Relazione di Elettronica

Amplificatori Operazionali

Francesco Forcher

Università di Padova, Facoltà di Fisica
francesco.forcher@studenti.unipd.it
Matricola: 1073458

Enrico Lusiani

Università di Padova, Facoltà di Fisica
enrico.lusiani@studenti.unipd.it
Matricola: 1073300

Laura Buonincontri

Università di Padova, Facoltà di Fisica
laura.buonincontri@studenti.unipd.it
Matricola: 1073131

9 maggio 2016

Sommario

L'obiettivo dell'esperienza è la misura della curva di trasferimento di un amplificatore (in configurazione invertente e non invertente) e lo studio della sua risposta in frequenza (in configurazione non invertente).

Indice

I. Schema Circuiti

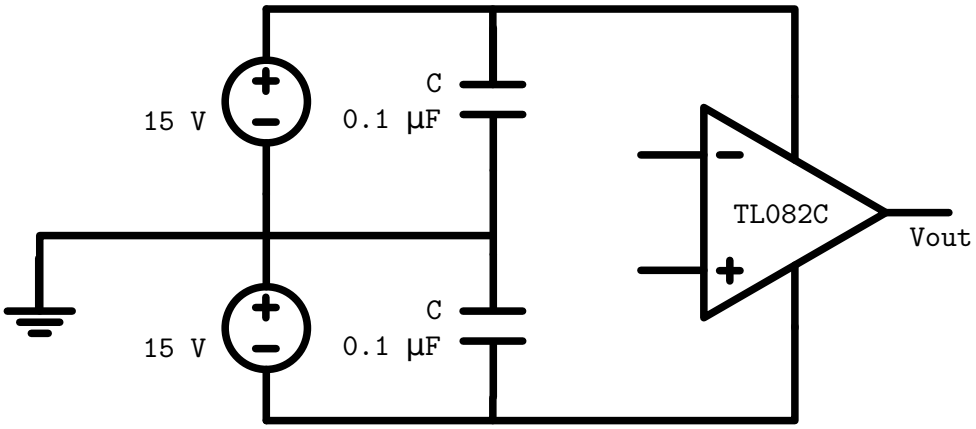


Figura 1: *Alimentazione OpAmp*

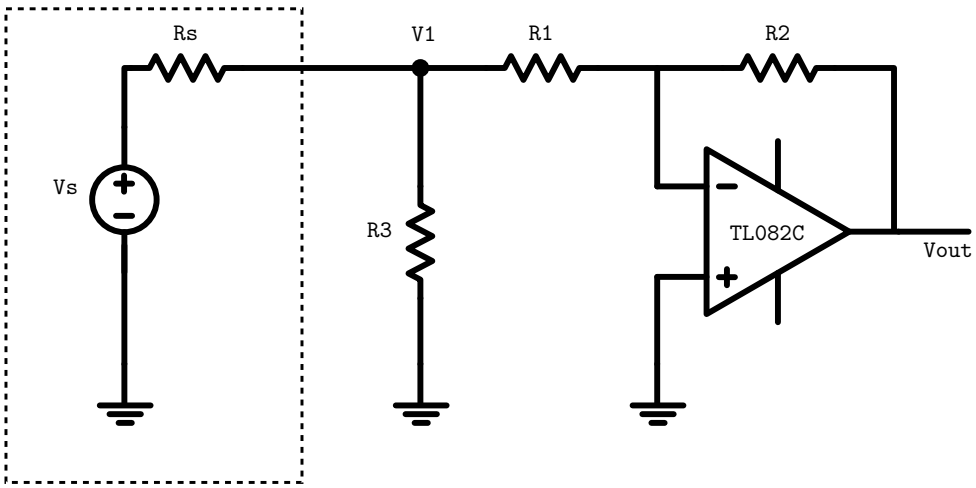


Figura 2: *Amplificazione configurazione invertente*

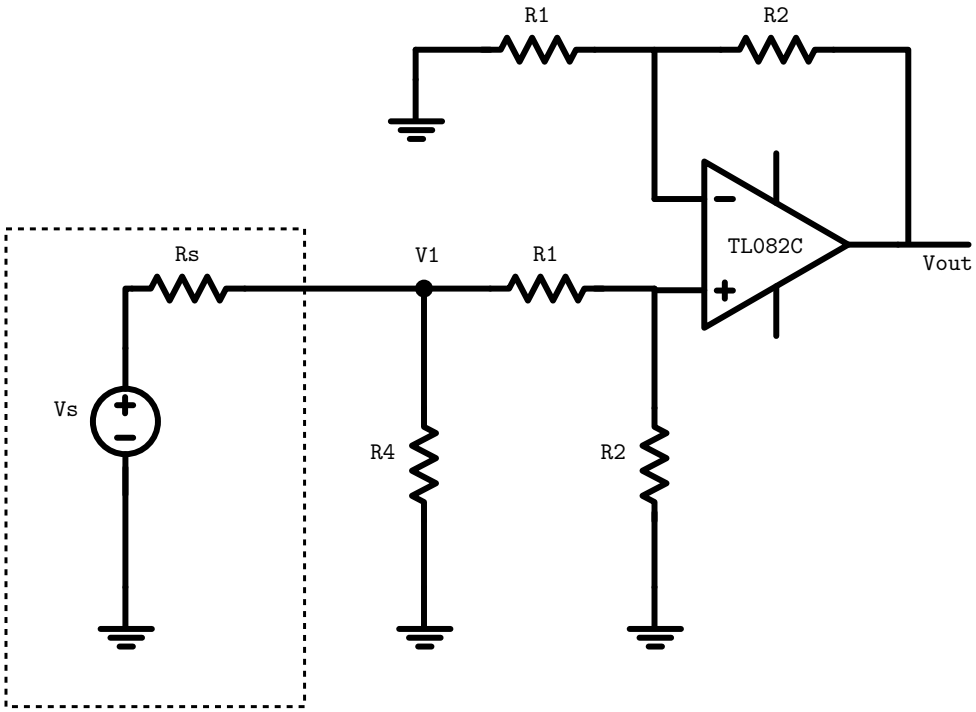


Figura 3: *Amplificazione configurazione non invertente*

II. PARTE I

II.I Amplificatore invertente

Schema amplificatore invertente: Le resistenze sono state scelte in modo da avere guadagno $A = -10\frac{V}{V}$
$R_1 = 9.85 \pm 0.05\,k\Omega$
$R_2 = 101.3 \pm 0.6\,k\Omega$
$R_3 = 56.0 \pm 0.3\,\Omega$

Per il calcolo degli errori sul valore delle resistenze, lette sull'Agilent U1232A, è stata utilizzata la seguente formula:

$$\sigma_{tot} = \sqrt{\sigma_{\%}^2 + \sigma_{dgt}^2}$$

Per il calcolo delle σ_{tot} è stato cercato del datasheet dello strumento, l'errore percentuale e di digit corrispondente al fondo scala utilizzato.

I.1 Calcolo amplificazione

La relazione tra le resistenze, affichè soddisfino la richiesta A=10 è la seguente:

$$\frac{V_1 - V_n}{R_1} = \frac{V_n - V_0}{R_2}$$
$$V_n = 0$$
$$\frac{V_1}{R_1} = \frac{-V_0}{R_2}$$
$$V_0 = -\frac{R_2}{R_1} \cdot V_1$$

Da cui si ricava la relazione per il calcolo di A.

