ETH Lectures 401-0663-00L Numerical Methods for Computer Science
401-2673-00L Numerical Methods for CSE

# Homework Problems and Projects

Prof. R. Hiptmair, SAM, ETH Zurich

Autumn Term 2021, Version of October 2, 2022
(C) Seminar für Angewandte Mathematik, ETH Zürich

**Link** to the current version of this homework collection

## 0.1 General Information

### 0.1.1 Weekly Homework Assignments

All problems will be published in this single ".pdf" file. Every week, we publish a list of problem numbers on the lecture Moodle page. These are the current assignments for that week. Details are given on the lecture Moodle page.

### 0.1.2 Importance of Homework

Homework assignments are **not** mandatory. However, it is **very important** that you constantly exercise with the material you learn. "Solving" the homework assignments one week before the main exam by looking at the solutions will likely result in failure.

We provide hints and solutions from the beginning. It is your responsibility to look at the solutions only if you are stuck with a difficult problem and you tried to find a solution for a sufficient amount of time.

Most homework problems come with an estimate for the time required for its solution. These times are what we expect a bright and well-prepared student to require for that problem during an exam.

### 0.1.3 Corrections and Grading of Assignments

Homework assignments will not be graded.

However, you may submit your handwritten solutions by handing them to your tutor directly or uploading them via the Moodle upload interface, see the lecture Moodle page for instructions. The assistants will

# Chapter 6

# Approximation of Functions in 1D

**Problem 6-1: Piecewise linear approximation on graded meshes**

One of the main point made in [Lecture → Section 6.2.3] is that the quality of an interpolant depends heavily on the choice of the interpolation nodes. If the function to be interpolated has a "bad behavior" in a small part of the domain, for instance it has very large derivatives of high order, more interpolation points are required in that area of the domain. Commonly used tools to cope with this task are *graded meshes*, which will be the topic of this problem.

Substantial implementation in C++ is requested.

Given a mesh $\mathcal{M} = \{0 = t_0 < t_1 < \cdots < t_{n-1} < t_n = 1\}$ on the unit interval $I = [0,1]$, $n \in \mathbb{N}$, we define the *piecewise linear interpolant*:

$$\mathsf{I}_{\mathcal{M}} : C^0(I) \to \mathcal{P}_{1,\mathcal{M}} = \{s \in C^0(I),\ s_{|[t_{j-1},t_j]} \in \mathcal{P}_1 \ \forall\, j\}, \quad \text{s.t.} \quad (\mathsf{I}_{\mathcal{M}} f)(t_j) = f(t_j), \quad j = 0,\ldots,n \tag{6.1.1}$$

See also [Lecture → Section 5.3.2].

**(6-1.a)** ⚀ (10 min.) If we choose the uniform mesh $\mathcal{M} = \{t_j\}_{j=0}^n$ with $t_j = j/n$, given a function $f \in C^2(I)$ what is the asymptotic behavior of the error $\|f - \mathsf{I}_{\mathcal{M}} f\|_{L^\infty(I)}$ when $n \to \infty$?

HINT 1 for (6-1.a): Look for a suitable estimate in [Lecture → Section 6.6.1]. ⌟

---

SOLUTION of (6-1.a):

Equation [Lecture → Eq. (6.6.1.9)] says (we write $f'' := f^{(2)}$ for the second derivative):

$$\|f - \mathsf{I}_{\mathcal{M}} f\|_{L^\infty(I)} \le \frac{1}{2n^2} \|f''\|_{L^\infty(I)}\,, \tag{6.1.2}$$

because the meshwidth is $h = 1/n$. Thus, the convergence is quadratic, i.e. algebraic with order 2:

$$\|f - \mathsf{I}_{\mathcal{M}} f\|_{L^\infty(I)} = O(n^{-2}) \quad \text{for} \quad n \to \infty\,. \tag{6.1.3}$$

---

▲

**(6-1.b)** ⚀ (5 min.) What is the smoothness class of the function $f : I := [0,1] \to \mathbb{R}$, $f(t) = t^\alpha$, $0 < \alpha < 2$? In other words, for which $k \in \mathbb{N}$ do we have $f \in C^k(I)$?

