

# Méthodes numériques et langage C

## Calcul numérique de fonctions et résolution d'équations

R. Flamary

17 octobre 2018

## Calcul numérique de fonctions et résolution d'équations

### Problèmes

- ▶ Calculer  $y$  tel que

$$y = f(x)$$

- ▶ Trouver  $x$  tel que

$$f(x) = 0$$

- ▶ En utilisant uniquement des opérations standards fournies par un processeur.
- ▶ Les deux problèmes sont liés mais différent dans leur objectif.

### Objectifs du cours

- ▶ Programmation (efficace) des séries entières.
- ▶ Méthodes classiques de résolution d'équation (Dichotomie, Newton).
- ▶ Exemples de calcul de fonctions usuelles.
- ▶ Représentation des fonctions en séries entières.

## Plan du cours

<b>Problèmes et précision</b>	<b>3</b>
Problèmes numériques	3
Précision numérique	4
Vitesse de convergence	6
<b>Programmation de séries</b>	<b>7</b>
Séries numériques	8
Récurrence	10
<b>Résolution d'équations</b>	<b>14</b>
Dichotomie	15
Méthode de Newton	18
Point fixe	22
<b>Calcul de fonctions usuelles</b>	<b>24</b>
Racine carrée	25
Fonctions trigonométriques	29
Fonction exponentielle	32

2 / 33

## Précision numérique

### Algorithme itératif

- ▶ Principe :
  - ▶ Initialisation approximative de la solution.
  - ▶ Dans une boucle : mise à jour de la solution (amélioration de la précision)
- ▶ De nombreux calculs numériques du cours sont sous cette forme.
- ▶ La question de la précision numérique détermine le nombre d'itérations.
- ▶ Il existe plusieurs approches pour la condition d'arrêt des itérations.

### Nombre $n$ d'itérations max.

- ▶ On arrête après  $n$  itérations.
- ▶ Typiquement boucle `for`
- ▶ Complexité du calcul facile à estimer  $T() = nT(\text{iteration})$ .
- ▶ Précision finale du calcul plus difficile à évaluer (en général).

### Précision minimale $\epsilon$

- ▶ On arrête lorsque la variation de la solution entre deux itérations est inférieure ou égale à  $\epsilon$ .
- ▶ Difficulté d'évaluer le nombre d'itérations nécessaires.
- ▶ Meilleur contrôle de la précision du résultat.

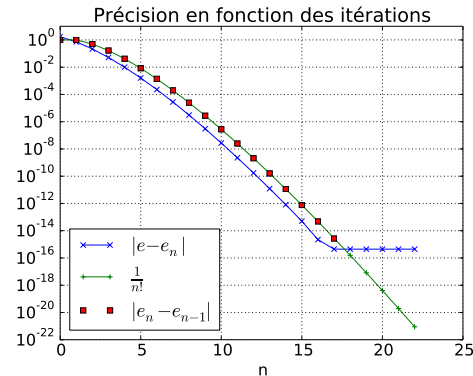
## Exemple de précision numérique

Soit la série suivante permettant de calculer la constante de Néper :

$$e = \sum_{k=0}^{\infty} \frac{1}{k!} \quad \text{et l'approximation} \quad e_n = \sum_{k=0}^n \frac{1}{k!}$$

### Calcul de la série

- ▶ Flottant 64 bits.
- ▶  $e - e_n$  erreur de la solution.
- ▶  $\frac{1}{n!}$  mise à jour de la solution.
- ▶  $|e_n - e_{n-1}|$  variation de la solution.



5 / 33

## Fonction analytique et série entière

Toute fonction analytique peut être exprimée sous la forme d'une série entière :

$$f(x) = \sum_{k=0}^{\infty} a_k (x - x_0)^k$$

où les  $a_k$  sont des réels et la série converge vers  $f(x)$  dans un voisinage de  $x_0$

- ▶ Les fonctions analytiques sont continues et infiniment différentiables.
- ▶ La série entière est une série de Taylor avec  $a_k = \frac{f^{(k)}(x_0)}{k!}$

### Applications

- ▶ Calcul numérique de fonctions complexes (exponentielle, trigonométriques, ...).
- ▶ Approximations physiques.

