Interpolation et approximation

Cours ENPC - Pratique du calcul scientifique

Cadre général de l'interpolation

Soients

- un segment $[a, b]$ de $\mathbb{R}$
- $n+1$ points distincts $a = x_0 < x_1 < \ldots < x_n = b$
- $n+1$ valeurs $u_0, u_1, \ldots, u_n$
- un sous-espace $\mathrm{Vect}(\varphi_0, \varphi_1, \ldots, \varphi_n)$ de l'e.v. des fonctions continues sur $[a, b]$

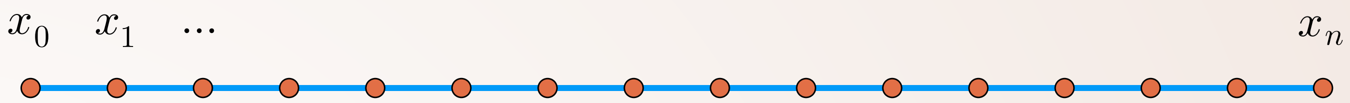
On cherche à identifier un élément de $\mathrm{Vect}(\varphi_0, \varphi_1, \ldots, \varphi_n)$ soit

$$\boxed{\widehat{u}(x) = \alpha_0 \varphi_0(x) + \cdots + \alpha_n \varphi_n(x)}$$

tel que

$$\boxed{\forall i \in \{0, \ldots, n\}, \qquad \widehat{u}(x_i) = u_i}$$

$$\boxed{\mathsf{A}\boldsymbol{\alpha} := \begin{pmatrix} \varphi_0(x_0) & \varphi_1(x_0) & \ldots & \varphi_n(x_0) \\ \varphi_0(x_1) & \varphi_1(x_1) & \ldots & \varphi_n(x_1) \\ \vdots & \vdots & & \vdots \\ \varphi_0(x_n) & \varphi_1(x_n) & \ldots & \varphi_n(x_n) \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_n \end{pmatrix} := \boldsymbol{b}}$$



$x_k = a + (b-a)\dfrac{k}{n}$ (nœuds équidistants)

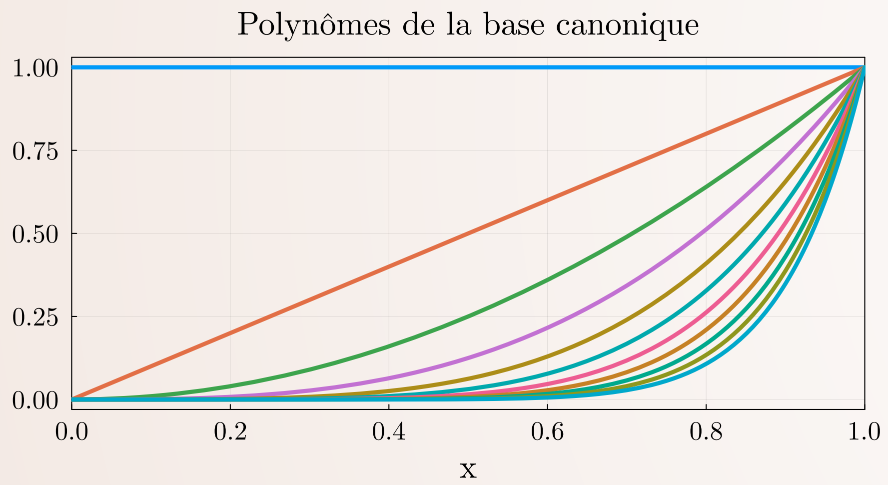
$$x_k = a + (b-a)\,\frac{1 - \cos\left(\pi \dfrac{k + \frac{1}{2}}{n+1}\right)}{2}$$ (nœuds de Tchebychev)

Familles de polynômes

Base canonique

$$\varphi_i(x) = x^i \quad \Rightarrow \quad \mathsf{A} = \begin{pmatrix} 1 & x_0 & \dots & x_0^n \\ 1 & x_1 & \dots & x_1^n \\ \vdots & \vdots & & \vdots \\ 1 & x_n & \dots & x_n^n \end{pmatrix}$$

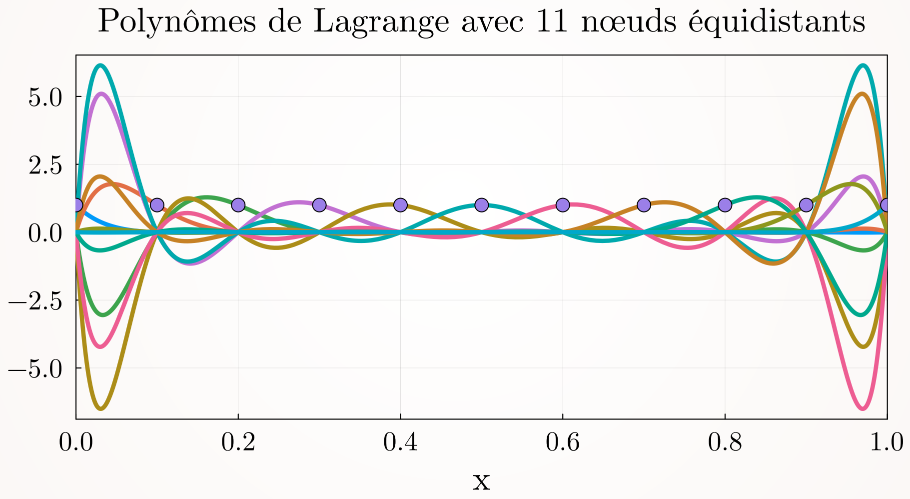
▶ Code



Polynômes de Lagrange

$$\varphi_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j} \quad \Rightarrow \quad \mathsf{A} = \mathsf{I}$$