I.2 Analisi

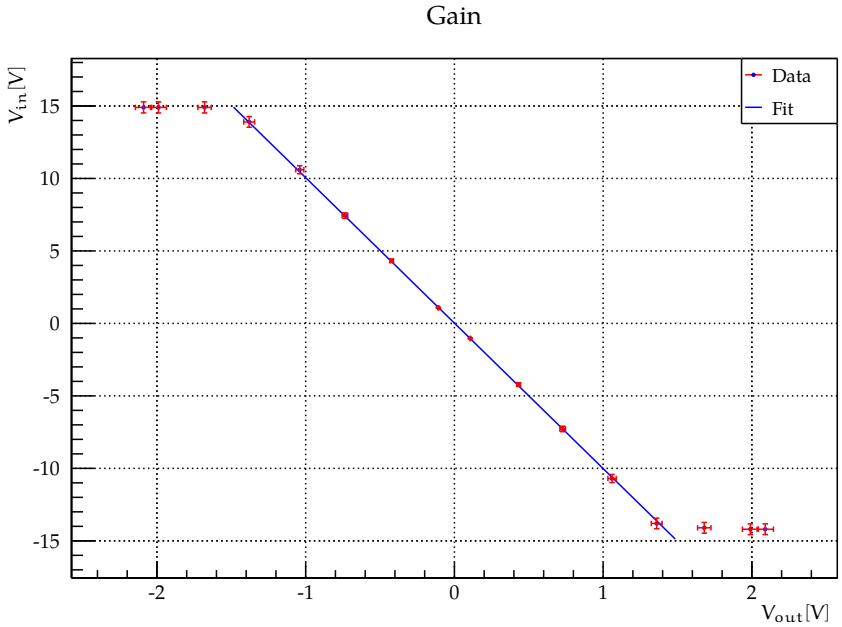
La stima di A teorica, a partire dalle resistenze misurate è:
$A_{teorica} = 10.28 \pm 0.08$

Le misure sono state fatte applicando una tensione sinusoidale di frequenza $f = 1\,kHz$, variando l'ampiezza tra $0.2V_{pp}$ e $4V_{pp}$.

In seguito è stato fatto il grafico della curva di trasferimento di un amplificatore invertente.

In seguito sono presentati i dati del **Grafico 1** acquisiti in laboratorio, con i rispettivi errori:

Grafico 1 Curva di trasferimento di un amplificatore invertente



Per il calcolo degli errori sui valori di V_{in} e V_{out} letti sull'oscilloscopio, è stata utilizzata la seguente formula:

$$\sigma_{tot} = \sqrt{(0.02 \cdot V_{letto})^2 + (0.06 \cdot V_{div})^2}$$

E' stata fatta l'interpolazione lineare dei punti nel **Grafico 1** pesata dei punti compresi tra 0 e 1.5 V.
$q = 0.02 \pm 0.03\,V$
$m = -10.0 \pm 0.1\,\frac{V}{V}$

II.II Amplificatore non invertente

Schema amplificatore non invertente: Le resistenze sono state scelte in modo da avere guadagno $A = 10\frac{V}{V}$
$R_{1,up} = 9.91 \pm 0.05\,k\Omega$
$R_{1,down} = 9.85 \pm 0.05\,k\Omega$
$R_{2,up} = 99.7 \pm 0.6\,k\Omega$
$R_{2,down} = 101.3 \pm 0.6\,k\Omega$
$R_4 = 56.0 \pm 0.3\,\Omega$

Tabella 1: *Dati curva di trasferimento*

$V_{in+} \pm \sigma_{V_{in+}}$ (V)	$V_{in-} \pm \sigma_{V_{in-}}$(V)	FS (V)	$V_{out+} \pm \sigma_{V_{out+}}$(V) (V)	$V_{out-} \pm \sigma_{V_{out-}}$(V)	FS (V)
1.06 ± 0.03	-1.04± 0.03	0.3	-10.7 ± 0.3	10.6± 0.3	3
0.107± 0.003	-0.108± 0.003	0.03	-1.04± 0.03	1.08± 0.03	0.3
0.43± 0.01	-0.422± 0.01	0.12	-4.2± 0.1	4.32± 0.1	1.2
0.73± 0.02	-0.74± 0.02	0.2	-7.3± 0.2	7.4± 0.2	2
1.36± 0.04	-1.38± 0.04	0.4	-13.8± 0.4	13.9± 0.4	4
1.68± 0.05	-1.68± 0.05	0.5	-14.1± 0.4	14.9± 0.4	4
1.99± 0.05	-1.99± 0.05	0.6	-14.2± 0.4	14.9± 0.4	4
2.09± 0.06	-2.09± 0.06	0.6	-14.2± 0.4	14.9± 0.4	4

II.1 Calcolo amplificazione

La relazione tra le resistenze, affichè soddisfino la richiesta A=10 è la seguente:
Nell'ingresso non invertente:

$$\frac{V_1 - V_p}{R_{1down}} = \frac{V_p}{R_{2down}}$$

$$\frac{V_1}{R_{1down}} = \frac{V_p}{R_{1down}} + \frac{V_p}{R_{2down}} = V_p \left(\frac{1}{R_{1down}} + \frac{1}{R_{2down}} \right)$$

Nell'ingresso invertente:

$$\frac{V_0 - V_n}{R_{2up}} = \frac{V_n}{R_{1up}}$$

$$\frac{V_0}{R_{2up}} = \frac{V_n}{R_{1up}} + \frac{V_n}{R_{2up}} = V_n \left(\frac{1}{R_{1up}} + \frac{1}{R_{2up}} \right)$$

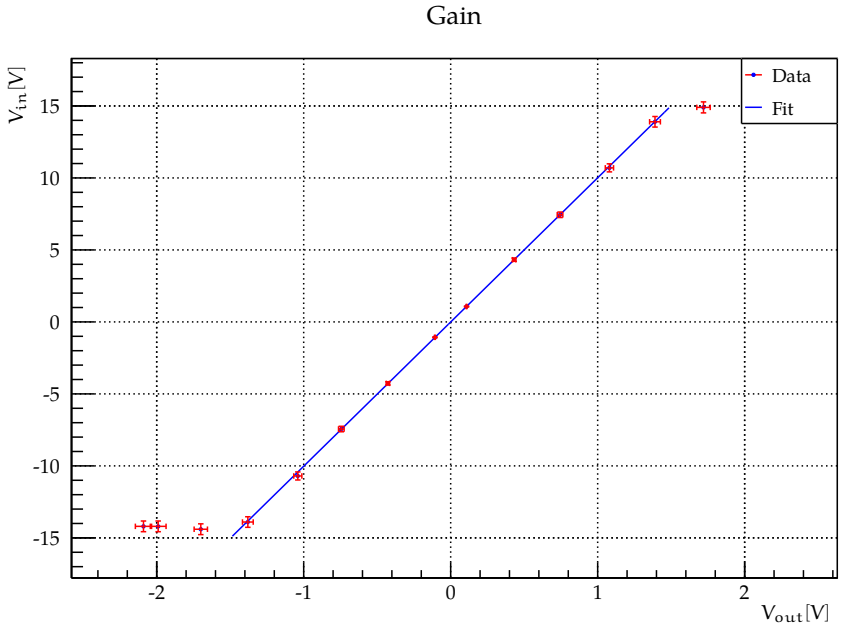
Poichè

$$V_p = V_n$$

$$\frac{V_1}{R_{1down}} \frac{1}{\left(\frac{1}{R_{1down}} + \frac{1}{R_{2down}} \right)} = \frac{V_0}{R_{2up}} \frac{1}{\left(\frac{1}{R_{1up}} + \frac{1}{R_{2up}} \right)}$$

$$V_0 = \frac{R_{2up}}{R_{1down}} \cdot V_1 \frac{\left(\frac{1}{R_{1up}} + \frac{1}{R_{2up}} \right)}{\left(\frac{1}{R_{1down}} + \frac{1}{R_{2down}} \right)}$$

