Gnu Scientific Library

Pascal Viot

September 6, 2016

Avant-propos

- La Gnu Scientifique Library est une bibliothèque écrite en C et offrant de nombreuses procedures pour le calcul numérique

Avant-propos

- La Gnu Scientifique Library est une bibliothèque écrite en C et offrant de nombreuses procedures pour le calcul numérique
- La version stable actuelle est la 2.2.1 (date Aout 2016)

Avant-propos

- La Gnu Scientifique Library est une bibliothèque écrite en C et offrant de nombreuses procedures pour le calcul numérique
- La version stable actuelle est la 2.2.1 (date Aout 2016)
- Cette bibliothèque comprend de nombreuses fonctions spéciales, générateurs de nombres aléatoires, recherche de zéros de polynômes de fonctions, résolution d'équations différentielles,

Avant-propos

- La Gnu Scientifique Library est une bibliothèque écrite en C et offrant de nombreuses procedures pour le calcul numérique
- La version stable actuelle est la 2.2.1 (date Aout 2016)
- Cette bibliothèque comprend de nombreuses fonctions spéciales, générateurs de nombres aléatoires, recherche de zéros de polynômes de fonctions, résolution d'équations différentielles,
- mais aussi transformées de Fourier, méthodes d'intégration numérique, problèmes d'algèbre linéaire, analyse spectrale, minimisation, méthodes d'ajustements,...

Avant-propos

- La Gnu Scientifique Library est une bibliothèque écrite en C et offrant de nombreuses procedures pour le calcul numérique
- La version stable actuelle est la 2.2.1 (date Aout 2016)
- Cette bibliothèque comprend de nombreuses fonctions spéciales, générateurs de nombres aléatoires, recherche de zéros de polynômes de fonctions, résolution d'équations différentielles,
- mais aussi transformées de Fourier, méthodes d'intégration numérique, problèmes d'algèbre linéaire, analyse spectrale, minimisation, méthodes d'ajustements,...
- En quelques transparents, on va illustrer avec quelques exemples son utilisation.

Plan

- Racines de polynômes

Plan

- Racines de polynômes
- Fonctions spéciales

Plan

- Racines de polynômes
- Fonctions spéciales
- Intégrales

Plan

- Racines de polynômes
- Fonctions spéciales
- Intégrales
- Equations différentielles

Plan

- Racines de polynômes
- Fonctions spéciales
- Intégrales
- Equations différentielles
- Analyse spectrale

Plan

- Racines de polynômes
- Fonctions spéciales
- Intégrales
- Equations différentielles
- Analyse spectrale
- Générateurs de nombres aléatoires

Plan

- Racines de polynômes
- Fonctions spéciales
- Intégrales
- Equations différentielles
- Analyse spectrale
- Générateurs de nombres aléatoires
- Distribution de nombres aléatoires

