NTNU – Trondheim
Norwegian University of
Science and Technology

The Empirical Interpolation Method

Stian Kristoffersen

Master of Science in Physics and Mathematics
Submission date: June 2013
Supervisor: Einar Rønquist, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Summary

In this thesis we look at the Empirical Interpolation Method (EIM) [15] and how it can be used in different applications. We propose a new formulation of EIM to make it easier to perform analytical operations like differentiation and integration of the basis functions as well as to apply EIM to a variety of problems. The new formulation is used to develop quadrature rules for the circle and semicircle, as well as for arbitrary simple polygons. The new formulation is also used to solve partial differential equations using a collocation approach on various domains including the circle, semicircle and triangle. The framework is briefly applied to compression of 3D animation in addition to recognition of images and sound.

Several of the methods show great potential, with exponential convergence for quadrature and collocation for regular problems. However, there are also serious issues that must be addressed if the methods are to be developed further. These issues are related to making the methods more robust and stable.

Sammendrag

I denne oppgaven ser vi på den empiriske interpolasjonsmetoden (EIM) [15] og hvordan den kan benyttes i forskjellige anvendelser. Vi presenterer en ny formulering av EIM for å gjøre det lettere å utføre analytiske operasjoner, som differensiering og integrasjon av basisfunksjonene, i tillegg til å bruke EIM i en rekke anvendelser. Den nye formuleringen brukes til å utvikle kvadraturregler for en sirkel og en halvsirkel, samt for vilkårlige enkle polygoner. Den nye formuleringen blir også brukt til å løse partielle differensiallikninger, via kollokasjon, på forskjellige domener inkludert en sirkel, en halvsirkel og en trekant. Rammeverket anvendes kort på komprimering av 3D animasjon i tillegg til gjenkjenning av bilder og lyd.

Flere av metodene virker lovende, med eksponentiell konvergens for kvadratur og kollokasjon for regulære problemer. Men det er også betydelige utfordringer som må løses hvis metodene skal utvikles videre. Disse utfordringene er relatert til å gjøre metodene mer robuste og stabile.

Preface

This thesis concludes my master's degree in Industrial Mathematics at the Norwegian University of Science and Technology (NTNU). The work was carried out during the spring of 2013.

I would like to thank my supervisor, Professor Einar M. Rønquist, for his encouragement, ideas and advice. I am grateful for our many discussions, including choosing the topic for this thesis, which I have found very exciting to work with.

Stian Kristoffersen
Trondheim
June 27, 2013

Contents

Chapter 1

Introduction

How we represent functions can greatly impact what we are able to do with them computationally. Some representations might leave us unable to perform basic operations like integration, differentiation or even just sampling the function in some point. There are for instance plenty of analytical integrals where an analytical solution is not known. By first representing the function in a way that allows us to perform basic operations we can more easily use the function in other applications like solving partial differential equations. Interpolation is a common approach to finding such a representation.

Interpolation, Figure 1.1, is a method to approximate a function on a domain by requiring that the approximation is exact in a finite set of points called interpolation points. The approximation is often a linear combination of a set of functions called basis functions. The basis functions are typically defined on the whole domain, making it possible to sample the approximation in the whole domain, rather than just in the interpolation points used to obtain the approximation. However, in order to ensure that we obtain a good approximation outside the interpolation points we must require that the function we wish to interpolate is analytical or sufficiently regular. This is the case if we use polynomials as basis functions. Then we can achieve exponentially decreasing error over the domain with the number of interpolations points used [4]. This, together with that polynomials are easily differentiated and integrated, makes polynomials a popular choice of basis functions.

Choosing good interpolation points is in general an open problem. The Gauss-Lobatto Legendre (GLL) points [4] have good properties. Interpolation is achieved by constructing polynomial basis functions through the GLL points. By integrating the basis functions, a numerical integration rule called GLL quadrature can be constructed. Furthermore, the GLL points, together with the polynomial basis functions, are used in spectral methods to solve partial differential equations. All three methods have exponential convergence given that the underlying functions

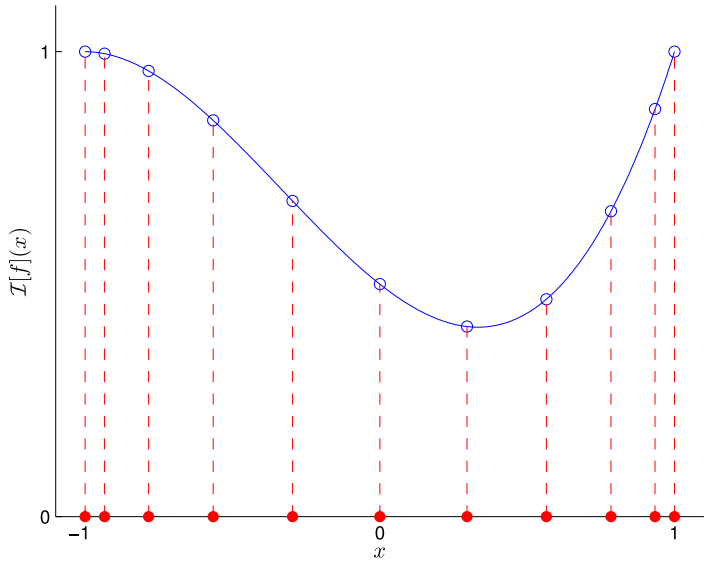

Figure 1.1: Interpolation (solid blue line) of a function f in the domain $\Omega = [-1, 1]$. The interpolation is required to be exact in the interpolation points (red dots), i.e. $\mathcal{I}[f](x_i) = f(x_i)$ for all interpolation points x_i, $i = 1, \ldots, M$.

are analytical. A serious limitation, however, is that the GLL points are only defined on the line, the square and other tensor product domains. If GLL points are to be used in general domains, the problem must be mapped from the actual domain over to a reference domain where the GLL points are defined. This can be tedious or impractical.

An alternative to using high order polynomials as basis functions is to use piecewise linear functions. The interpolation points might be equidistant or even arbitrary. The trapezoidal rule for numerical integration, and the linear finite element method to solve partial differential equations, can both be constructed in this way. While these methods are more flexible in terms of what domains they can handle, their convergence rates are algebraic (as low as $\mathcal{O}(M^{-2})$, using M sample points, for the trapezoidal rule [13]). Hence, traditional methods involves a trade-off between convergence rates and what domains they can easily handle.

The Empirical Interpolation Method (EIM) [15] was recently proposed to use as a general multipurpose interpolation procedure. The EIM constructs both the interpolation points, called magic points, as well as the accompanying basis functions. The EIM does not claim to be optimal, but it achieves exponential convergence rate for analytical functions, and it works on non-standard domains such as triangles and hexagons. Furthermore, the hierarchical nature of the EIM lets us add an interpolation point without having to recompute all the existing points. This is

in contrast with GLL, where all the interpolation points must be recomputed each time.

In [15] the authors suggested that EIM could be used for several applications including image and pattern recognition, numerical integration, and data compression. To our knowledge there have not been a lot of investigation into the feasibility of EIM for these applications. Given the good properties of EIM, it would be interesting to explore if EIM could solve the tradeoff between the convergence rate and domain flexibility in numerical integration and when solving partial differential equations.

In this thesis we will explore using the EIM to integrate functions numerically and to solve partial differential equations numerically on different domains. We will also look at the feasibility of applying the EIM to different problems including image and sound recognition as well as compression.

Chapter 2

The Empirical Interpolation Method

The EIM as it was presented in [15] is the cornerstone of this thesis. In this chapter we will present the EIM and some of its properties. In the next chapter we will present our own formulation of the EIM that will be convenient when applying the EIM to various applications in later chapters.

2.1 Introduction

Interpolation is used to approximate a function f on a domain Ω, $f : \Omega \to \mathbb{R}$ by requiring the approximation, $\mathcal{I}_M[f] : \Omega \to \mathbb{R}$, to be exact in a set of M interpolation points $\{\boldsymbol{x}_i\}_{i=1}^M, \boldsymbol{x}_i \in \Omega$

$$f(\boldsymbol{x}_i) = \mathcal{I}_M[f](\boldsymbol{x}_i) \qquad i = 1, \ldots, M. \tag{2.1}$$

Interpolation can be useful because the underlying function is not known in all points, or because approximating the function using well known functions such as polynomials is more convenient. In short, interpolation is about finding a good way to represent the underlying data. Polynomials are excellent to approximate analytical functions. For well chosen interpolation points, the error decreases exponentially with the number of interpolation points [4, 22]. They are also easily integrated and differentiated which we will exploit when developing quadrature rules and solving partial differential equations later in this thesis. More generally we will approximate the function f by finite sums of well chosen, pre-defined basis functions q_i,

$$f(\boldsymbol{x}) \approx \mathcal{I}_M[f](\boldsymbol{x}) = \sum_{i=1}^{M} \beta_i q_i(\boldsymbol{x}). \tag{2.2}$$

If the basis functions are 1 in one of the interpolation points and 0 in the others, then the basis is said to be nodal, i.e. if $q_i(\boldsymbol{x}_j) = \delta_{ij}$, where δ_{ij} is the Kronecker delta. This is called Lagrangian interpolation and is a classical way of approximating functions.

Finding the best interpolation points $\boldsymbol{x}_i$ and basis functions q_i to minimize the error is in general an open problem. One approach, using equidistant interpolation points and a nodal piecewise linear basis function, is shown in Figure 2.1. Another approach, using Gauss-Lobatto Legendre (GLL) points and nodal polynomial basis functions, is shown in Figure 2.2. As mentioned previously, the piecewise linear basis functions can be applied to a variety of domains, but typically yield algebraic convergence. Using polynomial basis functions over the GLL points yields exponential convergence for analytical functions, but only works on simple domains. The Empirical Interpolation Method (EIM) was proposed in [15] to find interpolation points and basis functions for general problems. It does not claim to be optimal, but works well in practice and can be shown to be well behaved under certain circumstances. Figure 2.3 shows interpolation points and a polynomial basis function chosen by the EIM.

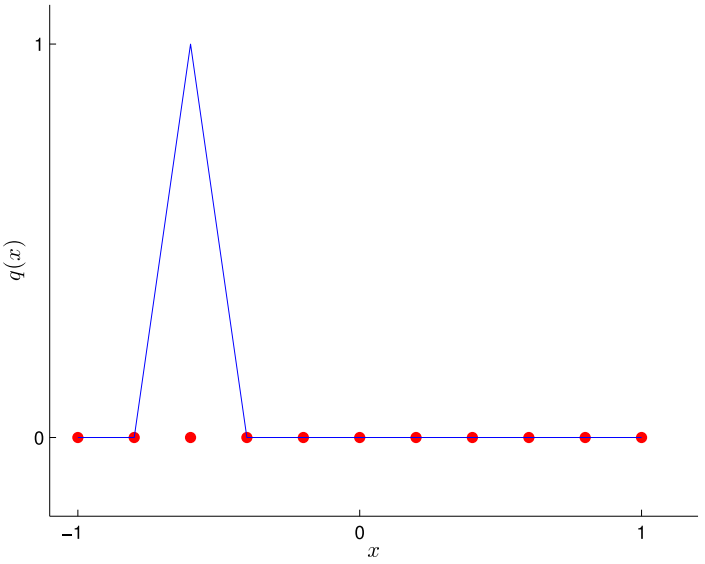

Figure 2.1: An example of a nodal basis function $q(x)$ (solid line) that is piecewise linear. Equidistant interpolation points are used (red dots).

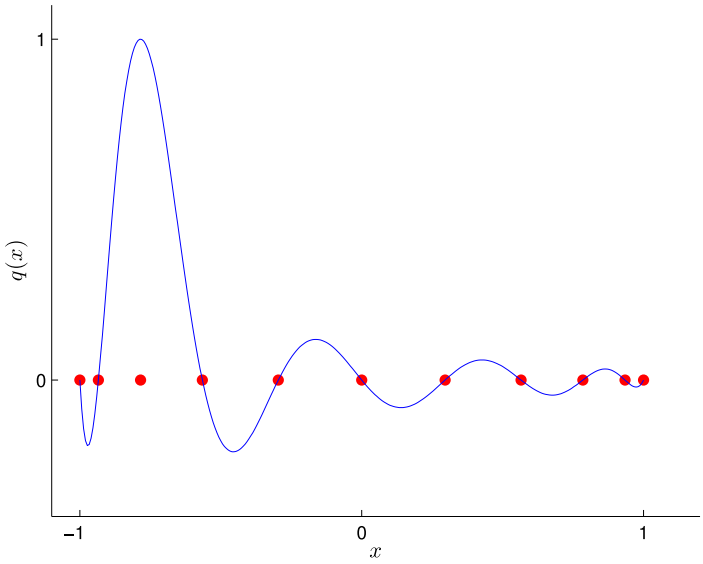

Figure 2.2: An example of a nodal polynomial basis function $q(x)$ (solid line). GLL distributed interpolation points are used (red dots).

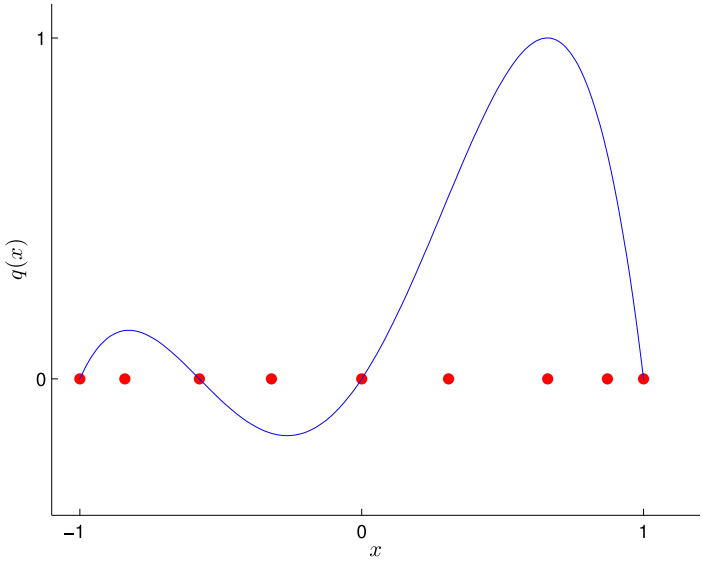

Figure 2.3: An example of a polynomial basis function $q(x)$ (solid line) and interpolation points (red dots) chosen by the EIM.

The EIM was first introduced in [3] and extended in [11]. It was originally developed to deal with non-affine functions in the Reduced Basis (RB) method [20]. The Reduced Basis method is a way to approximate the solution of parametrized Partial Differential Equations (PDEs). The solution $u(\boldsymbol{x};\boldsymbol{\mu})$ of parametrized PDEs does not only depend on the spatial position $\boldsymbol{x} \in \Omega$ but also on some parameter $\boldsymbol{\mu} \in \mathcal{D}$, where $\mathcal{D}$ is a parameter domain. The parameter $\boldsymbol{\mu}$ can for instance change attributes of the underlying problem like material properties or length scales. EIM was developed to approximate parametrized functions $f(\boldsymbol{x};\boldsymbol{\mu})$ as something that is affine in the parameter dependence. EIM allows us to find functions ϕ_i and q_i such that

$$f(\boldsymbol{x};\boldsymbol{\mu}) \approx \sum_{i=1}^{M} \phi_i(\boldsymbol{\mu}) q_i(\boldsymbol{x}). \tag{2.3}$$

By exploiting this approximation, we can represent more functions in a way that is compatible with the Reduced Basis method. Again, how we represent the function is essential to what we are able to do with it. In the original version of EIM, the approximation space was spanned by snapshots $f(\boldsymbol{x};\boldsymbol{\mu}_i)$, given by the parameters $\boldsymbol{\mu}_i$, of the function that was to be approximated. In [15] EIM was generalized to use generic approximation spaces rather than to approximate a single parametrized function $f(\boldsymbol{x};\boldsymbol{\mu})$. This decoupling of the approximation space and the problem specific function f is essential to develop e.g. generic quadrature rules. In this thesis we will make use of both generic function spaces as well as problem specific function spaces.

In addition to the GLL points mentioned earlier, alternatives to EIM include "best points" [16] and Fekete points [18]. These methods are not much used in practice. We will therefore compare some of our EIM results with the GLL method, because of its popularity and good properties.