Grafico 2 Curva di trasferimento di un amplificatore invertente



Da cui si ricava la relazione per il calcolo di A. Se poi si assume che $R_{1down} = R_{1up}$ e $R_{2down} = R_{2up}$, la relazione si semplifica a

$$V_0 = \frac{R_{2up}}{R_{1down}} \cdot V_1$$

II.2 Analisi

La stima di A teorica, a partire dalle resistenze misurate è:
$A_{teorica} = 10.08 \pm 0.07$

Le misure sono state fatte applicando una tensione sinusoidale di frequenza $f = 1$ kHz, variando l'ampiezza tra $0.2V_{pp}$ e $4V_{pp}$.

In seguito è stato fatto il grafico della curva di trasferimento di un amplificatore non invertente.

In seguito sono presentati i dati del **Grafico 2** acquisiti in laboratorio, con i rispettivi errori:

E' stata fatta l'interpolazione lineare pesata dei punti nel **Grafico 2** compresi tra 0 e 1.5 V.
$q = -0.007 \pm 0.03\,V$
$m = 10.0 \pm 0.1\,\frac{V}{V}$

Tabella 2: *Dati curva di trasferimento*

$V_{in+} \pm \sigma_{V_{in+}}$ (V)	$V_{in-} \pm \sigma_{V_{in-}}$(V)	FS (V)	$V_{out+} \pm \sigma_{V_{out+}}$(V) (V)	$V_{out-} \pm \sigma_{V_{out-}}$(V)	FS (V)
1.08± 0.03	-1.04± 0.03	0.3	10.7± 0.3	-10.7± 0.3	3
0.108± 0.003	-0.107± 0.003	0.03	1.07± 0.003	-1.07± 0.03	0.3
0.43± 0.01	-0.43± 0.01	0.120	4.3± 0.1	-4.3± 0.1	1.2
0.74± 0.02	-0.74± 0.02	0.2	7.4± 0.2	-7.4± 0.2	2
1.39± 0.04	-1.38± 0.04	0.4	13.9± 0.4	-13.9± 0.4	4
1.72± 0.05	-1.70± 0.05	0.5	14.9± 0.4	-14.4± 0.4	4
2.04± 0.05	-1.99± 0.05	0.6	14.7± 0.4	-14.2± 0.4	4
2.14± 0.06	-2.09± 0.06	0.6	14.9± 0.4	-14.2± 0.4	4

III. PARTE II

Le misure sono state effettuate sull'amplificatore non invertente utilizzato al punto precedente e applicando una tensione sinusoidale di frequenza variabile mantenendo l'ampiezza $V_s = 2V_{pp}$.

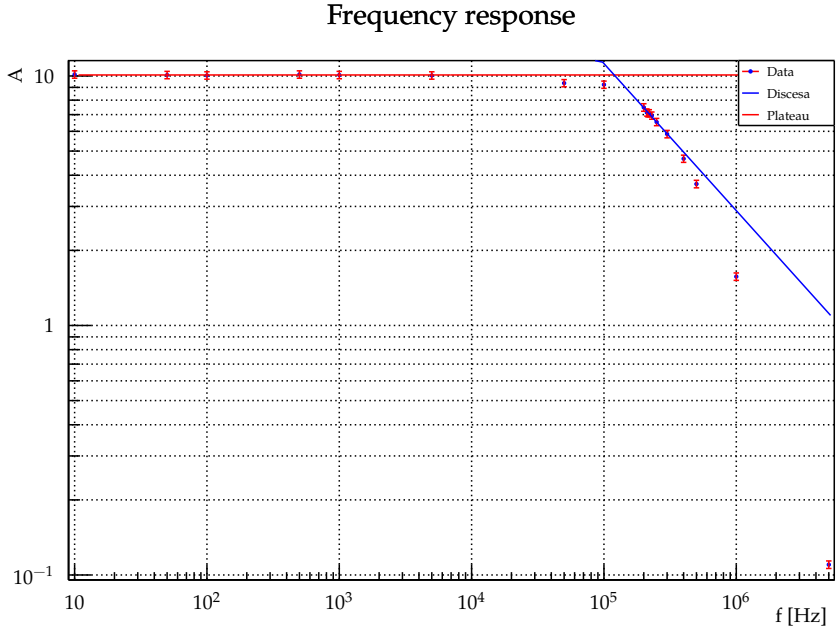
III.I Amplificatore con A=10

Le resistenze inserite sono le stesse dello schema precedente, e quindi anche l'amplificazione teorica. In seguito è presentato il grafico della risposta in frequenza di un amplificatore invertente con A=10. In seguito sono presentati i dati del **Grafico 3** acquisiti in laboratorio.

Sono state interpolate separatamente la zona di plateau e di discesa del **Grafico 3**, ottenendo come risultati:
$A_{plateau} = 10.1 \pm 0.1$
$q = 4.0 \pm 0.6$
$m = -0.6 \pm 0.1$

La closed-loop bandwidth, f_b, del circuito, è stata ricavata tradformando il grafico come $y' = log_{10}(y)$, interpolando la parte costante, traslandola di 3dB ($log_{10}(\sqrt{2})$) verso il basso e intersecandola con la retta ottenuta interpolando la parte di discesa. La f_b era poi l'esponenziale in base 10 dell'intersezione. L'errore è stato poi calcolato effettuando una propagazione degli errori sulla f_b, tenendo conto anche dell'alta correlazione tra i coefficienti della retta

Grafico 3 Risposta in frequenza di un amplificatore non invertente con A=10



(−0.9995).
$f_b = 215 \pm 7$ kHz
Il GBP è
$GBP = A_{CL} \cdot f_b = 2.17 \pm 0.08$ MHz