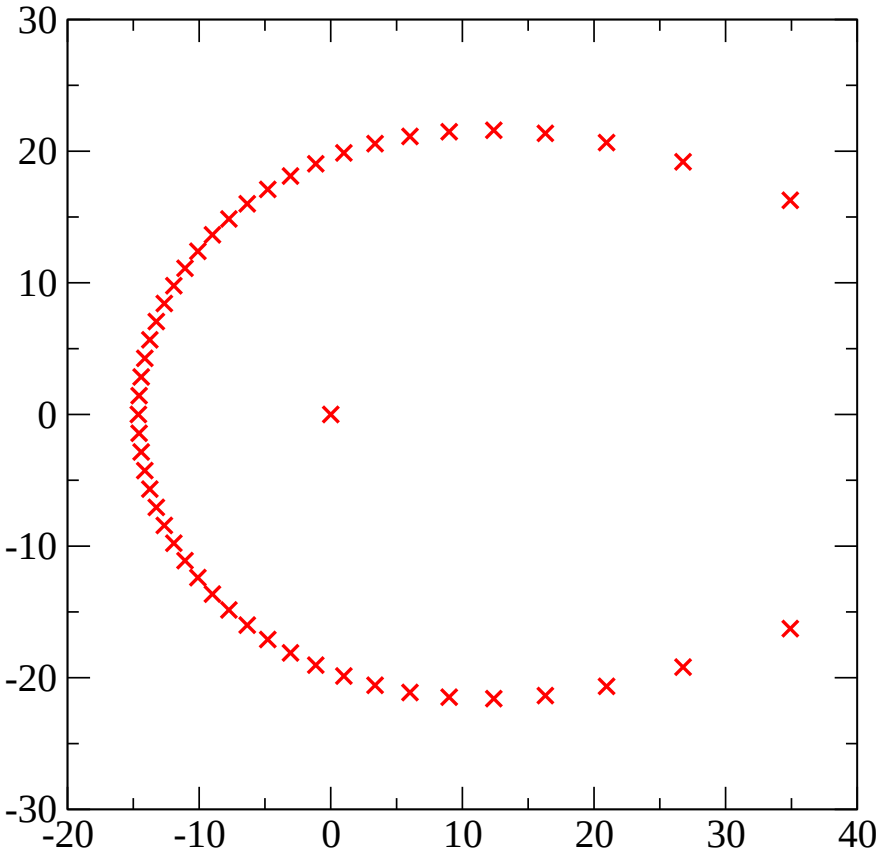
Plan

- Racines de polynômes
- Fonctions spéciales
- Intégrales
- Equations différentielles
- Analyse spectrale
- Générateurs de nombres aléatoires
- Distribution de nombres aléatoires
- Conclusion et références.

Racines de polynômes

```
#include <stdio.h>
#include <math.h>
#include <gsl/gsl_sf.h>
#include <gsl/gsl_poly.h>
main (void)
{
  int i;
  double a[50];
  double z[2*50];
 for(i=0;i<50;i++)
   a[i]=exp(-gsl_sf_lnfact(i));
  gsl_poly_complex_workspace * w
      = gsl_poly_complex_workspace_alloc (50);
  gsl_poly_complex_solve (a, 50, w, z);
  gsl_poly_complex_workspace_free (w);
  for (i = 0; i < 50; i++)
    {
      printf (" %+.18f %+.18f\n",i, z[2*i], z[2*i+1]);
    }
  return 0;
}
```

Racines de polynômes(2)

