

Régression linéaire de la mesure d'un capteur de température

J.-M Friedt, 2 avril 2026 (+1,5 points/réponse, -2 en copiant bêtement la réponse d'un chatbot)

Parmi les sources de tensions polarisant l'entrée du convertisseur analogique-numérique de l'Atmega32U4 se trouve une sonde de température. Bien que d'exactitude médiocre, la sonde de température permet au moins de déclencher une alarme en cas de surchauffe du système embarqué intégrant un Atmega32U4, ou de détecter une évolution croissante de la température qui pourrait se traduire à terme par un dysfonctionnement du système. Ainsi, AN212 d'Atmel¹ explique que “*The voltage has a linear relationship to temperature and the result has approximately a 1 LSB/°C correlation to temperature. The diode voltage is highly linear, but due to process variations the temperature sensor output voltage varies from one chip to another.*” Ainsi, la valeur lue sur le convertisseur analogique-numérique connecté à la source interne de tension représentative de la température est en première approximation la température en kelvin.

Nous nous proposons donc d'acquérir la température vue par le microcontrôleur, et d'en ajuster la tendance par une régression linéaire (*fit*).

Nous pourrions nous appuyer sur la bibliothèque `libLEEA` fournie à <https://github.com/jmfriedt/13ep> pour initialiser et lire une valeur sur le convertisseur analogique-numérique. Si on fait le choix de ne pas s'appuyer sur `libLEEA`, on prendra garde à l'emplacement de `MUX5` parmi les registres de l'Atmega32U4.

1. Sur quel canal du multiplexeur du convertisseur analogique-numérique se trouve la sonde de température ? combien de mesures faut-il avant d'obtenir une mesure valable ? Justifier l'obtention de ces informations (source et emplacement).
2. Proposer un programme qui lit la température et transmet l'information sur bus USB afin de permettre un affichage de la valeur sur un terminal du PC. Quelle référence de tension a été choisie pour cette mesure ? Pourquoi ?
3. Interpréter la valeur lue comme une température : la valeur semble-t-elle raisonnable ? Évolue-t-elle en plaçant le doigt sur le microcontrôleur ? Argumenter par rapport à AN122.

Maintenant que nous savons lire la température, nous désirons établir la tendance de son évolution avec le temps en vue de prédire sa valeur dans le futur selon une régression linéaire. Une bibliothèque classique de traitement numérique du signal est fournie par *Numerical Recipes : The Art of Scientific Computing*² déclinée en nombre de langages, et en particulier en C. Une copie numérique de l'ouvrage se trouve par exemple à³. En page 665 de l'ouvrage (689 du PDF) se trouve une fonction de régression linéaire (*fit*) prenant en entrée deux tableaux, abscisses et ordonnées, et éventuellement un tableau contenant l'incertitude sur ces mesures, et qui renvoie les coefficients de pente et d'ordonnée à l'origine de la régression linéaire.

La bibliothèque se trouve par exemple à <https://github.com/tranqv/Numerical-Recipes-in-C/> qui se compile selon un mode peu commun d'exécution d'un script shell tel que décrit dans `README.md` avec `bash sh.make lib`.

4. Compiler la bibliothèque de *Numerical Recipes* sur PC : comment s'appelle la bibliothèque statique résultante, et où se trouve-t-elle ? Comment indiquer à `gcc` de lier cette bibliothèque pour générer un exécutable pour en exploiter les fonctions (options d'emplacement des entêtes et d'édition de liens avec la bibliothèque) ?
5. Comment se nomme la fonction de *fit* dans l'implémentation de la bibliothèque ? Où, et comment, cette information a été trouvée ?
6. Proposer un exemple d'utilisation, sur PC, de la fonction de régression linéaire prenant en entrée 4 valeurs d'abscisses (1.0, 2.0, 3.0, 4.0) et 4 valeurs d'ordonnées (300.5, 301.5, 302.5, 303.5) avec une incertitude (écart type) de 0.1 sur chaque valeur. Le résultat est-il en accord avec les attentes ? Justifier. Quelle option de la fonction permet d'exploiter l'information d'incertitude fournie, au lieu de choisir une valeur arbitraire par défaut ?