7 / 33

## Vitesse de convergence

- ▶ Soit la suite  $(x_k)$  qui converge vers la valeur  $x^*$ .
- ▶ On dit que la suite est convergente d'ordre  $q$  vers  $x^*$  si il existe  $\mu \in ]0, 1[$  tel que

$$\lim_{k \rightarrow \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^q} = \mu$$

- ▶  $\mu$  est la vitesse de convergence qui majore la déviation à la limite entre 2 itérations.
- ▶ Les ordres  $q$  les plus communs sont :
  - ▶  $q = 1$ , convergence linéaire.
  - ▶  $q > 1$ , super-linéaire.
  - ▶  $q = 2$ , convergence quadratique.
- ▶ La convergence quadratique implique un doublement du nombre de chiffres précis à chaque itération.
- ▶ La vitesse de convergence des méthodes itératives permet de prédire un ordre de grandeur des nombres d'itérations nécessaires pour obtenir une précision donnée.

6 / 33

## Programmation de série numérique

$$S_n = \sum_{k=0}^n u_k$$

### Principe général

- ▶ Boucle for.
- ▶ Initialisation de l'accumulateur à 0 ou à  $u_0$ .
- ▶ Ajout de  $u_i$  pour chaque valeur de  $i$ .
- ▶ Programmation d'une fonction  $u(i)$  ?

### Implémentation type

```

1 float sum_u_i(int n)
2 {
3     float res=0;
4     for (int k=0; k<=n; k++)
5         res=res+u(k);
6     return res;
7 }
```

### Discussion

- ▶ On suppose que la fonction `float u(int i)` est définie.
- ▶ Efficacité dépend de  $u_k$
- ▶  $T(\text{sum\_u\_i}) =$
- ▶  $S(\text{sum\_u\_i}) =$

8 / 33

## Calcul de tous les éléments de la série

$$S_n = \sum_{k=0}^n u_k$$

### Principe

- ▶ Pour des raisons de complexité numérique, il est parfois nécessaire de calculer tous les éléments de la série  $S_k$  jusqu'à  $S_n$ .
- ▶ Utilise toujours une boucle for.
- ▶ La fonction retourne un pointeur vers les valeurs de la série.
- ▶ Nécessite une allocation mémoire (malloc), libérer la mémoire avec free.

### Implémentation type

```
1 float* sum_u_i(int n)
2 {
3     float *res=malloc((n+1)*
4         sizeof(float));
5     res[0]=u(0);
6     for (int k=1;k<=n;k++)
7         res[k]=res[k-1]+u(k);
8     return res;
9 }
```

### Discussion

- ▶ On suppose que la fonction `float u(int i)` est définie.
- ▶ Efficacité dépend de  $u_k$
- ▶  $T(\text{sum\_u\_i}) =$
- ▶  $S(\text{sum\_u\_i}) =$

9 / 33

## Récurrence pour le calcul de séries (1)

- ▶ Une récurrence dans la suite est une fonction simple permettant de calculer  $u_{k+1}$  à partir de  $u_k$  :

$$u_{k+1} = r(u_k, k)$$

- ▶ Trouver une récurrence dans une série numérique permet souvent de la calculer avec une complexité moindre.
- ▶ Si  $T(u) > T(r)$  il faut utiliser la relation de récurrence.
- ▶ Exemple d'un codage non efficace de la fonction suivante (Ex. 5 Cours 1) :

$$f_{unc}(x, n) = \sum_{i=1}^n x^i$$

### Code

```
1 float func(float x,int n)
2 {
3     float f=0;
4     for (int i=1;i<=n;i++)
5         f=f+puissance(x,i);
6     return f;
7 }
```

### Complexité

- ▶ Nb opérations :
- ▶ Complexité  $T()$  :
- ▶ Complexité  $S()$  :

11 / 33

## Fonction récursive (rappel)

- ▶ Une fonction récursive est une fonction qui s'appelle elle-même.
- ▶ Pour le calcul de séries, les fonctions récursives ont souvent une complexité spatiale  $S(n)$  ou  $S(\log(n))$  (à cause de la pile).
- ▶ Il est nécessaire de s'assurer que la fonction se termine quelle que soit la valeur donnée en entrée (théorème de terminaison).

