Solving PDEs using Spectral Methods in the Chebyshev basis

by example of the Heat Equation

Special Topic on Approximation of Functions

Candidate Number: 12345

Abstract

This work shall attempt to numerically solve the heat equation $u_t = \alpha u_{xx}$ with Dirichlet boundary conditions over the domain $[-1, 1] \times [0, T]$ by representing the spatial component as a *Chebfun* (Chebyshev series) and moving on in time by the Forward Euler numerical scheme.

Our Goal: Numerically obtain the solution $u(x, T)$ of

$$\begin{cases} \frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} & u : [-1, 1] \times [0, T] \mapsto \mathbb{R},\ T \in \mathbb{R}^+,\ \alpha \in \mathbb{R} \backslash \{0\} \\ u(x_j, 0) = u_0(x_j) & \forall x_j \in X_N,\ N \in \mathbb{N},\ N > 1,\ u_0 : [-1, 1] \mapsto \mathbb{R} \\ u(b, t) = u_0(b) & \forall b \in \{-1, 1\},\ \forall t \in (0, T]\,. \end{cases}$$

The implementation, centered around what we will refer to as **TschebFun**, including three major algorithms `TschebFun::interpolantThrough()`, `TschebFun::evaluateOn()` and `TschebFun::derivative()`, is done manually in C++, extended to work as a Python module and for demonstration, even features a high-level graphical interface to play with. Finally, we will compare the numerical results with the output of *Chebfun*'s high-level `pde15s()`.

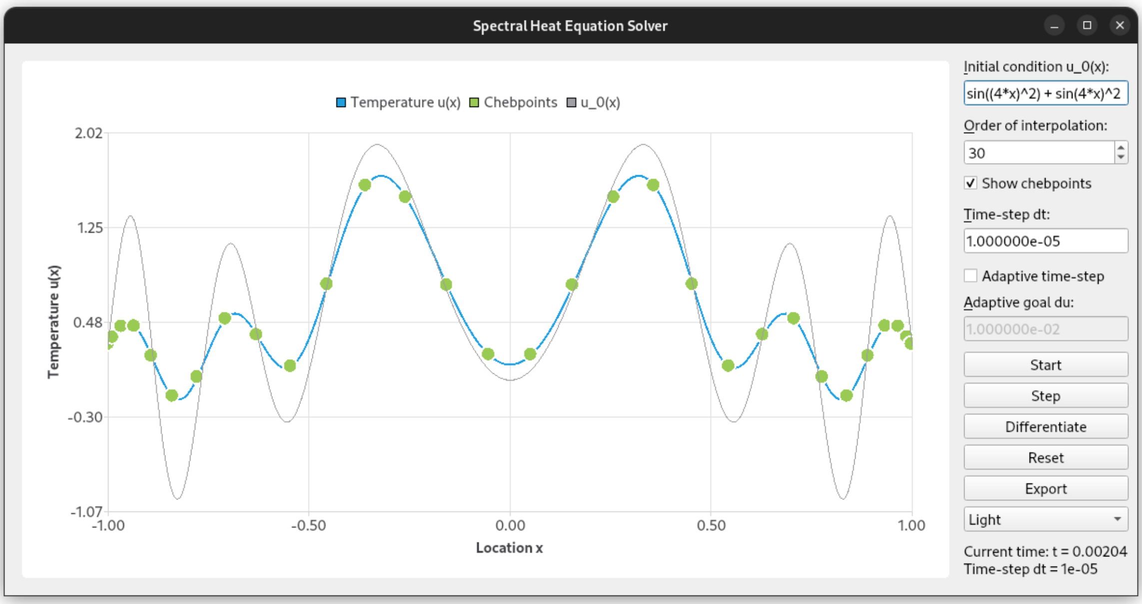

Figure 1: Screenshot of the graphical user interface. After entering an initial expression $u_0(x)$, depicted in grey, the simulation will run upon pressing 'Start'. The solution at time t, depicted in blue, is represented as a Chebyshev series of degree 29.