2.2 Derivation

We recall that we want to approximate a function f over a domain Ω by a linear combination of M pre-defined basis functions,

$$f(\boldsymbol{x}) \approx \mathcal{I}_M[f](\boldsymbol{x}) = \sum_{i=1}^{M} \beta_i q_i(\boldsymbol{x}). \tag{2.4}$$

The basis functions used will span a space W_M that is an approximation to the full function space $\mathcal{U}$ that contain f, i.e. $W_M \subseteq \mathcal{U}$. We will assume that the functions are at least continuous. Let $\mathcal{G}(\cdot;\boldsymbol{\mu})$ be a parametrized function that generate (span) the full function space by choosing parameters $\boldsymbol{\mu}$ from the parameter domain $\mathcal{D}$,

i.e. $\mathcal{U} = \text{span}\{\mathcal{G}(\cdot;\boldsymbol{\mu}) : \boldsymbol{\mu} \in \mathcal{D}\}$. When constructing W_M we will choose functions among a finite dimensional subset of dimension $\mathcal{M} \geq M$, spanned by the generating function, i.e. $W_M \subseteq \text{span}\{\mathcal{G}(\cdot;\boldsymbol{\mu}) : \boldsymbol{\mu} \in \Xi\}$. Here, $\Xi \subseteq \mathcal{D}$ is of dimension $\mathcal{M}$ and is called the training set. The snapshots $\mathcal{G}(\cdot;\boldsymbol{\mu}_i)$ chosen by EIM to span W_M is represented by the associated magic parameter $\boldsymbol{\mu}_i$. An example of a generating function is $\mathcal{G}(x;\mu) = x^\mu$, $\mu \in \Xi = [0,\ldots,\mathcal{M}-1]$, or simply the first $\mathcal{M}$ monomials. We will often use all of the parameters in the training set to span our approximation space, i.e. $W_M = \text{span}\{\mathcal{G}(\cdot;\boldsymbol{\mu}) : \boldsymbol{\mu} \in \Xi\}$. The $\mathcal{G}(\cdot;\cdot)$ abstraction is necessary in order to easily apply EIM to different types of problems.

In order for the approximation to interpolate the function f in the interpolation points we require

$$\sum_{j=1}^{M} q_j(x_i)\beta_j = f(x_i) \qquad i = 1,\ldots,M, \tag{2.5}$$

which on matrix form becomes

$$B^M \underline{\beta} = f(\underline{x}). \tag{2.6}$$

The EIM is an algorithm for constructing interpolation points and basis functions. EIM usually select the next interpolation point and basis function greedily. The greedy EIM works iteratively by adding the next function and point that is the worst approximated by the current approximation. It is listed in Algorithm 1. First, we select the initial basis function, described by the magic parameter $\boldsymbol{\mu}_1$, and the corresponding magic point $\boldsymbol{x}_1$. Then the first basis function $q_1(\boldsymbol{x})$ is set to be the snapshot $\mathcal{G}(\boldsymbol{x};\boldsymbol{\mu}_1)$ of the generating function, normalized to 1 for the magic point $\boldsymbol{x}_1$. Next, we use (2.5) to approximate each snapshot of the generating function, choosing the parameter of the snapshot that is the worst approximated to be $\boldsymbol{\mu}_2$. The worst approximated point is then set as the corresponding magic point $\boldsymbol{x}_2$. The snapshot $\mathcal{G}(\boldsymbol{x};\boldsymbol{\mu}_2)$ is normalized to form the second basis function $q_2(\boldsymbol{x})$. The algorithm continues to select the worst approximated function and point to form the next basis function and magic point until the maximum number of magic points is reached. In practice the discrete training sets $\bar{\Omega} \subset \Omega$ and $\Xi \subset \mathcal{D}$ are used rather than their continuous versions.

In some applications it can be desirable to select the basis functions in a predetermined manner rather than greedily. The GLL method uses polynomials of ascending degree when increasing the order of the method. We can do the same with the EIM by enforcing it to choose polynomials of ascending order. We will call this ascending EIM and we will use it extensively later on. When the target function can be approximated with exponential convergence using polynomials or trigonometric functions, the contribution of each basis function, added ascendingly, will also decrease exponentially [4, 5]. Therefore it makes sense to choose the ascending

Algorithm 1 Greedy EIM

$\boldsymbol{\mu}_1 = \arg\max_{\boldsymbol{\mu}\in\mathcal{D}} \|\mathcal{G}(\cdot;\boldsymbol{\mu})\|_{L^\infty(\Omega)}$
$\boldsymbol{x}_1 = \arg\max_{\boldsymbol{x}\in\Omega} |\mathcal{G}(\boldsymbol{x};\mu_1)|$
$q_1 = \mathcal{G}(\cdot;\boldsymbol{\mu}_1)/\mathcal{G}(\boldsymbol{x}_1;\boldsymbol{\mu}_1)$
for $m = 2 : M$ **do**
  $\boldsymbol{\mu}_m = \arg\max_{\boldsymbol{\mu}\in\mathcal{D}} \|\mathcal{G}(\cdot;\boldsymbol{\mu}) - \mathcal{I}_{m-1}[\mathcal{G}(\cdot;\boldsymbol{\mu})]\|_{L^\infty(\Omega)}$
  $\boldsymbol{x}_m = \arg\max_{\boldsymbol{x}\in\Omega} |\mathcal{G}(\boldsymbol{x};\boldsymbol{\mu}_m) - \mathcal{I}_{m-1}[\mathcal{G}(\cdot;\boldsymbol{\mu}_m)](\boldsymbol{x})|$
  $q_m = \frac{\mathcal{G}(\cdot;\boldsymbol{\mu}_m)-\mathcal{I}_{m-1}[\mathcal{G}(\cdot;\boldsymbol{\mu}_m)]}{\mathcal{G}(\boldsymbol{x}_m;\boldsymbol{\mu}_m)-\mathcal{I}_{m-1}[\mathcal{G}(\cdot;\boldsymbol{\mu}_m)](\boldsymbol{x}_m)}$
end for

version of EIM when the approximation space is constructed using polynomials and trigonometric functions.

2.3 Properties

2.3.1 Properties of the EIM algorithm

From [15] we have to following properties:

Lemma 2.1. *Assume that the space $W_M = span\{\mathcal{G}(\cdot;\boldsymbol{\mu}_1),\dots,\mathcal{G}(\cdot;\boldsymbol{\mu}_M)\}$ is of dimension M and that the $M\times M$ matrix B^M with entries $q_j(\boldsymbol{x}_i)$ is invertible, then we have $\mathcal{I}_M[f] = f$ for any $f \in W_M$.*

Theorem 2.1. *Assume that M is chosen such that $M < \mathcal{M}$; then, for any $m \le M$, the space $X_m = span\{q_1,\dots,q_m\}$ is of dimension m and coincides with $span\{\mathcal{G}(\cdot;\boldsymbol{\mu}_1),\dots,\mathcal{G}(\cdot;\boldsymbol{\mu}_m)\}$. In addition, the matrix B^M is lower triangular with unity diagonal (hence it is invertible).*

The interpolation using Algorithm 1 is thus exact for all function that lie in the approximation space. The q-basis functions result in a nested lower triangular system, which can be solved in $\mathcal{O}(M^2)$ operations.

2.3.2 Computational cost

We look at the computational cost in two stages. The first, the offline stage, is the work necessary to train EIM, i.e. finding the magic points and the accompanying basis functions. Afterwards we can reuse the magic points and basis functions to interpolate different functions, this is called the online stage. We will only consider the dominating contributions to the overall cost.

Offline stage

We will first look at the computational cost of the greedy EIM. For each pair of magic point and basis function we add, we iterate over the $\mathcal{O}(\mathcal{N})$ candidate functions to include in the approximation space. Each candidate is approximated by solving the lower triangular $M \times M$ system of linear equations at a cost of $\mathcal{O}(M^2)$. If we use discrete basis functions, we then find the function value in all $\mathcal{M}$ points by a matrix vector product at a cost of $\mathcal{O}(\mathcal{M}M)$, which dominates the cost of solving the system of linear equations. The cost of adding each candidate is thus $\mathcal{O}(\mathcal{M}M\mathcal{N})$. The total cost, for M magic points, is then $\mathcal{O}(\mathcal{M}\mathcal{N}M^2)$. The ascending version of EIM does not need to look through the $\mathcal{O}(\mathcal{N})$ candidate functions for each magic point, so the total cost is reduced to $\mathcal{O}(\mathcal{M}M^2)$. The storage cost of each basis vector is $\mathcal{O}(\mathcal{M})$, for a total storage cost of $\mathcal{O}(\mathcal{M}M)$. The two versions of EIM only differ in how they compute the EIM bases, once computed they are stored and used in the same way in the online stage.

Online stage

The interpolation is achieved by solving the lower triangular system at a cost of $\mathcal{O}(M^2)$. Evaluating the function in $\mathcal{M}$ points would add an extra cost of $\mathcal{O}(\mathcal{M}M)$.

2.3.3 Lebesgue constant

Let f be the function we wish to approximate and f^* be the best polynomial approximation of f in the space W_M, i.e

$$f^* = \arg \inf_{f^* \in W_M} \|f - f^*\|_{L^\infty(\Omega)}. \tag{2.7}$$

The polynomial f^* can be interpolated exactly by a polynomial interpolation, $\mathcal{I}[f^*] = f^*$. The Lebesgue constant can then be derived as in [18]

$$\begin{aligned} \|f - \mathcal{I}_M[f]\|_{L^\infty(\Omega)} &= \|f - f^* + \mathcal{I}_M[f^*] - \mathcal{I}_M[f]\|_{L^\infty(\Omega)} \\ &\leq \|f - f^*\|_{L^\infty(\Omega)} + \|\mathcal{I}_M\|_{L^\infty(\Omega)}\|f^* - f\|_{L^\infty(\Omega)} \\ &\leq (1 + \|\mathcal{I}_M\|_{L^\infty(\Omega)})\|f^* - f\|_{L^\infty(\Omega)} \end{aligned}$$

By choosing the Lebesgue constant to be $\Lambda_M = \|\mathcal{I}_M\|_{L^\infty(\Omega)}$ we arrive at the final result in the Lemma 2.2.

Lemma 2.2. *For any $f \in W$, the interpolation error satisfies*

$$\|f - \mathcal{I}_M[f]\|_{L^\infty(\Omega)} \leq (1 + \Lambda_M) \inf_{f^* \in W_M} \|f - f^*\|_{L^\infty \Omega}. \tag{2.8}$$

To actually compute the Lebesgue constant, we want to use the same method as in [15] to be able to compare results. We therefore choose to interpolate u, where $\|u\|_{L^\infty(\Omega)} = 1$, using nodal basis functions $h_i(\cdot) = \sum_{j=1}^{M} q_j(\cdot)[B]_{ji}^{-1}$ such that

$$\begin{aligned}\|\mathcal{I}_M[u]\|_{L^\infty(\Omega)} &= \|\sum_{i=1}^{M} u_i h_i(x)\|_{L^\infty(\Omega)} \\ &\leq \|\sum_{i=1}^{M} h_i(x)\|_{L^\infty(\Omega)} \leq \sum_{i=1}^{M} |h_i(x)|.\end{aligned}$$

The Lebesgue constant is a measure of how far from the best approximation f^* our interpolation $\mathcal{I}_M$ will be. Some Lebesgue constants involving different types of interpolation points have been reproduced from [15] and shown in Figure 2.4. The Lebesgue constants of the various EIM variations outperforms the uniform points by far, while not being too far from that of the Chebyshev points. The distribution of some of the interpolations points have also been reproduced from [15] and is shown in Figure 2.5. The the magic points resemble the Chebyshev points more than the equidistant points when looking at $\cos^{-1}$ of the distributions.

2.3.4 A priori error

An a priori error estimate is obtained in [15] and improved in [7]. The main results is that if there exists a finite-dimensional space allowing for an exponentially converging approximation, then the EIM will achieve an exponential rate of convergence.

2.3.5 A posteriori error

An approximation to the a posteriori error is given in [15]. Let ϵ_M be the interpolation error of f using M magic points, $\epsilon_M = \|f - \mathcal{I}_M[f]\|_{L^\infty(\Omega)}$. By refining the interpolation by performing another iteration of EIM, the error can only decrease or stay unchanged, i.e. $\epsilon_M \geq \hat{\epsilon}_M = \|f - \mathcal{I}_{M+1}[f]\|_{L^\infty(\Omega)} = |f(x_{M+1}) - \mathcal{I}_M[f](x_{M+1})|$. $\hat{\epsilon}$ is therefore a lower bound to the error ϵ, but if the error goes to 0 very fast, it is not unreasonable to use $\hat{\epsilon}$ as an approximation to the error. A rigorous upper bound of the interpolation error was presented in [8]. It requires the parametric derivatives of the function to be approximated. A method to approximate these derivatives is presented in [9].

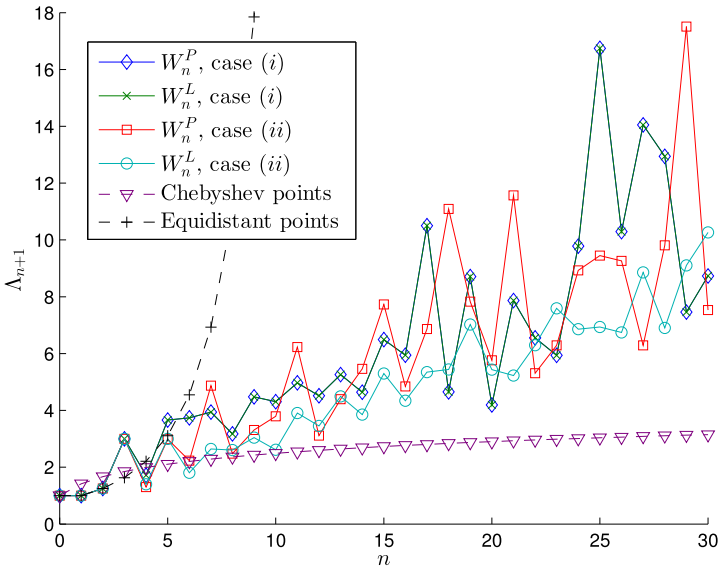

Figure 2.4: These results are reproduced from [15]. The Lebesgue constant Λ_{n+1} for $n+1$ interpolation points is given for monomial polynomials W_n^P and Legendre polynomials W_n^L. Case (i) refers to choosing the polynomials in an ascending order, while case (ii) is choosing greedily. The Lebesgue constant is also given for the Chebyshev points and equidistant points. The approximation spaces are spanned by $\{x^0, \ldots, x^n\}$, except in the case of greedy EIM where there approximation space is spanned by n of the monomials $\{x^0, \ldots, x^{30}\}$.

2.4 Numerical results

We have already motivated that we should use the ascending EIM when approximating analytical functions with polynomials, because the contributions will decrease exponentially. In Figure 2.6 we show results that support this. In Table 2.2 we can see that the greedy EIM chooses high order polynomials early, which might explain why the ascending EIM outperforms it in Figure 2.6. The computational cost of the ascending EIM is also lower than that of the greedy version. We will therefore use the ascending EIM whenever generic bases are used. Later, when using problem specific bases, there is no natural ascending order of the candidate functions, thus the greedy version of EIM is applied.

Because we will compute the magic points using a discrete domain $\bar{\Omega}$, it is interesting to see how sensitive the EIM interpolation is to the resolution of the domain. Figure 2.7 suggests that EIM interpolation is not sensitive to the location of the magic points. When interpolating functions like e^x, the EIM interpolation is actually even less sensitive. We will exploit this to use fairly coarse discrete domains

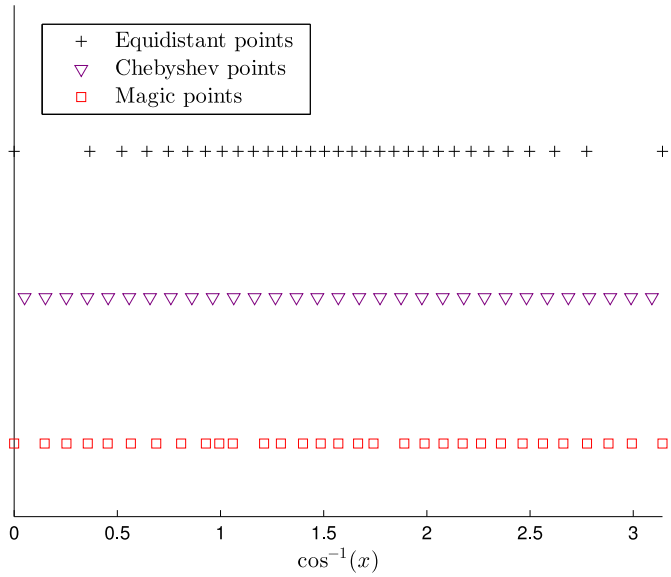

Figure 2.5: Reproduced results from [15]. We observe that $\cos^{-1}$ of the distribution of the magic points from greedy EIM, using monomial basis functions, are more similar to Chebyshev points than the equidistant points.

in the numerical results. The resolution will be given by the grid spacing Δ. The points in the domain will not lie further apart than Δ in each direction. In 2D this means that the Euclidean distance between two neighbouring points will not be greater than $\sqrt{2}\Delta$.