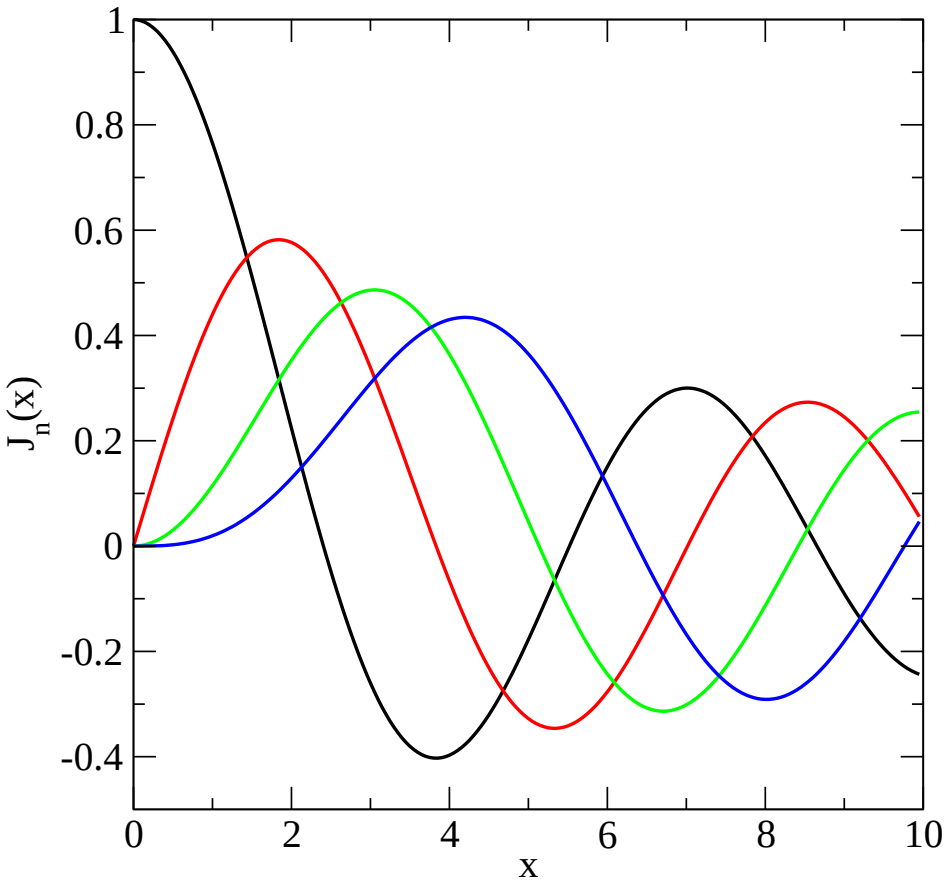


Fonctions spéciales

Beaucoup de fonctions spéciales disponibles avec la GSL
Fonctions de Bessel, Fonctions exponentielle intégrale, fonction Ψ,...

```
#include <stdio.h>
#include <math.h>
#include <gsl/gsl_sf.h>
int main (void)
{
  int i;double x;double y[4];
  for(i=0;i<200;i++)
        { x=i/20.0;
          int j;for(j=0;j<4;j++) y[j]=gsl_sf_bessel_Jn (j,x);
        printf("%e %e %e %e %e\n",x,y[0],y[1],y[2],y[3]);
        }
  return 0;
}
```

Fonctions spéciales(2)



Integration

Calcul de la densité du parking

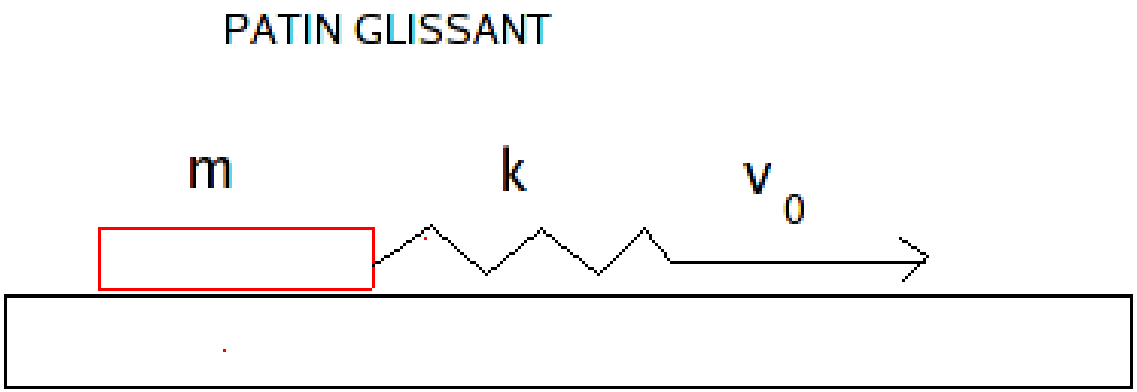
```
#include <stdio.h>
#include <math.h>
#include <gsl/gsl_integration.h>
#include <gsl/gsl_sf_expint.h>
double f (double x, void * params) {
  return exp(2.0*(-gsl_sf_expint_E1(x)-M_EULER))/(x*x);}
int main (void)
{
  gsl_integration_workspace * w
    = gsl_integration_workspace_alloc(2000);
  int i;double x;
  double result, error;
  double expected = .7475979203;
  double alpha = 1.0;

  gsl_function F;
  F.function = &f;
  F.params = &alpha;
  gsl_integration_qagiu (&F, 0,  1e-10, 9e-7, 2000,
                         w, &result, &error);
  printf("result          = % .18f\n", result);
  printf("exact result    = % .18f\n", expected);
  printf("estimated error = % .18f\n", error);
  printf("actual error    = % .18f\n", result - expected);
  printf("intervals =  %d\n", w->size);
  return 0;
}
```

Intégration(2)

result = 0.747597920165969376
exact result = 0.747597920300000052
estimated error = 0.000000619887179837
actual error = -0.000000000134030675

Equations différentielles



Equation du mouvement

$$mx'' = -k(x - v_0 t) - \alpha x' - \beta e^{-x'/v_f} \tag{1}$$

où x est la position du patin.
On introduit les paramètres suivants.

$$\omega_0^2 = \frac{k}{m} = 1, \tau = \frac{m}{\alpha}, \gamma = \frac{\beta}{m} \tag{2}$$