▶ Code

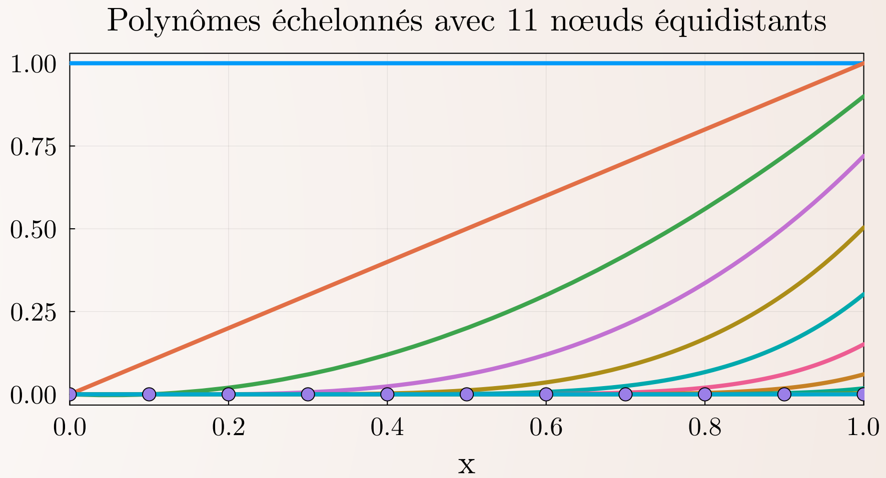


Polynômes échelonnés (Gregory-Newton généralisé)

$$\varphi_i(x) = (x - x_0)(x - x_1)(x - x_2)\dots(x - x_{i-1})$$

$\Rightarrow \quad$ A triangulaire inférieure

▶ Code



Contrôle de l'erreur

Théorème

- $u\colon [a,b] \to \mathbb{R}$ est une fonction dans $\mathcal{C}^{n+1}([a,b])$
- $a = x_0 < x_1 < \ldots < x_n = b$ sont $n+1$ nœuds distincts
- $\widehat{u}$ est un polynôme de degré au plus n, interpolateur de u aux points $x_0, x_1, \ldots, x_n$, i.e. $\widehat{u}(x_i) = u(x_i)$ pour tout $i \in \{0, \ldots, n\}$

Alors on a $\quad \forall\, x \in [a,b], \quad \exists\, \xi = \xi(x) \in [a,b]$

$$\boxed{e_n(x) := u(x) - \widehat{u}(x) = \frac{u^{(n+1)}(\xi)}{(n+1)!}(x - x_0)\ldots(x - x_n)}$$

Exercice : démontrer le théorème

1. Examiner le cas où x est l'un des x_i.
2. Posant $\omega_n(x) = \prod_{i=0}^{n}(x - x_i)$ et $g(t) = e_n(t)\omega_n(x) - e_n(x)\omega_n(t)$ pour un x donné différent des x_i, montrer que $g^{(k)}(t)$ avec $0 \le k \le n+1$ admet $n+2-k$ racines distinctes dans $[a,b]$.
3. Conclure.
4. Que dire du cas où u est un polynôme de degré au plus n ?
5. En supposant que u est dans $\mathcal{C}^\infty([a,b])$, fait-on systématiquement tendre l'erreur vers 0 lorsque le nombre de nœuds tend vers l'infini ?

Corollaire du théorème (mêmes hypothèses)

- On pose $C_{n+1} = \sup_{x\in[a,b]} \left|u^{(n+1)}(x)\right|$ et $h = \max_{i\in\{0,\ldots,n-1\}}|x_{i+1} - x_i|$

Alors on montre que $E_n := \sup_{x\in[a,b]}\left|e_n(x)\right| \leqslant \frac{C_{n+1}}{4(n+1)} h^{n+1}$

- Indications :
  - si $x \in [x_i, x_{i+1}]$, $(x - x_i)(x_{i+1} - x) = \left(\frac{x_{i+1}-x_i}{2}\right)^2 - \left(x - \frac{x_i + x_{i+1}}{2}\right)^2 \le \frac{h^2}{4}$
  - $|\omega_n(x)| \le \frac{h^2}{4} \times 2h \times 3h \times 4h \times \cdots \times nh$