HINT 1 for (6-1.b): Notice that $I$ is a closed interval and check the continuity of the derivatives in the endpoints of $I$. ⌟

---

SOLUTION of (6-1.b):

If $\alpha = 1$, $f(t) = t$ clearly belongs to $C^\infty(I)$. If $0 < \alpha < 1$, $f'(t) = \alpha t^{\alpha-1}$ blows up to infinity for $t$ going to $0$, therefore $f \in C^0(I) \setminus C^1(I)$. If $1 < \alpha < 2$, $f'$ is continuous but $f''(t) = \alpha(\alpha-1)t^{\alpha-2}$ blows up to infinity for $t$ going to $0$, therefore $f \in C^1(I) \setminus C^2(I)$.

More generally, for $\alpha \in \mathbb{N}$ we have $f(t) = t^\alpha \in C^\infty(I)$; on the other hand, if $\alpha > 0$ is not an integer, $f \in C^{\lfloor \alpha \rfloor}(I)$, where $\lfloor \alpha \rfloor = \mathrm{floor}(\alpha)$ is the largest integer not larger than $\alpha$.

---

▲

**(6-1.c)** ⚀ (30 min.) Implement a templated C++ function

```
template <typename FUNCTION>
double pwlintpMaxError(FUNCTION &&f, const Eigen::VectorXd &t);
```

that computes $\|f - I_{\mathcal{M}} f\|_{L^\infty(I)}$ when $f$ is passed through the functor `f` (with evaluation operator **double operator** `()(double)const`) and the mesh $\mathcal{M}$ through its *sorted* vector $[t_0, t_1, \ldots, t_n]^\top \in \mathbb{R}^{n+1}$, $n \in \mathbb{N}$, of node positions, which also defines the interval $I := [t_0, t_n]$.

The maximum norm should be approximated by sampling in $10^5$ equidistant points in $I$, see, e.g., [Lecture → Ex. 6.6.1.7].

---

SOLUTION of (6-1.c):

For efficient implementation it is essential to make use of the fact that the the nodes passed in `t` are sorted.

**C++ code 6.1.4: Implementation of `pwlintpMaxError()`**

```
2  template <typename FUNCTION>
3  double pwlintpMaxError(FUNCTION &&f, const Eigen::VectorXd &t) {
4    constexpr unsigned int N =
5        1e5;                // Sampling resolution for approximating L-infinty norm.
6    double maxerr = 0.0;  // L-infinity error so far.
7
8    // TODO: (6-1.c) Approximate the maximum norm of (f - If) on [0,1],
9    // where If is the piecewise linear interpolation of f on the mesh t.
10   // START
11   // We iterate over the sampling points x = 0, 1/N, 2/N, ... 1.
12   // For each x we need to interpolate f in the correct mesh interval.
13   // We are currently in the interval [tj, tj1] = [0,t(1)].
14   unsigned int j = 0;
15   double tj = t[j], tj1 = t[j + 1];
16   double fj = f(tj), fj1 = f(tj1);
17   for (unsigned int i = 0; i <= N; ++i) {
18     // We are currently in the mesh interval [tj, tj1].
19     double x = (1.0 * i) / N;
20     // If x is outside the current mesh interval,
```

```
    // then find the interval containing x.
    while (x > tj1) {  // x >= tj always satisfied.
      tj = tj1;
      fj = fj1;
      j++;
      tj1 = t[j + 1];
      fj1 = f(tj1);
    }
    // Now we have tj <= x <= tj1.

    // Evaluate linear interpolant [tj,tj1] at x.
    double lfx = fj + (fj1 - fj) / (tj1 - tj) * (x - tj);
    // Update L-infinity error on [0,x]
    double err = std::abs(f(x) - lfx);
    if (err > maxerr) {
      maxerr = err;
    }
  }
  // END
  return maxerr;
}
```

Get it on GitLab (gradedmeshes.hpp).

