

Introduzione ai circuiti e Studio Di Filtri Del Primo Ordine^a

Francesco Polleri^{1, b} e Mattia Sotgia^{1, c}

(Gruppo A1)

¹Dipartimento di Fisica,
Università degli Studi di Genova, I-16146 Genova,
Italia

(Dated: presa dati 19 ottobre 2021, analisi dati e relazione in data 26 ottobre 2021)

NOTE SULLA STRUMENTAZIONE

I. INTRODUZIONE

Vogliamo costruire e verificare il funzionamento di un filtro passa-basso (*low-pass filter*) o passa-alto (*high-pass filter*). Realizziamo quindi a partire da un circuito schematizzato in Figura 1 il prototipo sulla basetta in laboratorio, e quindi colleghiamo poi

II. METODI

III. RISULTATI

IV. CONCLUSIONE

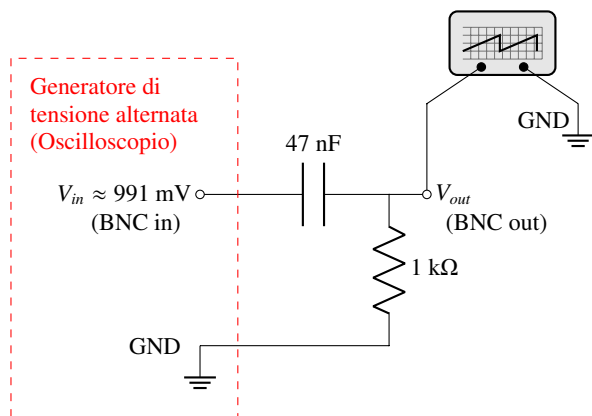


Figura 1 Circuito utilizzato per il filtro passa-alto progettato nell'esperienza, i valori di R e C sono i valori nominali riportati sul componente.

Appendice A: Output analisi dati

```
Processing analisi_RC_filter.C...

*****
PRIMO DIAGRAMMA DI BODE (AMPIEZZA)
*****

FCN=7.55295 FROM MIGRAD STATUS=CONVERGED 22 CALLS 23 TOTAL
EDM=4.53221e-08 STRATEGY= 1 ERROR MATRIX ACCURATE
EXT PARAMETER STEP FIRST
NO. NAME VALUE ERROR SIZE DERIVATIVE
1 p0 3.84779e+03 6.24227e+01 8.91082e-02 -4.82311e-06

** CH12 / NDF ( PROB. ) 7.55295 / 9 ( 0.579748 )
```

^a Esperinza n. 1

^b s5025011@studenti.unige.it; In presenza in laboratorio per la presa dati

^c s4942225@studenti.unige.it

Frequenza di Taglio da $|H(\omega)|$, $v = 3847.79 \pm 62.4227$ Hz

SECONDO DIAGRAMMA DI BODE (FASE)

EXT	PARAMETER	VALUE	ERROR	SIZE	DERIVATIVE
NO.	NAME	VALUE	ERROR	SIZE	DERIVATIVE
1	p0	3.78558e+03	6.12585e+01	3.28702e-05	-3.66507e-02

** CHI2 / NDF (PROB.) 20.0293 / 9 (0.0177321)

Frequenza di Taglio da $\phi(\omega)$, $v = 3785.58 \pm 61.2585$ Hz

** Verifica compatibilita => COMPATIBILE

Appendice B: Programma di 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_19_seconda_versione.txt";
std::string old_rawdata = "../dati/test_dati.txt";
// I dati sono stati ricavati dal file dati.dat forniti su aulaweb, svolgendo i seguenti calcoli per rendere il file come
// previsto per l'esperienza nel formato: Vin | scalaVin | Vout | scalaVout | T | scalaT | dt | scaladt
// * Vin e' fissato al valore di 5V, Vout e' quindi ricavato come ampiezza * 5
// * scalaVin e scalaVout sono state impostate a 10mV, ovvero 0.01V
// * dal valore di v, la frequenza, e' ottenuto il valore di T, come T=1/v, e la scalaT e' scelta come 1/100 del valore
// * il valore di dt e' ricavato dal valore della fase: se la fase vale phi = 2 * M_PI * dt / T, allora posso ricavare
// dt come dt = phi * T / ( 2 * M_PI ), e la scaladt e' scelta come 1/100 del valore di dt

// fisso i valori di R e C ???
const double R = 50;
const double C = 0.000000000001;

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";
    }
    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)          | %.7%          | ?          | variabile

double get_VRangeErr(double errPercent, int partitions, double rangel){
    return errPercent * partitions * rangel; // TODO: controllare calcolo errori
}
double get_TRangeErr(double rangel, double errPercent = 0.0016, int partition = 10){
    return rangel * 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_RC_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->SetLineStylePS(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);
    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(1+(pow([0]/x,2)))");
    H_fit->SetParameter(0, 3e3);

    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([0]/x)", 100, 1e5);
    phi_fit->SetParameter(0, 3e3);
    phi_fit->SetParLimits(0, 1e3, 1e4);

    TGraphErrors* phi_resd = new TGraphErrors();
    TF1* phi_res_f = new TF1("phi_rf", "0", 100, 1e5);
    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 << " " << dt << " " << fsdt << std::endl;
        double eVin, eVout;
        if(fsVin<=0.01){
            eVin = max_to_stat(get_VRangeErr(0.045, 8, fsVin)); // ! RIVEDERE calcolo errore
        }else{

```

```

    eVin = max_to_stat(get_VRangeErr(0.035, 8, fsVin)); // ! RIVEDERE calcolo errore
}
if(fsVout<=0.01){
    eVout = max_to_stat(get_VRangeErr(0.045, 8, fsVout)); // ! RIVEDERE calcolo errore
}else{
    eVout = max_to_stat(get_VRangeErr(0.035, 8, fsVout)); // ! RIVEDERE calcolo errore
}
double eT = max_to_stat(get_TRangeErr(fsT)); // ! RIVEDERE calcolo errore
double edt = max_to_stat(get_TRangeErr(fsdt)); // ! RIVEDERE calcolo errore

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)); // // ! RIVEDERE calcolo errore

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)); // // ! RIVEDERE calcolo errore

    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", "", "", 100, 1e5);

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

header->DrawLatexNDC(0.35, 0.15, ("#splitline{#bf{A}}_{\omega}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->GetY()[i]);
    H_resd->SetPointError(i, 0, 1);
}
H_resd->Draw("ap");
H_res_f->Draw("same");

double frequenza_taglio_amp = H_fit->GetParameter(0);
double err_frequenza_taglio_amp = H_fit->GetParError(0);

std::cout << "Frequenza_di_Taglio_da_{H(w)}_{\omega}=" << 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", "", "", 100, 1e5);

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

phi_header->DrawLatexNDC(0.35, 0.15, ("#splitline{#bf{B}}_{\omega}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->GetY()[i]);
    phi_resd->SetPointError(i, 0, 1);
}
phi_resd->Draw("ap");
phi_res_f->Draw("same");

double frequenza_taglio_fase = phi_fit->GetParameter(0);
double err_frequenza_taglio_fase = phi_fit->GetParError(0);

std::cout << "Frequenza_di_Taglio_da_{phi(w)}_{\omega}=" << frequenza_taglio_fase << " +/- " << err_frequenza_taglio_fase << " Hz" << std::endl;

std::cout << "**_Verifica_compatibilita_{\omega}=" << 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]}");

c1->SaveAs("../fig/RC_bode.pdf");
// TODO: salvare il file come pdf

    return;
}

#ifdef __CINT__
int main(){
    analisi_RC_filter();
    return 0;
}
#endif

```