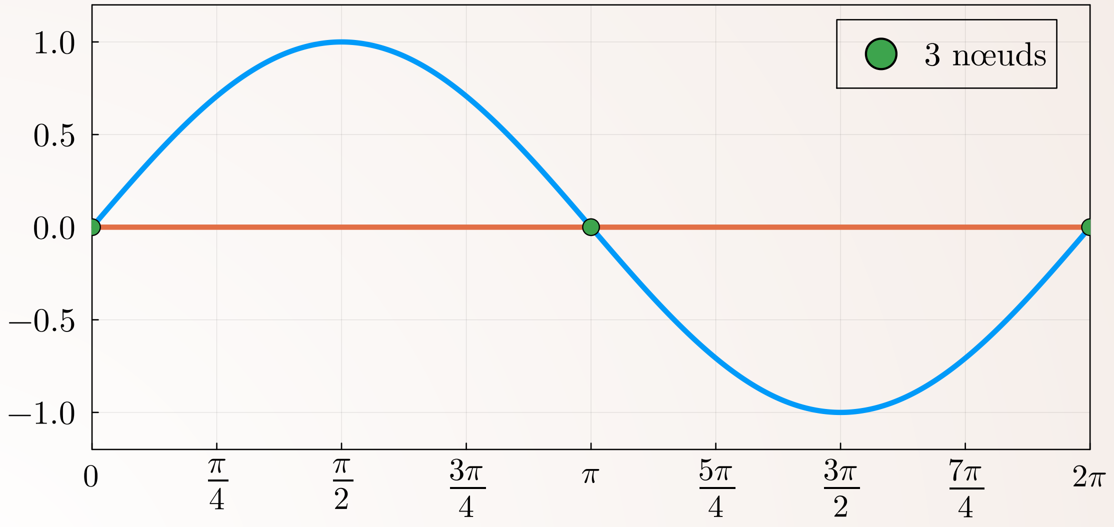
Remarques

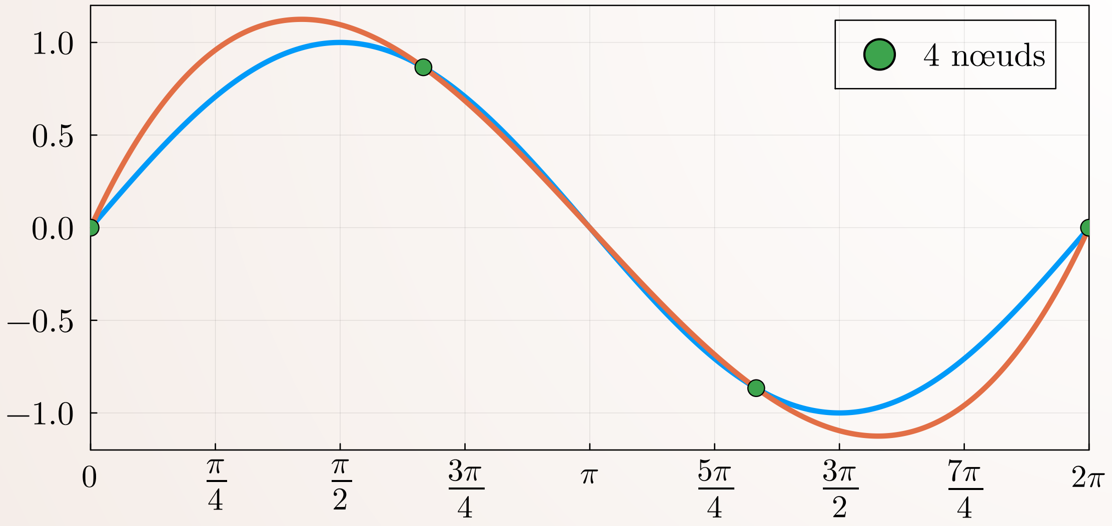
- h dépend de n et du choix des nœuds (en $1/n$ pour nœuds équidistants)
- C_n ne peut *a priori* être contrôlé
- Dans certains cas, C_n ne croît pas trop vite avec n de sorte que $E_n \underset{n\to\infty}{\to} 0$ (ex : sinus)
- Contre-exemple avec la fonction de Runge $u(x) = \frac{1}{1+25x^2}$ pour laquelle le majorant de E_n tend vers l'infini si les nœuds sont équidistants
- Optimiser l'interpolation ? → optimisation des nœuds ou interpolation par morceaux