▲

**(6-1.d)** ⚂ (15 min.) [ depends on Sub-problem (6-1.c) ]

We consider the root functions $f(t) := t^\alpha$, $\alpha > 0$. Implement a C++ function

```
void cvgplotEquidistantMesh(const Eigen::VectorXd &alpha);
```

that uses MATPLOTLIBCPP to create log-log plots of the the error norms $\|f - I_{\mathcal{M}} f\|_{L^\infty(I)}$ as a function of $n := \sharp\mathcal{M} - 1 \to \infty$ for families of *equidistant meshes*. Use meshes with $n = 2^k$, $k = 5, 6, \ldots, 12$, and values for $\alpha$ passed in `alpha`. Do not forget axis labels, plot title, and a meaningful legend. Save the plot in `cvgplotequidistand.png`.

Describe your observations qualitatively.

SOLUTION of (6-1.d):

**C++ code 6.1.5: Implementation of `cvgplotEquidistantMesh()`**

```
void cvgplotEquidistantMesh(const Eigen::VectorXd &alpha) {
  const unsigned int n_alphas = alpha.size();
  plt::figure();
  // TODO: (6-1.d) Create log-log plots of the maximum norm
  // errors (obtained by pwlintMaxError()) with
  // number of mesh intervals 32, 64, ..., 4096.
  // START
  constexpr unsigned int kmin = 5, kmax = 12;
  const unsigned int n_meshes = kmax - kmin + 1;
  // n=2^kmin, 2^(kmin+1), ..., 2^kmax.
  Eigen::ArrayXd N =
      Eigen::pow(2, Eigen::ArrayXd::LinSpaced(n_meshes, kmin, kmax));

  // Errors(i,j) = L-infinity error of interpolation of t^alpha(j)
```

```
16   // using a mesh M with #M = N(i)+1.
17   Eigen::MatrixXd Errors(n_meshes, n_alphas);
18   for (unsigned int i = 0; i < n_meshes; ++i) {
19     const unsigned int n = N(i);  // The mesh has #M = (n+1) = (N(i)+1) nodes.
20     Eigen::VectorXd mesh = Eigen::VectorXd::LinSpaced(n + 1, 0.0, 1.0);
21     for (unsigned int j = 0; j < n_alphas; ++j) {
22       // Compute L-infinity error for f_j(t) = t^alpha(j).
23       const double alphaj = alpha(j);
24       auto fj = [alphaj](double t) { return std::pow(t, alphaj); };
25       Errors(i, j) = pwlintpMaxError<std::function<double(double)>>(fj, mesh);
26     }
27   }
28
29   // Define a few line-styles to distinguish plots.
30   // Colours change automatically.
31   constexpr unsigned int n_styles = 4;
32   std::vector<std::string> style;
33   style.push_back("o--");
34   style.push_back("v-.");
35   style.push_back("s:");
36   style.push_back("D-");
37   // Plot errors
38   for (unsigned int j = 0; j < n_alphas; j++) {
39     char lab[20];
40     sprintf(lab, "alpha = %.2f", alpha(j));
41     plt::loglog(N, Errors.col(j), style[j % n_styles], {{"label", lab}});
42   }
43   plt::title("Error of pw. lin. intp. of f(t)=t^alpha on uniform meshes");
44   plt::ylabel("Error");
45   plt::xlabel("Number of mesh intervals");
46   plt::legend("lower left", std::vector<double>());
47   // END
48   plt::savefig("./cx_out/cvgplotequidistant.png");
49 }
```

Get it on GitLab (gradedmeshes.hpp).

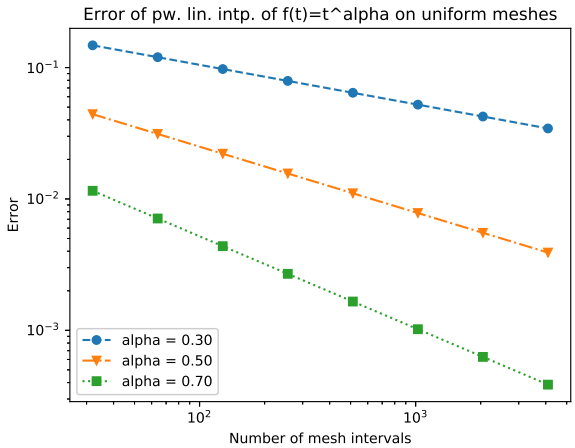



Fig. 51

◁ The data points $(n, \mathrm{errornorm(n)})$ almost exactly lie on straight lines in the doubly logaritmic plot, *cf.* [Lecture → § 6.2.2.9].

algebraic convergence in $n^{-1}$

We conjecture an asymptotic behavior $\|f - I_{\mathcal{M}} f\|_{L^\infty(I)} = O(n^{-\beta})$ with $\beta = \beta(\alpha)$ an increasing function of $\alpha$, because $\beta$ can be read off the slope of the lines in the plot.