1 Motivation

Partial differential equations are notoriously hard to solve. One more possible approach to make way in this important class of problems is by the technique of spectral methods, incidentally closely related to finite element methods. The key idea is to perform the problem solution by representation of the occurring functions in a certain basis. For non-periodic problem settings, CHEBYSHEV series are a fantastic choice.

2 Chebyshev and why we like his polynomials

Let $\mathbb{N}$ denote the nonnegative integers, so $0 \in \mathbb{N}$.

2.1 Definition: Chebyshev polynomial

Chebyshev[1] polynomials $T_k : \mathbb{R} \mapsto \mathbb{R}$ are functions satisfying

$$T_k(x) = T_k(\cos\theta) := \cos(k\theta) = \frac{1}{2}(z^k + z^{-k})$$
$$z := \mathrm{e}^{i\theta}, \quad x := \mathfrak{Re}(z) = \cos(\theta) = \frac{1}{2}(z + z^{-1})$$

for degree $k \in \mathbb{N}$. Then, $T_0(x) = 1$, $T_1(x) = x$, $T_2(x) = 2x^2 - 1$, and so on.

These relations between x, z and θ reveal fundamental connections between three famous basis sets (as we will confirm later): CHEBYSHEV, LAURENT and FOURIER.

2.1 Theorem: Chebyshev Recursion Formula

The Chebyshev polyomials satisfy the three-term recurrence relation

$$T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x)\,.$$

[1] named after Pafnuty Lvovich CHEBYSHEV, alternatively transliterated as Tchebycheff, Tchebyshev (French) or TSCHEBYSCHOW (German)

Proof. Theorem 2.1 For $k > 1$,

$$
\begin{aligned}
2xT_k(x) - T_{k-1}(x) &= 2x\frac{1}{2}(z^k + z^{-k}) - \frac{1}{2}(z^{k-1} + z^{-(k-1)}) \\
&= 2\frac{1}{2}(z + z^{-1})\frac{1}{2}(z^k + z^{-k}) - \frac{1}{2}(z^{k-1} + z^{-k+1}) \\
&= \frac{1}{2}(z^{k+1} + z^{k-1} + z^{-k+1} + z^{-k-1}) - \frac{1}{2}(z^{k-1} + z^{-k+1}) \\
&= \frac{1}{2}(z^{k+1} + z^{-(k+1)}) = T_{k+1}(x)
\end{aligned}
$$

□

The Chebyshev polynomials also satisfy an *orthogonality relation*,

$$
\langle T_m, T_n \rangle := \int_{-1}^{1} T_m(x)T_n(x)\frac{1}{\sqrt{1-x^2}}\mathrm{d}x = \int_{\pi}^{0} \cos(m\theta)\cos(n\theta)\frac{-\sin(\theta)}{\sqrt{1-\cos^2(\theta)}}\mathrm{d}\theta\,,
$$

which becomes, with the fitting substitution $x = \cos(\theta)$ and $\mathrm{d}x = -\sin(\theta)\mathrm{d}\theta$,

$$
\begin{aligned}
\langle T_m, T_n \rangle &= \int_0^{\pi} T_m(\cos\theta)T_n(\cos\theta)\frac{\sin\theta}{\sin\theta}\mathrm{d}\theta = \int_0^{\pi} \cos(m\theta)\cos(n\theta)\mathrm{d}\theta \\
&= \frac{1}{2}\int_0^{\pi} \Big(\underbrace{\cos((m+n)\theta)}_{=\cos(2m\theta)\text{ for } m=n} + \underbrace{\cos((m-n)\theta)}_{=1 \text{ for } m=n} \Big)\mathrm{d}\theta
\end{aligned}
$$

along with the knowledge that $\int_0^{\pi} \cos(k\theta)\mathrm{d}\theta = k^{-1}\left[\sin(k\theta)\right]_0^{\pi} = 0$ for $k \in \mathbb{Z}\backslash\{0\}$,