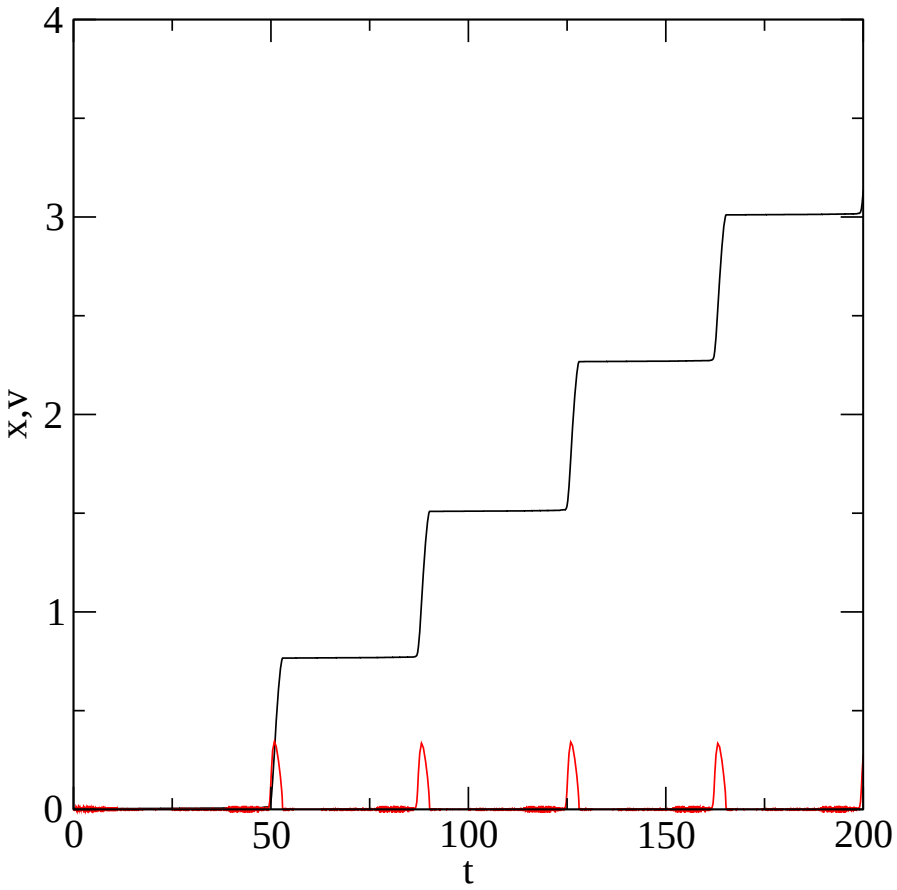
Equations différentielles (2)

```
#include <stdio.h>
#include<math.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_odeiv.h>
struct param_ODE { double tau;double gamma; double vf;double v0;};
int func (double t, const double y[], double f[],
           void *params)
{
  struct param_ODE *para=(struct param_ODE *)params;
  double tt,tau=para->tau;
  double gamma=para->gamma;
  double vf=para->vf,v0=para->v0;
  f[0] = y[1];
  if (y[1]>0) {tt=1.0;}else{tt=-1;}
  if (y[1]==0) {tt=0.0;}
  f[1] = -y[0]+v0*t-y[1]/tau-gamma*tt*exp(-y[1]/vf);
  return GSL_SUCCESS;
     }
...
```

Equations différentielles (3)

```
...
int   main (void)
{ const gsl_odeiv_step_type * T  = gsl_odeiv_step_rk4;
 /* on choisit la methode */
  gsl_odeiv_step * s   = gsl_odeiv_step_alloc (T, 2);
  gsl_odeiv_control * c  = gsl_odeiv_control_y_new (1e-4, 0.0);
  gsl_odeiv_evolve * e  = gsl_odeiv_evolve_alloc (2);
   int (* jac) (double t, const double y[], double * dfdy,
       double dfdt[], void * params)=NULL;
 /* initialisation des parametres */
  struct param_ODE para={0.5,1.0,0.1,0.02};
  /*initialisation des paramètres */
  gsl_odeiv_system sys = { func,jac,2, &para };
  double t = 0.0, t1 = 200.0; /* valeur initiale et finale du temps */
  double h = 1e-4; /* pas d'intégration initial, par la suite le solveur choisit so
  double y[2] = { 0.0, 0.0, };
  FILE *fd; fd= fopen("patin.res","w");
  while (t < t1){
    int status = gsl_odeiv_evolve_apply (e, c, s, &sys, &t, t1,&h, y);
    if (status != GSL_SUCCESS)  break;
    fprintf (fd,"%.5e %.5e %.5e\n", t, y 0], y[1]); }
  gsl_odeiv_evolve_free (e);
  gsl_odeiv_control_free (c);
  gsl_odeiv_step_free (s);
  fclose(fd);
  return 0;
}
```