▲

**(6-1.e)** (30 min.) [ depends on Sub-problem (6-1.c) ]

Create a C++ function

```
Eigen::VectorXd cvgrateEquidistantMesh(const Eigen::VectorXd
```

```
    &alpha);
```

that estimates rates of asymptotic algebraic convergence (→ [Lecture → § 6.2.2.9]) of $\|f - \mathrm{I}_{\mathcal{M}} f\|_{L^\infty(I)}$ for $f(t) := t^\alpha$ in terms of $n := \sharp\mathcal{M} - 1 \to \infty$ for families of *equidistant meshes* for the passed values of the parameter $\alpha$ and returns the estimates. These regression-based estimates should rely on meshes with $n = 2^k$, $k = 5, 6, \ldots, 12$.

HINT 1 for (6-1.e): Determine the rate of algebraic convergence using 1D linear regression, see which is implemented, as a special case, in the auxiliary function `polyfit`, see [Lecture → Code 5.7.0.21].

SOLUTION of (6-1.e):

**C++ code 6.1.6: Implementation of `cvgrateEquidistantMesh()`**

```
2  Eigen::VectorXd cvgrateEquidistantMesh(const Eigen::VectorXd &alpha) {
3    const unsigned int n_alphas = alpha.size();
4    Eigen::VectorXd Rates(n_alphas);
5
6    // TODO: (6-1.e) Fill in the entries of Rates.
7    // Hint: For each alpha(j), use polyfit() to estimate convergence
8    // rates of the maximum norm (obtained by pwlintMaxError()) with
9    // number of mesh intervals 32, 64, ..., 4096.
10   // START
11   constexpr unsigned int kmin = 5, kmax = 12;
12   const unsigned int n_meshes = kmax - kmin + 1;
13   // n=2^kmin, 2^(kmin+1), ..., 2^kmax.
14   Eigen::ArrayXd N =
15       Eigen::pow(2, Eigen::ArrayXd::LinSpaced(n_meshes, kmin, kmax));
16
17   // Errors(i,j) = L-infinity error of interpolation of t^alpha(j)
18   // using a mesh M with #M = N(i)+1.
19   Eigen::MatrixXd Errors(n_meshes, n_alphas);
20   for (unsigned int i = 0; i < n_meshes; ++i) {
21     const unsigned int n = N(i);  // The mesh has #M = (n+1) = (N(i)+1) nodes.
22     Eigen::VectorXd mesh = Eigen::VectorXd::LinSpaced(n + 1, 0.0, 1.0);
23     for (unsigned int j = 0; j < n_alphas; ++j) {
24       // Compute L-infinity error for f_j(t) = t^alpha(j).
25       const double alphaj = alpha(j);
26       auto fj = [alphaj](double t) { return std::pow(t, alphaj); };
27       Errors(i, j) = pwlintpMaxError<std::function<double(double)>>(fj, mesh);
28     }
29   }
30
31   for (unsigned int j = 0; j < n_alphas; ++j) {
32     // Estimate convergence rate for f_j(t) = t^alpha(j) by the slope
33     // of line fitted through the errors on a log-log scale.
34     Eigen::VectorXd coeff =
35         polyfit(N.array().log(), Errors.col(j).array().log(), 1);
36     Rates(j) = -coeff[1];
37   }
38   // END
39   return Rates;
40 }
```

Get it on GitLab (gradedmeshes.hpp).

▲

**(6-1.f)** ⚂ (15 min.) Tabulate and plot the empiric rate of asymptotic $n$-convergence (in maximum norm) of the piecewise linear interpolant of $f(t) = t^\alpha$, $0 < \alpha < 2$, on *equidistant meshes*. To conduct this experiment implement a C++ function

```
void testcvgEquidistantMesh(void);
```

that plots (use MATPLOTLIBCPP) and tabulates the measured convergence rates based on $\alpha = 0.05, 0.15, 0.25, \ldots, 2.95$. Save the plot in the file `cvgRateEquidistant.png`.