1. https://ww1.microchip.com/downloads/en/AppNotes/Atmel-8108-Calibration-of-the-AVRs-Internal-Temperature-Reference_ApplicationNote_AVR122.pdf

2. <https://www.amazon.com/Numerical-Recipes-Scientific-Computing-Second/dp/0521431085>

3. https://www.cec.uchile.cl/cinetica/pcordero/MC_libros/NumericalRecipesinC.pdf

Une fois le bon fonctionnement du code démontré sur PC, nous désirons l'exploiter sur microcontrôleur Atmega32U4. Pour ce faire :

7. **Modifier le script `sh.make` afin de compiler une version de la bibliothèque exécutable sur microcontrôleur. Pour ce faire, on pensera à modifier le nom du compilateur (`gcc`) et de l'archiveur (`ar`) pour cibler le microcontrôleur AVR, et inclure le type de microcontrôleur ciblé (`-mmcu=atmega32u4`). Démontrer la compilation croisée de la bibliothèque de *Numerical Recipes* afin de cibler l'Atmega32U4.**
8. **Est-ce que l'option d'optimisation proposée par le script de compilation est appropriée pour un microcontrôleur aux ressources de mémoire réduites ? Modifier cette option d'optimisation pour mieux répondre aux exigences d'une faible empreinte mémoire, et recompiler la bibliothèque.**

Nous n'aurons aucune confiance dans les fonctions de gestion des affichages fournies par `avr-libc` qui gèrent très mal les nombres à virgule (`sprintf("%f", ...)` par exemple).

9. **Fournir une fonction qui, recevant un nombre à virgule flottante (`float`) et un pointeur sur une chaîne de caractères (`char*`), remplit le tableau de caractères avec les symboles ASCII représentant en décimale le nombre à virgule flottante, pour la partie entière supposée inférieure à 10^3 , et les trois premières décimales. Un symbole "." devra séparer la partie entière de la partie fractionnaire. Démontrer le bon fonctionnement de cette fonction avec l'affichage du contenu d'une variable de type `float` initialisée avec 123.456 depuis le microcontrôleur au travers du bus USB.**
10. **Recompiler l'exemple de régression linéaire validé auparavant sur PC pour qu'il s'exécute sur microcontrôleur, en remplaçant la fonction d'affichage par une communication via le bus USB exploitant la fonction que nous venons de rédiger. Démontrer le bon fonctionnement de la bibliothèque sur microcontrôleur. Les résultats sont-ils cohérents avec ceux obtenus sur PC ?**
11. **Quelle est la taille du fichier binaire effectivement transféré au microcontrôleur ? Comparer avec la taille de la mémoire non-volatile (flash) de l'Atmega32U4. Est-ce que le programme pourrait s'exécuter sur Atmega16U4 ? Justifier.**
12. **Quel mécanisme est intégré au microcontrôleur pour le redémarrer en cas de corruption de la mémoire qui induirait l'arrêt de l'exécution séquentielle des instructions du programme ?**

Maintenant que nous sommes convaincus de savoir acquérir des températures d'une part, et de savoir effectuer une régression linéaire d'autre part, nous fusionnons les deux codes.

13. **Proposer un programme qui lit la température du microcontrôleur, mémorise 4 valeurs et fournit la tendance (ordonnée à l'origine et pente) de ces mesures.**
14. **Modifier le programme pour que le calcul soit glissant, i.e. que chaque nouvelle acquisition s'ajoute aux précédentes et n'élimine que la plus ancienne, avant d'effectuer le calcul de régression linéaire.**

1. 0b100111=39. La documentation technique précise *The propagation delay of this driver is approximately 2 μ s. Therefore two successive conversions are required. The correct temperature measurement will be the second one.*
2. La documentation technique précise *The internal 2.56V voltage reference must also be selected for the ADC voltage reference source in the temperature sensor measurement.* Le programme suivant permet d'acquérir et afficher la température :

```

1 #define F_CPU 16000000
2 #define F_USB 16000000
3 #include "VirtualSerial.h"
4 #include <util/delay.h>
5 #include <stdio.h>
6 #include "adc.h"
7
8 extern USB_ClassInfo_CDC_Device_t VirtualSerial_CDC_Interface;
9 extern FILE USBSerialStream;
10
11 int main(void)
12 {char ReportString[20] = "World\r\n\0";
13  adc_init(3);
14
15  SetupHardware();
16  CDC_Device_CreateStream(&VirtualSerial_CDC_Interface, &USBSerialStream);
17  GlobalInterruptEnable();
18
19  for (;;)
20  {sprintf(ReportString, "%d\r\n", adc_read(39));
21   fputs(ReportString, &USBSerialStream);
22   CDC_Device_ReceiveByte(&VirtualSerial_CDC_Interface);
23   CDC_Device_USBTask(&VirtualSerial_CDC_Interface);
24   USB_USBTask();
25   _delay_ms(500);
26  }
27 }