$$
\langle T_m, T_n \rangle = \int_0^{\pi} T_m(\cos\theta)T_n(\cos\theta)\mathrm{d}\theta = \begin{cases} 0 & \text{for } m \neq n \\ \pi/2 & \text{for } m = n \neq 0 \\ \pi & \text{for } m = n = 0 \end{cases}
$$

which can be effectively utilised to define a function space $(\mathcal{T}, +, \cdot)$ in the *orthogonal* basis of Chebyshev polynomials $\mathcal{T} := \{T_k\}_{k\in\mathbb{N}}$. Note that the operation $\langle\cdot,\cdot\rangle$ satisfies all axioms of an authentic inner product (linearity, etc.) over a function space due to the linearity of the integral.

In the following proceedings, we will restrict our view on functions over the interval $[-1, 1] \subset \mathbb{R}$. Any (real) Lipschitz-continuous function $f \in \mathcal{C}_L$, where $\mathcal{C}_L := \{g : [-1,1] \mapsto \mathbb{R} \mid \exists L \text{ s.t. } \forall x_1, x_2 \in \mathbb{R},\ |g(x_1) - g(x_2)| \leq L \cdot |x_1 - x_2|\}$ can be represented in the Chebyshev basis $\mathcal{T}$, as Lipschitz continuity is a sufficient condition for absolute and uniform convergence of the corresponding series representation

$$
f(x) = \sum_{k=0}^{\infty} a_k T_k(x), \quad a_k \in \mathbb{R}, \quad k \in \mathbb{N}\,.
$$

Utilising orthogonality, for any $f \in \mathcal{C}_L$, we find coefficients $a_l \in \mathbb{R}$ by 'right-multiplying' the equation $f = \sum_{k=0}^{\infty} a_k T_k$ with any one of the Chebyshev polynomials T_l.

$$\begin{aligned}
\langle f, T_l \rangle = \langle \sum_{k=0}^{\infty} a_k T_k, T_l \rangle &= \int_0^{\pi} a_k T_k(\cos\theta) T_l(\cos\theta) \mathrm{d}\theta \\
&= \sum_{k=0}^{\infty} a_k \langle T_k, T_l \rangle \qquad \text{by linearity} \\
&= \begin{cases} a_0 \pi & \text{for } l = 0 \\ a_l \pi/2 & \text{for } l \neq 0 \end{cases}
\end{aligned}$$

which can easily be rearranged to give explicit relations for a_0 and a_k

$$\begin{aligned}
a_0 &= \frac{1}{\pi}\langle f, T_0 \rangle = \frac{1}{\pi}\int_0^{\pi} f(\cos\theta) \mathrm{d}\theta \\
a_k &= \frac{2}{\pi}\langle f, T_k \rangle = \frac{2}{\pi}\int_0^{\pi} f(\cos\theta)\cos(k\theta) \mathrm{d}\theta, \quad k \neq 0\,.
\end{aligned}$$

Dealing with a numerical problem, we shall approximate the above two integrals by the rectangular integral rule. A different approach for the derivation of the explicit coefficient integrals can be found in Trefethen 2019 along with a complex analysis styled proof.

As computers rarely allow us to store infinitely many coefficients a_k, we will work with the truncated Chebyshev series

$$f_N(x) = \sum_{k=0}^{N-1} a_k T_k(x), \quad k \in \{0, \ldots, N\}, \quad N \in \mathbb{N}, N > 1$$

which approximates the function with a degree $N-1$ polynomial up to

$$f(x) - f_N(x) = \sum_{k=0}^{\infty} a_k T_k(x) - \sum_{k=0}^{N-1} a_k T_k(x) = \sum_{k=N}^{\infty} a_k T_k(x)\,.$$

2.2 Theorem: Rectangular integral rule

$$\int_a^b f(x)\mathrm{d}x = \lim_{N\to\infty} \frac{b-a}{N} \sum_{k=0}^{N} f(x_k), \quad x_k := a + \frac{b-a}{N} k$$