### Code source

```
1 int factorielle(int i)
2 {
3     if (i<=0)
4         return 1;
5     else
6         return i*factorielle(i-1);
7 }
8
9 int main()
10 {
11     for (int i=0;i<=8;i++)
12         printf("i=%d, i!=%d\n",i,
13             factorielle(i));
14 }
```

### Sortie

```
1 ./ex_factoriel
2 i=0, i!=1
3 i=1, i!=1
4 i=2, i!=2
5 i=3, i!=6
6 i=4, i!=24
7 i=5, i!=120
8 i=6, i!=720
9 i=7, i!=5040
10 i=8, i!=40320
```

10 / 33

## Récurrence pour le calcul de séries (2)

Fonction à calculer :

$$f_{unc}(x, n) = \sum_{i=1}^n x^i$$

- ▶ La récurrence est évidente :

$$x^i = x^{i-1} * x$$

- ▶ On calculera donc dans la boucle le terme  $x^i$  qui sera mis à jour à chaque itération.

### Code

```
1 float func(float x,int n)
2 {
3     float f=0,xi=x;
4     for (int i=1;i<=n;i++)
5     {
6         f=f+xi;
7         xi=xi*x;
8     }
9     return f;
10 }
```

### Complexité

- ▶ Nb opérations :
- ▶ Complexité  $T()$  :
- ▶ Complexité  $S()$  :

12 / 33

## Exercice 1

Soit la série de Taylor de la fonction exponentielle :

$$\exp(x) = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

Coder une implémentation efficace utilisant la récurrence permettant de calculer la série jusqu'à l'ordre  $n$ .

Quelle est sa complexité? Quelle serait la complexité si on utilisait les fonctions puissance et factorielle toutes deux de complexité  $T() = O(n)$ .

### Solution

### Complexité

▶ Nb opérations :

▶ Complexité  $T()$  :

▶ Complexité  $S()$  :

Si 2 fonctions  $O(n)$  dans la boucle :

▶ Complexité  $T()$  :

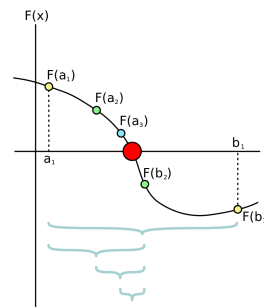
▶ Complexité  $S()$  :

13 / 33

## Dichotomie

### Diviser pour mieux régner

- ▶ Mise à jour de l'intervalle  $[a, b]$  assurant une contraction.
- ▶ Calculer  $f(c)$  avec  $c = \frac{a+b}{2}$ .
- ▶ Si  $f(a)f(c) > 0$  alors  $a \leftarrow c$ , sinon  $b \leftarrow c$
- ▶  $c$  est la solution de l'équation.



### Propriétés

- ▶ Précision numérique  $\epsilon_n = b_n - a_n$  simple à calculer :  $\epsilon_n = \frac{\epsilon_0}{2^n}$  (conv. linéaire).
- ▶ Nombre d'itérations jusqu'à convergence :  $n = \log_2 \frac{\epsilon_0}{\epsilon}$ .
- ▶ On prend souvent  $\epsilon = \frac{a+b}{2}\delta$  où  $\delta$  est la précision machine ( $\delta = 10^{-16}$  en 64 bits).
- ▶ Dans tous les cas  $n$  est majoré par le nombre de bits de la mantisse.

15 / 33

## Résolution d'équations unidimensionnelles

### Objectif

Trouver un réel  $x^* \in [a, b]$  satisfaisant la condition :

$$f(x) = 0$$

- ▶  $f(x)$  est une fonction continue sur l'intervalle  $[a, b]$ .
- ▶ On suppose également que  $f(a)f(b) < 0$ .
- ▶ Le passage de la fonction en 0 est assuré par le théorème des valeurs intermédiaires.