---

SOLUTION of (6-1.f):

**C++ code 6.1.7: Implementation of `testcvgEquidistantMesh()`**

```
2  void testcvgEquidistantMesh() {
3    plt::figure();
4    // TODO: (6-1.f) Plot and tabulate the convergence rates of pw.
5    // lin. intp. of t^α using equidistant meshes on [0,1],
6    // for alpha = 0.05, 0.15, 0.25 ..., 2.95.
7    // START
8    constexpr unsigned int n_alphas = 30;
9    Eigen::VectorXd alpha = Eigen::VectorXd::LinSpaced(n_alphas, 0.05, 2.95);
10
11   Eigen::VectorXd ConvRates = cvgrateEquidistantMesh(alpha);
12   std::cout << std::setw(10) << "alpha" << std::setw(20) << "Convergence rate"
13             << std::endl;
14   for (unsigned int j = 0; j < n_alphas; ++j) {
15     std::cout << std::setw(10) << alpha(j) << std::setw(20) << ConvRates(j)
16               << std::endl;
17   }
18
19   plt::plot(alpha, ConvRates);
20   plt::xlabel("alpha");
21   plt::ylabel("conv. rate");
22   plt::title("Cvg. rates of pw. lin. intp. of f(t)=t^alpha on uniform meshes");
23   // END
24   plt::savefig("./cx_out/cvgRateEquidistant.png");
25 }
```

Get it on GitLab (gradedmeshes.hpp).

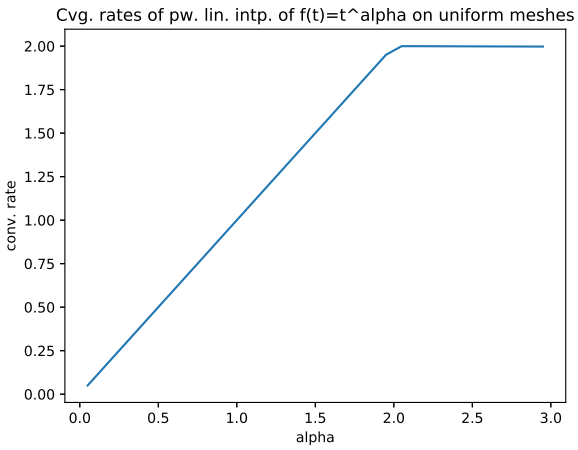



Fig. 52

The empiric rate of algebraic convergence is equal to $\alpha$, if $\alpha$ is smaller than $2$, and equal to $2$ otherwise. In brief, we can say that the rate is $\min\{\alpha, 2\}$.
Be careful with the case $\alpha = 1$: here the interpolant exactly agrees with the solution on any mesh. This is why we avoided integers when sampling values for $\alpha$.

---

▲

**(6-1.g)** ⚂ (5 min.) In which mesh interval do you expect $|f - \mathrm{I}_{\mathcal{M}} f|$ to attain its maximum?

HINT 1 for (6-1.g): Recall [Lecture → Thm. 6.2.2.15] and apply it in the piecewise linear setting. ⌟

---

SOLUTION of (6-1.g):

The error representation [Lecture → Eq. (6.5.2.27)] in the linear case ($n = 1$) for some $\tau_t \in (t_j, t_{j+1})$ reads as:

$$\forall\, t \in (t_j, t_{j+1}) \qquad \left| f(t) - \left(\mathrm{I}_{\mathcal{M}} f\right)(t) \right| = \frac{1}{2} |f''(\tau_t)\ (t - t_j)(t - t_{j+1})| \\ \leq \frac{1}{8} |f''(\tau_t)|\ (t_{j+1} - t_j)^2 = \frac{1}{8n^2} |f''(\tau_t)|$$

Therefore the error can be large only in the subintervals where the second derivative of $f$ is large. $|f''(t)| = \left|\alpha(\alpha-1)t^{\alpha-2}\right|$ is monotonically decreasing for $0 < \alpha < 2$: therefore, we can expect a large error in the first subinterval, the one that is closest to $0$.

