mesma precisão de `x`.

- Para ver o efeito da quantização do sinal de entrada, mas fazendo as contas internas do filtro na precisão normal do computador, use o comando

  ```
  sq = Fixed{Int16,B0-1}.(s)
  yq = fxfilt(b, a, Float64.(sq))
  ```

  Isto faz com que a saída do filtro seja calculada em precisão completa, mas considerando o erro de quantização com `B0` bits na entrada (o comando é `Fixed{Int16, B0-1}` porque um bit é de sinal, o restante é a parte fracionária).

- Para quantizar o sinal com `B0` bits, mas fazer as contas com `B` bits (usando precisão dupla para contas intermediárias), use o comando

  ```
  sq = Fixed{Int16,B0-1}.(s)
  yq = fxfilt(b, a, Fixed{Int16,B-1}.(sq), true)
  ```

- Para quantizar os coeficientes com `Bc` bits, use o código

  ```
  bq = Fixed{Int16,Bc-1}.(b)
  aq = Fixed{Int16,Bc-1}.(a)
  sq = Fixed{Int16,B0-1}.(s)
  yq = fxfilt(bq, aq, Fixed{Int16,B-1}.(sq), true)
  ```

A função `quantize2(x, B)` no Matlab arredonda o valor de `x` para o número de bits especificado em `B` (em Matlab você não precisa subtrair um de `B`). Repare que as últimas duas linhas do código escolhem se você quer aproximar o efeito de overflow ou não. Nesta experiência vamos evitar overflow, por isto a última linha está descomentada, e a penúltima está comentada. Para fazer as contas em precisão infinita, use o comando `filter` normal do Matlab, para fazer as contas em ponto fixo, use `filterfx`.

Tarefas: Calcule a relação sinal/ruído (SNR), em dB.

Vamos usar o seguinte filtro para reduzir o efeito do ruído:

- Um filtro IIR $H_I(z)$ usando a aproximação elíptica. Você pode usar as funções `ellipord` e `ellip` para o projeto (seu uso é semelhante ao das funções usadas na Experiência 3).

Projete o filtro usando as especificações abaixo (escolha adequadamente ω_p)

- Faixa de passagem: $0 \leq \omega \leq \omega_p$,
- Faixa de rejeição: $0{,}2\pi$ rad/amostra $= \omega_r \leq \omega \leq \pi$ rad/amostra,
- Atenuação mínima na faixa de rejeição: 40 dB,
- Queda máxima na faixa de passagem $\delta_p = 0{,}05$.

Verifique se os filtros projetados realmente obedecem às especificações. Em seguida, complete as seguintes tarefas:

1. Quantize os coeficientes do filtro, e calcule a resposta em frequência considerando os coeficientes quantizados. Compare com a resposta em frequência desejada (não é necessário refazer o projeto para atender às especificações).

2. Seja $y_Q[n]$ a saída do filtro IIR. Considerando o filtro implementado com precisão infinita (mas usando os coeficientes quantizados), determine a função valor esperado $m_{y_Q}[n] = \mathrm{E}\{y_Q[n]\}$.

3. Seja $y[n]$ a resposta do filtro com a entrada limpa (sem quantização). Defina $\varepsilon_Q[n] = y_Q[n] - y[n]$. Esse sinal pode ser considerado um processo estacionário? Qual é o valor DC e a potência média de $\varepsilon_Q[n]$? Calcule a SNR na saída. Você pode usar a função `impz` para calcular a resposta ao impulso do filtro IIR[1].

4. Use a função `filter` (Matlab) ou `filt` (Julia) para calcular a saída $y_Q[n]$ para uma realização do filtro, usando como entrada o sinal quantizado $s_q[n]$. Use o conceito de ergodicidade para calcular a potência média do ruído na saída do filtro, e compare com o valor teórico do item anterior (você pode considerar o ruído de quantização como sendo um processo ergódico).

[1] Quem mostrar como esse cálculo pode ser feito teoricamente (sem usar a função `impz`) ganha um bônus na correção

5. Use agora a função `filterfx.m` (Matlab) ou `fxfilt` (Julia) para gerar os sinais de saída considerando que o filtro seja implementado usando aritmética de precisão finita com 12 bits para as contas (Use `Fixed{Int16,11}` em Julia, para evitar overflow). Considere que cálculos intermediários possam usar um registrador de precisão dupla.

6. Novamente, calcule o sinal $\varepsilon_T[n]$ para este exemplo, use ergodicidade para calcular a potência média, e calcule a SNR de saída.

7. Para o caso do filtro implementado usando 12 bits do item anterior, calcule teoricamente a potência média do ruído total na saída do filtro. Leia a função `filterfx.m` (ou `fxfilt`) e considere exatamente como as contas são feitas. Compare o resultado teórico com o do item anterior.

8. Qual seria o resultado para o filtro IIR se fosse usada a forma direta canônica vista em aula? (ou seja, aquela em que a parte com realimentação é feita antes da parte FIR, veja a apostila) Não é preciso implementar o filtro, apenas calcular o valor teórico da SNR.