Equations différentielles (4)



Analyse spectrale

On cherche à calculer les valeurs propres d'une matrice réelle symétrique

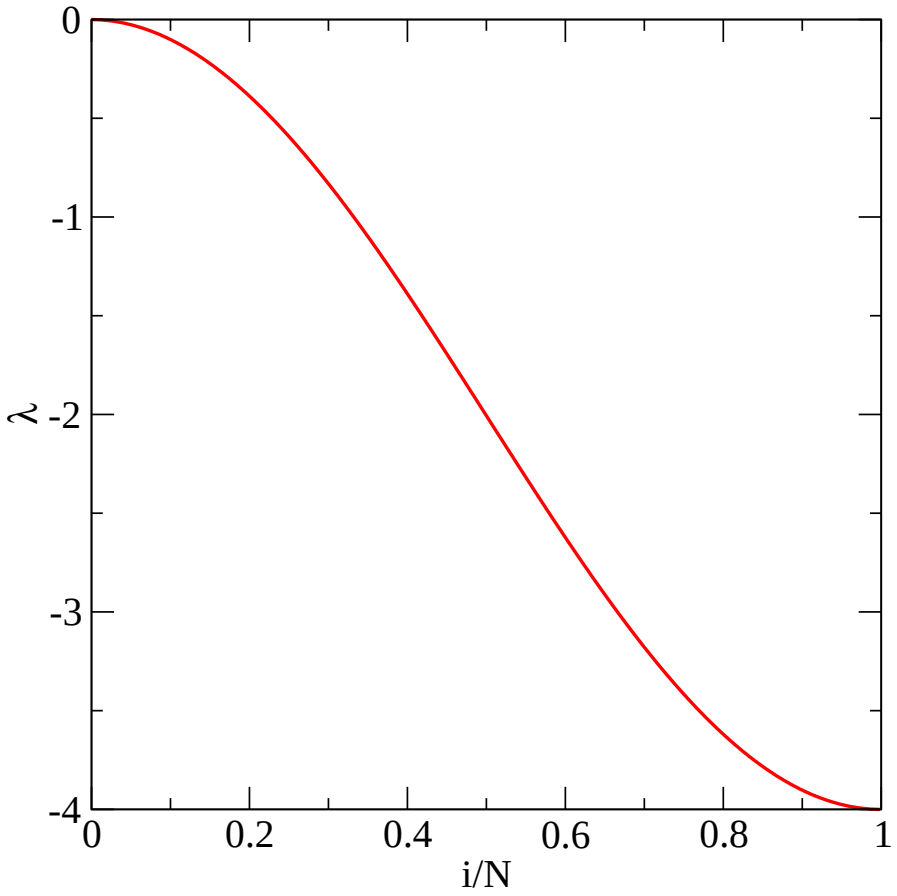
$$\begin{pmatrix} -2 & 1, & 0 & \dots & 0 \\ 1 & -2 & 1 & 0 & \dots \\ 0 & 1 & -2 & 1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & -1 & 2t \end{pmatrix} \tag{3}$$

La dimensions linéaire de la matrice est de 500 On cherche toutes les valeurs propres et on veut les classer de manière croissante.
Cette matrice correspond à une diffusion sur réseau à une dimension avec conditions aux bords adsorbants. On peut vérifier que toutes les valeurs propres sont strictement négatives.

Analyse spectrale (2)

```
#include <stdio.h>
 #include <stdlib.h>
 #include <gsl/gsl_math.h>
 #include <gsl/gsl_eigen.h>
 const int ndim=500;
 int main (void)
 { int i;
  double *data;
  data= (double*) malloc(ndim*ndim*sizeof(double*));
  for(i=0;i<ndim;i++)  data[i*ndim+i]=-2.0;
  for(i=1;i<ndim;i++)  {data[i*ndim+i-1]=1.0;data[(i-1)*ndim+i]=1.0;}
   gsl_matrix_view m = gsl_matrix_view_array (data, ndim, ndim);
   gsl_vector *eval = gsl_vector_alloc (ndim);
   gsl_matrix *evec = gsl_matrix_alloc (ndim, ndim);
   gsl_eigen_symmv_workspace * w = gsl_eigen_symmv_alloc (ndim);
   gsl_eigen_symmv (&m.matrix, eval, evec, w);
   gsl_eigen_symmv_free (w);
   gsl_eigen_symmv_sort (eval, evec, GSL_EIGEN_SORT_ABS_ASC);
   { int i; for (i = 0; i < ndim; i++)
       { double eval_i = gsl_vector_get (eval, i);
         gsl_vector_view evec_i = gsl_matrix_column (evec, i);
         printf (" %d %g\n",i,eval_i);
       }
   }
   gsl_vector_free (eval);
   gsl_matrix_free (evec);
   return 0;
 }
```