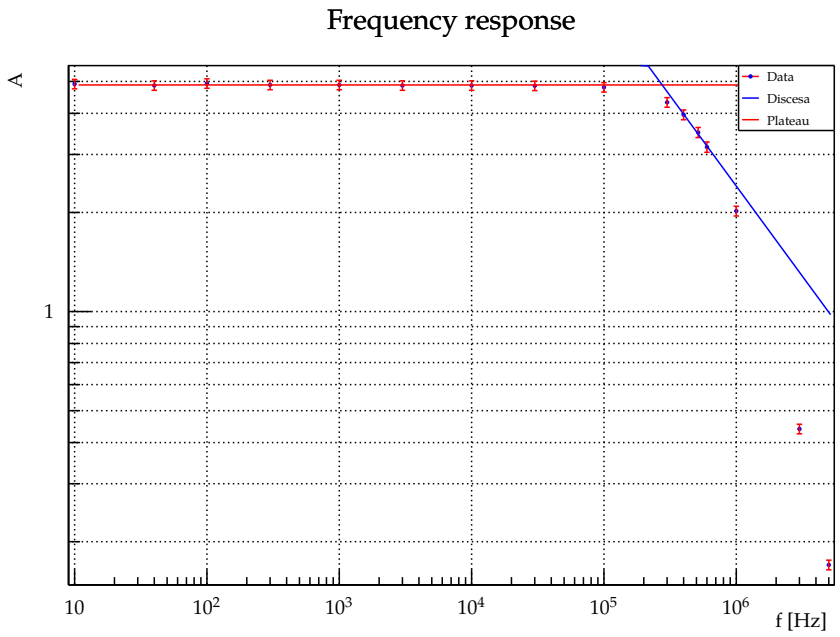
III.II Amplificatore con A=5

Le resistenze inserite sono state sostituite con:
$R_{1,up} = 5.54 \pm 0.03\,k\Omega$
$R_{1,down} = 5.54 \pm 0.03\,k\Omega$
$R_{2,up} = 26.9 \pm 0.1\,k\Omega$
$R_{2,down} = 26.9 \pm 0.2\,k\Omega$
$R_4 = 56.0 \pm 0.3\,\Omega$

L'amplificazione teorica è perciò
$A_{teorica} = 4.86 \pm 0.03$ In seguito è presentato il grafico della risposta in frequenza di un amplificatore non invertente con A=5. In seguito sono presentati i dati del **Grafico 4** acquisiti in laboratorio.

Sono state interpolate separatamente la zona di plateau e di discesa del **Grafico 4**, ottenendo come risultati:
$A_{plateau} = 4.88 \pm 0.06$
$q = 3.7 \pm 0.7$
$m = -0.5 \pm 0.1$

Grafico 4 Risposta in frequenza di un amplificatore non invertente con A=5



Dall'interpolazione si è poi ricavato $f_b = 520 \pm 20\,\text{kHz}$
Il GBP è
$\text{GBP} = A_{CL} \cdot f_b = 2.5 \pm 0.1\,\text{MHz}$

III.III Amplificatore con A=1

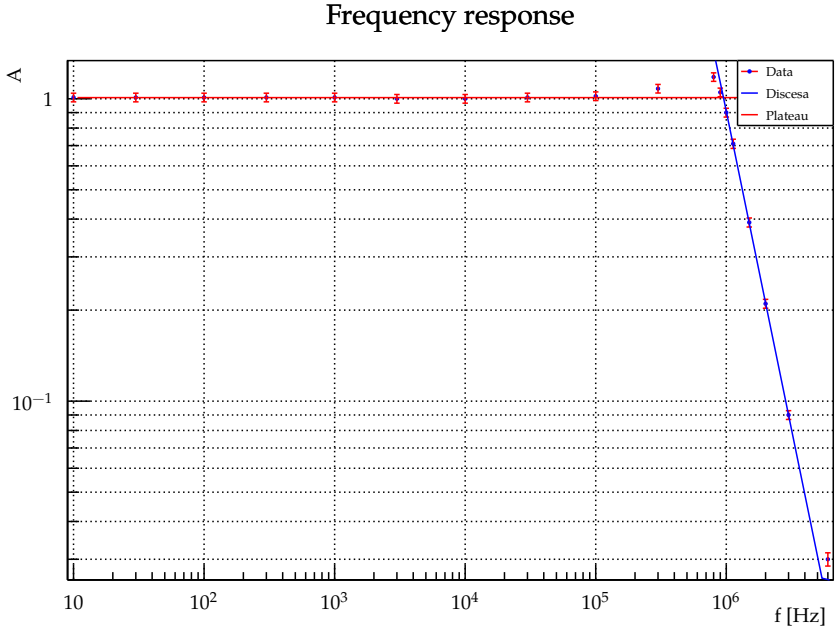
Le resistenze inserite sono state sostituite con:
$R_{1,up} = 32.6 \pm 0.2\,\text{k}\Omega$
$R_{1,down} = 32.6 \pm 0.2\,\text{k}\Omega$
$R_{2,up} = 32.7 \pm 0.2\,\text{k}\Omega$
$R_{2,down} = 32.5 \pm 0.2\,\text{k}\Omega$
$R_4 = 56.0 \pm 0.3\,\Omega$

L'amplificazione teorica è perciò
$A_{teorica} = 1.000 \pm 0.005$ In seguito è presentato il grafico della risposta in frequenza dell'amplificatore non invertente con A=1.

In seguito sono presentati i dati del **Grafico 5** acquisiti in laboratorio: Sono state interpolate separatamente la zona di plateau e di discesa del **Grafico 5**, ottenendo come risultati:
$A_{plateau} = 1.01 \pm 0.01$
$q = 12.6 \pm 0.4$
$m = -2.11 \pm 0.06$

Dall'interpolazione si è poi ricavato $f_b = 1.12 \pm 0.01\,\text{MHz}$
Il GBP è

Grafico 5 Risposta in frequenza di un amplificatore non invertente con A=1



$$GBP = A_{CL} \cdot f_b = 1.13 \pm 0.02\,MHz$$

Grafico 6 Risposta in frequenza di un amplificatore non invertente a varie amplificazioni

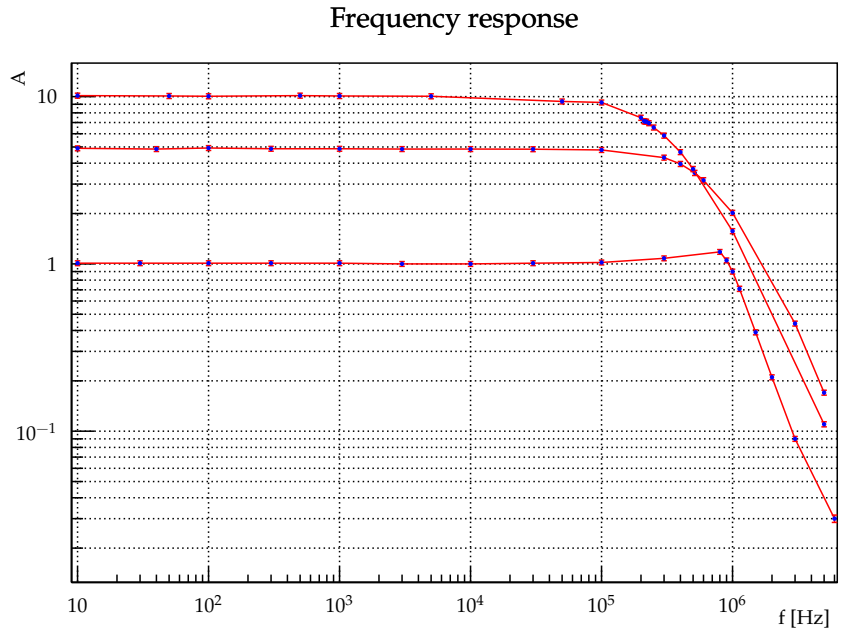


Tabella 3: *Dati risposta in frequenza*

f(Hz)	A
10	1.01 ± 0.03
30	1.01 ± 0.03
100	1.01 ± 0.03
300	1.01 ± 0.03
1000	1.01 ± 0.03
3000	1.00 ± 0.03
10000	1.00 ± 0.03
30000	1.01 ± 0.03
100000	1.02 ± 0.03
300000	1.08 ± 0.04
800000	1.18 ± 0.04
900000	1.05 ± 0.03
1000000	0.90 ± 0.03
1130000	0.71 ± 0.02
1500000	0.39 ± 0.01
2000000	0.210 ± 0.007
3000000	0.090 ± 0.003
6000000	0.030 ± 0.001