---

▲

**(6-1.h)** ⚄ (20 min.) [ depends on Sub-problem (6-1.g) ]

Compute ("on paper") the exact value of $\|f - \mathrm{I}_{\mathcal{M}} f\|_{L^\infty(I)}$ for $f(t) := t^\alpha$, $0 < \alpha < 2$. Compare the order of convergence obtained with the one observed numerically in (6-1.b).

HINT 1 for (6-1.h): Use the insight gained in Sub-problem (6-1.g) to simplify the problem. ⌟

---

SOLUTION of (6-1.h):

From (6-1.g) we expect that the maximum appears in the first subinterval. Then, for $t \in ]0, 1/n[$ and $0 < \alpha < 2$ (with $\alpha \neq 1$), let us find this maximum by considering the error function $\varphi$:

$$\varphi(t) = f(t) - \left(\mathrm{I}_{\mathcal{M}} f\right)(t) = t^\alpha - t\,\frac{1}{n^{\alpha-1}} \quad , \quad (\varphi(0) = \varphi(1/n) = 0)\,,$$

$$\varphi'(t) = \alpha t^{\alpha-1} - \frac{1}{n^{\alpha-1}}\,,$$

$$\blacktriangleright \quad \varphi'(t^*) = 0 \quad \text{if} \quad t^* = \frac{1}{n}\alpha^{-1/(\alpha-1)}\,.$$

This means

$$\max_{t\in(0,1/n)} |\varphi(t)| = |\varphi(t^*)| = \left| \frac{\alpha^{-\alpha/(\alpha-1)}}{n^\alpha} - \frac{\alpha^{-1/(\alpha-1)}}{n^\alpha} \right| \\ = \frac{1}{n^\alpha} \left| \alpha^{-\alpha/(\alpha-1)} - \alpha^{-1/(\alpha-1)} \right| = \mathcal{O}(n^{-\alpha}) = \mathcal{O}(h^\alpha)\,.$$

The order of convergence in $h = 1/n$ is equal to the parameter $\alpha$, as we could have guessed from Fig. 52.

---

▲

**(6-1.i)** ⚂ (20 min.) Since the interpolation error is concentrated in the left part of the domain, it seems reasonable to use a finer mesh only in this part.

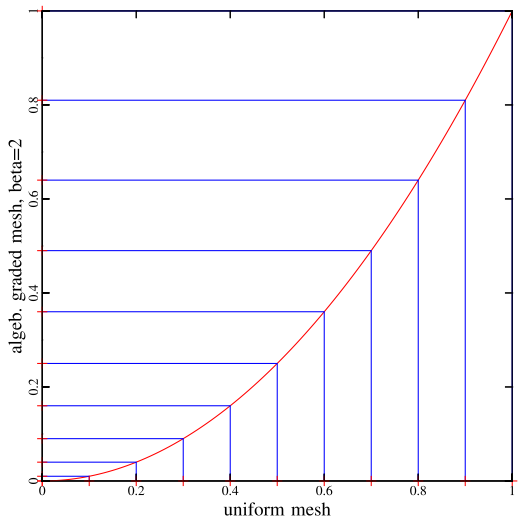



Fig. 53

A suitable choice is an **algebraically graded mesh**, defined as

$$\mathcal{G} = \left\{ t_j = \left(\frac{j}{n}\right)^{\beta}, j = 0, \ldots, n \right\} \tag{6.1.8}$$

for the **grading parameter** $\beta > 1$.

◁ Algebraically graded mesh for $\beta = 2$.

Realize a C++ function

```
Eigen::MatrixXd cvgrateGradedMesh(const Eigen::VectorXd &alpha,
                                  const Eigen::VectorXd &beta);
```

that estimates rates of asymptotic algebraic convergence (→ [Lecture → § 6.2.2.9]) of $\|f - \mathrm{I}_{\mathcal{G}} f\|_{L^\infty(I)}$ for $f(t) := t^\alpha$ in terms of $n := \sharp\mathcal{M} - 1 \to \infty$ for families of *graded meshes* for the passed values of the parameter $\alpha$ (argument `alpha`) and of the grading parameter $\beta > 0$ (argument `beta`). The function should return the measured rates as entries of a matrix, a column for each $\alpha$-value, a row for each $\beta$-value. The regression-based estimates should rely on meshes with $n = 2^k$, $k = 5, 6, \ldots, 12$.