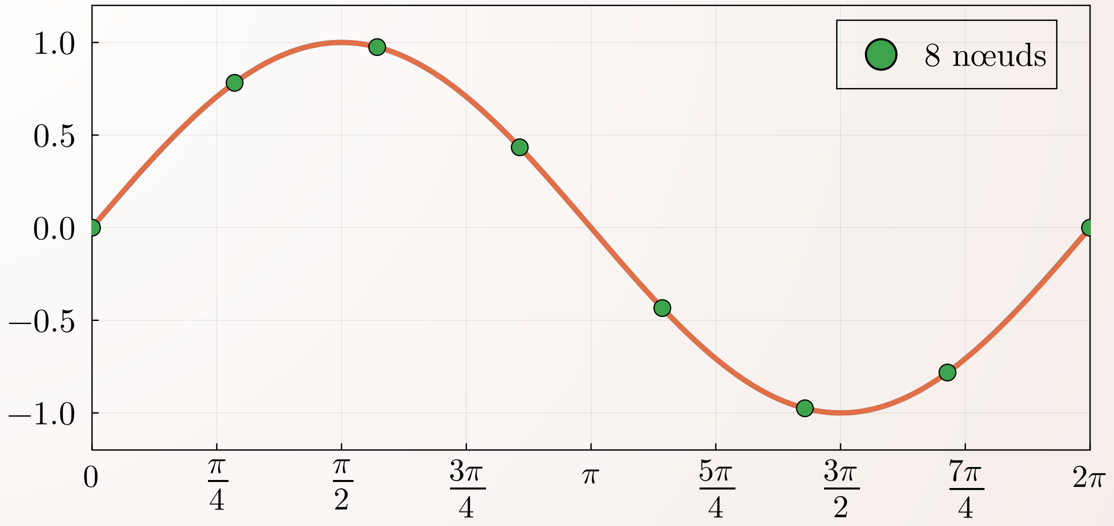
Fonction sinus avec nœuds équidistants

$$\boxed{u(x) = \sin x}$$

$$\boxed{x_k = a + (b-a)\frac{k}{n} \quad (0 \le k \le n)}$$



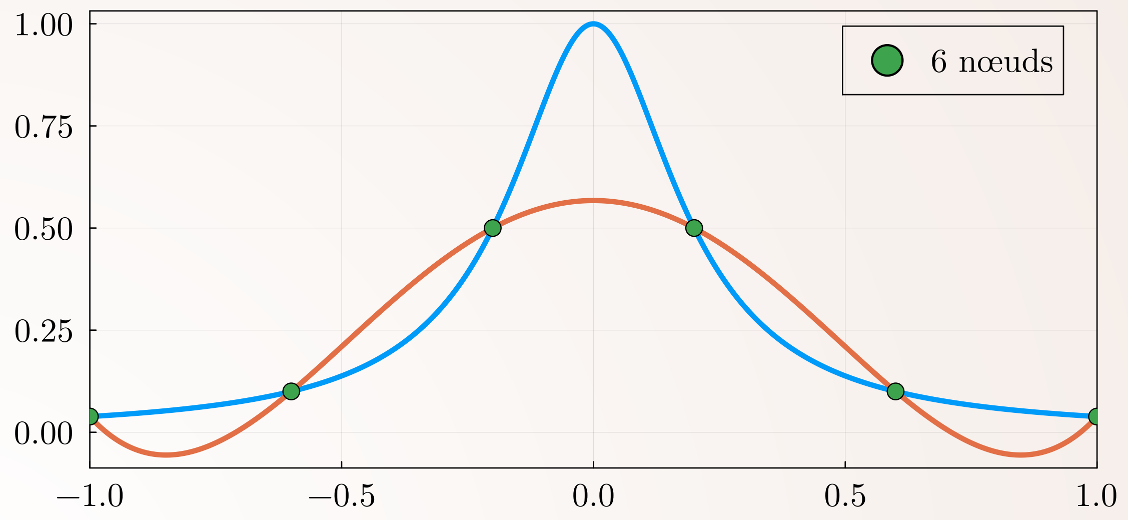


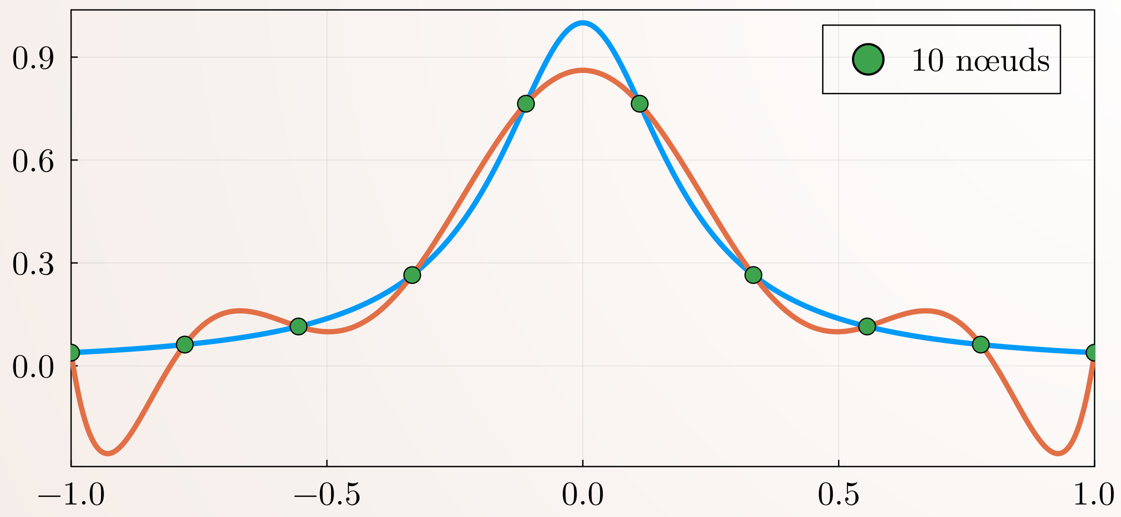


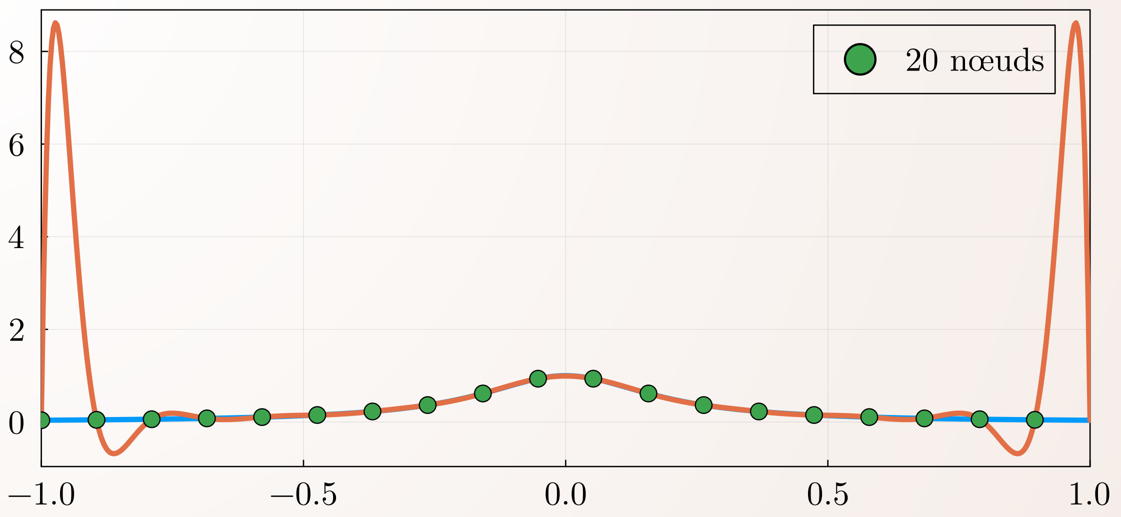
Fonction de Runge avec nœuds équidistants