The rest of this chapter consists of various numerical results when applying EIM to interpolation in both 1D and 2D. We will look at the basis functions that different versions of EIM produce, both graphically and as expressions like $0.50 + 0.50\cos(\pi x)$. How this is achieved will become clear in later chapters. We observe that the basis functions of the ascending EIM in Table 2.1 share some similarities with the Legendre polynomials in Table 2.3.

The magic points from EIM on a square in Figure 2.13 show similarities with the GLL points in Figure 2.14. Interpolation using monomial EIM shows similarities with GLL interpolation in Figures 2.9 and 2.15. In Figure 2.11 trigonometric EIM interpolation outperforms GLL interpolation, because trigonometric basis functions are better suited than polynomials in that particular case.

So far the performance of the EIM interpolation is competitive with that of the GLL interpolation. The distribution of the magic points show similarities with both GLL points and Chebyshev points. In later chapters we will apply EIM to domains where the GLL points are not directly applicable.

Index i	Function $q_i(x)$
1	1.00
2	0.50 + 0.50x
3	1.00 - 1.00x^2
4	-2.60x + 2.60x^3
5	1.25x + 2.17x^2 - 1.25x^3 - 2.17x^4

Table 2.1: The five first basis functions $q_i(x)$ from ascending EIM using monomial generating functions (x^μ).

Index i	Function $q_i(x)$
1	1.00
2	0.50 + 0.50x
3	1.00 - 1.00x^2
4	-1.17x + 1.17x^{29}
5	0.58x + 0.66x^2 - 0.58x^{29} - 0.66x^{30}

Table 2.2: The five first basis functions $q_i(x)$ from greedy EIM using monomial generating functions (x^μ). The result depends on how large the function space is. In this case $M = 31$ and the candidate functions are $\{x^0, \ldots, x^{30}\}$.

Index	Function
1	1.00
2	x
3	-0.50 + 1.50x^2
4	-1.50x + 2.50x^3
5	0.375 - 3.75x^2 + 4.375x^4

Table 2.3: The five first Legendre polynomials.

Index i	Function $q_i(x)$
1	1.00
2	0.50 + 0.50cos(πx)
3	1.00sin(πx)
4	0.25 - 0.50sin(πx) - 0.25cos($2\pi x$)
5	-1.00sin($2\pi x$)

Table 2.4: The five first basis functions $q_i(x)$ from greedy EIM using trigonometric generating functions (sin(μx),cos(μx)).

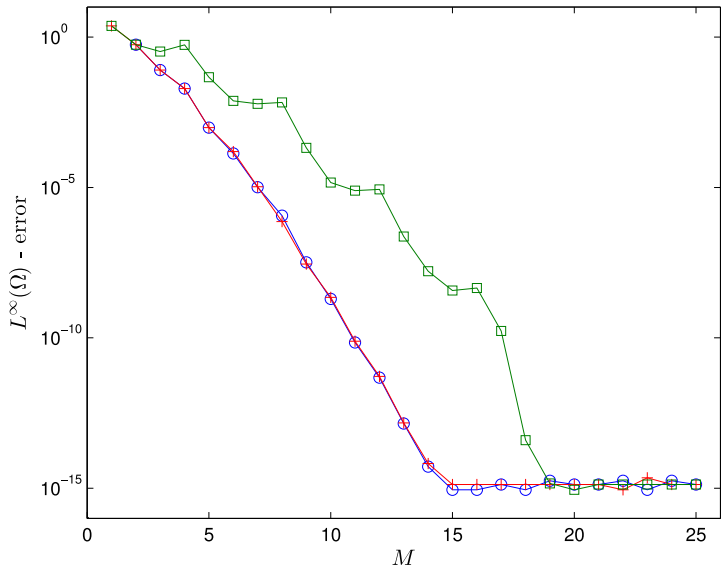

Figure 2.6: We look at EIM interpolation of e^x. The nested nature of EIM allows us to interpolate using the M first monomials, selected ascendingly (red plus signs) or greedily (green squares), from some larger space (W_{25} in this case). This may produce a different convergence rate than using all of the M first monomials (blue circles), from a space W_M of dimension M. In this case the basis functions were selected greedily, but when we use all of the basis functions, they will span the same space as the ascending EIM. When interpolating e^x we see that using the polynomials $\{x^0, \ldots, x^{M-1}\}$ produce the best results, suggesting that the ascending EIM is a good choice.

Index i	Function $q_i(x, y)$
1	1.00
2	0.50 + 0.50x
3	0.50 + 0.50y
4	1.00 - 1.00x^2
5	0.25 - 0.25x + 0.25y - 0.25xy

Table 2.5: The five first basis functions $q_i(x, y)$ from ascending EIM on a square using monomial generating functions ($x^m y^n$).

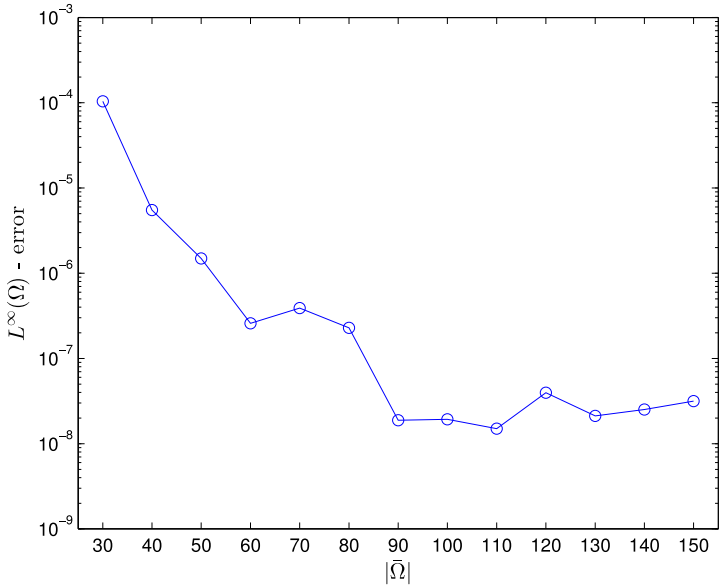

Figure 2.7: Interpolation error of $1/(1 + 2x^2)$ for different number of points in the domain $\bar{\Omega}$ when training EIM. The number of of magic points used in the interpolation is 30. The error is computed in 300 equidistant points. We get exponential convergence that yields good results for a training set $\bar{\Omega}$ that is only 2-3 times larger than the number of magic points used. This suggests that EIM is not particularly sensitive to the location of the magic points, enabling us to use a coarse grid when computing the discrete max.

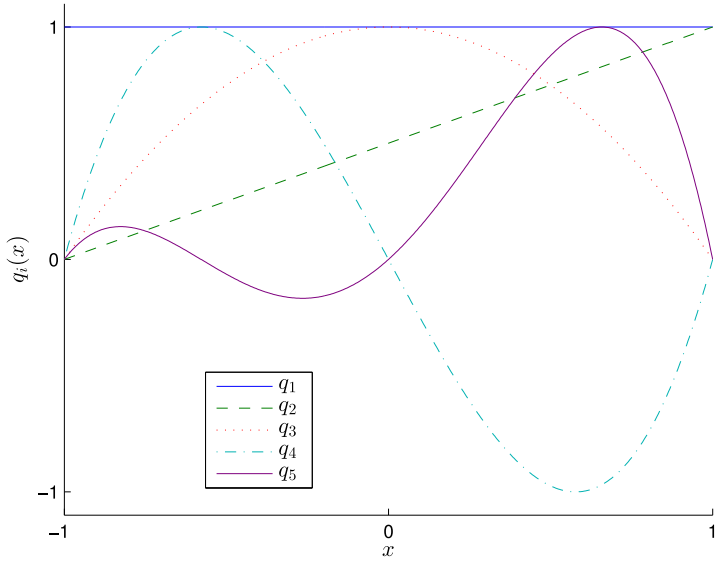

Figure 2.8: The five first basis functions $q_i(x)$ from ascending EIM using monomial generating functions (x^μ) on $\Omega = [-1, 1]$.

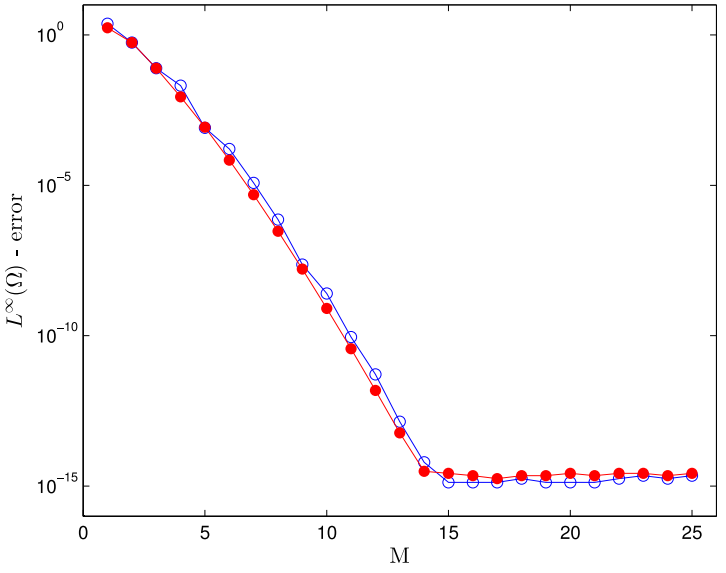

Figure 2.9: Convergence of the interpolation error of e^x as a function of interpolation points M. EIM (blue circle) and GLL (red dots) converge at the same rate. Here, $\Omega = [-1, 1]$ with a grid spacing of $\Delta = 4 \cdot 10^{-3}$.

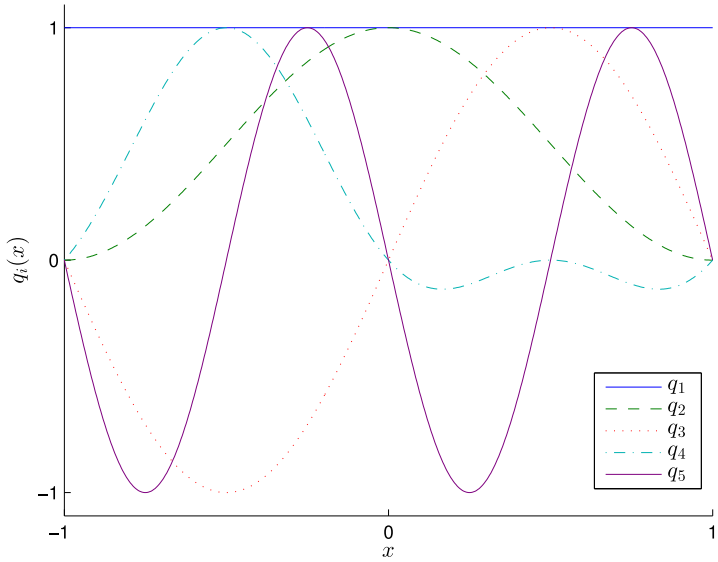

Figure 2.10: The five first basis functions $q_i(x)$ from ascending EIM using trigonometric generating functions ($\sin(\mu x)$,$\cos(\mu x)$) on $\Omega = [-1, 1]$.

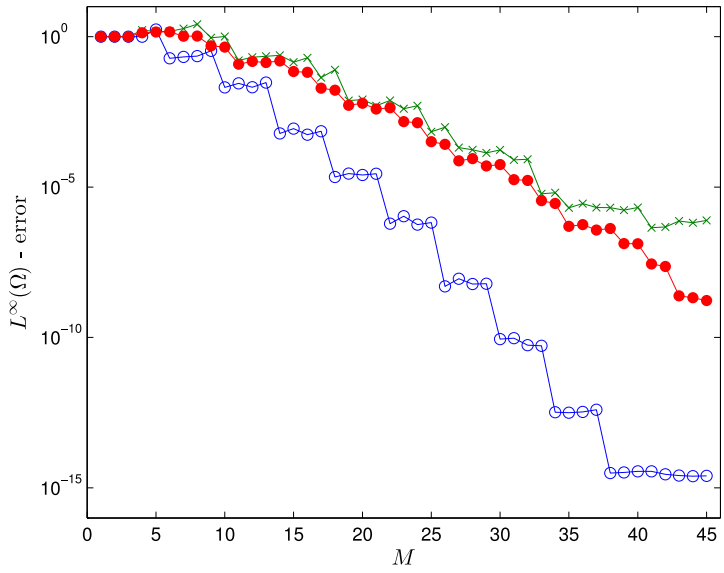

Figure 2.11: Convergence of the interpolation error of $\sin(\pi\cos(\pi(x+1)))$ as a function of the number of interpolation points M. Here, $\Omega = [-1, 1]$ with a grid spacing of $\Delta = 4 \cdot 10^{-3}$. The trigonometric EIM performs the best (blue circles), while GLL (red dots) performs somewhat better than monomial EIM (green crosses).

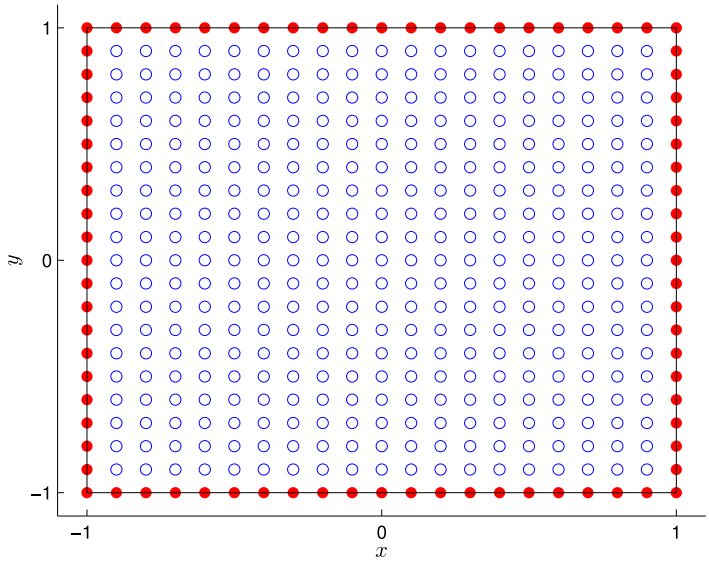

Figure 2.12: Spatial training set for a square domain with a grid spacing of $\Delta = 10^{-1}$. All the primitive spatial training sets are built in the same way. First the boundary is discretized by placing points along each segment of the boundary (red dots). Then the interior is filled with a uniform grid (blue circles).

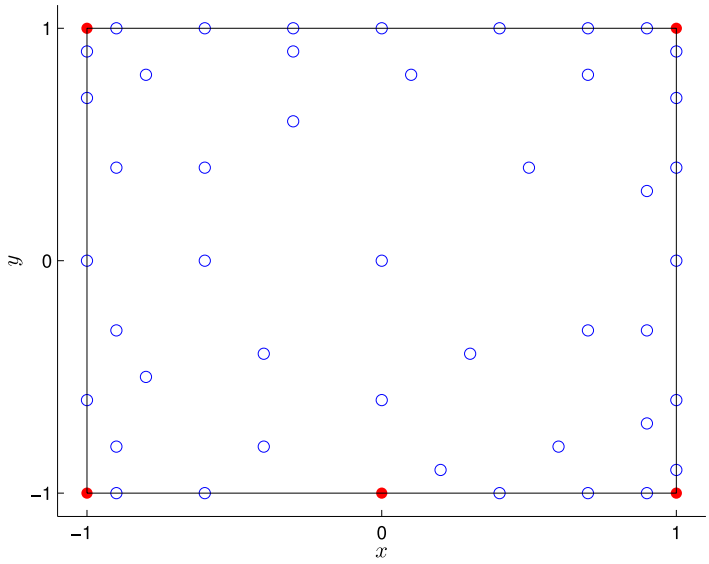

Figure 2.13: Distribution of the first 50 spatial magic points on a square domain. The red dots show the first five magic points.

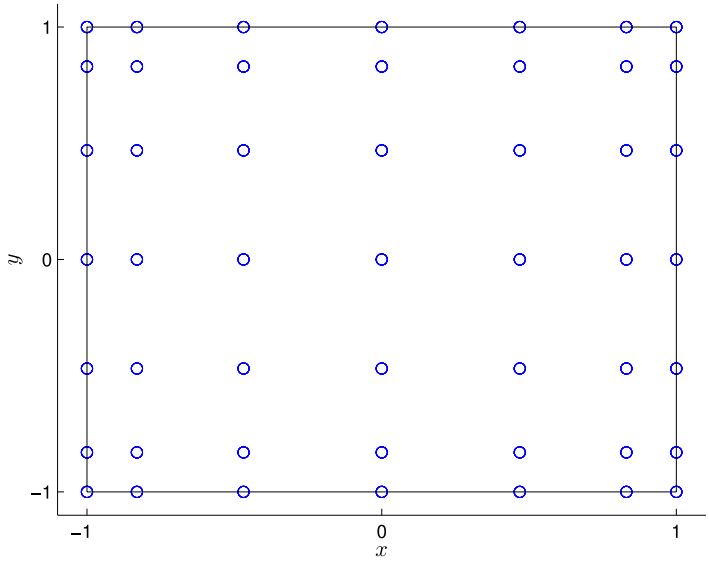

Figure 2.14: Distribution of the first 49 GLL points on a square domain.