Bonthuis and Schachinger 2021, p. 128

```cpp
TschebFun TschebFun::interpolantThrough(Vector y) {
  int order = y.size(), degree = order - 1;
  Vector j = (xt::linspace((double)degree, 0.0, order) + 0.5) * (pi /
   ↪ order);
  Vector coeffs = xt::zeros_like(y); // as many coefficients as data
   ↪ points
  coeffs[0] = xt::sum(y)() / order;
  for (size_t k = 1; k < order; k++)
    coeffs[k] = (2.0 / order) * xt::sum(y * xt::cos(j * k))();
  return TschebFun(coeffs);
}
```

Most importantly, this quadrature-style integral approximation is only one way of numerically determining the coefficients a_k. Another is to recognise the structure of the above integral for $k \neq 0$ as a cosine transform of the function $(f \circ \cos)$.

2.2 Definition: Cosine Transform

2.3 Definition: Discrete Cosine Transform

Most significantly, this approach via the Discrete Cosine Transform can be sped up by means of the *Fast Fourier Transform* (Cooley and Tukey 1965).

Numerically speaking, a significant improvement to these two approaches can be made by using the *Barycentric interpolation formula in Chebyshev points* (Trefethen 2019). Given more time, one should implement this feature in TschebFun as well.

2.4 Definition: Chebyshev points

From the equispaced points

$$\Theta_N := \{\theta_j := jN/\pi \mid j = 0, ..., N\},$$

we can further define the Chebyshev points as the corresponding $\cos(\theta_j)$,

$$X_N := \{x_j := \cos(\theta_j) \mid \theta_j \in \Theta_N\}.$$

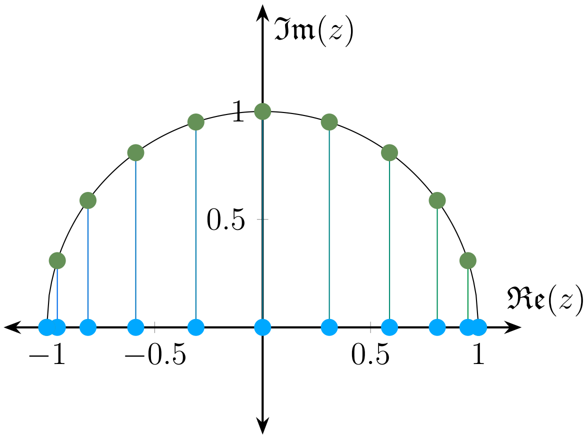

Figure 2: The Chebyshev points $\{x_j = \cos(\theta_j)\}$ are projections of the equispaced points $\{\theta_j\}$ on the unit circle onto the x-axis.

```cpp
#pragma once

#include <xtensor/xarray.hpp>
#include <xtensor/xindex_view.hpp>
#include <xtensor/xview.hpp>

typedef xt::xarray<double> Vector;

class TschebFun {
 public:
  xt::xarray<double> coefficients;

 public:
  TschebFun(Vector coeffs);
  size_t order() { return coefficients.size(); };
  size_t degree() { return coefficients.size() - 1; };
  Vector evaluateOn(Vector x);
  TschebFun derivative();
  TschebFun operator+(const TschebFun &other);
  TschebFun operator-(const TschebFun &other);
  TschebFun operator*(const double &factor);
  static TschebFun interpolantThrough(Vector y);
  static Vector chebpoints(size_t N);
  static Vector modifiedChebpoints(size_t N);
};
```

3 The spectral method

Differentiation matrix D_N according to Trefethen 2000.

$$\boldsymbol{a}^{(t+dt)} = \boldsymbol{a}^{(t)} - \alpha \Delta t \cdot D_N^2 \boldsymbol{a}^{(t)}$$

3.1 Enforcing boundary conditions

One way of forcing the boundary conditions, at least the first that came to my mind when thinking of this issue, is to pin down the two highest-order coefficients in the series representation after the iteration.