$$u(x) = \frac{1}{1+25x^2}$$

$$x_k = a + (b-a)\frac{k}{n} \quad (0 \le k \le n)$$







Optimisation des nœuds : nœuds de Tchebychev

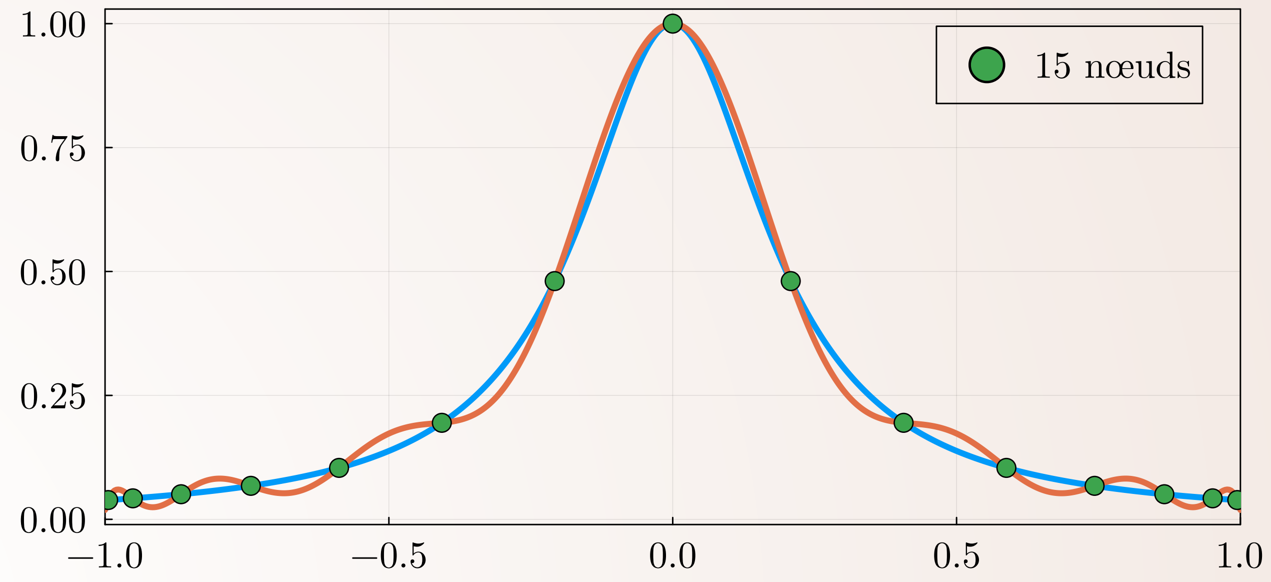
Contrôle du majorant de l'erreur

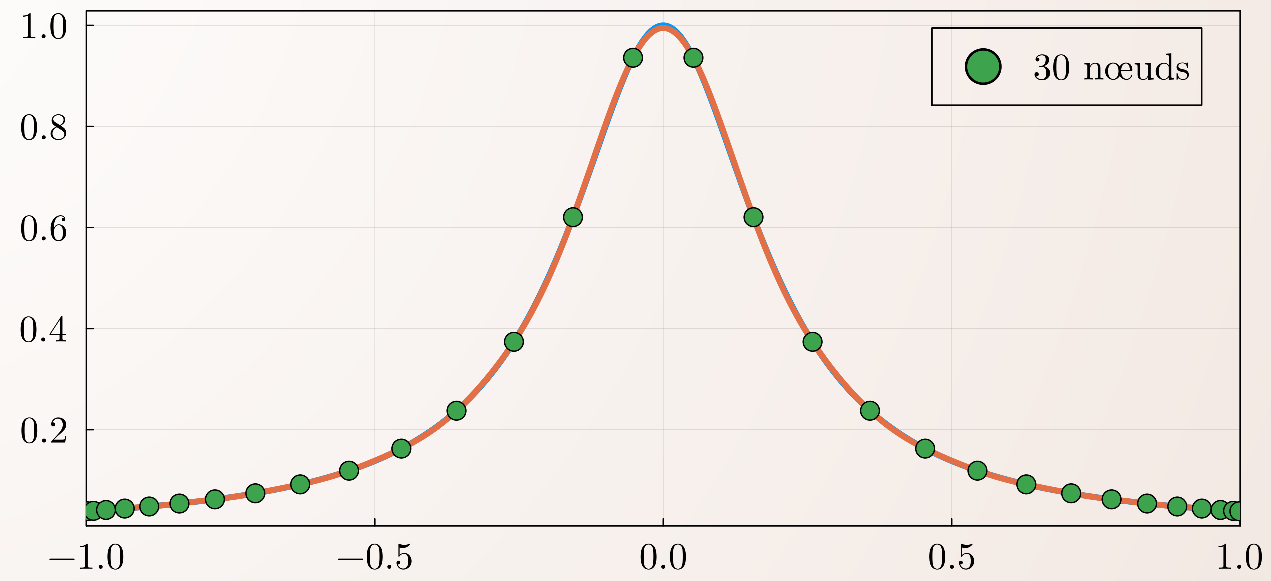
$$\boxed{e_n(x) := u(x) - \widehat{u}(x) = \frac{u^{(n+1)}(\xi)}{(n+1)!}(x-x_0)\ldots(x-x_n)}$$