Analyse spectrale (3)



Générateur de nombres aléatoires

On fixe le générateur comme le Mersenne et on initialise deux fois deux sequences différentes

```
 #include <stdio.h>
#include <gsl/gsl_rng.h>
   gsl_rng * r;  /* global generator */
   int main (void)
   { int i;
     const gsl_rng_type * T=gsl_rng_mt19937 ;
     r = gsl_rng_alloc (T);
     printf ("generator type: %s\n", gsl_rng_name (r));
     gsl_rng_set(r,3); // première graine
     for(i=0;i<10;i++) printf (" %e\n", gsl_rng_uniform (r));
      printf ("\n");
     gsl_rng_set(r,1); //seconde graine
     for(i=0;i<10;i++) printf (" %e\n", gsl_rng_uniform (r));
     gsl_rng_free (r);
     return 0;
   }
```

Générateur de nombres aléatoires

5.507979e-01	4.170220e-01
7.072488e-02	9.971848e-01
7.081478e-01	7.203245e-01
8.399490e-01	9.325574e-01
2.909047e-01	1.143811e-04
1.213286e-01	1.281244e-01
5.108276e-01	3.023326e-01
5.693113e-01	9.990405e-01
8.929470e-01	1.467559e-01
4.370619e-01	2.360890e-01

Sauver et lire une graine

Changer la graine pour recommencer une simulation ou relire une graine si l'on veut poursuivre une simulation

Sauver une graine

```
#include <stdio.h>
#include <gsl/gsl_rng.h>
#include <sys/time.h>
#include <time.h>
     int main (void)
     { int i;
       gsl_rng * r;  /* global generator */
       const gsl_rng_type * T=gsl_rng_mt19937 ;
       r = gsl_rng_alloc (T);
       gsl_rng_set(r,time(NULL));
       printf("%ld \n",time(NULL));
       printf ("generator type: %s\n", gsl_rng_name (r));
       for(i=0;i<10;i++) printf (" %e\n", gsl_rng_uniform (r));
        printf ("\n");

       FILE * fd;
       fd=fopen("sauve_rng","w");
       gsl_rng_fwrite(fd,r);
       fclose(fd);
       for(i=0;i<10;i++) printf (" %e\n", gsl_rng_uniform (r));
        printf ("\n");
       gsl_rng_free (r);
       return 0;
     }
```

Sauver et lire une graine (2)

Lire une graine

```
#include <stdio.h>
#include <gsl/gsl_rng.h>
#include <sys/time.h>
#include <time.h>

     int main (void)
     { int i;
       gsl_rng * r;  /* global generator */
       const gsl_rng_type * T=gsl_rng_mt19937 ;
       r = gsl_rng_alloc (T);
/* on relit le fichier qui a sauvegarde l etat du generateur */
       FILE * fd;
       fd=fopen("sauve_rng","r");
       gsl_rng_fread(fd,r);
       fclose(fd);
       printf ("generator type: %s\n", gsl_rng_name (r));
       for(i=0;i<10;i++) printf (" %e\n", gsl_rng_uniform (r));
        printf ("\n");
       gsl_rng_free (r);
       return 0;
     }
```

Générateur de nombres aléatoires

```
1360847968
generator type: mt19937
1.478347e-01
8.033008e-01
4.449535e-01
5.775761e-01
3.470892e-01
1.647841e-01
9.416657e-01
6.827526e-02
8.547487e-01
5.650302e-01
5.885839e-01
2.894531e-01
4.488714e-01
8.878881e-01
4.852855e-01
9.770472e-01
4.668554e-01
3.293718e-01
```