Let $l := u_0(-1)$, $r := u_0(1)$. Recognise that

$$T_k(-1) = T_k(\cos\pi) = \cos(k\pi) = (-1)^k$$
$$T_k(1) = T_k(\cos 0) = \cos(k0) = 1$$

which leads to

$$\begin{aligned}
u(-1,t) = \sum_{k=0}^{N-1} a_k^{(t)} T_k(-1) \quad &= \overbrace{\sum_{k=0}^{N-3} a_k^{(t)} (-1)^k}^{:=\Sigma_1} \quad +(-1)^{N-2} a_{N-2} \quad +(-1)^{N-1} a_{N-1} \quad = l \\
u(1,t) = \sum_{k=0}^{N-1} a_k^{(t)} T_k(1) \quad &= \underbrace{\sum_{k=0}^{N-3} a_k^{(t)}}_{:=\Sigma_2} \quad +a_{N-2} \quad +a_{N-1} \quad = r
\end{aligned}$$

By adding up the above two equations, one obtains

$$\Sigma_1 + \Sigma_2 + \underbrace{\left((-1)^{N-2} + 1\right)}_{\in\{0,2\}} a_{N-2} + \underbrace{\left((-1)^{N-1} + 1\right)}_{\in\{0,2\}} a_{N-1} = l + r \tag{1}$$

For N even: Equation 1 has one unkown $a_{N-2} = \frac{l+r-\Sigma_1-\Sigma_2}{2}$, $a_{N-1} = r - a_{N-2} - \Sigma_2$.

For N odd: Equation 1 has one unkown $a_{N-1} = \frac{l+r-\Sigma_1-\Sigma_2}{2}$, $a_{N-2} = r - a_{N-1} - \Sigma_2$.

3.2 Clenshaw Algorithm

3.1 Theorem: Clenshaw recurrence relation

(Press et al. 1987, pp. 172–178).

```
Vector TschebFun::evaluateOn(Vector x) {
  Vector U_kp2;
  Vector U_kp1 = xt::zeros_like(x);
  Vector U_k = xt::ones_like(x) * coefficients[coefficients.size() -
↪   1];
  for (int k = coefficients.size() - 2; k >= 0; k--) {
    U_kp2 = U_kp1;
    U_kp1 = U_k;
    U_k = 2.0 * x * U_kp1 - U_kp2 + coefficients[k];
  }
  return (U_k - U_kp2 + coefficients[0]) / 2.0;
}
```

4 The heat equation and its solution

5 Differentiation

```
TschebFun TschebFun::derivative() {
  int n = coefficients.size();
  n = n - 1; // differentiation reduces the degree (order) by 1
  Vector coeffs = coefficients; // make a copy
  Vector derivative = xt::zeros<double>({n});
  for (size_t j = n; j > 2; j--) {
    derivative[j - 1] = (2 * j) * coeffs[j];
    coeffs[j - 2] += (j * coeffs[j]) / (j - 2);
  }
  if (n > 1)
    derivative[1] = 4 * coeffs[2];
  derivative[0] = coeffs[1];
  return TschebFun(derivative);
}
```

5.1 Extension to a Python module

Using `pybind11`.

```python
import heatfun
u0 = lambda x: np.exp(-(x**2) * 12)
x_to_plot = np.linspace(-1.0, 1.0, 500)
cheb_x = heatfun.modifiedChebpoints(30)
solution = heatfun.solve(u0(cheb_x), 0.01, x_to_plot)
```

6 Results

```matlab
% example adapted from 'Exploring ODEs', page 282
u0 = chebfun('exp(-x^2 * 12)');
pdefun = @(t, x, u) diff(u, 2);
bc.left = @(t, u) u;
bc.right = @(t, u) u;
opts = pdeset('plot', 'off');
[t, u] = pde15s(pdefun, [0 0.005 0.010], u0, bc, opts);

x = linspace(-1.0, 1.0, 500).';
all_outputs = u(x);
output = all_outputs(:, end);
dlmwrite('matlab.csv', output, 'precision', '%.16f')
```