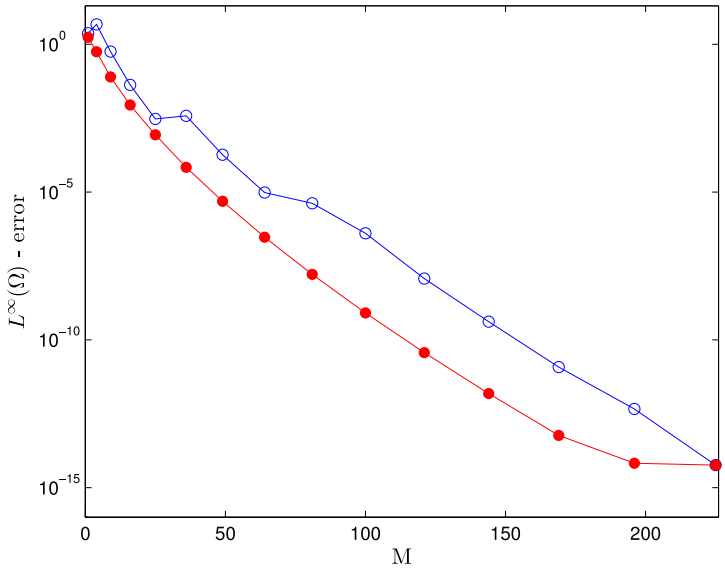

Figure 2.15: Convergence of the interpolation error of e^{xy} as a function of interpolation points M. The domain Ω is a square with a grid spacing of $\Delta = 5 \cdot 10^{-2}$. The convergence rate of of GLL (red dots) is slightly better than that of EIM (blue circles).

Chapter 3

The semi-analytical formulation of EIM

The semi-analytical formulation of the EIM was developed to offer a practical approach to apply EIM to a variety of problems. Its main advantage is the ability to perform analytical operations on the basis functions which will prove useful when performing quadrature and solving partial differential equations in later chapters. Furthermore, it allows the software implementation of EIM to be abstract in much the same way as the underlying mathematics. This makes it very easy to apply EIM to a variety of problems. The difference between the semi-analytical formulation and the usual formulation of EIM is essentially a change of basis, which in some circumstances lead to an extra computational cost. In this chapter we present the semi-analytical formulation of EIM and some of its properties.

3.1 Introduction

When working on this thesis it soon became clear that it would be desirable to have a unified computational framework for the range of problems we wanted to look at. We also wanted to be able to perform analytical operations on the basis functions. While several implementations of EIM use discrete basis vectors, resulting from a finite training set $\bar{\Omega} \subset \Omega$ or as in Discrete EIM (DEIM) [6, 24], the semi-analytical EIM treats the basis functions as functions, thus allowing analytical operations. The the magic points, however, are typically computed by sampling the function in a finite set of points. The max in Algorithm 1 could in some cases be computed analytically, or at least numerically, but since the EIM method is quite accurate, even for fairly coarse grids $\bar{\Omega}$, it is more practical to sample the function in a finite set of points. This way we can easily handle any function, and is why we call

it semi-analytical: we get the convenience of a discrete max, combined with the flexibility of analytical basis functions.

The difference between the semi-analytical formulation and the standard formulation of EIM is essentially that the basis functions q are stored in a different basis spanned by snapshots of $\mathcal{G}(\cdot;\cdot)$. In the previous chapter we computed the basis functions q using EIM. If these basis functions were to be stored discretely, we could loose the information of which functions actually span our approximation space, and thus our ability to perform analytical operations on the basis functions. This space is by construction spanned by the generating function $\mathcal{G}(\cdot;\cdot)$ that was introduced in the previous chapter. By representing everything using linear combinations of instantiations of $\mathcal{G}(\cdot;\cdot)$, we can isolate the functions that span the space. If we can perform linear analytical operations on each of the functions that span the space, then we can perform those operations on all the functions in the space.

An example is the fifth monomial basis function, $q_5(x,y) = 0.25 - 0.25x + 0.25y - 0.25xy$, resulting from ascending EIM on a square. In the semi-analytical formulation, this basis function is stored as pairs of coefficients and snapshots of the generating function $\mathcal{G}(\cdot;\cdot)$, i.e. $(.25, 1)$, $(-.25, x)$, $(.25, y)$ and $(-.25, xy)$. It is easy to perform linear analytical operation on each of the functions, e.g. xy, and thus on linear combinations of those functions. As long as each of the functions are easy to work with separately we can combine them into an approximation space. In this thesis we have used monomials as well as sine and cosine functions, but other combinations of simple functions will work equally well.

Furthermore, if $\mathcal{G}(\cdot;\cdot)$ is either compactly described, e.g. as monomials, x^μ, or already stored as e.g. discrete functions, then the extra cost of storing the basis functions is limited to storing the basis coefficients and which snapshots of the generating function they belong to. By hiding away the particulars of each problem in $\mathcal{G}(\cdot;\cdot)$ we can implement software that works with a generic generating function. Changing the application will then be limited to changing the generating functions. More about this in the Software Implementation chapter.

The key features when using the $\mathcal{G}(\cdot,\cdot)$ basis are:

- Sampling of functions at arbitrary points in the domain
- Analytical integration and differentiation (and other linear operations)
- Concise storing of basis functions and coefficients
- A practical approach to adapt EIM to a variety of applications

3.2 Derivation

We recall that the standard version of EIM approximates a function f as

$$f(\boldsymbol{x}) \approx \mathcal{I}_M[f](\boldsymbol{x}) = \sum_{i=1}^{M} \beta_i q_i(\boldsymbol{x}), \tag{3.1}$$

by using magic points $\boldsymbol{x}_i$ and basis functions q_i obtained from Algorithm 1 and solving

$$\sum_{j=1}^{M} q_j(\boldsymbol{x}_i)\beta_j = f(\boldsymbol{x}_i) \qquad i = 1, \ldots, M. \tag{3.2}$$

The basic idea of the semi-analytical approach is to always express functions as a linear combination of the generating function $\mathcal{G}(\cdot;\cdot)$, as opposed to storing the linear combination as a single discrete vector. One immediate advantage is that if we can perform linear operations analytically on $\mathcal{G}(\cdot;\cdot)$, we can also perform those operations on linear combinations of $\mathcal{G}(\cdot;\cdot)$. This can involve sampling the function anywhere in the domain, integration and differentiation. It will also prove useful in compression, in producing human readable output of the solution, and to easily apply EIM to a variety of applications.

Assuming that f is a linear combination of the generating function, given some magic parameters, we can express it as

$$f(\cdot) = \sum_{i=1}^{M} \alpha_i^f \mathcal{G}(\cdot; \boldsymbol{\mu}_i). \tag{3.3}$$

The snapshots of the generating function $\mathcal{G}(\cdot;\cdot)$ does not need to be linearly independent. But the functions $\mathcal{G}(\cdot;\boldsymbol{\mu}_i)$ should be linearly independent if they were chosen by EIM. This is because of the property that EIM interpolates functions in the approximation space exactly, so that only linearly independent snapshots are selected. Algorithm 1 will, by construction, produce basis functions $\{q_i\}_{i=1}^M$ that are linear combinations of the generating function $\mathcal{G}(\cdot;\cdot)$. We can therefore write

$$q_j = \sum_{i=1}^{M} \alpha_{ij}^Q \mathcal{G}(\cdot; \boldsymbol{\mu}_i). \tag{3.4}$$

By collecting all the coefficients in a transformation matrix $T^Q \in \mathbb{R}^{M\times M}$ we can make the connection to the usual collection of basis functions $\{q_i\}_{i=1}^M$ as columns in the matrix Q. We thus rename the α-coefficients

$$T_{ij}^Q = \alpha_{ij}^Q, \tag{3.5}$$

and insert into (3.4)

$$q_j = \sum_{i=1}^{M} \alpha_{ij}^{Q} \mathcal{G}(\cdot, \boldsymbol{\mu}_i) = \sum_{i=1}^{M} T_{ij}^{Q} \mathcal{G}(\cdot; \boldsymbol{\mu}_i). \tag{3.6}$$

By introducing underline to indicate a vector of values, $\underline{\boldsymbol{\mu}} = [\boldsymbol{\mu}_1 \cdots \boldsymbol{\mu}_M]^T$, and writing a vector of functions as

$$\mathcal{G}(\cdot, \underline{\boldsymbol{\mu}}^T) = [\mathcal{G}(\cdot; \boldsymbol{\mu}_1) \cdots \mathcal{G}(\cdot, \boldsymbol{\mu}_M)], \tag{3.7}$$

we can write (3.6) in the more compact form

$$Q = \mathcal{G}(\cdot; \underline{\boldsymbol{\mu}}^T) T^Q. \tag{3.8}$$

Written out this becomes

$$Q = [q_1 \cdots q_M] = [\mathcal{G}(\cdot, \boldsymbol{\mu}_1) \cdots \mathcal{G}(\cdot, \boldsymbol{\mu}_M)] \begin{bmatrix} \alpha_{11}^{Q} & \cdots & \alpha_{1M}^{Q} \\ \vdots & \ddots & \vdots \\ \alpha_{M1}^{Q} & \cdots & \alpha_{MM}^{Q} \end{bmatrix}. \tag{3.9}$$

The discrete version is obtained by sampling in a set of discrete points $\underline{\boldsymbol{x}}^{\text{train}} = \bar{\Omega} \subset \Omega$, $\mathcal{G}(\underline{\boldsymbol{x}}^{\text{train}}; \underline{\boldsymbol{\mu}}^T) \in \mathbb{R}^{\mathcal{N} \times M}$, with $\mathcal{N}$ spatial points and M selected parameter values,

$$\mathcal{G}(\underline{\boldsymbol{x}}^{\text{train}}; \underline{\boldsymbol{\mu}}^T)_{ij} = \mathcal{G}(\boldsymbol{x}_i^{\text{train}}; \boldsymbol{\mu}_j). \tag{3.10}$$

The discrete version of Q, written $\bar{Q}$, which uses discrete basis vectors rather than functions, can thus be written

$$\bar{Q} = \mathcal{G}(\underline{\boldsymbol{x}}^{\text{train}}; \underline{\boldsymbol{\mu}}^T) T^Q, \tag{3.11}$$

where each column is a discrete basis vector $\bar{q}$. Written out we get

$$\bar{Q} = [\bar{q}_1 \cdots \bar{q}_M] = \begin{bmatrix} \mathcal{G}(\boldsymbol{x}_1^{\text{train}}, \boldsymbol{\mu}_1) & \cdots & \mathcal{G}(\boldsymbol{x}_1^{\text{train}}, \boldsymbol{\mu}_M) \\ \vdots & \ddots & \vdots \\ \mathcal{G}(\boldsymbol{x}_{\mathcal{N}}^{\text{train}}, \boldsymbol{\mu}_1) & \cdots & \mathcal{G}(\boldsymbol{x}_{\mathcal{N}}^{\text{train}}, \boldsymbol{\mu}_M) \end{bmatrix} \begin{bmatrix} \alpha_{11}^{Q} & \cdots & \alpha_{1M}^{Q} \\ \vdots & \ddots & \vdots \\ \alpha_{M1}^{Q} & \cdots & \alpha_{MM}^{Q} \end{bmatrix}. \tag{3.12}$$

To summarize we can write our Q basis as a linear combination of functions from our generating function $\mathcal{G}(\cdot, \cdot)$. The underlying functions can be handled analytically,

but the magic points are generally computed discretely, hence the name semi-analytical. The usual discrete representation can be obtained by sampling the underlying generating function in a set of discrete points.

Sometimes it is useful to construct a nodal basis. We will use h to represent nodal basis functions and require, $h_i(x_j) = \delta_{ij}$, where δ_{ij} is the Kronecker delta. We can express our H basis in a similar manner

$$h_j = \sum_{i=1}^{M} \alpha_{ij}^H \mathcal{G}(\cdot, \boldsymbol{\mu}_i) = \sum_{i=1}^{M} T_{ij}^H \mathcal{G}(\cdot, \boldsymbol{\mu}_i) \tag{3.13}$$

where

$$T_{ij}^H = \alpha_{ij}^H \tag{3.14}$$

and we get

$$H = \mathcal{G}(\cdot, \underline{\boldsymbol{\mu}}^T) T^H. \tag{3.15}$$

A change of basis from q to h is limited to computing $T^H \in \mathbb{R}^{M \times M}$ and does not affect our ability to perform analytical operations on the underlying generating function.

3.3 Adapting EIM to various problems

By choosing the appropriate domain Ω, parameter domain $\mathcal{D}$, and function space W, EIM can be adapted to a variety of applications. The semi-analytic formulation makes it easier to isolate these variations in practical software implementations by abstracting the function space into the generating function $\mathcal{G}(\cdot;\cdot)$. Examples are given in Table 3.1. For both quadrature and collocation we use the generic bases. The only difference between a polynomial basis on a triangle and a circle is the domain Ω and that the polynomials $x^i y^j$ are defined on the corresponding domain.

Application	$\boldsymbol{x} \in \Omega$	$\boldsymbol{\mu} \in \mathcal{D}$	$\mathcal{G}(\boldsymbol{x};\boldsymbol{\mu}) \in W$
Generic 1D Basis	$\boldsymbol{x} = (x)$	$\boldsymbol{\mu} = (i)$	x^i, $\sin(ix)$, $\cos(ix)$
Generic 2D Basis	$\boldsymbol{x} = (x, y)$	$\boldsymbol{\mu} = (i, j)$	$x^i y^j$
Generic 3D Basis	$\boldsymbol{x} = (x, y, z)$	$\boldsymbol{\mu} = (i, j, k)$	$x^i y^j z^k$
Image recognition	pixel in 2D image	frame number	look up pixel value
Animation	vertex in 3D	frame number	look up displacement

Table 3.1: Overview of different applications of EIM.

3.4 Linear operators

Because all functions are expressed as a linear combination of instances of the generating function $\mathcal{G}(\cdot,\cdot)$, as long as it is simple to perform linear analytical operations on those functions, it will also be possible to perform analytical operations on the linear combinations. This will prove useful when applying EIM to quadrature and solving partial differential equations. Then it will be necessary to integrate and differentiate the functions accurately.

3.5 Choice of basis

As mentioned earlier, the semi-analytical EIM stores the basis functions as linear combinations of the generating function $\mathcal{G}(\cdot;\cdot)$. However, we still use the q-basis functions when computing the EIM basis, and when performing interpolation. Conversion between the $\mathcal{G}$-basis and q-basis is done with the transformation matrix T^Q, while T^H is used to obtain the nodal h-basis functions. We could of course have used the $\mathcal{G}$-basis functions when computing the EIM basis and performing the interpolation, but this would not lead to a lower triangular system of equations as with the q-basis. Furthermore, using $\mathcal{G}$ directly seems to lead to ill-conditioned system of linear equations.

3.6 Computational cost

To be able to sample a function, we have to find the α's, even if we are using the q- or h-basis functions. Converting is done by applying either T^Q or T^H as described earlier. This is done at a cost of $\mathcal{O}(M^3)$. Compared to only finding the coefficients by solving the lower triangular system from before at a cost of $\mathcal{O}(M^2)$, this will incur an extra cost in both the offline and online stages. Previously, it was sampling the functions in $\mathcal{M}$ points at a cost of $\mathcal{O}(M\mathcal{M})$ that dominated solving the system of linear equations at a cost of $\mathcal{O}(M^2)$. Now, computing the coefficients will dominate as long as $\mathcal{M} < M^2$. We could of course use the $\mathcal{G}$-functions directly in EIM, but this would not result in a lower triangular system so the cost of obtaining the α's would still be $\mathcal{O}(M^3)$. This is the cost of the added flexibility and opportunities of the semi-analytical formulation of EIM.

The storage cost, given that we have already stored the generating function $\mathcal{G}$, is limited to the magic points and the magic parameters, thus $\mathcal{O}(M)$, which is potentially much less expensive than the $\mathcal{O}(M\mathcal{M})$ cost when using discrete vectors.

Chapter 4

Quadrature

The need to evaluate integrals, $\int_\Omega f \, \mathrm{d}\Omega$, occur in many applications. Performing the integration analytically can be prohibitive due to a complex function f or domain Ω. Quadrature, i.e. numerical integration, offers computationally feasible approximations. We will start by looking at some existing quadrature rules and how we can use EIM to produce more flexible rules in terms of the domains it can handle. As mentioned earlier GLL quadrature offers exponential convergence for analytic integrands, but is only applicable to simple domains like a line, a square and other tensor product domains. We will use EIM to develop quadrature rules for simple polygons, the semicircle and the circle. We will combine the good approximation property of the EIM interpolant together with exact integration of the basis functions to achieve quadrature rules with exponential convergence for analytic functions and a cost that scales linearly with the number of magic points. We will compare the EIM quadrature to GLL quadrature on the line and the square.