### Méthodes de résolution

- ▶ Dichotomie.
- ▶ Méthode de la fausse position.
- ▶ Méthode de Newton.
- ▶ Méthode du point fixe.

14 / 33

## Exercice 2 : Dichotomie en C

### Code

```

1 double dichotom(double a, double b,
2     double eps)
3 {
4     double c=(a+b)/2;
5     double fa=f(a),fc=f(c);
6
7
8
9
10
11
12
13
14     return c;
15 }

```

### Exercice

- ▶ Compléter la fonction à gauche.
- ▶ On suppose que la fonction `double f(double x)` est définie.
- ▶ Possibilité d'utiliser les fonctions de `<math.h>`
- ▶ Convergence atteinte lorsque  $|b - a| < \epsilon$ .
- ▶ Minimiser le nombre d'opérations sachant que l'appel à  $f(x)$  est potentiellement coûteux.

16 / 33

## Méthode de la fausse position

### Variante de la dichotomie

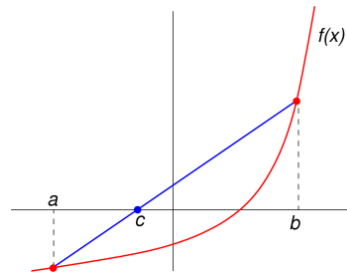
- ▶ On trace une droite entre les points  $(a, f(a))$  et  $(b, f(b))$ .
- ▶  $c$  est l'abscisse du point d'ordonnée nulle de cette droite :

$$\frac{f(a) - f(b)}{a - b}(c - b) + f(b) = 0$$

Ce qui nous donne :

$$c = a - \frac{a - b}{f(a) - f(b)}f(a)$$

- ▶ Si  $f(a)f(c) > 0$  alors  $a \leftarrow c$ , sinon  $b \leftarrow c$



### Propriétés

- ▶ Convergence superlinéaire (plus rapide que la dichotomie).
- ▶ Une variation s'appelant la méthode de la sécante ne nécessite pas que  $f(a)f(b) < 0$ .

17 / 33

## Convergence de la méthode de Newton

La convergence de la méthode de Newton vers une solution de l'équation est assurée si la fonction  $f(x)$  vérifie les conditions suivantes :

1.  $f$  est de classe  $C^2$  sur  $[a, c]$  (continue et dérivable deux fois).
2.  $f(a)f(b) < 0$ . ( $\Rightarrow \exists x, f(x) = 0$  avec la continuité).
3.  $\forall x \in [a, b] \quad f'(x) \neq 0$ .
4.  $f''(x)$  est de signe constant sur  $[a, b]$  (une seule solution dans l'intervalle).
5.  $|f(a)|/|f'(a)| < b - a$  et  $|f(b)|/|f'(b)| < b - a$  (on reste dans l'intervalle  $[a, b]$ ).

### Exercice 3 :

Vérifier la convergence sur  $[1, A]$  pour  $f(x) = x^2 - A = 0$  avec  $A > 1$

Source: [3, Chap 4]

19 / 33

## Méthode de Newton

### Principe

- ▶ Méthode itérative (pas d'intervalle).
- ▶ Initialisation  $x_0$ .
- ▶ La dérivée de la fonction en  $f(x_k)$  donne une approximation linéaire :

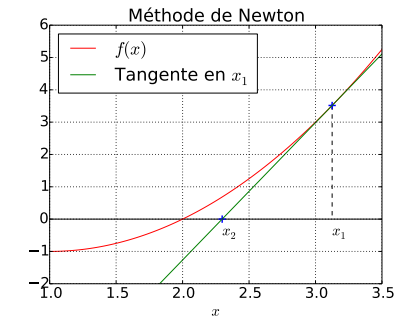
$$\tilde{f}(x) = f(x_k) + (x - x_k)f'(x_k)$$

- ▶ La nouvelle valeur est obtenue en annulant l'approximation :

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

### Propriétés

- ▶ Convergence quadratique à proximité d'une solution.
- ▶ Itérations peu précises loin d'une solution.
- ▶ Peut diverger si  $f$  a une dérivée nulle.
- ▶ Méthode de la fausse position est une Newton approchée (gradient approché).