```

3. la valeur lue est $1^\circ/\text{LSB}$ puisque 1.1.2 d'AN122 indique " *T_{ADC} is the output from the ADC converted to $^\circ\text{C}$ by subtracting 273 from the ADC value.*" donc la valeur lue en bits est interprétée comme une température en kelvin. L'affichage indique 300 K donc 27°C et placer le doigt sur le microcontrôleur élève la température autour de 303 K.
4. en consultant la dernière ligne de la compilation
`ar rcs $HOME/apps/shared/lib/libNRinC.a ...`
nous constatons que la bibliothèque statique `libNRinC.a` est placée dans `$HOME/apps/shared/lib/` donc `-I$HOME/apps/shared/include` pour les entêtes, et `-L$HOME/apps/shared/lib/ -lNRinC` pour lier la bibliothèque aux objets lors de la compilation d'un exécutable qui en exploite les fonctions.
5. `grep -r fit $HOME/apps/shared/include/` indique qu'il existe 6 fonctions contenant le terme `fit` préfixées de `nrc_` et `nrc_fit()` correspond au prototype de la fonction fournie dans l'ouvrage.
6. le programme

```

1 #include "nrc_complex.h"
2 #include "nrc.h"
3 #include "nrc_types.h"
4 #include "nrc_util.h"
5
6 #define N 4
7 int main()
8 {double x[N]={1.,2.,3.,4.},y[N]={300.5,301.5,302.5,303.5},sig[N]={0.1,0.1,0.1,0.1};
9  double a,b,siga,sigb,chi2,q;
10 nrc_fit(x,y, N, sig, 1, &a, &b, &siga, &sigb, &chi2, &q);

```

```

11 printf("%lf_%lf\n",a,b);
12 }

```

se contente de charger tous les entêtes fournis dans `include` du répertoire produit par la bibliothèque, et `gcc -o ex pc.c -I $HOME/apps/shared/include/ -L $HOME/apps/shared/lib/ -lnRinC -lm` compile un programme `ex` dont l'exécution se traduit par 299.499840 1.000050 proche des 299.5 (ordonnée à l'origine) et 1.0 (pente) attendus.

7. Nous modifions `COMP="gcc"` par `COMP="avr-gcc"` et `ar $arFLAG ${mylib} $List` par `avr-ar $arFLAG ${mylib} $List` ainsi que `DEPS="-ansi"` par `DEPS="-ansi -mmcu=atmega32u4"`.
8. l'option `-O3` utilisée par défaut optimise autant que possible le code, mais au détriment de l'occupation mémoire. L'option `-Os` est plus appropriée pour un microcontrôleur contraint par ses ressources mémoires.

On notera que cette conclusion était loin d'être triviale, et ce n'est que grâce à l'exécution du programme optimisé en `-O3` dans `simavr` fournissant un serveur GDB que nous avons constaté le problème. En effet, l'exécution du programme compilé avec l'option `-g3` peut être tracée par `avr-gdb` pour constater l'échec de la fonction `nrc_gcf()` qui indique `nrc_error (error_text=0x80015d "a too large, ITMAX too small in nrc_gcf")` at `nrcutil.c:22` Ce n'est qu'en compilant par `-Os` que nous évitons cette erreur et aboutissons à un résultat correct.

9. la fonction suivante permet d'afficher un nombre à virgule flottante avec trois décimales :

```

1 void print_float(char *c,float f)
2 {float div=1000.;
3  int k=0;
4  do
5    {c[k]=(int)(f/div)+'0';
6     f-=(int)(f/div)*div;
7     div/=10.;
8     k++;
9     if ((div<1.) && (div>=0.1)) {c[k]='.';k++;}
10  }
11  while (div>.0005);
12 }

```

10. le programme

```

1 #include "nrc_complex.h"
2 #include "nrc.h"
3 #include "nrc_types.h"
4 #include "nrc_util.h"
5
6 #define N 4
7
8 #define F_CPU 16000000
9 #define F_USB 16000000
10
11 #include <stdio.h>
12 #include "VirtualSerial.h"
13 #include <util/delay.h>
14
15 extern USB_ClassInfo_CDC_Device_t VirtualSerial_CDC_Interface;
16 extern FILE USBSerialStream;
17
18 int main(void)
19 {char ReportString[20] = "World\r\n\0";
20
21  SetupHardware();
22  CDC_Device_CreateStream(&VirtualSerial_CDC_Interface, &USBSerialStream);
23  GlobalInterruptEnable();
24
25  double x[N]={1.,2.,3.,4.},y[N]={300.5,301.5,302.5,303.5},sig[N]={0.1,0.1,0.1,0.1};