4.1 Derivation

Quadrature rules typically approximate the integral by a weighted sum of function evaluations

$$\int_\Omega f(\boldsymbol{x}) \, \mathrm{d}\Omega \approx \sum_{i=1}^{M} \rho_i f(\boldsymbol{x}_i). \tag{4.1}$$

This way we only need to be able to sample the function at a set of points to approximate the integral. GLL quadrature is one such method. It yields exponential convergence for analytic integrands, but is only defined on a line, a square or other

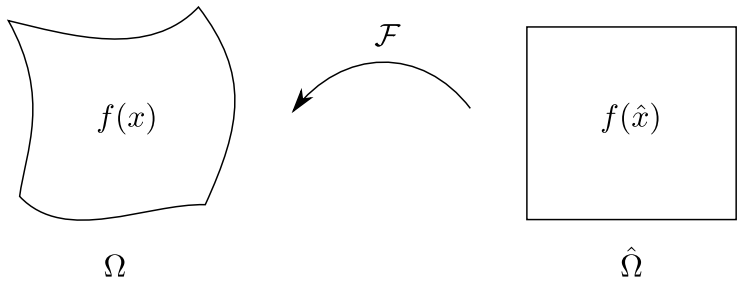

Figure 4.1: Mapping $\mathcal{F}$ from a reference domain $\hat{\Omega}$ to the domain Ω, i.e. $\Omega = \mathcal{F}(\hat{\Omega})$.

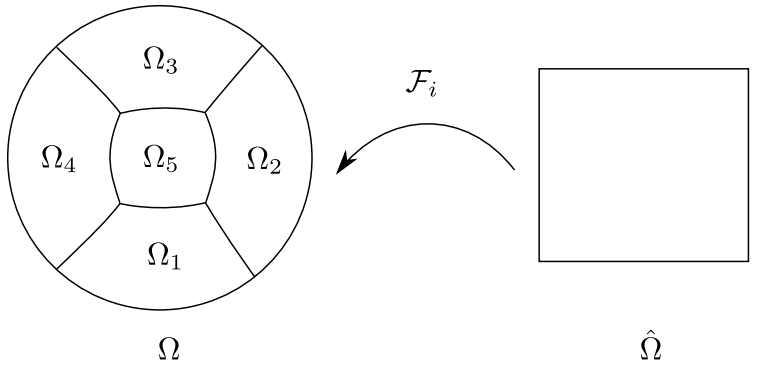

Figure 4.2: When integrating over a circular domain Ω, a common approach is to first partition into five subdomains, and then map each of the subdomains Ω_i from the reference domain $\hat{\Omega}$, via the mapping $\mathcal{F}_i$, i.e. $\Omega_i = \mathcal{F}_i(\hat{\Omega})$, $i = 1, \ldots, 5$.

tensor product domains. In general it is therefore necessary to map the domain Ω to a reference domain $\hat{\Omega}$ as shown in Figure 4.1. Sometimes, we first need to partition the domain into subdomains before we can map each of them to the reference domain separately, as shown in Figure 4.2. This can be achieved by applying e.g. the Gordon-Hall algorithm [22]. However, the domain decomposition and the corresponding mappings incur and additional complexity and an additional cost.

Monte Carlo integration is an alternative approach where random points in the domain are sampled directly, before averaged and multiplied with the volume V of the domain

$$\int_\Omega f(\boldsymbol{x}) \, \mathrm{d}\Omega = V \frac{1}{M} \sum_{i=1}^{M} f(\boldsymbol{x}_i). \tag{4.2}$$

The convergence rate, however, is only $\frac{1}{\sqrt{M}}$, which is only improved to $\frac{1}{M}$ in the quasi-Monte Carlo method [13].

A desired improvement would be to obtain exponential convergence as the GLL method, but only be required to sample points directly in the domain as in the Monte Carlo method. We will develop such a quadrature rule using EIM in this chapter. Using EIM to perform numerical integration was first suggested in [15], an $\mathcal{O}(M^2)$ rule was developed in [1] and reiterated in [10]. It is based on approximating the desired integrand f by the usual EIM linear combination. The underlying basis functions are then integrated and used as weights.

$$\int_\Omega f(\boldsymbol{x})\,\mathrm{d}\Omega \approx \int_\Omega f_M(\boldsymbol{x})\,\mathrm{d}\Omega \tag{4.3}$$

$$= \int_\Omega \Big(\sum_{j=0}^{M} \beta_j^M q_j(\boldsymbol{x})\Big)\,\mathrm{d}\Omega \tag{4.4}$$

$$= \sum_{j=1}^{M} \beta_j^M \underbrace{\int_\Omega q_j(\boldsymbol{x})\,\mathrm{d}\Omega}_{\rho_j} \tag{4.5}$$

$$= \sum_{j=1}^{M} \beta_j^M \rho_j \tag{4.6}$$

$$= \underbrace{\int_\Omega \mathcal{G}(\cdot;\underline{\boldsymbol{\mu}}^T)\,\mathrm{d}\Omega\, T^Q}_{\underline{\rho}^T}\, \underline{\beta}^M \tag{4.7}$$

To compute (4.7), we need to obtain the interpolation coefficients $\underline{\beta}$, which is done in $\mathcal{O}(M^2)$ operations because of the lower triangular structure of B. This is the simplest version of the EIM quadrature, where we simply integrate the q-basis functions to obtain the quadrature weights ρ:

1. Compute the q-basis functions using EIM.
2. The q-bases are actually stored as linear combinations of $\mathcal{G}$, so the integrals of the q-basis functions are computed by integrating the snapshots $\mathcal{G}(\cdot;\boldsymbol{\mu}_i)$ and applying the transformation matrix T^Q to get the integrals of the q-basis functions and thus the weights $\underline{\rho}$.
3. Compute the EIM interpolation coefficients $\underline{\beta}$.
4. Compute the the weighted sum of the coefficients $\underline{\beta}$ and the weights $\underline{\rho}$ to obtain the final approximation.

An alternative approach to approximating integrals in inner products can be found in [2]. Their goal is to efficiently estimate inner products of results from the Re-

duced Basis method. Instead of using the q-basis functions they use nodal h-basis functions. This allows samples of the integrand to be used directly without having to solve for the β-basis coefficients. The result is that the online cost is reduced to $\mathcal{O}(M)$. The framework applied is that of the Discrete EIM [6], which is a formulation of EIM that uses vectors. Using the semi-analytical formulation presented in the last chapter we can change the basis from q to h to achieve the same computational advantage as follows

$$\int_\Omega f(\boldsymbol{x})\,\mathrm{d}\Omega \approx \int_\Omega \mathcal{G}(\cdot;\underline{\boldsymbol{\mu}}^T)\,\mathrm{d}\Omega\, T^Q \underline{\beta}^M \tag{4.8}$$

$$= \underbrace{\int_\Omega \mathcal{G}(\cdot;\underline{\boldsymbol{\mu}}^T)\,\mathrm{d}\Omega\, T^H}_{\underline{\rho}^T} f(\underline{\boldsymbol{x}}) \tag{4.9}$$

We will refer to this version of the EIM quadrature as the one where we first change to the h-basis and then integrate the basis functions.

1. Compute the q-basis functions using EIM.
2. Convert to h-basis functions.
3. The h-bases are actually stored as linear combinations of $\mathcal{G}$, so the integrals of the h-basis functions are computed by integrating the snapshots $\mathcal{G}(\cdot;\boldsymbol{\mu}_i)$ and applying the transformation matrix T^H to get the integrals of the h-basis functions and thus the weights $\underline{\rho}$.
4. Compute the nodal interpolation coefficients by simply sampling the target function in the magic points $f(\boldsymbol{x}_i)$.
5. Compute the the weighted sum of the coefficients $f(\underline{\boldsymbol{x}})$ and the weights $\underline{\rho}$ to obtain the final approximation.

Changing basis from q to h has been numerically unstable in some of our experiments using the semi-analytical EIM, therefore we will later present an alternative approach.

The previous work [1, 10, 2] suggest that the basis functions can be integrated with existing quadrature rules. This, of course, introduces the same limitations that we wanted to avoid by developing a new quadrature rule. The cost of integrating the basis functions is limited to the offline stage, and is therefore not critical for the online performance. However, in some cases the semi-analytic EIM will make it easier to compute these integrals. We will demonstrate its feasibility on arbitrary simple polygons, i.e. polygons that are non-overlapping and without holes, and on the circle and semicircle. Our focus will be on creating quadrature rules to integrate analytical non-parametrized functions over domains where traditional methods such as GLL quadrature is not well suited.

4.2 Choice of integration rule

If we were to integrate the basis functions using an existing numerical method, the trapezoidal rule and Simpson's method would yield algebraic convergence in 1D. GLL would yield exponential convergence for analytic integrands if we were able to find a suitable mapping from the reference domain onto the target domain. Otherwise we could triangulate the domain and approximate the target function over each triangle as e.g. linear polynomials, but the convergence rate would again be slow. In many cases we are therefore limited to algebraic convergence, which can be very expensive if we want highly accurate quadrature rules. We will therefore look at integrating each basis function analytically, or at least exactly. This is possible with the semi-analytical EIM if we are able to integrate each of the snapshots of the generating function $\mathcal{G}(\cdot;\cdot)$ separately. If so, we are able to integrate any function in the space it spans. We will use polynomials, e.g. $x^m y^n$ to span our approximation space. It is possible to integrate these polynomials analytically over the domains we look at.

4.3 Alternative derivation of the quadrature rule

As mentioned before, the straight forward change of basis from q to h and integrating the basis functions directly has been numerically unstable in practice when the semi-analytic EIM. We can circumvent the problem with the change of basis by implicitly integrating the basis functions. This is achieved by assuming that the generating functions selected by EIM, $\mathcal{G}(\cdot;\boldsymbol{\mu}_i)$, can be integrated exactly using a quadrature rule on the form

$$\int_\Omega \mathcal{G}(\cdot;\boldsymbol{\mu}_i)\,\mathrm{d}\Omega = \sum_{i=1}^{M} \rho_i \mathcal{G}(x_i;\boldsymbol{\mu}_i), \tag{4.10}$$

where the weights ρ_i are computed from nodal basis functions h_i,

$$\rho_i = \int_\Omega h_i(\boldsymbol{x})\,\mathrm{d}\Omega \qquad i = 1,\ldots,M. \tag{4.11}$$

The nodal basis functions h can be found explicitly by a change of basis as in the quadrature rule defined earlier. However, what we want is the weight ρ_i given by the integral of h_i, not the function h_i itself. Inaccuracies from first finding q and then h may be the problem with the first version of the quadrature rule. Because we know h exist and $\text{span}\{h_i\} = \text{span}\{\mathcal{G}(\cdot;\boldsymbol{\mu}_i)\}$, then we know that using the h_i basis would interpolate the generating functions exactly, and that (4.10) will thus integrate exactly. In addition, we assume that the generating function $\mathcal{G}(\cdot;\cdot)$ can be integrated analytically, as is the idea behind the semi-analytical formulation of

EIM. We can therefore build a system of linear equations where the weights ρ_i are the unknowns and the basis functions h_i are used implicitly

$$\int_\Omega \mathcal{G}(\cdot;\boldsymbol{\mu}_i)\,\mathrm{d}\Omega = \int_\Omega \Big(\sum_{j=0}^{M} \mathcal{G}(\boldsymbol{x}_j;\boldsymbol{\mu}_i) h_j(\boldsymbol{x})\Big)\,\mathrm{d}\Omega \qquad i = 1,\dots,M \tag{4.12}$$

$$= \sum_{j=0}^{M} \mathcal{G}(\boldsymbol{x}_j;\boldsymbol{\mu}_i) \int_\Omega h_j(\boldsymbol{x})\,\mathrm{d}\Omega \qquad i = 1,\dots,M \tag{4.13}$$

which written out becomes

$$\begin{bmatrix} \mathcal{G}(\boldsymbol{x}_1;\boldsymbol{\mu}_1) & \cdots & \mathcal{G}(\boldsymbol{x}_M;\boldsymbol{\mu}_1) \\ \vdots & \ddots & \vdots \\ \mathcal{G}(\boldsymbol{x}_1;\boldsymbol{\mu}_M) & \cdots & \mathcal{G}(\boldsymbol{x}_M;\boldsymbol{\mu}_M) \end{bmatrix} \begin{bmatrix} \int_\Omega h_1(\boldsymbol{x})\,\mathrm{d}\Omega \\ \vdots \\ \int_\Omega h_M(\boldsymbol{x})\,\mathrm{d}\Omega \end{bmatrix} = \begin{bmatrix} \int_\Omega \mathcal{G}(\cdot;\boldsymbol{\mu}_1)\,\mathrm{d}\Omega \\ \vdots \\ \int_\Omega \mathcal{G}(\cdot;\boldsymbol{\mu}_M)\,\mathrm{d}\Omega \end{bmatrix}, \tag{4.14}$$

or $\underline{A}\underline{\rho} = \underline{b}$ where $A_{ij} = \mathcal{G}(\boldsymbol{x}_j;\boldsymbol{\mu}_i)$, $\rho_j = \int_\Omega h_j(\boldsymbol{x})\,\mathrm{d}\Omega$ and $b_i = \int_\Omega \mathcal{G}(\boldsymbol{x};\boldsymbol{\mu}_i)\,\mathrm{d}\Omega$. We will refer to this version of the EIM quadrature as the one where we compute the weights in the h-basis implicitly.

1. Compute the q-basis functions using EIM.
2. Throw away the q-basis functions, but use the underlying snapshots $\mathcal{G}(\cdot;\boldsymbol{\mu}_i)$ to compute the integrals of the h-basis function and thus the weights $\underline{\rho}$ implicitly.
3. Compute the nodal interpolation coefficients by simply sampling the target function in the magic points $f(\boldsymbol{x}_i)$.
4. Compute the the weighted sum of the coefficients $f(\underline{\boldsymbol{x}})$ and the weights $\underline{\rho}$ to obtain the final approximation.

4.4 Quadrature on simple polygons in 2D

We have now presented three ways of performing quadrature using the semi-analytical framework. All of the variations require us to be able integrate the generating function $\mathcal{G}(\cdot;\cdot)$ over the domain, i.e. we need to be able to integrate

$$\int_\Omega \mathcal{G}(\boldsymbol{x};\boldsymbol{\mu}_i)\,\mathrm{d}\Omega, \qquad i = 1,\dots,M. \tag{4.15}$$

Even when using polynomials as basis functions this might not always be straight forward, not even when dealing with polygon domains. To integrate over simple

polygons, i.e. non-overlapping polygons without holes, we might need to partition the domain in order to obtain more manageable subdomains to integrate over. An alternative approach is to use Green's theorem to integrate over the boundary instead, see Figure 4.3. We will use Green's theorem to develop a method to integrate polynomials exactly over arbitrary simple polygons. By using this method in our EIM quadrature, we can develop quadrature rules for arbitrary simple polygons. Later we will demonstrate this method in practice on three polygonal domains: a triangle, a square and a flag. While the semicircle and circle require different analytical integration of the generating function $\mathcal{G}(\cdot;\cdot)$, all the polygons will share the same integration rule, which we now develop.

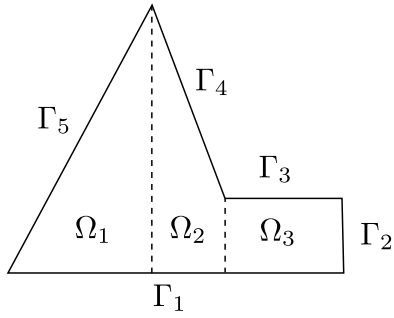

Figure 4.3: In order to integrate over a polygonal domain Ω we might want to partition the domain into subdomains Ω_i, $i = 1, 2, 3$, that would lead to simpler limits in the integrals. By exploiting Green's theorem we can instead convert the integral over the domain to an integral over the boundary segments Γ_i, $i = 1, \ldots, 5$.

We will apply Green's theorem to convert the integral in (4.15) to an integral over the boundary $\partial\Omega$ rather than over the domain Ω. In the case of polygons this will produce an exact integration rule for arbitrary simple polygons.

