Realizzazione e caratterizzazione di un circuito filtro passa banda RLC[a]

Francesco Polleri[1, b] e Mattia Sotgia[1, c]
(Gruppo A1)
[1] *Dipartimento di Fisica,*
Università degli Studi di Genova, I-16146 Genova,
Italia

(Dated: presa dati 26 ottobre 2021, analisi dati <date>, relazione in data 27 ottobre 2021)

I. INTRODUZIONE

II. METODI

III. RISULTATI

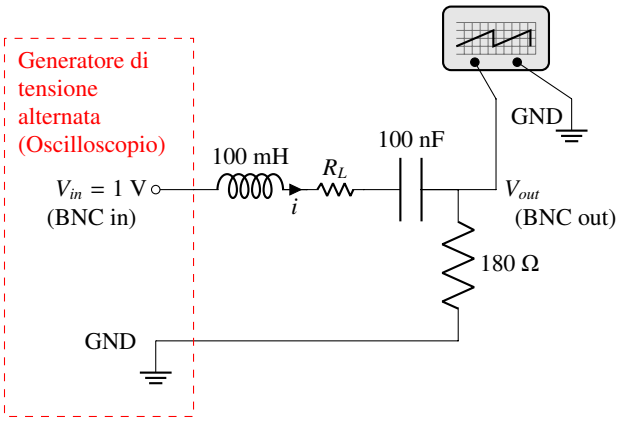

Figura 1 Circuito utilizzato per il filtro passa-banda progettato nell'esperienza, i valori di R, L e C sono i valori nominali riportati sul componente. La resistenza R_L è la resistenza interna all'induttanza, che verifichiamo non essere nulla.

IV. CONCLUSIONE

Appendice A: Programma per analisi dati

```
#include<vector>
#include<cmath>
#include<iostream>
#include<fstream>
#include<string>

#include<TCanvas.h>
#include<TGraphErrors.h>
#include<TF1.h>
#include<TStyle.h>
#include<TAxis.h>
#include<TMath.h>
#include<TLatex.h>
#include<TLegend.h>

const double title_size = 21;

std::string rawdata = "../dati/presa_dati_2021_10_26.txt";

const double R = 175.7; // indicativo valore nominale 200 ohm?
const double C = 100; // indicativo valore nominale 100 nF
const double L = 97.6; // indicativo valore nominale 100 mH
const double R_L = 79.1;

const double R_Hi = 1.785; // kohm

// Con i valori scelti otteniamo che circa v = 1.6 kHz
// Q circa = 5
// A = 1 (che va bene finche' R_L << R)

void print_mmsg(std::string mmsg){
    std::cout << std::endl
        << " **********" << std::endl
        << "    " << mmsg << std::endl
        << " **********" << std::endl
        << std::endl;
}

void print_stat(TF1* _f){
    std::cout << std::endl
        << "** " << "CHI2 / NDF ( PROB. ) "
        << _f->GetChisquare() << " / " << _f->GetNDF() << " ( " << _f->GetProb() << " )"
        << std::endl << std::endl;
}

std::string compatible(double G1, double errG1,
                       double G2, double errG2){
    double abs_values = abs(G2-G1);
    double err_abs_val = 3*sqrt(pow(errG1, 2) + pow(errG2, 2));
    if(abs_values<err_abs_val){
        return "COMPATIBILE";
```