---

SOLUTION of (6-1.i):

**C++ code 6.1.9: Implementation of `cvgrateGradedMesh()`**

```
2  Eigen::MatrixXd cvgrateGradedMesh(const Eigen::VectorXd &alpha,
3                                    const Eigen::VectorXd &beta) {
4    const unsigned int n_alphas = alpha.size();
5    const unsigned int n_betas = beta.size();
6    Eigen::MatrixXd Rates(n_betas, n_alphas);
7
8    // TODO: (6-1.i) Fill in the entries of Rates.
9    // START
10   constexpr unsigned int kmin = 5, kmax = 12;
11   const unsigned int n_meshes = kmax - kmin + 1;
12   // n=2^kmin, 2^(kmin+1), ..., 2^kmax.
13   Eigen::ArrayXd N =
14       Eigen::pow(2, Eigen::ArrayXd::LinSpaced(n_meshes, kmin, kmax));
15
16   // Errors(i,k*n_alphas+j) = L-infinity error of interpolation of t^alpha(j)
17   // using a mesh M with #M = N(i)+1 and grading parameter beta(k).
18   Eigen::MatrixXd Errors(n_meshes, n_alphas * n_betas);
19   for (unsigned int i = 0; i < n_meshes; ++i) {
20     const unsigned int n = N(i);  // The mesh has #M = (n+1) = (N(i)+1) nodes.
21     Eigen::VectorXd mesh = Eigen::VectorXd::LinSpaced(n + 1, 0.0, 1.0);
```

```
22      for (unsigned int k = 0; k < n_betas; ++k) {
23        // Transform the mesh using the grading parameter beta.
24        Eigen::VectorXd gradedMesh = pow(mesh.array(), beta(k)).matrix();
25        for (unsigned int j = 0; j < n_alphas; ++j) {
26          // Compute L-infinity error for f_j(t) = t^alpha(j).
27          const double alphaj = alpha(j);
28          auto fj = [alphaj](double t) { return std::pow(t, alphaj); };
29          const unsigned int ind = k * n_alphas + j;
30          Errors(i, ind) =
31              pwlintpMaxError<std::function<double(double)>>(fj, gradedMesh);
32        }
33      }
34    }
35
36    for (unsigned int k = 0; k < n_betas; ++k) {
37      for (unsigned int j = 0; j < n_alphas; ++j) {
38        // Estimate convergence rate for f_j(t) = t^alpha(j) by the slope
39        // of line fitted through the errors on a log-log scale.
40        const unsigned int ind = k * n_alphas + j;
41        Eigen::VectorXd coeff =
42            polyfit(N.array().log(), Errors.col(ind).array().log(), 1);
43        Rates(k, j) = -coeff[1];
44      }
45    }
46    // END
47    return Rates;
48  }
```

Get it on GitLab (gradedmeshes.hpp).

▲

**(6-1.j)** ⚄ (20 min.) [ depends on Sub-problem (6-1.i) ]

When using algebraically graded meshes for the approximation of $t \mapsto t^\alpha$, $0 < \alpha < 2$, by means of piecewise linear interpolation, how do you have to choose the grading parameter $\beta = \beta(\alpha)$ as a function of $\alpha$ in order to recover an asymptotic algebraic convergence like $O(n^{-2})$ of $\|f - \mathrm{I}_{\mathcal{G}} f\|_{L^\infty(I)}$ for $\sharp\mathcal{G} \sim n \to \infty$?

To arrive at a conjecture rely on the function `cvgrateGradedMesh()` from Sub-problem (6-1.i) and write another function

```
void testcvgGradedMesh(void);
```

that calls `cvgrateGradedMesh()` for

$$\alpha \in \{0.05, 0.15, 0.25, \ldots, 2.95\}\,,$$
$$\beta \in \{0.1, 0.2, 0.3, \ldots, 2.0\}\,,$$

and creates a useful plot of the data (unsing MATPLOTLIBCPP) and stores it in the file `alphabeta.png`.