- Augmenter le nombre de nœuds $n+1$ change $u^{(n+1)}$ ⇒ difficile à contrôler
- Idée : pour n donné, choisir les nœuds pour minimiser $\sup_{x\in[a,b]} |(x-x_0)\ldots(x-x_n)|$
- Pour tout polynôme p unitaire de degré n, on montre que $\sup_{x\in[-1,1]} |p(x)| \geq \frac{1}{2^{n-1}}$
- Borne $\frac{1}{2^{n-1}}$ atteinte pour $\frac{T_n(x)}{2^{n-1}}$ avec $T_n(x) = \cos(n \arccos x)$ (polynôme de Tchebychev)
- Les zéros de T_n sont donc les $x_k = -\cos(\pi \frac{k+\frac{1}{2}}{n})$ $(0 \leq k < n)$ (signe "-" pour ordre)
- Soit $x_k = -\cos(\pi \frac{k+\frac{1}{2}}{n+1})$ $(0 \leq k \leq n)$ pour $n+1$ points
- Dans le cas $[a,b]$, pour n points

$$\boxed{x_k = a + (b-a)\frac{1-\cos(\pi\frac{k+\frac{1}{2}}{n})}{2} \qquad (0 \leq k < n)}$$

- Application à la fonction de Runge $u(x) = \frac{1}{1+25x^2}$





Interpolation par morceaux

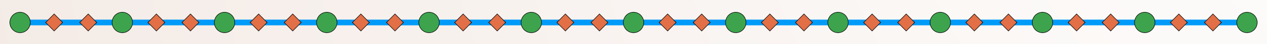
Interpolation continue par morceaux

$$e_n(x) := u(x) - \widehat{u}(x) = \frac{u^{(n+1)}(\xi)}{(n+1)!}(x - x_0)\dots(x - x_n)$$

$$E_n := \sup_{x\in[a,b]} \left|e_n(x)\right| \leqslant \frac{C_{n+1}}{4(n+1)} h^{n+1}$$
$$\text{avec } C_{n+1} = \sup_{x\in[a,b]} \left|u^{(n+1)}(x)\right| \text{ et } h = \max_{i\in\{0,...,n-1\}} |x_{i+1} - x_i|$$

Idée pour contrôler le majorant :

- découper l'intervalle $[a, b]$ avec $n + 1$ nœuds, par exemple $h = \frac{b-a}{n}$
- interpoler avec un polynôme de degré m sur $[x_i, x_{i+1}]$



$$\sup_{x\in[x_i,x_{i+1}]} \left|e_n(x)\right| \leqslant \frac{C_{m+1}}{4(m+1)} \left(\frac{h}{m}\right)^{m+1}$$

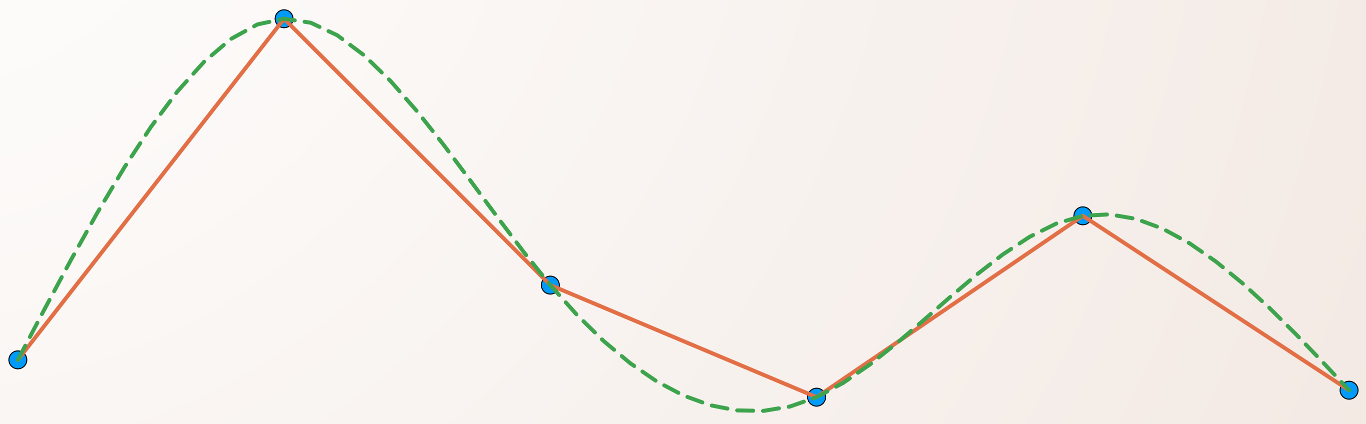
- m est fixé petit donc l'erreur est majorée uniformément par Ch^{m+1}