18 / 33

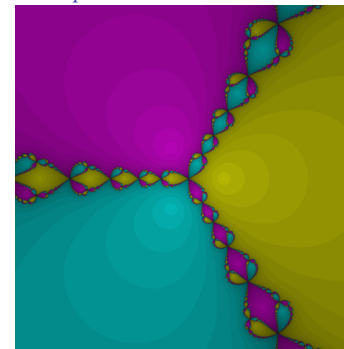
## Fractale de Newton

Soit l'équation complexe suivante :

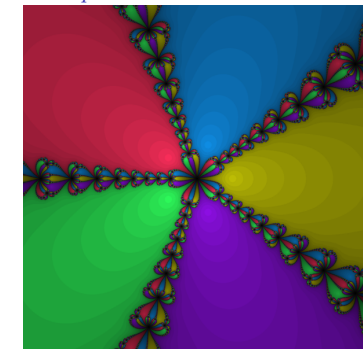
$$f(z) = z^p - 1 = 0$$

La méthode de Newton converge vers les solutions du problème de la forme  $z = e^{k2i\pi/p}$ . Si on détermine le point de convergence pour chaque pixel d'initialisation on obtient une fractale de Newton :

Pour  $p = 3$



Pour  $p = 5$



20 / 33

## Exercice 4 : Méthode de Newton

### Code

```
1 double newton(double x0, double eps)
2 {
3     double x;
4
5
6
7
8
9
10
11
12
13
14
15     return x;
16 }
17 }
```

### Exercice

- ▶ Compléter la fonction à gauche.
- ▶ On suppose que la fonction `double f(double x)` ainsi que sa dérivée `double df(double x)` sont définies.
- ▶ Possibilité d'utiliser les fonctions de `<math.h>`
- ▶ Convergence atteinte lorsque  $|x_k - x_{k-1}| < \epsilon$ .
- ▶ Minimiser le nombre d'opérations sachant que l'appel à  $f(x)$  et  $f'(x)$  est potentiellement coûteux.

21 / 33

## Convergence de la méthode du point fixe

### Théorème du point fixe de Banach

Soit  $E$  un espace métrique complet et  $g : E \rightarrow E$  une application contractante ( $k$ -lipschitzienne pour  $k < 1$ ).

- ▶  $g$  possède un unique point fixe  $x^*$  et ce point fixe est attractif.
- ▶ Toute suite de  $E$  vérifiant la récurrence  $x_{k+1} = g(x_k)$  converge vers  $x^*$

Résultat majeur pour prouver la convergence de suites

### Application lipschitzienne

Soient  $E$  une partie de  $\mathbb{R}$ ,  $f : E \rightarrow \mathbb{R}$  une application et  $k$  un réel positif, on dit que  $f$  est  $k$ -lipschitzienne si

$$\forall (x, y) \in E^2, |f(x) - f(y)| \leq k |x - y|.$$

Si  $k \in [0, 1[$  on dit que  $f$  est contractante.

23 / 33

## Méthode du point fixe

### Principe

- ▶ Si on peut réécrire  $f(x) = 0$  sous la forme

$$g(x) = x$$

L'équation ci dessus est une équation du point fixe associée à  $f(x) = 0$ .

- ▶ Algorithme

1. On initialise  $x_0 \in \mathbb{R}$
2. Pour chaque itération  $k$  :

$$x_{k+1} \leftarrow g(x_k)$$

### Propriétés

- ▶ Cet algorithme converge vers la solution de l'équation  $f(x) = 0$  sous certaines conditions.
- ▶ Il peut exister plusieurs fonctions  $g$  pour une équation donnée.

### Exercice 5

Soit l'équation de Ferrari  $f(x) = x^4 + 6x^2 - 60x + 36$ . Donner 2 fonctions  $g(x)$  point fixe de cette équation.