Tabella 4: *Dati risposta in frequenza*

f(Hz)	A
10	4.9±0.2
40	4.9±0.2
100	4.9±0.2
300	4.9±0.2
1000	4.9±0.2
3000	4.9±0.2
10000	4.9±0.2
30000	4.9±0.2
100000	4.8±0.2
300000	4.3±0.1
400000	4.0±0.1
515000	3.5±0.1
600000	3.2±0.1
1000000	2.02±0.07
3000000	0.44±0.01
5000000	0.170±0.006

Tabella 5: *Dati risposta in frequenza*

f(Hz)	A
10	10.1±0.3
50	10.1±0.3
100	10.1±0.3
500	10.1±0.3
1000	10.1±0.3
5000	10.1±0.3
50000	9.4±0.3
100000	9.2±0.3
200000	7.5±0.3
211000	7.1±0.2
215000	7.1±0.2
220000	7.1±0.2
230000	6.9±0.2
250000	6.5±0.2
300000	5.9±0.2
400000	4.7±0.2
500000	3.69±0.1
1000000	1.57±0.05
5000000	0.110±0.003

IV. Discussioni e conclusioni

E' stata fatta la simulazione per la risposta in frequenza utilizzando Spice, con l'accortezza di scaricare la libreria del componente dal sito della Texas Instrument. I **Grafico 8** e **Grafico 9** mostrano il confronto della simulazione con i dati sperimentali e mostrano un buon accordo tra i dati e la simulazione. Nel **Grafico 7**, relativo all'ampificazione A=1, si nota una deviazione dei dati sperimentali dovuta alla risonanza, causata dalle induttanze parassite nel circuito reale, che comportano lo spostamento verso le basse frequenze del polo.

Queste differenze si ripercuotono sul Gain-Bandwidth Product. Infatti è più basso di quello previsto nel datasheet fornito dal produttore ($3 \cdot 10^6$), differenza visibile soprattutto con amplificazione A=1. Leggendo il datasheet della Texas Instrument è emerso che una possibile giustificazione a questa discrepanza è che non sono state seguite alcune linee guida indicate. Non sono stati posti, ad esempio, i condensatori vicino all'amplificatore operazionale, accorgimento che avrebbe ridotto l'induttanza del circuito di alimentazione ad alte frequenze. Inoltre l'uso di una breadboard ha introdotto numerose capacità e induttanze parassite, che hanno interferito nella delicata misura di f_b.

Grafico 7 Risposta in frequenza con Spice confrontata coi dati sperimentali (A=1)

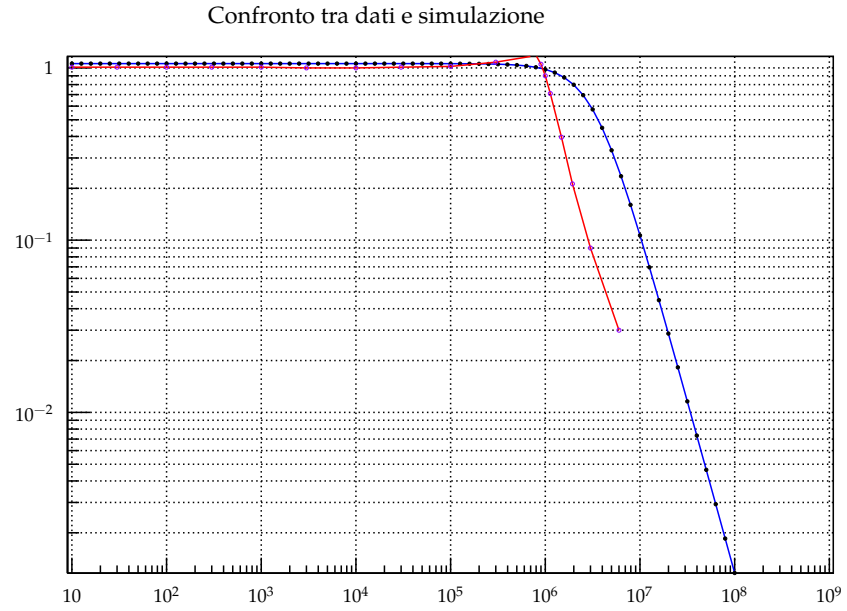


Grafico 8 Risposta in frequenza con Spice confrontata coi dati sperimentali (A=5)

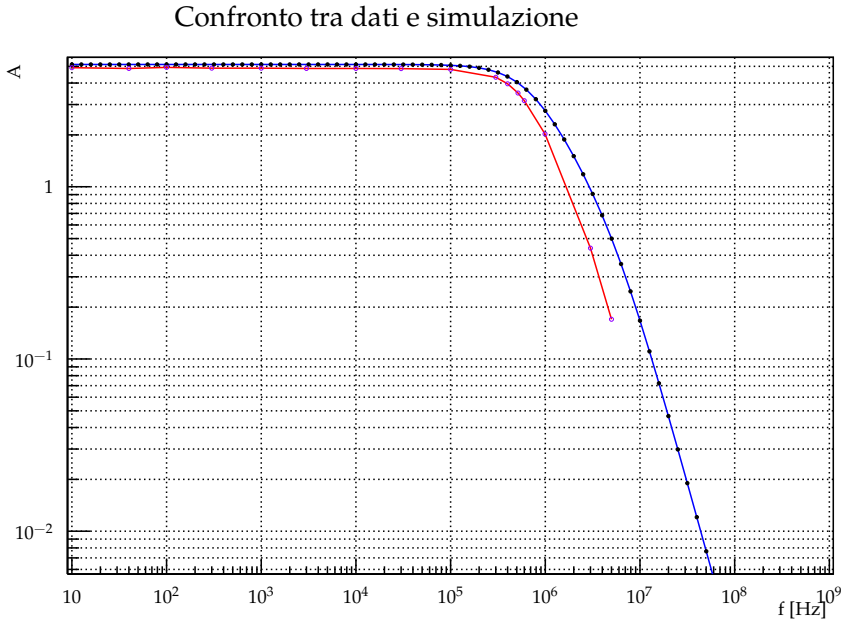
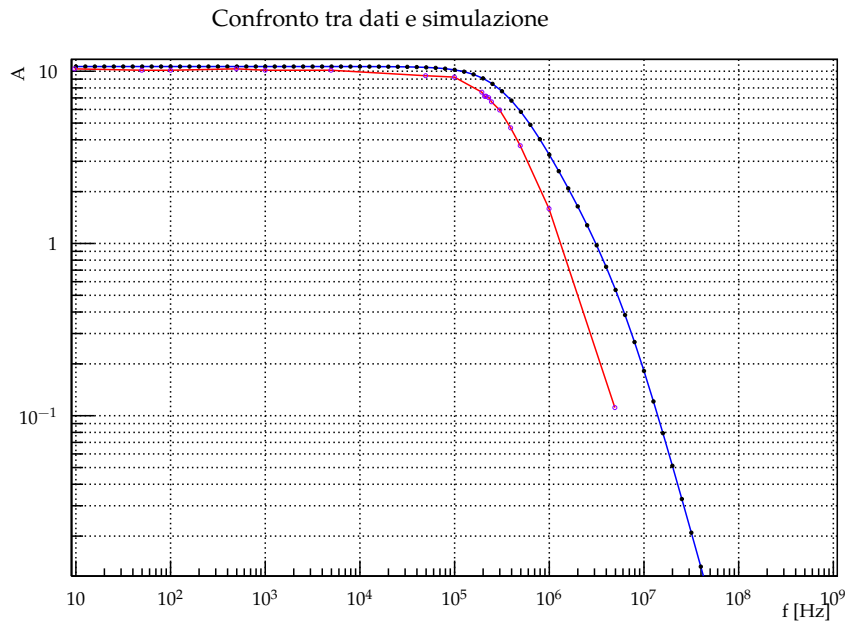