Régularité supérieure

- L'interpolation précédente n'assure que la continuité $\widehat{u}(x_i^-) = \widehat{u}(x_i^+)$
- Pour améliorer la régularité, on peut imposer des conditions sur les dérivées supérieures

  → splines cubiques :

  - polynôme de degré 3 sur chaque sous-intervalle : $4n$ inconnues
  - interpolation aux nœuds x_i $(0 \leq i \leq n)$: $2n$ équations
  - raccord des dérivées aux nœuds x_i $(0 < i < n)$: $n - 1$ équations
  - raccord des dérivées secondes aux nœuds x_i $(0 < i < n)$: $n - 1$ équations
  - nombre total d'équations : $4n - 2 \Rightarrow$ il en faut 2 de plus (en général dérivées secondes nulles aux extrémités)



Approximation par moindres carrés

Position du problème

- $n+1$ nœuds distincts $a = x_0 < x_1 < \ldots < x_n = b$
- $n+1$ valeurs $u_0, u_1, \ldots, u_n$
- un sous-espace $\mathrm{Vect}(\varphi_0, \varphi_1, \ldots, \varphi_m)$ de l'e.v. des fonctions continues sur $[a,b]$ (en général des polynômes) mais ici $m < n$

On cherche à identifier un élément de $\mathrm{Vect}(\varphi_0, \varphi_1, \ldots, \varphi_m)$ soit

$$\boxed{\widehat{u}(x) = \alpha_0 \varphi_0(x) + \cdots + \alpha_m \varphi_m(x)}$$

tel que

$$\boxed{\forall i \in \{0, \ldots, n\}, \qquad \widehat{u}(x_i) \approx u_i}$$

$$\boxed{\mathsf{A}\boldsymbol{\alpha} := \begin{pmatrix} \varphi_0(x_0) & \varphi_1(x_0) & \ldots & \varphi_m(x_0) \\ \varphi_0(x_1) & \varphi_1(x_1) & \ldots & \varphi_m(x_1) \\ \varphi_0(x_2) & \varphi_1(x_2) & \ldots & \varphi_m(x_2) \\ \vdots & \vdots & & \vdots \\ \varphi_0(x_{n-2}) & \varphi_1(x_{n-2}) & \ldots & \varphi_m(x_{n-2}) \\ \varphi_0(x_{n-1}) & \varphi_1(x_{n-1}) & \ldots & \varphi_m(x_{n-1}) \\ \varphi_0(x_n) & \varphi_1(x_n) & \ldots & \varphi_m(x_n) \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_m \end{pmatrix} \approx \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{n-2} \\ u_{n-1} \\ u_n \end{pmatrix} =: \boldsymbol{b}}$$

Minimisation de

$$\boxed{J(\boldsymbol{\alpha}) = \frac{1}{2} \sum_{i=0}^{n} |\widehat{u}(x_i) - u_i|^2 = \frac{1}{2} \sum_{i=0}^{n} \left(\sum_{j=0}^{m} \alpha_j \varphi_j(x_i) - u_i \right)^2 = \frac{1}{2} \|\mathsf{A}\boldsymbol{\alpha} - \boldsymbol{b}\|^2}$$

On cherche donc α tel que

$$\nabla J(\boldsymbol{\alpha}) = \frac{1}{2} \nabla \Big((\mathsf{A}\boldsymbol{\alpha} - \boldsymbol{b})^T (\mathsf{A}\boldsymbol{\alpha} - \boldsymbol{b}) \Big) = \mathbf{0}$$

soit

$$\mathrm{d}J = (\mathsf{A}\mathrm{d}\boldsymbol{\alpha})^T (\mathsf{A}\boldsymbol{\alpha} - \boldsymbol{b}) = \mathrm{d}\boldsymbol{\alpha}^T \mathsf{A}^T (\mathsf{A}\boldsymbol{\alpha} - \boldsymbol{b}) \Rightarrow \nabla J(\boldsymbol{\alpha}) = \mathsf{A}^T (\mathsf{A}\boldsymbol{\alpha} - \boldsymbol{b}) = \mathbf{0}$$

ce qui donne le système à résoudre

$$\boxed{\mathsf{A}^T \mathsf{A} \boldsymbol{\alpha} = \mathsf{A}^T \boldsymbol{b}}$$

avec $\mathsf{A}^T\mathsf{A}$ matrice $m \times m$ inversible si A est de rang m (colonnes indépendantes)

Remarque

Sous `Julia` le résultat du calcul $\alpha = (\mathsf{A}^T\mathsf{A})^{-1}\mathsf{A}^T\boldsymbol{b}$ peut simplement être obtenu par

```
α = A\b
```

Bibliothèque `Polynomials.jl`

- Dépôt GitHub https://github.com/JuliaMath/Polynomials.jl
- Doc https://juliamath.github.io/Polynomials.jl/stable/

Home   Edit on GitHub

Polynomials.jl

Polynomials.jl is a Julia package that provides basic arithmetic, integration, differentiation, evaluation, and root finding for univariate polynomials.

To install the package, run

```
(v1.6) pkg> add Polynomials
```

As of version `v3.0.0` Julia version `1.6` or higher is required.