22 / 33

## Calcul de fonctions usuelles

### Problème

- ▶ Calculer les valeurs de fonctions mathématiques usuelles.
- ▶ Utiliser uniquement des opérations disponibles sur le processeur (somme, multiplication).
- ▶ Questions :
  - ▶ Implémentation en virgule flottante.
  - ▶ Série de Taylor, méthode de Newton ?
  - ▶ Précision d'une implémentation physique ?

### Fonctions étudiées

- ▶ Racine carrée et racine carrée inverse.
- ▶ Fonctions trigonométriques.
- ▶ Exponentielle.

### Discussion

- ▶ Calculs approchés implémentés en matériel.
- ▶ Extensions INTEL récentes (SSE, MMX, AVX, SMVL).
- ▶ Calcul sur des séquences en mémoire (très efficaces).

24 / 33

## Racine carrée

Soit la fonction racine carrée

$$\text{sqrt}(x) = \sqrt{x}$$

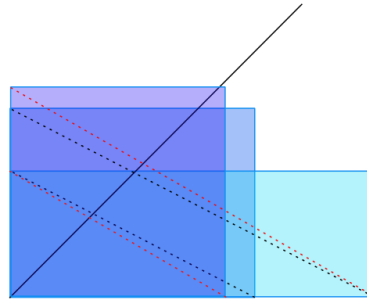
- ▶ Problème équivalent à résoudre l'équation :

$$f(x) = x^2 - S$$

- ▶ Si on applique la méthode de Newton on trouve l'itération :

$$x_{k+1} = x_n - \frac{f(x_n)}{f'(x_n)} =$$

- ▶ Méthode exposée par Héron d'Alexandrie dans son ouvrage *Metrica*.
- ▶ Certains calculs égyptiens suggèrent que le méthode est encore plus ancienne.
- ▶ Interprétation géométrique simple (moyenne des cotés d'un rectangle).
- ▶ Vitesse de convergence quadratique (Newton).



25 / 33

## Racine carrée (2)

### Initialisation

- ▶ Vitesse de convergences des méthodes itérative dépendent de l'initialisation.
- ▶ Pour un flottant en base binaire de la forme  $m \cdot 2^{2n}$  on initialise par :

$$x_0 = 2^n$$

### Autres variantes et généralisation

- ▶ Méthodes de calculs décimale par décimale (Bâtons de Neper, Algorithme de la potence). Plus lente que la méthode de Héron mais ne nécessite pas d'inversion.
- ▶ Approximation de Bakhshali. Équivalente à 2 itération de la méthode de Newton.
- ▶ Méthode de Newton pour la racine  $n$ ième :

$$x_{k+1} = \frac{1}{n} \left( (n-1)x_k + \frac{A}{x_k^{n-1}} \right)$$

26 / 33

## Racine carrée inverse

$$\text{isqrt}(x) = \frac{1}{\sqrt{x}}$$

### Méthode de Newton

- ▶ Équation :

$$f(x) =$$

- ▶ Itérations :

$$x_{k+1} =$$

### Cas d'utilisation

- ▶ Illumination dans les scènes 3D en infographie.
- ▶ Calcul rapide de vecteur normal :

$$\hat{\mathbf{v}} = \frac{\mathbf{v}}{\sqrt{\|\mathbf{v}\|}}$$



27 / 33

## Racine carrée inverse rapide

### Principe

Soit  $x$  un flottant IEEE 754.

- ▶ Calcul sur les flottants peut être plus coûteux que sur des entiers.
- ▶ Approximation de  $\log_2(S)$  en traitant le flottant comme un entier :

$$I_S \approx L \log_2(S) + C, \quad \log_2(S) \approx \frac{1}{L} (I_S + C)$$

- ▶  $\log_2(x) = -1/2 \log_2(S)$  on trouve donc :

$$I_x \approx$$

Dans le code  $-\frac{3}{2}C = 0x5f3759df$

- ▶ La dernière étape est une itération de Newton (et une seconde commentée).
- ▶ Tiré du code source de Quake 3.