Grafico 9 Risposta in frequenza con Spice confrontata coi dati sperimentali (A=10)



V. Codice

É presentata qua la parte fondamentale del codice in c++ usato per i calcoli numerici. Inoltre è stato usato per i calcoli Mathematica.

V.I Parte 1

```
/*
 * Gain.cpp
 *
 *  Created on: 30/apr/2016
 *      Author: enrico
 */

#include <string>
#include <array>
#include <memory>
#include <iostream>

#include "OpampAnalisys.h"

using namespace std;

int main(int argc, char** argv)
{
  OpampAnalisys::basename = string(argv[1]);
  array<string, 2> type{"inv", "noninv"};

  for (auto i: type)
  {
    cout << i << endl;
    string filename = "/amp_" + i;
    unique_ptr<OpampAnalisys> oA(new OpampAnalisys(filename));
    oA->analisys();
    cout << "amplificatore " + i + " completato\n\n\n";
  }
  return 0;
}
```

../src/opamp_p1/Gain.cpp

```
/*
 * Graph.h
 *
 *  Created on: 01/mag/2016
 *      Author: enrico
 */

#ifndef GRAPH_H_
#define GRAPH_H_

```

```
11 #include <vector>
12
13 class Graph
14 {
15   public:
16     Graph();
17     virtual ~Graph();
18     double* x()
19     {
20       return &(vx[0]);
21     };
22     double* y()
23     {
24       return &(vy[0]);
25     };
26     double* ex()
27     {
28       return &(vex[0]);
29     };
30     double* ey()
31     {
32       return &(vey[0]);
33     };
34
35     unsigned int n()
36     {
37       return num;
38     };
39
40     void add(double x, double y, double ex, double ey);
41   private:
42     std::vector<double> vx;
43     std::vector<double> vy;
44     std::vector<double> vex;
45     std::vector<double> vey;
46
47     unsigned int num;
48 };
49
50 #endif /* GRAPH_H_ */
```

../src/opamp_p1/Graph.h

```
1 /*
2  * Graph.cpp
3  *
4  *  Created on: 01/mag/2016
5  *      Author: enrico
6  */
7
8 #include "Graph.h"
9
```

```
10 Graph::Graph()
11   :num(0)
12 {
13 }
14
15 Graph::~Graph()
16 {
17 }
18
19 void Graph::add(double x, double y, double ex, double ey)
20 {
21   vx.push_back(x);
22   vy.push_back(y);
23   vex.push_back(ex);
24   vey.push_back(ey);
25   ++num;
26 }
```

../src/opamp_p1/Graph.cpp

```
1  /*
2   * OpampAnalisys.h
3   *
4   *  Created on: 01/mag/2016
5   *      Author: enrico
6   */
7
8  #ifndef OPAMPANALISYS_H_
9  #define OPAMPANALISYS_H_
10
11 #include <string>
12 #include <memory>
13
14 class TGraphErrors;
15
16 class OpampAnalisys final
17 {
18   public:
19     OpampAnalisys(std::string filename);
20     virtual ~OpampAnalisys();
21
22     static std::string basename;
23
24     void analisys();
25   private:
26     std::unique_ptr<TGraphErrors> g;
27     std::string filename;
28 };
29
30 #endif /* OPAMPANALISYS_H_ */
```

../src/opamp_p1/OpampAnalisys.h

```
1  /*
2   * OpampAnalisys.cpp
3   *
4   *  Created on: 01/mag/2016
5   *      Author: enrico
6   */
7
8  #include "OpampAnalisys.h"
9  #include "Graph.h"
10
11 #include <TROOT.h>
12 #include <TGraph.h>
13 #include <TGraphErrors.h>
14 #include <TF1.h>
15 #include <TCanvas.h>
16 #include <TAxis.h>
17 #include <TFitResult.h>
18 #include <TFrame.h>
19 #include <TLegend.h>
20
21 #include <iostream>
22
23 using namespace std;
24
25 string OpampAnalisys::basename ="";
26
27 unique_ptr<Graph> readGraph(string);
28
29 OpampAnalisys::OpampAnalisys(string filename)
30   :filename(filename)
31 {
32   string name = basename + filename + ".txt";
33   unique_ptr<Graph> gr = readGraph(name);
34
35   cout << gr->n() << endl;
36   g = unique_ptr<TGraphErrors>(new TGraphErrors(gr->n(), gr->x(),
       gr->y(), gr->ex(), gr->ey()));
37 }
38
39 OpampAnalisys::~OpampAnalisys()
40 {
41   // TODO Auto-generated destructor stub
42 }
43
44 void OpampAnalisys::analisys()
45 {
46   TCanvas c("Interpolazione Opamp");
47   c.SetGrid();
48
49   g->SetFillColor(1);
50   g->SetLineColor(2);
51   g->SetLineWidth(1);
```

```
52   g->SetMarkerColor(4);
53   g->SetMarkerSize(0.7F);
54   g->SetMarkerStyle(1);
55   g->SetTitle("Gain");
56   g->GetXaxis()->SetTitle("V_{out} [V]");
57   g->GetYaxis()->SetTitle("V_{in} [V]");
58   g->Draw("AP");
59
60   TF1* f = new TF1("fit", "[0]+[1]*x");
61   f->SetParName(1, "m");
62   f->SetParName(0, "q");
63   f->SetLineColor(4);
64   f->SetLineWidth(1);
65
66   TFitResultPtr r = g->Fit(f, "S", "", -1.5, 1.5);
67   r->Print("V");
68   for (unsigned int i = 0; i < r->NPar(); ++i)
69   {
70     clog << r->ParName(i)
71       << " " << r->Parameter(i)
72       << " " << r->ParError(i) << endl;
73   }
74
75   TLegend *leg = new TLegend(0.8, 0.8, 0.9, 0.9);
76   leg->AddEntry(g.get(), "Data", "lp");
77   leg->AddEntry(f, "Fit", "l");
78   leg->Draw();
79
80   c.Update();
81   c.GetFrame()->SetFillColor(0);
82   c.GetFrame()->SetBorderSize(12);
83   c.Modified();
84
85   string name = "Result" + filename + ".tex";
86   c.Print(name.c_str());
87 }
```