We start by stating Green's theorem [14]

$$\int_{\Omega} (\nabla \times F) \cdot \hat{k} \, \mathrm{d}\Omega = \oint_{\partial\Omega} F \cdot \hat{t} \, \mathrm{d}\gamma, \tag{4.16}$$

where $\hat{k}$ is the unit vector in the same direction as $\nabla \times F$, and $\hat{t}$ is the tangential unit vector. In order to apply Green's theorem we construct the artificial vector field F, with two components L and G,

$$F = \begin{bmatrix} L \\ G \end{bmatrix}. \tag{4.17}$$

We want to integrate $\mathcal{G}(\boldsymbol{x}; \boldsymbol{\mu}_i)$ over the domain using (4.16) and therefore need our

artificial vector field F to fulfil $(\nabla \times F) \cdot \hat{k} = \mathcal{G}(\boldsymbol{x}; \boldsymbol{\mu}_i) = g(x, y)$, which is achieved by choosing

$$F = \begin{bmatrix} 1 \\ \int g(x,y)\,\mathrm{d}x \end{bmatrix} = \begin{bmatrix} 1 \\ G \end{bmatrix} \tag{4.18}$$

By applying Green's theorem and integrating over each boundary segment Γ_k separately we get

$$\begin{aligned} \int_\Omega \mathcal{G}(\boldsymbol{x}; \boldsymbol{\mu}_i)\,\mathrm{d}\Omega &= \int_\Omega (\nabla \times F) \cdot \hat{k}\,\mathrm{d}\Omega && (4.19)\\ &= \sum_{k=1}^{K} \oint_{\Gamma_k} F \cdot \hat{t}\,\mathrm{d}\gamma && (4.20)\\ &= \sum_{k=1}^{K} \int_{\Gamma_k} (\hat{t}_x + G\hat{t}_y)\,\mathrm{d}\gamma && (4.21)\\ &= \sum_{k=1}^{K} \int_{-1}^{1} (\hat{t}_x + G(\boldsymbol{x}_{\Gamma_k}(s))\,\hat{t}_y) \frac{\mathrm{d}\gamma}{\mathrm{d}s}\,\mathrm{d}s && (4.22) \end{aligned}$$

In the last step a change of variable is made by parametrizing each boundary segment Γ_k using $\boldsymbol{x}_{\Gamma_k}(s)$. We have now changed the original integral over the domain to instead integrate, using a single variable, over each boundary segment. It is worth noting that the result so far is not limited to polynomial generating functions and polygonal domains. However, applying the result to other generating functions and domains can be more difficult in terms of dealing with the integrals and the mapping $\boldsymbol{x}_{\Gamma_k}$, which is why we now focus on polynomials.

When choosing $\mathcal{G}(\cdot, \cdot) = x^m y^n$ as the generating function, $\int g(x, y)\,\mathrm{d}x$ can be found analytically and $G(\boldsymbol{x}(s))$ will be a polynomial of degree of at most $2M + 1$, , so GLL quadrature of order $M+1$ will integrate it exactly [4]. Computing the integral analytically can be a bit tedious, which is why we choose to use GLL quadrature instead. Our exact integration rule for $x^m y^n$ on arbitrary simple polygons in 2D is thus

$$\int_\Omega \mathcal{G}(\boldsymbol{x}; \boldsymbol{\mu}_i)\,\mathrm{d}\Omega = \sum_{k=1}^{K} \frac{|\Gamma_k|}{2} \sum_{j=1}^{M+1} \omega_j (\hat{t}_x + G(\boldsymbol{x}_{\Gamma_k}(s_j))\,\hat{t}_y) \tag{4.23}$$

where K is the total number of edges, $|\Gamma_k|$ the length of edge k, ω_j is the jth GLL weight, and s_j the jth GLL point. Similar results might be possible in higher dimensions by via Stoke's theorem.

Index i	Function $q_i(x, y)$
1	1.00
2	0.50 + 0.50x
3	0.50 + 0.50y
4	1.00 - 1.00x^2
5	1.00 + 1.00x + 1.00y + 1.00xy

Table 4.1: The five first basis functions $q_i(x, y)$ from ascending EIM on a triangular domain using monomial generating functions ($x^m y^n$).

Index i	Function $q_i(x, y)$
1	1.00
2	0.50 + 0.50x
3	1.00y
4	1.00 - 1.00y - 1.00x^2
5	2.00xy

Table 4.2: The five first basis functions $q_i(x, y)$ from ascending EIM on a semicircular domain using monomial generating functions ($x^m y^n$).

4.5 Numerical results

We will apply EIM quadrature to a line, a square, a semicircle, a circle and a flag. For the domains that have not been used in previous results, we will also include their magic points, basis functions and EIM interpolation error.

It is interesting to observe that, in Figure 4.5, EIM reaches machine precision in about 4 times as many interpolation points as GLL in 2D. Compared to Figure 4.4, we see that in 1D the ratio is a bit less than 2. It would be interesting to see if this has anything to do with GLL quadrature being exact for polynomials of degree $2M + 1$, while the EIM method is only exact for polynomials of degree M.

The different plateaus of the convergence in e.g. Figure 4.7 suggests that some of the basis functions do not contribute to decrease the error further. We will look at a way to remove superfluous basis functions when discussing encoding in the next chapter.

The main takeaway from the numerical results is that both EIM interpolation and EIM quadrature works with exponential convergence for the various domains. However, the integration rule introduced for arbitrary simple polygons seems to introduce some rounding error as shown in Figure 4.8. A more serious problem is that EIM chooses the same magic point twice for both the triangle and the semicircle. This will be discussed further in the next section.

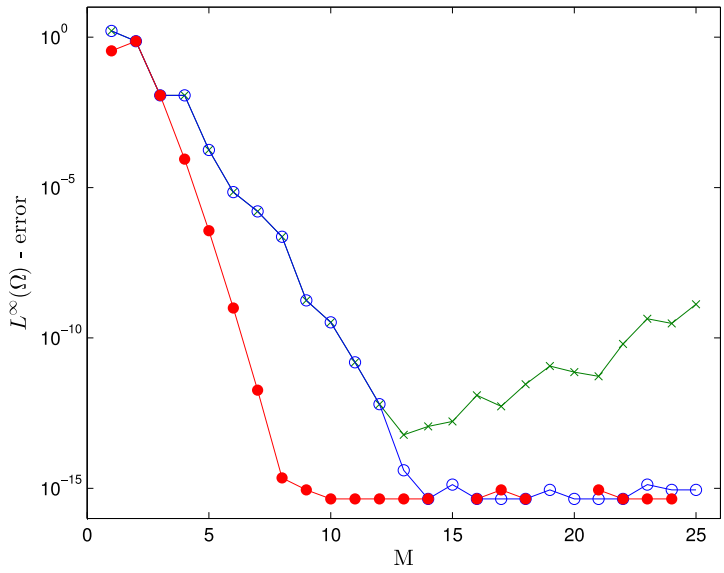

Figure 4.4: Convergence of the quadrature error when approximating $\int_{-1}^{1} e^x \, \mathrm{d}x$ with M quadrature points. Standard semi-analytical EIM quadrature (green crosses) is unstable. The alternative semi-analytical EIM (blue circle) is stable but not as good as GLL (red dots). Some of the GLL marks are missing because the error is 0 to machine precision in those points. The grid spacing is $\Delta = 4 \cdot 10^{-3}$.

4.6 Issues

Initially, the semi-analytical implementation of EIM seemed to produce good results for various applications. That is, the method yielded exponential convergence for the various test cases, even down to machine precision given enough magic points. However, when the final results where to be produced issues started to occur when the error approached machine precision. When looking into the issues it became clear that the error was quite severe in certain cases.

As mentioned, the semi-analytical implementation of EIM produces exponential convergence down to machine precision for many test cases. Such results would normally not be possible with even small implementation issues. It was therefore reasonable to believe that the implementation was correct. The triangular domain turned out to cause problems both when performing numerical integration and when solving partial differential equations. The interpolation error in the magic points should, by construction, be zero. This is because we require our approximation to be equal to the function we are interpolating in the magic points. Figure 4.18 show that the error is far from zero for the triangular domain. The error is so

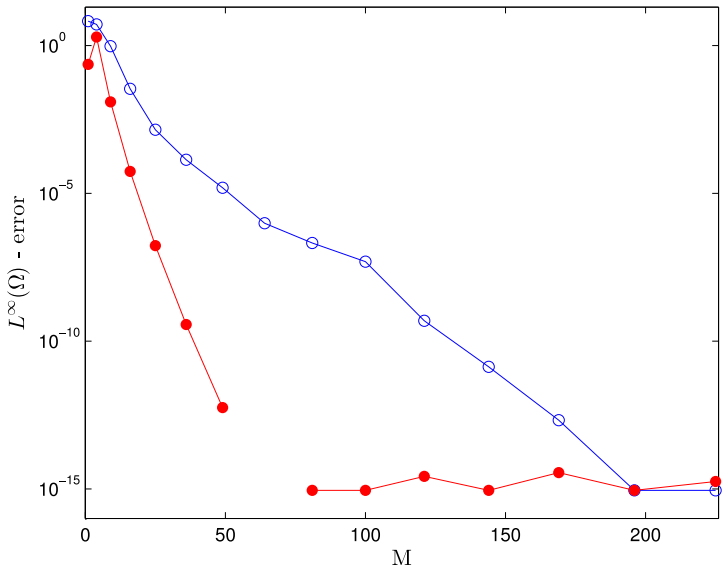

Figure 4.5: Convergence of the quadrature error when approximating $\int_\Omega e^{xy} \mathrm{d}\Omega$ on a square domain using M quadrature points. GLL quadrature (red dots) performs better than the EIM quadrature (blue circles). The missing GLL point is because the error is zero to machine precision. The grid spacing is $\Delta = 5 \cdot 10^{-2}$.

severe in the magic points that when training the EIM on the triangle, the error in the magic points are occasionally larger than elsewhere in the training set. The EIM will then choose some of the magic points more than once. This should of course never happen, but leads to singular matrices in a couple of the applications. The EIM could be implemented not to choose the same magic point more than once, but this would not solve the problem of large errors. The reason why changing basis from q to the nodal basis h and then integrating the basis functions does not work might also be due to the same issues. Early computations using discrete basis vectors did seem to work fine when changing the basis.

The error in Figure 4.18 starts at zero and increases gradually. This is to be expected if the error is related to accumulation of round-off errors. The semi-analytical formulation is centred around the generating function $\mathcal{G}(\cdot, \cdot)$ rather than the q basis functions which are used in the traditional formulation. To express a single q-basis function we therefore need to take a linear combination of the generating functions. We recall from (3.4) that this could be written $q_j = \sum_{i=1}^{M} \alpha_{ij}^{Q} \mathcal{G}(\cdot; \boldsymbol{\mu}_i)$. If this way of dealing with the basis functions introduces round-off errors, the error should be reduced if the generating functions are used directly rather than using the q basis functions. As Figure 4.19 shows, this is indeed the case for the triangle. Figures 4.20 and 4.21 show that this is also the case for the square shaped domain.

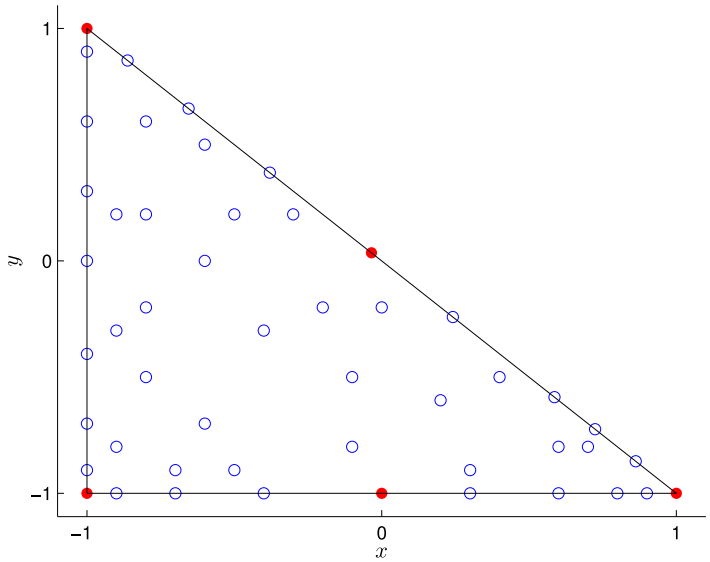

Figure 4.6: Distribution of the first 50 spatial magic points on a triangular domain. The red dots mark the first five magic points.

Using the generating functions directly seems to solve several of problems encountered: no duplicate magic points are chosen and the convergence seems to be better behaved for the cases tested. It does not, however, solve the problem in general as the error is still large when using the generating functions directly. It is worth noting that when using $\mathcal{G}(\cdot;\cdot)$ directly in the computations, the systems of linear equations are ill-condition, which could introduce the errors that are observed.

Another potential source of issues is that the approximations spaces in 2D are generated by all polynomials $x^m y^n$ of degree $m+n \leq N_1$, but some of the polynomials are missing due to a bug. The bug added the extra requirement $m, n \leq N_2$, where $N_2 < N_1$ causing polynomials like $x^0 y^{N_1}$ to be missing. The missing polynomials

Index i	Function $q_i(x, y)$
1	1.00
2	0.50 - 0.50x
3	-1.00y
4	0.50 + 0.50y - 0.50x^2
5	-2.00xy

Table 4.3: The five first basis functions $q_i(x, y)$ from ascending EIM on a circular domain using monomial generating functions ($x^m y^n$).

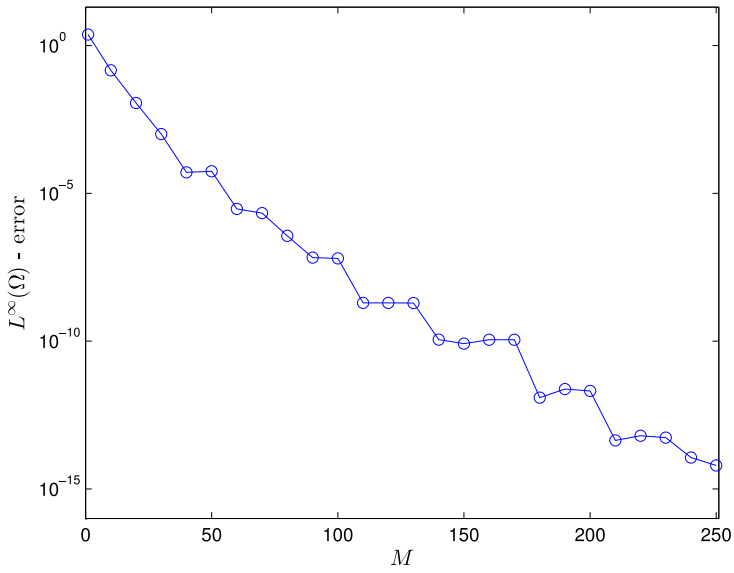

Figure 4.7: Convergence of the interpolation error of e^{xy} on a triangular domain as a function of magic points M. The grid spacing is $\Delta = 5 \cdot 10^{-2}$.

would be replaced by higher degree polynomials.

This is an issue that deserves further investigation, but time did not allow us to do so here. The reader should keep in mind that some of the figures are deceiving when it comes to the problems highlighted in this section which is why they were discovered so late.

Index i	Function $q_i(x, y)$
1	1.00
2	0.50 + 0.25x
3	0.50 + 0.50y
4	1 - 0.25x^2
5	0.25 - 0.125x + 0.25y - 0.125xy

Table 4.4: The five first basis functions $q_i(x, y)$ from ascending EIM on a flag-shaped domain using monomial generating functions $(x^m y^n)$.

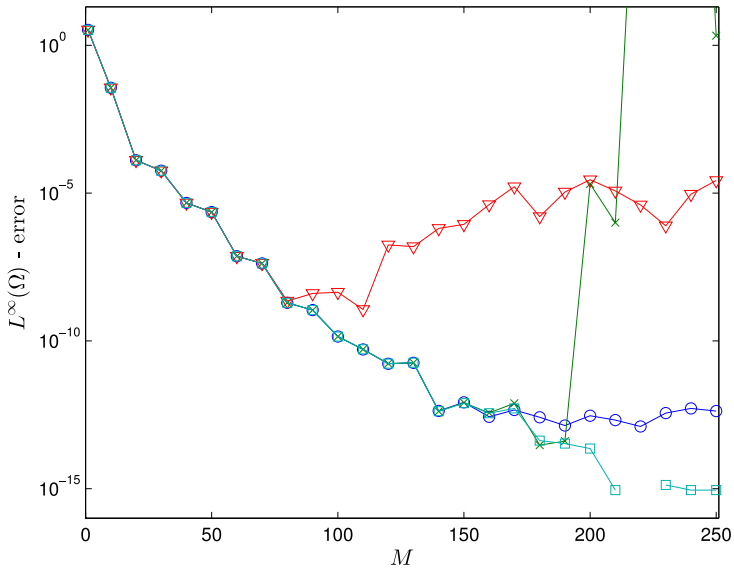

Figure 4.8: Convergence of the quadrature error of $\int_\Omega e^{xy} \mathrm{d}\Omega$ on a triangular domain as a function of magic points M. The initial EIM quadrature using the h-basis functions (red triangles) is unstable. The alternative EIM quadrature using the h-basis functions implicitly (green crosses) does not work here because the EIM basis is faulty (it chose the same magic point twice). Better results are achieved when interpolating and integrating the q-basis functions directly using the generic polygon domain integration (blue circles) or analytically for the triangle (cyan squares). The grid spacing is $\Delta = 5 \cdot 10^{-2}$.