### Code

```

1 float Q_rsqrt( float number )
2 {
3     long i;
4     float x2, y;
5     const float threehalfs = 1.5F;
6
7     x2 = number * 0.5F;
8     y = number;
9     i = * ( long * ) &y; // evil
10    floating point bit level
11    hacking
12    i = 0x5f3759df - ( i >> 1 ); //
13    what the fuck?
14    y = * ( float * ) &i;
15    y = y * ( threehalfs - ( x2 * y * y
16    ) ); // 1st iteration
17    // y = y * ( threehalfs - ( x2 * y *
18    y ) ); // 2nd iteration, this
19    can be removed
20
21    return y;
22 }
```

28 / 33

## Fonction sinus

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

### Implémentation dans la Glibc (Bibliothèque C standard)

- ▶ Méthodes de réduction de l'intervalle :  $\sin(x + k2\pi) = \sin(x)$ .
- ▶ Utilisation de la fonction `fsin` des processeurs intel en 32 et 64 bits.
- ▶ Sinon implémentation détaillée dans `sysdeps/ieee754/dbl-64/s.sin.c`

### Algorithme simplifié dans `fdlibm`

- ▶ Méthodes de réduction de l'intervalle sur  $[-\frac{\pi}{4}, \frac{\pi}{4}]$ .
- ▶ Fonction sinus impaire donc calcul uniquement pour  $x > 0$ .
- ▶ Si  $|x| < 2^{-27}$  alors on retourne  $x$ .
- ▶ Sinon on utilise une approximation polynomiale de degré 13.

29 / 33

## Exercice 6 : Fonction Sinus

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

### Code

```
1 double sinslow(double x)
2 {
3     double res=x,temp=x;
4
5
6
7
8
9
10
11
12     return res;
13 }
```

### Exercice

- ▶ Compléter la fonction à gauche.
- ▶ Utiliser la série de Taylor pour calculer la valeur du sinus.
- ▶ Convergence atteinte lorsque  $\frac{|x_k - x_{k-1}|}{|x_k|} < 1e - 16$ .

### Discussion

- ▶ Problèmes ?

30 / 33

## Exercice 7 : Fonction Sinus

### Code

```
1 double sinfast(double x)
2 {
3
4
5
6
7
8
9
10
11 }
```

### Exercice

- ▶ Compléter la fonction à gauche.
- ▶ Coder la méthode de réduction de l'intervalle sur  $[-\pi, \pi]$ .
- ▶ Appeler la fonction `sinslow` lorsque vous avez réduit l'intervalle.

31 / 33

## Fonction exponentielle

$$\exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

### Implémentation dans `fdlibm`

- ▶ Réduction de l'intervalle, on cherche l'entier  $k$  tel que

$$x = k * \ln(2) + r, \quad |r| \leq \frac{\ln(2)}{2}$$

- ▶ Calcul précis de  $\exp(r)$  avec approximation polynomiale.
- ▶  $\exp(x) = 2^k \exp(r)$
- ▶ Approximation polynomiale de Remez très précise autour de 0 (bornée par  $2^{-59}$ ).
- ▶ Cas particuliers :
  - ▶  $\infty$  pour  $x > 7.09782712893383973096e + 02$
  - ▶  $0$  pour  $x < -7.45133219101941108420e + 02$

32 / 33



## Ressources bibliographiques I

- [1] "Freely distributable libm (fdlibm)," <http://www.netlib.org/fdlibm/>, version du 2015-09-28.
- [2] "The gnu c library," <http://www.gnu.org/software/libc/>, version du 2015-09-28.
- [3] J. Bastien and J.-N. Martin, *Introduction à l'analyse numérique : applications sous Matlab : cours et exercices corrigés*. Dunod, 2003.
- [4] R. P. Brent and P. Zimmermann, *Modern computer arithmetic*. Cambridge University Press, 2010, vol. 18.
- [5] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys (CSUR)*, vol. 23, no. 1, pp. 5–48, 1991.
- [6] D. Henderson, "Elementary functions : Algorithms and implementation," *Mathematics and Computer Education*, vol. 34, no. 1, p. 94, 2000.
- [7] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes in C*. Cambridge university press Cambridge, UK, 1992.