../src/opamp_p1/OpampAnalisys.cpp

```
1  /*
2   * readGraph.cpp
3   *
4   *  Created on: 01/mag/2016
5   *      Author: enrico
6   */
7  #include <memory>
8  #include <fstream>
9  #include <string>
10 #include <cmath>
11
12 #include "Graph.h"
13
```

```
14 using namespace std;
15
16 double error(double x, double fs)
17 {
18   return sqrt((0.06*fs)*(0.06*fs) +
19       (0.02*x)*(0.02*x));
20 }
21
22 unique_ptr<Graph> readGraph(string filename)
23 {
24   unique_ptr<Graph> g(new Graph());
25   fstream fin(filename.c_str(), iostream::in);
26   double x1, x2, y1, y2,  ex, ey, fsx, fsy;
27   while(fin>>x1>>x2>>fsx>>y1>>y2>>fsy)
28   {
29     ex = error(x1, fsx);
30     ey = error(y1, fsy);
31     g->add(x1, y1, ex, ey);
32     ex = error(x2, fsx);
33     ey = error(y2, fsy);
34     g->add(x2, y2, ex, ey);
35   }
36
37   return g;
38 }
```

../src/opamp_p1/readGraph.cpp

V.II Parte 1

```
1 /*
2  * Gain.cpp
3  *
4  *  Created on: 30/apr/2016
5  *      Author: enrico
6  *
7  *  Scritto per root 6.06.02
8  */
9
10 #include <string>
11 #include <array>
12 #include <memory>
13 #include <iostream>
14
15 #include <TCanvas.h>
16 #include <TAxis.h>
17
18 #include "OpampAnalysis.h"
19 #include "AdHocParameters.h"
20
21 using namespace std;
22
```

```
23 int main(int argc, char** argv)
24 {
25   // initialize folder name, where data is stored
26   OpampAnalysis::basename =
27     string(argv[1]);
28
29   // initialize parameters
30   array<AdHocParameters, 3> parameters{{
31     {"/amp_noninv_A1", 100000, 1000000, 2000000},
32     {"/amp_noninv_A5", 99999, 300001, 999999},
33     {"/amp_noninv_A10", 5001, 100001, 399999}
34   }};
35
36   for (auto i: parameters)
37   {
38     unique_ptr<OpampAnalysis> oA(new OpampAnalysis(i));
39     oA->analysis();
40   }
41
42   string name = "Result/amp_noninv_all.tex";
43
44   // canvas to store the graphs of all amplitudes
45   TCanvas *cAll = new TCanvas("Risposta in frequenza");
46   cAll->SetGrid();
47   cAll->SetLogx();
48   cAll->SetLogy();
49
50   OpampAnalysis::gAll->SetTitle("Frequency response");
51
52   OpampAnalysis::gAll->Draw("A");
53   OpampAnalysis::gAll->GetXaxis()->SetTitle("f [Hz]");
54   OpampAnalysis::gAll->GetYaxis()->SetTitle("A");
55
56   OpampAnalysis::gAll->Draw("A");
57
58   cAll->Print(name.c_str());
59
60   return 0;
61 }
```

../src/opamp_p2/Gain.cpp

```
1 /*
2  * Graph.h
3  *
4  *  Created on: 01/mag/2016
5  *      Author: enrico
6  */
7
8 #ifndef GRAPH_H_
9 #define GRAPH_H_
10
```

```
11 #include <vector>
12
13 class Graph
14 {
15   // class storing vector of points as vector of doubles
16   // and can return pointers to the data, for root's TGraph
17   public:
18     Graph();
19     virtual ~Graph();
20     double* x()
21     {
22       return vx.data();
23     };
24     double* y()
25     {
26       return vy.data();
27     };
28     double* ex()
29     {
30       return vex.data();
31     };
32     double* ey()
33     {
34       return vey.data();
35     };
36
37     unsigned int n()
38     {
39       return num;
40     };
41
42     void add(double x, double y, double ey);
43
44   private:
45     std::vector<double> vx;
46     std::vector<double> vy;
47     std::vector<double> vex;
48     std::vector<double> vey;
49
50     unsigned int num;
51 };
52
53 #endif /* GRAPH_H_ */
```

../src/opamp_p2/Graph.h

```
1 /*
2  * Graph.cpp
3  *
4  *  Created on: 01/mag/2016
5  *      Author: enrico
6  */
```

```
7
8  #include "Graph.h"
9
10 Graph::Graph()
11   :num(0)
12 {
13 }
14
15 Graph::~Graph()
16 {
17 }
18
19 void Graph::add(double x, double y, double ey)
20 {
21   vx.push_back(x);
22   vy.push_back(y);
23   vey.push_back(ey);
24   ++num;
25 }
```

../src/opamp_p2/Graph.cpp

```
1  /*
2   * OpampAnalysis.h
3   *
4   *  Created on: 01/mag/2016
5   *      Author: enrico
6   *
7   *  Scritto per root 6.06.02
8   */
9
10 #ifndef OPAMPAnalysis_H_
11 #define OPAMPAnalysis_H_
12
13 #include <string>
14 #include <memory>
15
16 #include <TMultiGraph.h>
17
18 #include "AdHocParameters.h"
19
20 class TGraphErrors;
21
22 class OpampAnalysis final
23 {
24   // actual analysis class
25   public:
26     OpampAnalysis(const AdHocParameters &p);
27     virtual ~OpampAnalysis();
28
29     // folder name, the program expects the files to be there
30     static std::string basename;
```

```
31
32      // multigraph to store graphs of all amplitudes
33      static TMultiGraph* gAll;
34
35      void analysis();
36    private:
37      std::unique_ptr<TGraphErrors> g;
38      AdHocParameters p;
39  };
40
41  #endif /* OPAMPAnalysis_H_ */
```