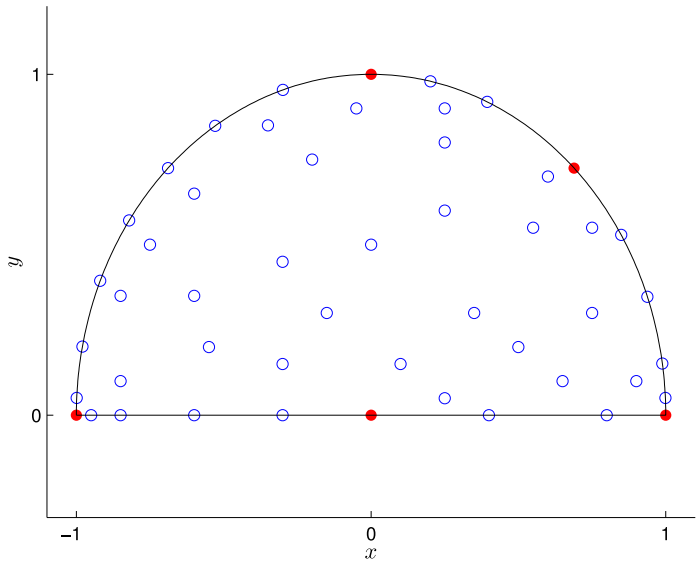

Figure 4.9: Distribution of the first 50 spatial magic points on a semicircular domain. The red dots mark the first five magic points.

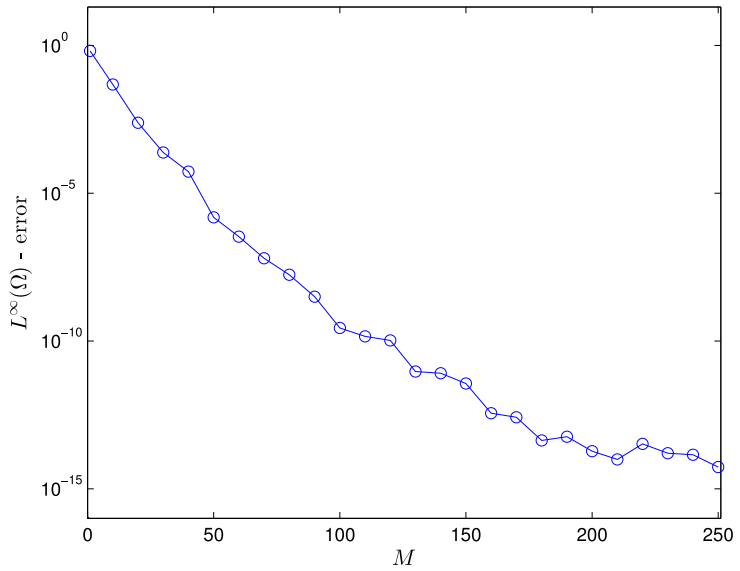

Figure 4.10: Convergence of the interpolation error of e^{xy} on a semicircular domain as a function of magic points M. The grid spacing is $\Delta = 5 \cdot 10^{-2}$.

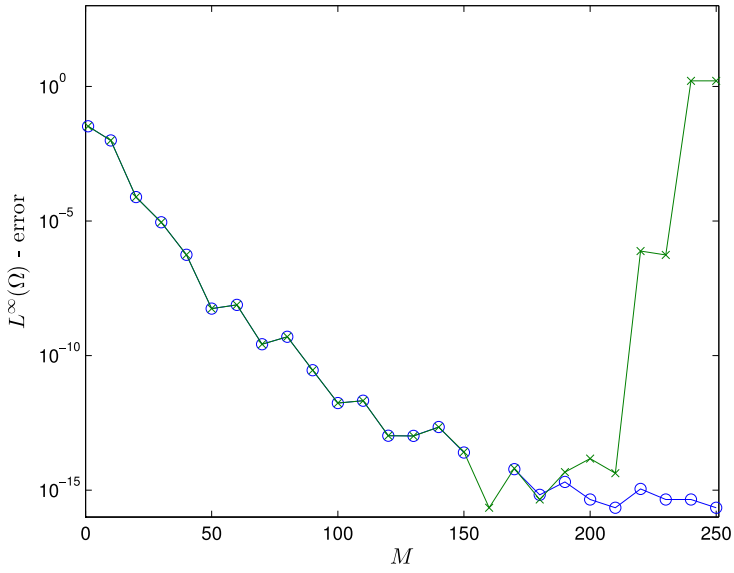

Figure 4.11: Convergence of the quadrature error of $\int_\Omega e^{xy} \mathrm{d}\Omega$ on a semicircular domain as a function of magic points M. The alternative EIM quadrature, using the h-basis functions implicitly (green crosses) fails due to a faulty EIM basis (EIM chose the same magic point twice). However, interpolating and integrating the q-basis functions directly (blue circles) works. The grid spacing is $\Delta = 5 \cdot 10^{-2}$.

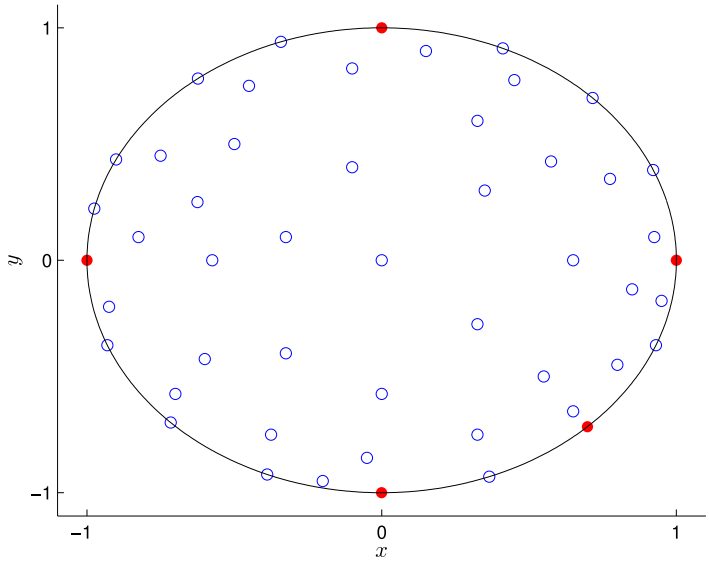

Figure 4.12: Distribution of the first 50 spatial magic points on a circular domain. The red dots mark the first five magic points.

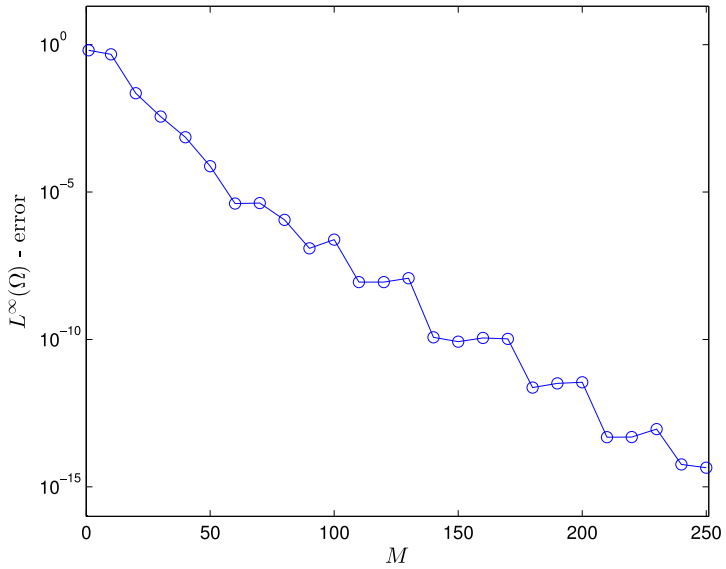

Figure 4.13: Convergence of the interpolation error e^{xy} on a circular domain as a function of magic points M. The grid spacing is $\Delta = 5 \cdot 10^{-2}$.

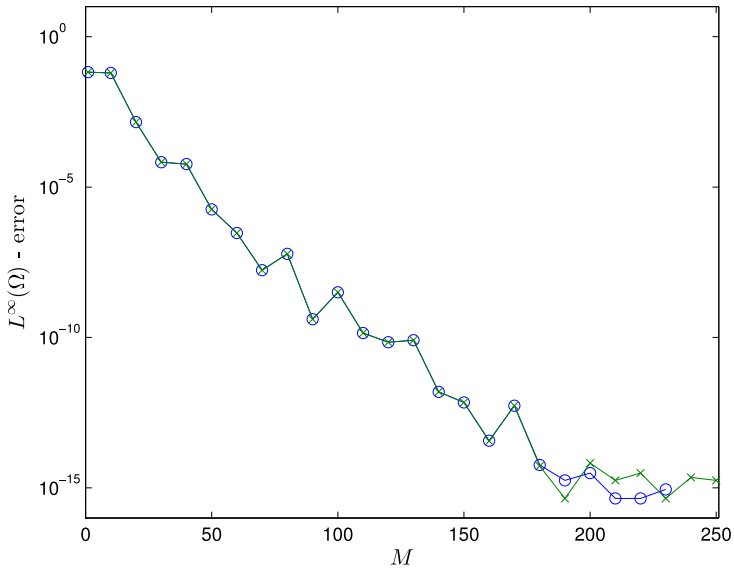

Figure 4.14: Convergence of the quadrature error of $\int_\Omega e^{xy} \mathrm{d}\Omega$ on a circular domain as a function of the magic points M. Both the alternative EIM quadrature (green crosses) and integrating the q-basis functions directly (blue circles) works well. The grid spacing is $\Delta = 5 \cdot 10^{-2}$.

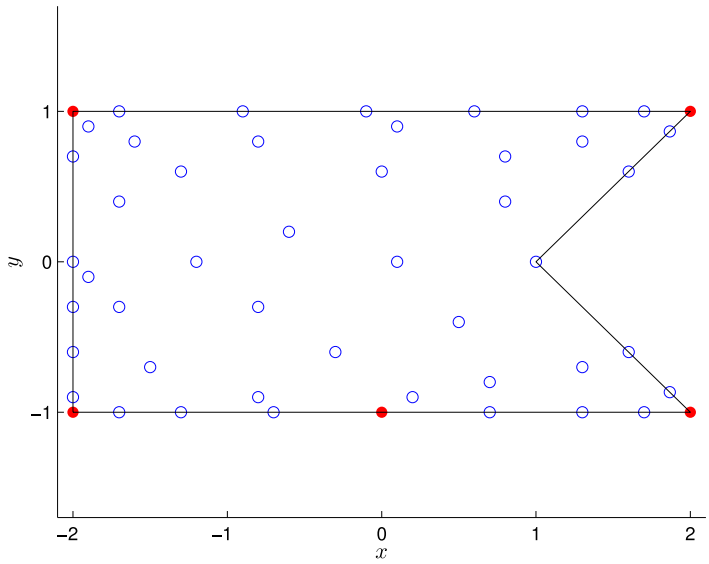

Figure 4.15: Distribution of the first 50 spatial magic points on a flag-shaped domain. The red dots mark the first five magic points.

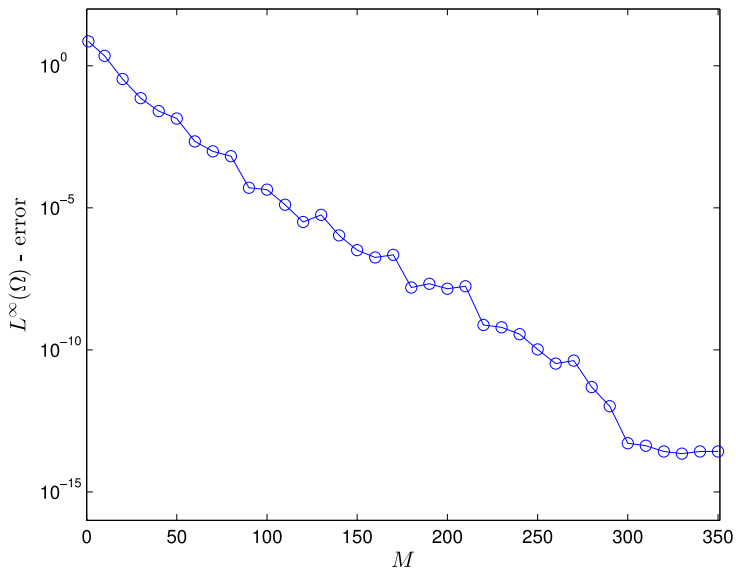

Figure 4.16: Convergence of the interpolation error of e^{xy} on a flag-shaped domain as a function of magic points M. The grid spacing is $\Delta = 2 \cdot 10^{-2}$.

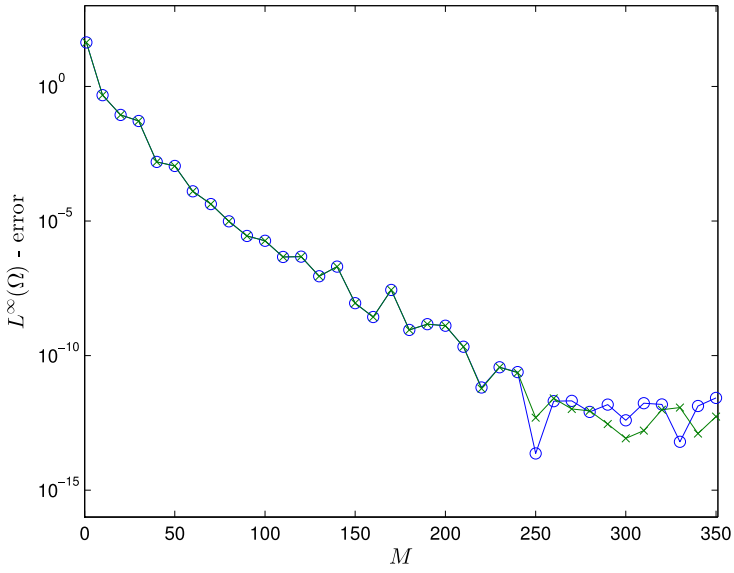

Figure 4.17: Convergence of the quadrature error of $\int_\Omega e^{xy} \mathrm{d}\Omega$ on a flag-shaped domain as a function of the magic points M. Both the alternative EIM quadrature (green crosses) and integrating the q-basis functions directly (blue circles) works well, except that they seem to converge at a higher error level than the other domains. The grid spacing is $\Delta = 2 \cdot 10^{-2}$.

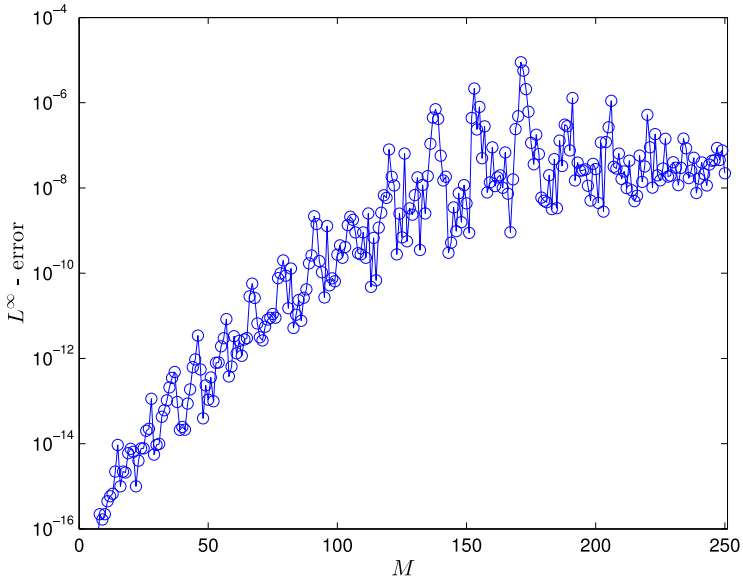

Figure 4.18: Interpolation error in the magic points for EIM using q-basis functions on a triangular domain. The grid spacing is $\Delta = 5 \cdot 10^{-2}$.