SOLUTION of (6-1.j):

**C++ code 6.1.10: Implementation of `testcvgGradedMesh()`**

```
2  void testcvgGradedMesh() {
```

```
  plt::figure();
  // TODO: (6-1.j) Plot the convergence rates from
  // cvgrateGradedMesh() using alpha = 0.05, 0.15, ..., 2.95
  // and beta = 0.1, 0.2, ..., 2.0.
  // Note: Running this code may take a while, so start by
  // using fewer values for alpha and beta.
  // Hint: You can use plt::plot_surface(X,Y,Z) where X, Y, Z
  // are matrices that all have the same dimensions.
  // You can use x.replicate() to create X and Y
  // for an appropriate vector x.
  // START
  constexpr unsigned int n_alphas = 30;
  Eigen::VectorXd alpha = Eigen::VectorXd::LinSpaced(n_alphas, 0.05, 2.95);
  constexpr unsigned int n_betas = 20;
  Eigen::VectorXd beta = Eigen::VectorXd::LinSpaced(n_betas, 0.1, 2.0);

  // ConvRates(i,j) = convergence rate using beta(i) and alpha(j).
  Eigen::MatrixXd ConvRates = cvgrateGradedMesh(alpha, beta);
  // Create meshgrid of alpha and beta values for plotting,
  // such that alphaGrid(i,j) = alpha(j)
  // and betaGrid(i,j) = beta(i).
  Eigen::MatrixXd alphaGrid = alpha.replicate(1, n_betas).transpose();
  Eigen::MatrixXd betaGrid = beta.replicate(1, n_alphas);

  plt::plot_surface(alphaGrid, betaGrid, ConvRates);
  plt::xlabel("alpha");
  plt::ylabel("beta");
  plt::title("Cvg. rates of pw. lin. intp. of f(t)=t^alpha on graded meshes");
  // END
  plt::savefig("./cx_out/alphabeta.png");
}
```

Get it on GitLab (gradedmeshes.hpp).

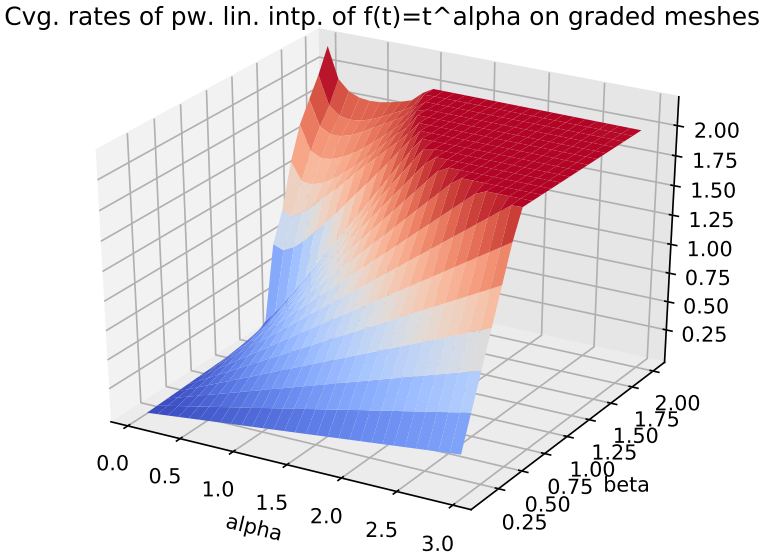



Fig. 54

The comparison of this plot with the analogous ones for different values of $\alpha$ suggests that the choice of $\beta = 2/\alpha$ guarantees quadratic convergence in $n^{-1}$.

Proceeding as in (6-1.g), we can see that the maximal error in the first subinterval $(0, t_1) = (0, 1/n^\beta)$ is equal to $1/n^{\alpha\beta}\,(\alpha^{-\alpha/(1-\alpha)} - \alpha^{-1/(1-\alpha)}) = \mathcal{O}(n^{-\alpha\beta})$ (replace the interval size $1/n$ with $1/n^\beta$ in those equations). This implies that a necessary condition to have quadratic convergence is $\beta \geq 2/\alpha$. In order

to find an upper bound on the optimal $\beta$, we should control the error committed in every subinterval. Here the exact computation of $\varphi(t^*)$ becomes very long and complicated.

---

▲

**End Problem 6-1** , 170 min.