../src/opamp_p2/OpampAnalysis.h

```
1  /*
2   * OpampAnalysis.cpp
3   *
4   *  Created on: 01/mag/2016
5   *      Author: enrico
6   *
7   *  Scritto per root 6.06.02
8   */
9
10 #include "OpampAnalysis.h"
11 #include "Graph.h"
12
13 #include <TROOT.h>
14 #include <TGraph.h>
15 #include <TGraphErrors.h>
16 #include <TF1.h>
17 #include <TF2.h>
18 #include <TCanvas.h>
19 #include <TAxis.h>
20 #include <TFitResult.h>
21 #include <TFrame.h>
22 #include <TLegend.h>
23
24 #include <iostream>
25
26 using namespace std;
27
28 std::string OpampAnalysis::basename = "";
29 TMultiGraph* OpampAnalysis::gAll = new TMultiGraph();
30
31 unique_ptr<Graph> readGraph(string);
32
33 OpampAnalysis::OpampAnalysis(const AdHocParameters &p)
34   :p(p)
35 {
36   // read graph from datafile
37   string name = basename + p.filename + ".txt";
38   unique_ptr<Graph> gr = readGraph(name);
```

```
39
40    // initialize TGraphErrors
41    g = unique_ptr<TGraphErrors>(new TGraphErrors(gr->n(), gr->x(),
        gr->y(), gr->ex(), gr->ey()));
42
43    // graph clone, to be stored in the multigraph
44    TGraphErrors* gClone = dynamic_cast<TGraphErrors*>(g->Clone());
45    gClone->SetFillColor(1);
46    gClone->SetLineColor(2);
47    gClone->SetLineWidth(1);
48    gClone->SetMarkerColor(4);
49    gClone->SetMarkerSize(0.7F);
50    gClone->SetMarkerStyle(1);
51    gAll->Add(gClone, "PL");
52  }
53
54  OpampAnalysis::~OpampAnalysis()
55  {
56    // RAII
57  }
58
59  void OpampAnalysis::analysis()
60  {
61    // Create canvas and update settings
62    TCanvas c("Interpolazione Opamp");
63    c.SetGrid();
64    c.SetLogy();
65    c.SetLogx();
66    c.GetFrame()->SetFillColor(0);
67    c.GetFrame()->SetBorderSize(12);
68
69    // Graph settings
70    g->SetFillColor(1);
71    g->SetLineColor(2);
72    g->SetLineWidth(1);
73    g->SetMarkerColor(4);
74    g->SetMarkerSize(0.7F);
75    g->SetMarkerStyle(1);
76    g->SetTitle("Frequency response");
77    g->GetXaxis()->SetTitle("f [Hz]");
78    g->GetYaxis()->SetTitle("A");
79    g->Draw("AP");
80    g->Print();
81
82    // Interpolation function, works on log10(y)
83    TF1* fFall = new TF1("Discesa", "[0] + [1]*log10(x)");
84    fFall->SetParName(0, "q");
85    fFall->SetParName(1, "m");
86    fFall->SetParameter(1, -1.);
87    fFall->SetLineColor(4);
88    fFall->SetLineWidth(1);
89
```

```
90  // Interpolation function, works on log10(y)
91  TF1* fPlat = new TF1("Plateau", "[0]");
92  fPlat->SetParName(0, "A");
93  fPlat->SetParameter(0, 10.);
94  fPlat->SetLineColor(2);
95  fPlat->SetLineWidth(1);
96
97  // Function to change the graph to logarithmic
98  TF2 *t = new TF2("transform", "log10(y)");
99
100 // Clone of the current graph, where t will be applied
101 // this one will be interpolated
102 TGraphErrors* gLog = dynamic_cast<TGraphErrors*>(g->Clone());
103 gLog->Apply(t);
104
105 // Fit the logarithmic graph, on the constant part
106 TFitResultPtr rPlat = gLog->Fit(fPlat, "SM", "", 10, p.endPlat);
107 rPlat->Print("V");
108 for (unsigned int i = 0; i < rPlat->NPar(); ++i)
109 {
110   clog << rPlat->ParName(i)
111     << " " << rPlat->Parameter(i)
112     << " " << rPlat->ParError(i) << endl;
113 }
114 // Actual function that will be drawn
115 TF1* fPlatNorm = new TF1("Plateau",
116     [=](Double_t* x, Double_t *p)
117     {
118       return pow(10, rPlat->Parameter(0));
119     },
120     10, 10000000, 0);
121 fPlatNorm->SetLineColor(2);
122 fPlatNorm->SetLineWidth(1);
123 fPlatNorm->Draw("LSAME");
124
125 // Fit the logarithmic graph, on the "linear" part
126 TFitResultPtr rFall = gLog->Fit(fFall, "S+", "", p.startFall,
      p.stopFall);
127 rFall->Print("V");
128 for (unsigned int i = 0; i < rFall->NPar(); ++i)
129 {
130   clog << rFall->ParName(i)
131     << " " << rFall->Parameter(i)
132     << " " << rFall->ParError(i) << endl;
133 }
134 // Actual function that will be drawn
135 TF1* fFallNorm = new TF1("Discesa",
136     [=](Double_t* x, Double_t *p)
137     {
138       return pow(10, rFall->Parameter(0) +
      rFall->Parameter(1)*log10(x[0]));
139     },
```

```
140         10, 10000000, 0);
141     fFallNorm->SetLineColor(4);
142     fFallNorm->SetLineWidth(1);
143     fFallNorm->Draw("LSAME");
144 
145     // Calculate f_b
146     double A = rPlat->Parameter(0) - log10(2)/2;
147     double eA = rPlat->ParError(0);
148 
149     double m = rFall->Parameter(1);
150     double em = rFall->ParError(1);
151 
152     double q = rFall->Parameter(0);
153     double eq = rFall->ParError(0);
154 
155     double covmq = rFall->CovMatrix(0,1);
156 
157     double fb = pow(10, (A - q)/m);
158 
159     double dA = 1/m;
160     double dq = -dA;
161     double dm = -(A - q)/(m*m);
162     double efb = fb*log(10)*
163            sqrt(dA*dA*eA*eA +
164               dq*dq*eq*eq +
165               dm*dm*em*em +
166               2*dq*dm*covmq);
167 
168     clog << "fb " << fb << " " << efb << endl;
169 
170 
171     // Draw legend
172     TLegend *leg = new TLegend(0.8, 0.8, 0.9, 0.9);
173     leg->AddEntry(g.get(), "Data", "lp");
174     leg->AddEntry(fFallNorm, "Discesa", "l");
175     leg->AddEntry(fPlatNorm, "Plateau", "l");
176     leg->Draw();
177 
178     // Save result as image on disk
179     string name = "Result" + p.filename + ".tex";
180     c.Print(name.c_str());
181 
182     name = "Result" + p.filename + ".svg";
183     c.Print(name.c_str());
184 
185 }
```

../src/opamp_p2/OpampAnalysis.cpp

```
1 /*
2  * readGraph.cpp
3  *
```

```
 *  Created on: 01/mag/2016
 *      Author: enrico
 */
#include <memory>
#include <fstream>
#include <string>
#include <cmath>

#include "Graph.h"

using namespace std;

const double fsin = 0.2;

double error(double A, double fsout)
{
  return sqrt((0.06*fsout)*(0.06*fsout) +
    (A*0.06*fsin)*(A*0.06*fsin) +
      2*(0.02*A)*(0.02*A));
}

unique_ptr<Graph> readGraph(string filename)
{
  unique_ptr<Graph> g(new Graph());
  fstream fin(filename.c_str(), iostream::in);
  double x, y, ey, fsy;
  while(fin>>x>>y>>fsy)
  {
    ey = error(y, fsy);
    g->add(x, y, ey);
  }
  return g;
}
```

../src/opamp_p2/readGraph.cpp

```
/*
 * AdHocParameters.h
 *
 *  Created on: 07/mag/2016
 *      Author: enrico
 */

#ifndef ADHOCPARAMETERS_H_
#define ADHOCPARAMETERS_H_

struct AdHocParameters
{
  // simple structure to store the parameters that change
  // according to the amplitude considered
  public:
    std::string filename;
```

```
17      double endPlat;
18      double startFall;
19      double stopFall;
20  };
21
22  #endif /* ADHOCPARAMETERS_H_ */
```

../src/opamp_p2/AdHocParameters.h