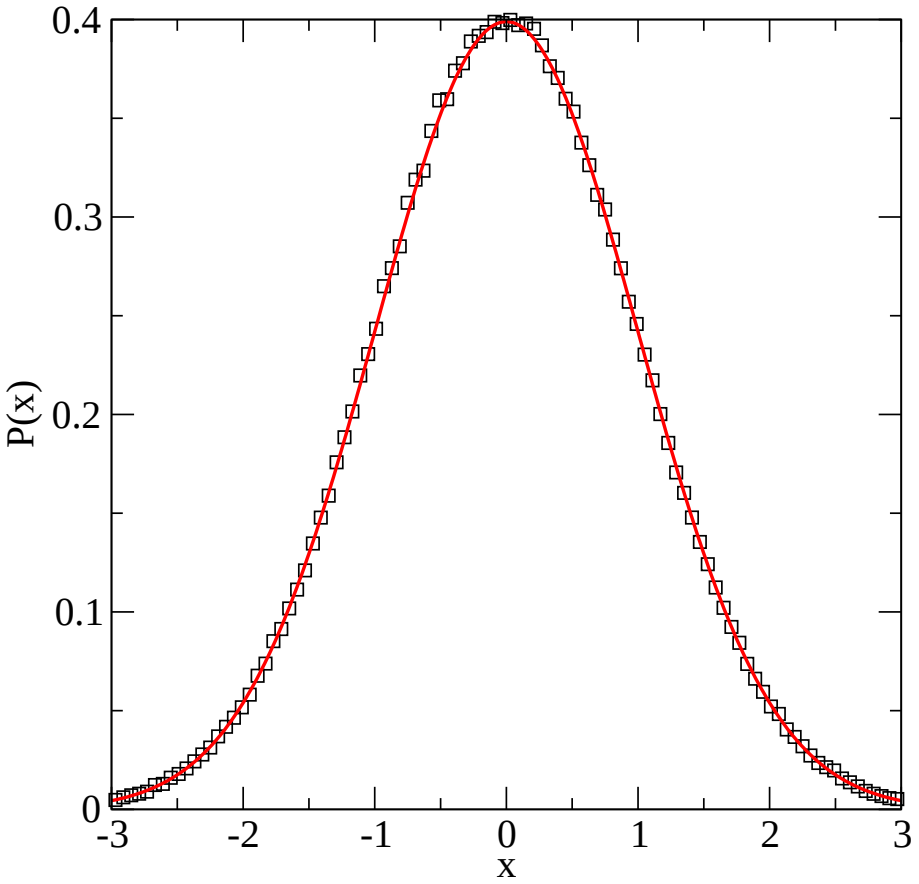
```
generator type: mt19937
5.885839e-01
2.894531e-01
4.488714e-01
8.878881e-01
4.852855e-01
9.770472e-01
4.668554e-01
3.293718e-01
7.670982e-01
2.296747e-01
```

Distribution de nombres aléatoires

On peut générer des nombres de distribution particulière (par exemple: une distribution gaussienne) et calculer un histogramme des résultats.

```
#include <stdio.h>
 #include <gsl/gsl_rng.h>
 #include <gsl/gsl_randist.h>
 #include <gsl/gsl_histogram.h>
    int main (void)
    { int i;
       double data;
      gsl_rng * r;  /* global generator */
      const gsl_rng_type * T=gsl_rng_mt19937 ;
      r = gsl_rng_alloc (T);
      gsl_histogram  *histo = gsl_histogram_alloc (100);
      gsl_histogram_set_ranges_uniform (histo, -3, 3);
      printf ("generator type: %s\n", gsl_rng_name (r));
      gsl_rng_set(r,3); //  graine
      for(i=0;i<1000000;i++) {data= gsl_ran_gaussian(r,1);
          gsl_histogram_increment (histo, data); }
     double sum=gsl_histogram_sum(histo);
      FILE *fichier;
      fichier=fopen("histo.res","w");
      for(i=0;i<100;i++) if ( histo->bin[i] > 0 )
       fprintf(fichier,"%e %e\n",-3+(i+0.5)*6.0/(double) 100,
            (double) histo->bin[i]/(6*sum)*100);
      gsl_histogram_free (histo);
      gsl_rng_free (r);
      return 0;
    }
```

Distribution de nombres aléatoires (2)



Conclusion et références

- La bibliothèque GSL offre un nombre abondant de procédures sur des domaines très variés de méthodes numériques.

Conclusion et références

- La bibliothèque GSL offre un nombre abondant de procédures sur des domaines très variés de méthodes numériques.
- La richesse a un prix! Il faut utiliser des fonctions peu intuitives et de la programmation avancée.