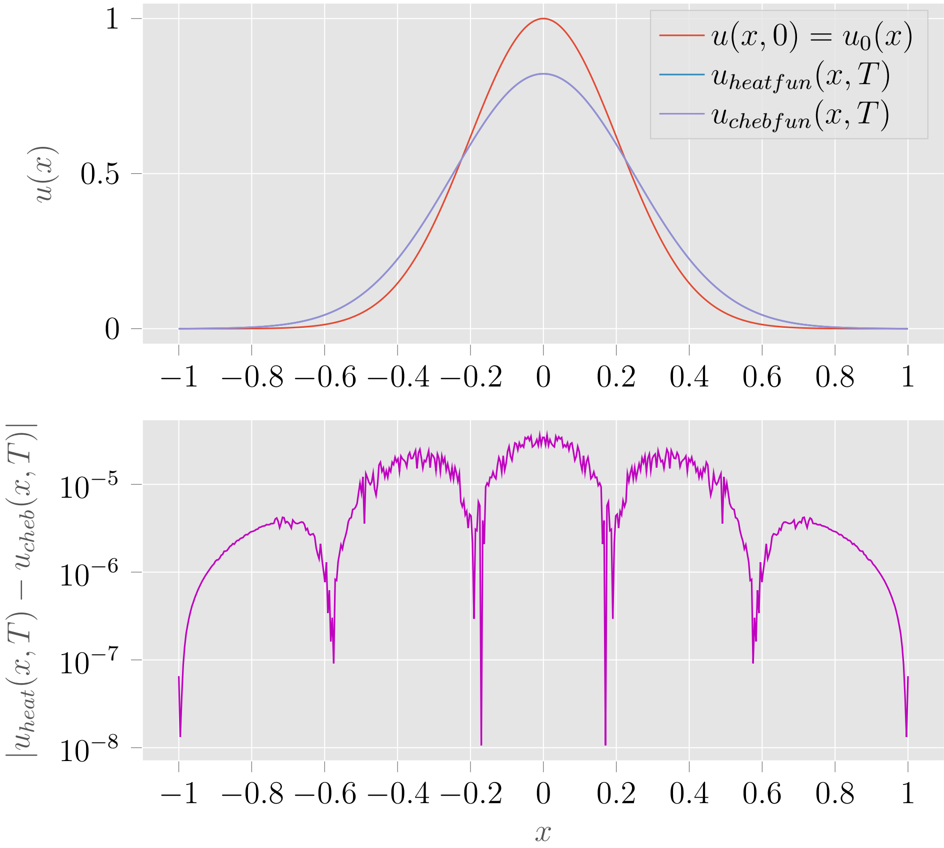

Figure 3: Comparison of heatfun and chebfun

7 Discussion

References

Bonthuis, Douwe and Ewald Schachinger (2021). *Lecture Notes in Computational Physics*. Institute of Theoretical and Computational Physics, Technical University of Graz.

Cooley, James W. and John W. Tukey (1965). 'An algorithm for the machine calculation of complex Fourier series'. In: *Mathematics of Computation* 19, pp. 297–301. DOI: 10.2307/2003354.

Press, William H., Saul A. Teukolsky, William T. Vetterling and Brian P. Flannery (1987). 'Numerical Recipes in FORTRAN - The Art of Scientific Computing, 2nd Edition'. In.

Trefethen, Lloyd N. (2000). *Spectral Methods in MATLAB*. Society for Industrial and Applied Mathematics. DOI: 10.1137/1.9780898719598. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9780898719598.

– (2019). *Approximation Theory and Approximation Practice, Extended Edition*. Philadelphia, PA: Society for Industrial and Applied Mathematics. DOI: 10.1137/1.9781611975949. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611975949.

A Title of Appendix

Appendices are definitely not necessary and assessors are not obliged to read them so only use them for non-vital text, figures or calculations.