[a] Esperinza n. 2
[b] s5025011@studenti.unige.it
[c] s4942225@studenti.unige.it; In presenza in laboratorio per la presa dati

```
    }
    return "NON-COMPATIBILE";
}

void set_TGraphAxis(TGraphErrors* g, std::string ytitle){
    g->SetTitle("");
    g->GetYaxis()->SetTitle(ytitle.c_str());
    g->GetYaxis()->SetTitleOffset(2);
    g->GetYaxis()->SetTitleFont(43);
    g->GetYaxis()->SetTitleSize(title_size);
    g->GetYaxis()->SetLabelFont(43);
    g->GetYaxis()->SetLabelSize(12);
    g->GetYaxis()->CenterTitle();

    g->GetXaxis()->SetTickLength(0.05);
}

void set_ResidualsAxis(TGraphErrors* rg, std::string xtitle, std::string ytitle="Residui [#sigma]"){
    rg->GetXaxis()->SetTitle(xtitle.c_str());
    rg->GetXaxis()->SetTitleOffset(5);
    rg->GetXaxis()->SetTitleFont(43);
    rg->GetXaxis()->SetTitleSize(title_size);

    rg->GetYaxis()->SetTitle(ytitle.c_str());
    rg->GetYaxis()->SetTitleOffset(2);
    rg->GetYaxis()->SetTitleFont(43);
    rg->GetYaxis()->SetTitleSize(title_size);
    rg->GetYaxis()->CenterTitle();

    rg->GetYaxis()->SetLabelFont(43);
    rg->GetYaxis()->SetLabelSize(12);
    rg->GetYaxis()->SetNdivisions(5, 5, 0);
    rg->GetXaxis()->SetLabelFont(43);
    rg->GetXaxis()->SetLabelSize(12);
    rg->GetXaxis()->CenterTitle();

    rg->GetXaxis()->SetTickLength(0.08);
}

double max_to_stat(double value){
    return value/(std::sqrt(3));
}

// funzione calcolo incertezza a partire da fondo scala (per Qualsiasi grandezza)
// tab. VALORI  | Grandezza misurata | errPercent | partitions | fondoscala (range1)
//              | V (tensione)       | 3.5%       | 8          | variabile
//              | T (periodi)        | ?.?%       | ?          | variabile

double get_VRangeErr(double errPercent, int partitions, double range1){
  return errPercent * partitions *  range1; // TODO: controllare calcolo errori
}
double get_TRangeErr(double range1, double errPercent = 0.0016, int partition = 10){
    return range1 * errPercent * partition;
}

double getH(double vin, double vout){
    return vout / vin;
}

double get_HErr(double Vin, double Vout, double eVin, double eVout){
  return sqrt(pow(eVout / Vin, 2) + pow(eVin * Vout / pow(Vin, 2), 2));
}

double get_phi(double T, double dt){
    return 2 * M_PI * dt / T;
}

double get_phiErr(double T, double dt, double eT, double edt){
    return 2 * M_PI * sqrt(pow(edt/T, 2) + pow(dt * eT/(pow(T, 2)), 2));
}

void analisi_RLC_filter(){

    // todo:
    // * leggere file formato:
    // Vin | scalaVin | Vout | scalaVout | T | scalaT | dt | scaladt
    // * trascrivere i file con gli errori:
    // Vin | eVin | Vout | eVout | T | eT | dt | edt
    // * calcolare i valori H, eH, phi, ephi, w, ew
    // H | eH | phi | ephi | w | ew
    // H = Vin/Vout
    // phi = 2 * pi * dt / T
    // w = 2 * pi / T -> meglio forse usare v = 1 / T [Hz]?


    gStyle->SetFrameLineWidth(0);
    gStyle->SetTextFont(43);
    gStyle->SetLineScalePS(1);

    std::ifstream data(rawdata.c_str());

    std::ofstream out_rawdata("../misc/rawdata.txt"); // carbon copy of original data
    std::ofstream out_cleandata("../misc/cleandata.txt"); // values from rawdata with error
    std::ofstream out_computeddata("../misc/computeddata.txt"); // computed data for final graph

    double Vin, fsVin, Vout, fsVout, T, fsT, dt, fsdt;