The package can then be loaded into the current session through

```
using Polynomials
```

Quick Start

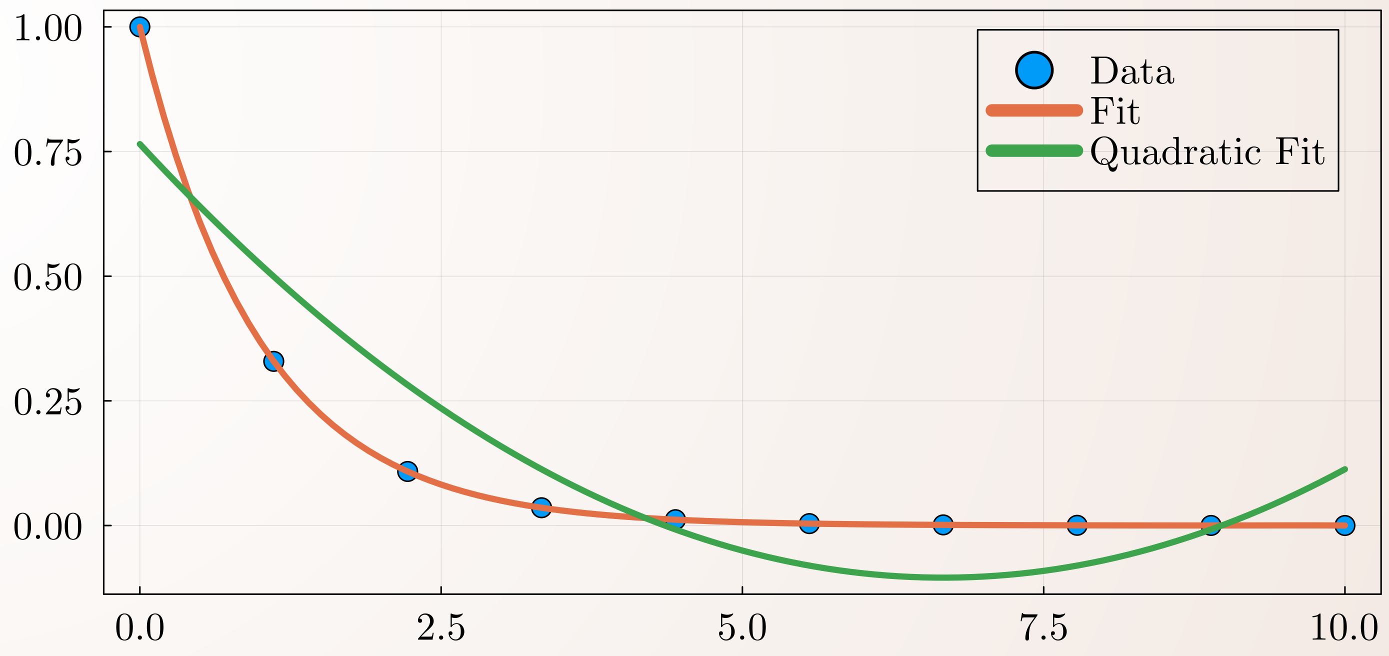
Construction and Evaluation

Construct a polynomial from its coefficients, lowest order first.

Exemple d'utilisation de `fit`

```
using Plots, Polynomials
xs = range(0, 10, length=10)
ys = @. exp(-xs)
f = fit(xs, ys) # degree = length(xs) - 1
f2 = fit(xs, ys, 2) # degree = 2

scatter(xs, ys, label="Data")
plot!(f, extrema(xs)..., label="Fit")
plot!(f2, extrema(xs)..., label="Quadratic Fit")
plot!(legend=:topright)
```



Bibliothèque `Interpolations.jl`

- Dépôt GitHub https://github.com/JuliaMath/Interpolations.jl
- Doc http://juliamath.github.io/Interpolations.jl/stable/

Home Edit on GitHub

Interpolations

Dec 2022 v0.14.7 | PkgEval fail | CI passing | Dependents 481 | docs stable | docs latest

The package Interpolations.jl implements a variety of interpolation schemes for the Julia language. It has the goals of ease-of-use, broad algorithmic support, and exceptional performance.

Currently this package supports B-splines and irregular grids. The API has been designed with intent to support more options. Pull-requests are more than welcome! It should be noted that the API may continue to evolve over time.

There are many other interpolation packages implemented in Julia. For a listing, see Other Interpolation Packages.

Some of these packages support methods that `Interpolations` does not, so if you can't find what you need here, check one of them or submit a pull request here.

Installation

Interpolations.jl can be installed via the following invocation since it is a registered Julia package.

```
using Pkg
Pkg.add("Interpolations")
```

Exemple d'utilisation `ici` (section `Convenience Constructors`)

```
using Interpolations, Plots
a = 1.0 ; b = 10.0 ; xᵢ = a:1.0:b # bounds and knots
F(x) = cos(x^2/9)
yᵢ = F.(xᵢ) # function application by broadcasting
itp_linear = linear_interpolation(xᵢ,yᵢ) ; itp_cubic = cubic_spline_interpolation(x
F_linear(x) = itp_linear(x) ; F_cubic(x) = itp_cubic(x)
x_new = a:0.1:b # smoother interval, necessary for cubic spline
scatter(xᵢ, yᵢ, markersize=10,label="Data points")
plot!(F_linear, x_new, w=4,label="Linear interpolation")
plot!(F_cubic, x_new, linestyle=:dash, w=4, label="Cubic Spline interpolation")
plot!(F, x_new, linestyle=:dot, w=4, label="Initial function")
width, height = 1500, 800 ; plot!(size = (width, height), legend = :bottomleft)
```