```

```

26 double a,b,siga,sigb,chi2,q;
27 nrc_fit(x,y, N, sig, 1, &a, &b, &siga, &sigb, &chi2, &q);
28 for (;;)
29     {sprintf(ReportString, "%ld,%ld\r\n", (long long)(a*1000.), (long long)(b*1000.));
30     fputs(ReportString, &USBSerialStream);
31     CDC_Device_ReceiveByte(&VirtualSerial_CDC_Interface);
32     CDC_Device_USBTask(&VirtualSerial_CDC_Interface);
33     USB_USBTask();
34     _delay_ms(500);
35     }
36 }

```

qui se compile par `avr-gcc -Os -mmcu=atmega32u4 -o ex avr.c -I $HOME/apps/shared/include/ -L $HOME/apps/shared/lib/ -I l3ep/LUFA_light/VirtualSerial_example/ -I l3ep/LUFA_light/lufa-LUFA -L l3ep/LUFA_light/VirtualSerial_lib/ -lnRinC -lm -lVirtualSerial` fournit le résultat 299500 1002 donc 299.5 pour l'ordonnée à l'origine et 1.002 pour la pente. On prendra soin que l'int sur AVR est codé sur 16 bits uniquement et induit un dépassement de capacité pour afficher la valeur 300000 : il faut donc penser à afficher un long long par %ld.

11. alors que l'exécutable `ex` au format ELF occupe 51480 octets, le binaire lui-même obtenu par `avr-objcopy -Obinary ex ex.bin` indique une taille de 16620 octets. Ce programme tient dans les 32768 octets de la mémoire flash de l'Atmega32U4, mais pas dans les 16 KB de l'Atmega16U4.
12. le chien de garde (*watchdog*) redémarre le microcontrôleur s'il n'a pas été réinitialisé avant un intervalle de temps prédéfini.
13. le programme suivant fusionne les deux programmes précédents de mesure de température et de régression linéaire :

```

1 #define F_CPU 16000000
2 #define F_USB 16000000
3 #include "VirtualSerial.h"
4 #include <util/delay.h>
5 #include <stdio.h>
6 #include "adc.h"
7 #include "nrc_complex.h"
8 #include "nrc.h"
9 #include "nrc_types.h"
10 #include "nrc_util.h"
11
12 #define N 4
13
14 #define F_CPU 16000000
15 #define F_USB 16000000
16
17 extern USB_ClassInfo_CDC_Device_t VirtualSerial_CDC_Interface;
18 extern FILE USBSerialStream;
19
20 int main(void)
21 {char ReportString[20] = "World\r\n\0";
22  int k,tmp;
23  double x[N]={1.,2.,3.,4.},y[N]={3.,4.,5.,6.},sig[N]={0.1,0.1,0.1,0.1};
24  double a,b,siga,sigb,chi2,q;
25
26  adc_init(3);
27  SetupHardware();
28  CDC_Device_CreateStream(&VirtualSerial_CDC_Interface, &USBSerialStream);
29  GlobalInterruptEnable();
30
31  for (;;)
32      {for (k=0;k<N;k++)
33          {tmp=adc_read(39);
34           y[k]=(double)tmp;

```

```

35     sprintf(ReportString,"%d_", tmp);
36     fputs(ReportString, &USBSerialStream);
37     }
38     nrc_fit(x,y, N, sig, 1, &a, &b, &sig_a, &sig_b, &chi2, &q);
39     sprintf(ReportString,"->%d_%d\r\n", a,b);
40     fputs(ReportString, &USBSerialStream);
41     CDC_Device_ReceiveByte(&VirtualSerial_CDC_Interface);
42     CDC_Device_USBTask(&VirtualSerial_CDC_Interface);
43     USB_USBTask();
44     _delay_ms(500);
45     }
46 }

```

14. nous remplaçons la boucle d'acquisition par

```

1 ...
2 {for (k=0;k<N-1;k++) y[k]=y[k+1];
3   tmp=adc_read(39);
4   y[N-1]=(double)tmp;
5   sprintf(ReportString,"%d_", tmp);
6   fputs(ReportString, &USBSerialStream);
7   nrc_fit(x,y, N, sig, 1, &a, &b, &sig_a, &sig_b, &chi2, &q);
8 ...

```

avec le reste du programme identique au cas précédent.