    TCanvas* c1 = new TCanvas("c1", "", 600, 1000); // ! Modificare per avere grafico orizzontale piu pratico
    c1->SetMargin(0.16, 0.06, 0.12, 0.06);
    c1->SetFillStyle(4000);
    c1->Divide(1,2);

    // Analisi 1mo diagramma di BODE, |H(w)| su w
    c1->cd(1);

    TGraphErrors* H_plot = new TGraphErrors();
    H_plot->SetName("H_plot");
    TF1* H_fit = new TF1("Hf", "1/sqrt([0]+pow([1],2)*(pow(x/[2]-[2]/x, 2)))"); // ! Controllare formule
    H_fit->SetParameters(1, 5, 2000);
    // [0] = A = (1 + R_L / R)^2
    // [1] = Q = fattore di qualita = 1/(R C w_0)
    // [2] = w_0


    TGraphErrors* H_resd = new TGraphErrors();
    TF1* H_res_f = new TF1("H_rf", "0", 10, 10e6);
    H_res_f->SetLineStyle(2);

    TLatex* header = new TLatex();
    header->SetTextFont(43);
```

```
header->SetTextSize(15);

TPad* Hp1 = new TPad("", "", 0.0, 0.3, 1.0, 1.0);
TPad* Hp2 = new TPad("", "", 0.0, 0.0, 1.0, 0.295);
Hp1->SetMargin(0.14, 0.06, 0.0, 0.06);
Hp1->SetFillStyle(4000);
Hp1->SetLogx();
Hp1->SetLogy();
Hp1->Draw();
Hp2->SetMargin(0.14, 0.06, 0.4, 1.0);
Hp2->SetFillStyle(4000);
Hp2->SetLogx();
Hp2->Draw();


// Analisi 2do diagramma di BODE, phi su w
c1->cd(2);

TGraphErrors* phi_plot = new TGraphErrors();
phi_plot->SetName("phi_plot");
TF1* phi_fit = new TF1("phi_f", "-atan([1]*(x/[2]-[2]/x)/sqrt([0]))"); // ! Controllare formule
phi_fit->SetParameters(1, 5, 2000);
// [0] = A = (1 + R_L / R)^2
// [1] = Q = fattore di qualita = 1/(R C w_0)
// [2] = w_0

TGraphErrors* phi_resd = new TGraphErrors();
TF1* phi_res_f = new TF1("phi_rf", "0");
phi_res_f->SetLineStyle(2);

TLatex* phi_header = new TLatex();
phi_header->SetTextFont(43);
phi_header->SetTextSize(15);

TPad* phi_p1 = new TPad("", "", 0.0, 0.3, 1.0, 1.0);
TPad* phi_p2 = new TPad("", "", 0.0, 0.0, 1.0, 0.295);
phi_p1->SetMargin(0.14, 0.06, 0.0, 0.06);
phi_p1->SetFillStyle(4000);
phi_p1->SetLogx();
phi_p1->Draw();
phi_p2->SetMargin(0.14, 0.06, 0.4, 1.0);
phi_p2->SetFillStyle(4000);
phi_p2->SetLogx();
phi_p2->Draw();


for(int i=0; data >> Vin >> fsVin >> Vout >> fsVout >> T >> fsT >> dt >> fsdt; i++){
    out_rawdata << Vin << " " << fsVin << " " << Vout << " " << fsVout << " " << T << " " << fsT << " " << dt << " " << fsdt << std::endl;
    double eVin, eVout;
    if(fsVin<=0.01){
        eVin = max_to_stat(get_VRangeErr(0.045, 8, fsVin));
    }else{
        eVin = max_to_stat(get_VRangeErr(0.035, 8, fsVin));
    }
    if(fsVout<=0.01){
        eVout = max_to_stat(get_VRangeErr(0.045, 8, fsVout));
    }else{
        eVout = max_to_stat(get_VRangeErr(0.035, 8, fsVout));
    }
    double eT = max_to_stat(get_TRangeErr(fsT));
    double edt = max_to_stat(get_TRangeErr(fsdt));

    out_cleandata << Vin << " " << eVin << " " << Vout << " " << eVout << " " << T << " " << eT << " " << dt << " " << edt << std::endl;

    H_plot->SetPoint(i, 1 / T, Vout / Vin);
    H_plot->SetPointError(i, eT/pow(T, 2), get_HErr(Vin, Vout, eVin, eVout));

    phi_plot->SetPoint(i, 1 / T, 2 * M_PI * dt / T);
    phi_plot->SetPointError(i, eT/pow(T, 2), get_phiErr(T, dt, eT, edt));


    out_computeddata << Vout / Vin << " " << get_HErr(Vin, Vout, eVin, eVout) << " "
                     << 2 * M_PI * dt / T << " " << 2 * M_PI * sqrt(pow(edt/T, 2) + pow(dt * eT/(pow(T, 2)), 2)) << " "
                     << 1 / T << " " << eT/pow(T, 2) << std::endl;
}
out_rawdata << "EOF" << std::endl;
out_cleandata << "EOF" << std::endl;
out_computeddata << "EOF" << std::endl;


// Grafico 1 Bode
print_mmsg("PRIMO DIAGRAMMA DI BODE (AMPIEZZA)");
Hp1->cd();
H_plot->Draw("ap");
H_plot->Fit("Hf");

std::string H_stat="#chi^{2}/ndf (prob.) = "
        +std::to_string(H_fit->GetChisquare())+"/"
        +std::to_string(H_fit->GetNDF())
        +" ("+std::to_string(H_fit->GetProb())+")";

header->DrawLatexNDC(0.35, 0.15, ("#splitline{#bf{A} #it{1#circ diagramma di Bode}}{" + H_stat + "}").c_str());

print_stat(H_fit);

// RESIDUI
Hp2->cd();

for(int i=0; i<H_plot->GetN(); i++){
        H_resd->SetPoint(i, H_plot->GetX()[i], (H_plot->GetY()[i] - H_fit->Eval(H_plot->GetX()[i]))/H_plot->GetEY()[i]);
        H_resd->SetPointError(i, 0, 1);
}
H_resd->Draw("ap");
H_res_f->Draw("same");

double A_amp= H_fit->GetParameter(0);
double err_A_amp = H_fit->GetParError(0);
double Q_amp = H_fit->GetParameter(1);
double err_Q_amp = H_fit->GetParError(1);
double frequenza_taglio_amp = H_fit->GetParameter(2);
double err_frequenza_taglio_amp = H_fit->GetParError(2);

std::cout << "A da |H(w)| = (1 + R_L / R)^2 = " << A_amp << " +/- " << err_A_amp << std::endl
        << "Fattore di Qualita' da |H(w)|, Q = " << Q_amp << " +/- " << err_Q_amp << std::endl
        << "Frequenza di Taglio da |H(w)|, v = " << frequenza_taglio_amp << " +/- " << err_frequenza_taglio_amp << " Hz" << std::endl;

// Grafico 2 Bode
print_mmsg("SECONDO DIAGRAMMA DI BODE (FASE)");
phi_p1->cd();
phi_plot->Draw("ap");
```

```
    phi_plot->Fit("phi_f");

    std::string phi_stat="#chi^{2}/ndf (prob.) = "
            +std::to_string(phi_fit->GetChisquare())+"/"
            +std::to_string(phi_fit->GetNDF())
            +" ("+std::to_string(phi_fit->GetProb())+")";

    phi_header->DrawLatexNDC(0.35, 0.15, ("#splitline{#bf{B} #it{2#circ diagramma di Bode}}{" + phi_stat + "}").c_str());

    print_stat(phi_fit);

    // RESIDUI
    phi_p2->cd();

    for(int i=0; i<phi_plot->GetN(); i++){
            phi_resd->SetPoint(i, phi_plot->GetX()[i], (phi_plot->GetY()[i] - phi_fit->Eval(phi_plot->GetX()[i]))/phi_plot->GetEY()[i]);
            phi_resd->SetPointError(i, 0, 1);
    }
    phi_resd->Draw("ap");
    phi_res_f->Draw("same");

    double A_fase = H_fit->GetParameter(0);
    double err_A_fase = H_fit->GetParError(0);
    double Q_fase = H_fit->GetParameter(1);
    double err_Q_fase = H_fit->GetParError(1);
    double frequenza_taglio_fase = H_fit->GetParameter(2);
    double err_frequenza_taglio_fase = H_fit->GetParError(2);

    std::cout << "A da phi(w) = (1 + R_L / R)^2 = " << A_fase << " +/- " << err_A_fase << std::endl
            << "Fattore di Qualita' da phi(w), Q = " << Q_fase << " +/- " << err_Q_fase << std::endl
            << "Frequenza di Taglio da phi(w), v = " << frequenza_taglio_fase << " +/- " << err_frequenza_taglio_fase << " Hz" << std::endl;

    std::cout << std::endl << "** Verifica compatibilita => " << compatible(frequenza_taglio_amp, err_frequenza_taglio_amp, frequenza_taglio_fase,
           err_frequenza_taglio_fase) << std::endl;


    set_TGraphAxis(H_plot, "#left|H(#nu)#right| [a. u.]");
    set_ResidualsAxis(H_resd, "Frequenza #nu [Hz]");


    set_TGraphAxis(phi_plot, "Fase #varphi(#nu) [rad]");
    set_ResidualsAxis(phi_resd, "Frequenza #nu [Hz]");

    // TODO: salvare il file come pdf

    return;
}

#ifndef __CINT__
int main(){
    analisi_RLC_filter();
    return 0;
}
#endif
```