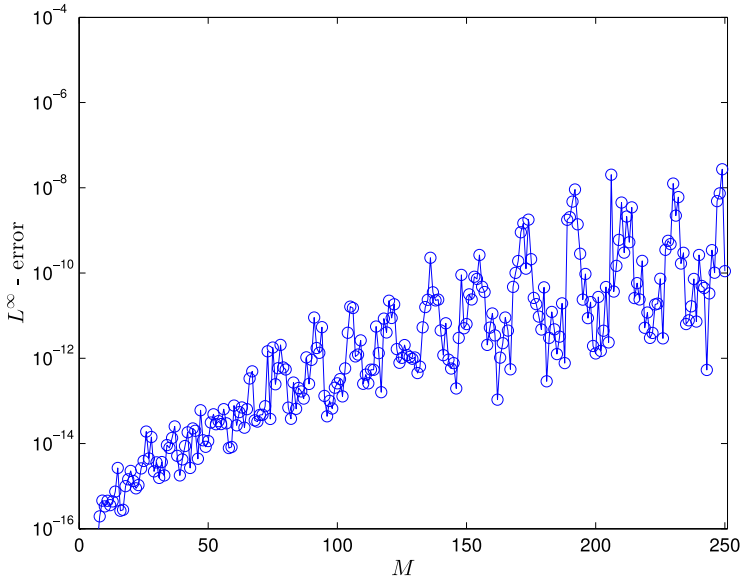

Figure 4.19: Interpolation error in the magic points for EIM using $\mathcal{G}$-basis functions on a triangular domain. The grid spacing is $\Delta = 5 \cdot 10^{-2}$.

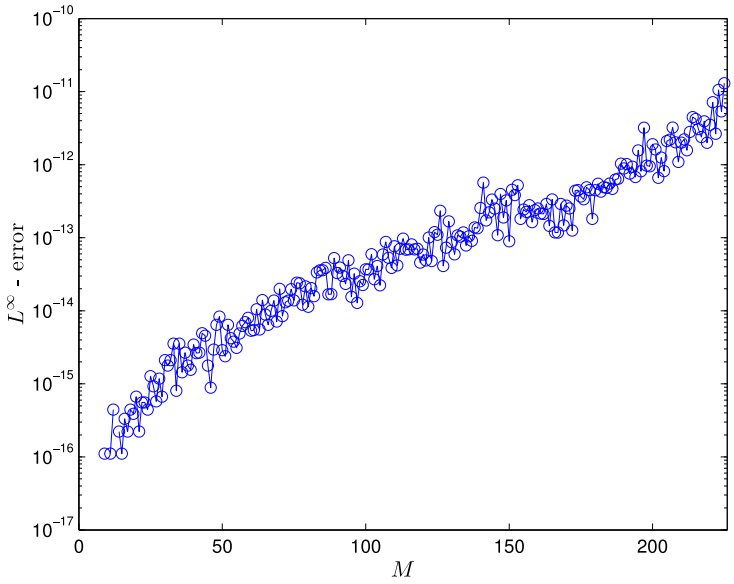

Figure 4.20: Interpolation error in the magic points for EIM using q-basis functions on a square domain. The grid spacing is $\Delta = 5 \cdot 10^{-2}$.

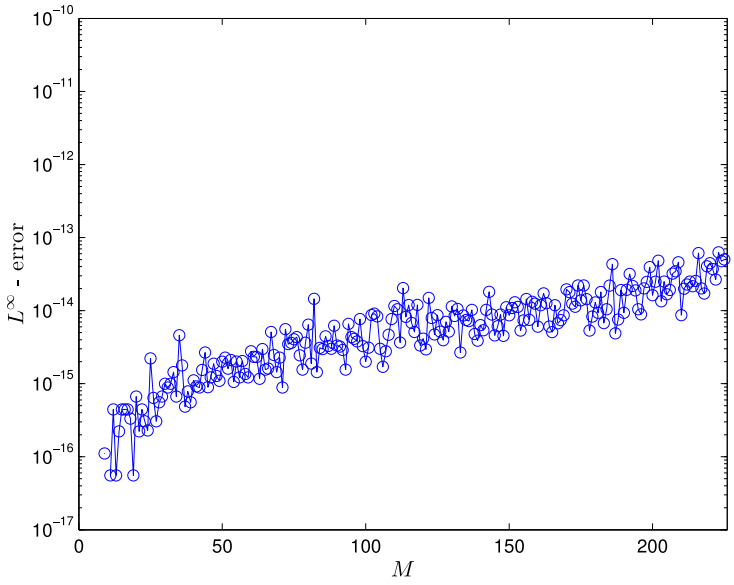

Figure 4.21: Interpolation error in the magic points for EIM using $\mathcal{G}$-basis functions on a square domain. The grid spacing is $\Delta = 5 \cdot 10^{-2}$.

Chapter 5

Compression

Compression can be desirable in the presence of limited bandwidth, storage or computational resources. One scenario is transmitting data from a server to a client where resources may be limited or even varying. Another is storing large amounts of data, and a third can be to manage complex models by pre-computing results and storing them in an efficient manner. Our model problem will be from 3D animation, but as usual our only assumption is that the underlying data is sufficiently smooth. At the end of the chapter we will suggest how the semi-analytical framework could be used in field reconstruction, e.g. to make a computational surrogate of 3D animation that could be sampled continuously in time and space.

5.1 Introduction

It was suggested that EIM could be applied to animation in [15]. For the first time in this thesis we will construct our approximation space from the data we want to approximate. As mentioned, EIM was initially developed to approximate parametrized functions $f(\boldsymbol{x}; \boldsymbol{\mu})$, using snapshots of the function itself to generate the approximation space. We will view the 3D animation as nothing more than a parametrized function that deforms a 3D model in time (the parameter) and space (the domain). The generating function $\mathcal{G}(\boldsymbol{x}; t)$ is thus the displacement in time, given by t, and space, given by $\boldsymbol{x}$. By approximating the displacement using EIM, we can view each snapshot in time as a linear combination of some other snapshots that were chosen by EIM. The only benefit the semi-analytical framework could have, apart from making the compression simple to implement, is that if we have already stored the generating function $\mathcal{G}(\cdot; \cdot)$ we can reuse those snapshots directly (by taking linear combinations of them) rather than storing new basis functions q. The semi-analytical framework is thus of limited use, but it will play a more important role in the field reconstruction presented at the end of this chapter.

There is of course nothing special with 3D animation, and the generating function could just as easily describe other sufficiently smooth data.

5.2 Encoding

Before looking specifically at compression of 3D animation we will look at compression in general. In particular, we will look at how data can be encoded (represented), using the semi-analytical EIM framework, to achieve reductions in the computational costs at different stages. Because approximations produced by EIM are written

$$f(\cdot) \approx \sum_{i=1}^{M} \alpha_i^f \mathcal{G}(\cdot; \boldsymbol{\mu}_i), \tag{5.1}$$

where $\mathcal{G}(\cdot;\cdot)$ can both be continuous or discrete, we can approximate a function without having access to B, T^Q or T^H. All we need are the coefficients $\underline{\alpha}^f$ and the corresponding magic parameters $\underline{\boldsymbol{\mu}}$. The tuple $(\underline{\alpha}, \underline{\boldsymbol{\mu}})_{\mathcal{G}_M}^M$ is referred to as the encoding of f using the basis $\mathcal{G}_M$ and all M coefficients. An approximation $(\underline{\alpha}, \underline{\boldsymbol{\mu}})_{\mathcal{G}_M}^m$, $m < M$ is given by omitting $(M - m)$ pairs $(\alpha, \boldsymbol{\mu})$ that contribute the least to the approximation of f. Some of the α's might be very small or even zero, so that a sufficient accuracy can be obtained without including them. By sorting the α's, taking the maximum of each corresponding basis function into account, the least significant contributions can be removed.

The potential savings of this approach is particularly visible when encoding an even trigonometric function as shown in Table 5.1. The most significant contributions are made by even functions with an odd number of periods. These types of functions make up a fourth of the total approximation space, and their contributions decrease exponentially. Good approximations can thus be achieved by omitting coefficients after the function has been interpolated. This illustrates that the bases, especially the generic ones, can contain information that is not useful for the particular function we want to approximate. Encoding can help remove some of this superfluous data.

We now have four levels of reducing the computational cost:

1. When computing the EIM basis, fewer magic points/basis functions can be included, i.e. by exiting the algorithm early.
2. When interpolating, the nested structure of EIM enables us to easily use fewer magic points/basis functions.
3. When encoding, the coefficients from the interpolation are sorted by contribution and only the most significant can be stored/transmitted.

Contribution α	Snapshot of $\mathcal{G}(\cdot;\cdot)$
6.7e-01	$\cos(3\pi x)$
5.7e-01	$\cos(\pi x)$
1.0e-01	$\cos(5\pi x)$
6.8e-03	$\cos(7\pi x)$
2.5e-04	$\cos(9\pi x)$
5.9e-06	$\cos(11\pi x)$
9.5e-08	$\cos(13\pi x)$
1.1e-09	$\cos(15\pi x)$
1.1e-11	$\cos(17\pi x)$
7.7e-14	$\cos(19\pi x)$

Table 5.1: The most significant contributions to the encoding of $\sin(\pi\cos(\pi(x+1)))$. The function itself is even and all of the most significant contributions are also even, with an odd number of periods. EIM was trained on both sine and cosine functions.

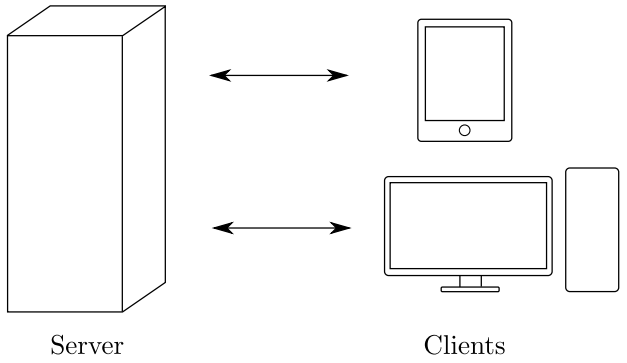

Figure 5.1: Server client model. The server is usually resourceful, but is serving multiple clients, each of which may have very limited resources.

4. When sampling, the least significant coefficients from the encoding can be omitted (because they were sorted in the last step).

The server client model is illustrated in Figure 5.1. One scenario can be to compute an animation on the server and transmit it for viewing on the clients. The server is resourceful and can manage the full animation, while the clients can only receive and playback data of limited complexity. The server computes the animation and compresses it using steps 1 through 3. The reduction in computational cost in these steps can vary with the available resources of both the server and the clients from time to time. The resulting encoding of the animation is then transmitted to one or more of the clients. If a client is unable to receive the whole encoding, or does not have time to process the whole encoding, step 4 can be applied to reduce the complexity further.

5.3 Compression of 3D animation

We will now apply EIM to compression of 3D animation. Our domain will be the flag used earlier. The flag itself is a discrete polygon in 2D (Figure 5.2), but it is deformed in 3D (Figure 5.3). The animation is also discrete in time. The displacement for a given point t in time is given by $\mathcal{G}(\cdot;t) : \Omega \to \mathbb{R}^3$, where the domain Ω is the reference flag in 2D and the displacement is given as a vector in 3D. The greedy version of EIM will select the snapshots in time that best represent the whole sequence. The EIM will be applied separately in each spatial direction. The flag was modelled and animated, using a wind simulation, in a software package used to create 3D animation for films and games. However, the initial results were not particularly good. The problem might have been that the spatial or temporal resolution was not sufficient for the simulation to produce sufficiently smooth data, but we did not have time to look into this, so we will only present some results using synthetic animation. Each animation will be a sequence of displacement snapshots in time called frames. The error is measured as the maximum error in any vertex in any frame of the sequence.

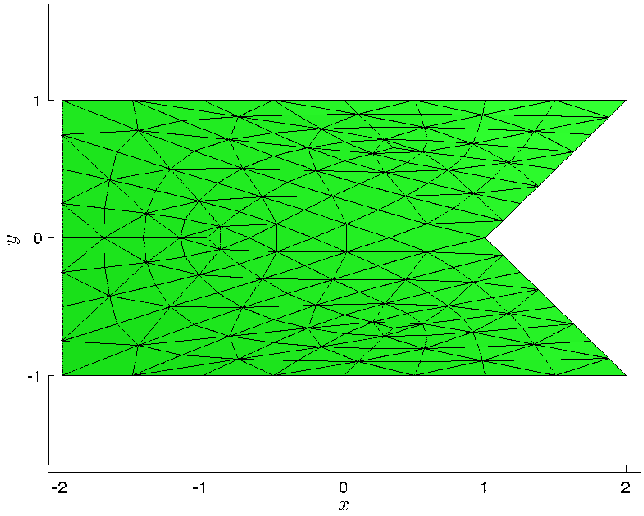

Figure 5.2: The flag model shown in its neutral pose.

The synthetic animations is designed to show the feasibility of applying EIM to animation. The first example displaces each vertex in time as a phase shifted trigonometric function. Each component (x, y and z) of the displacement is given by a simple sine or cosine that is phase shifted with time. As expected, the compression converges with 3 magic points in each dimension as show in Figure 5.4.

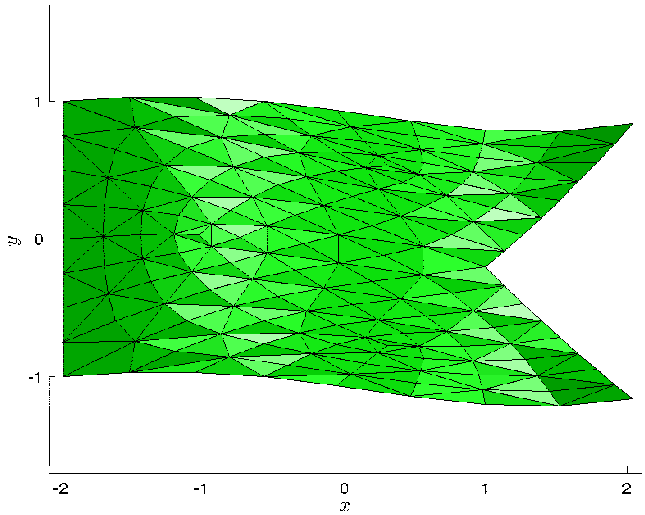

Figure 5.3: The flag model displaced synthetically in 3D.

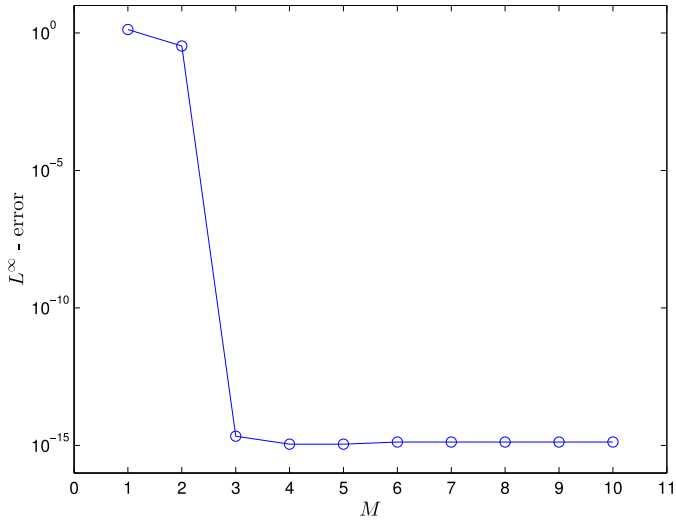

Figure 5.4: Convergence of the compression error of a simple synthetic flag animation as the number of magic points M in each of the 3 dimensions. The error is measured as the maximum error in each vertex and over all the frames in the sequence. The synthetic animation sequence consists of 100 frames. As expected, the compression converges in 3 magic points.

The second example is also based on phase shifted trigonometric functions, but this time the amplitude has an exponential dependency to make the convergence rate slower. The convergence of the error is shown in Figure 5.5. The accompanying magic points are shown Figure 5.6. Although the convergence looks good, EIM actually selects the same magic point twice towards the end. This also happened for the square and semicircular domains earlier. But this time it might not be as severe because the problem only occurred after machine precision was reached. In this case it might therefore be enough to limit EIM from selecting the same magic point twice.

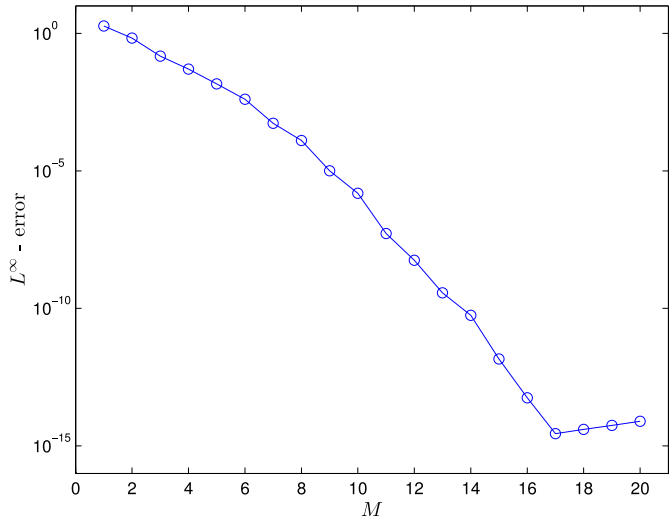

Figure 5.5: Convergence of the compression error of a synthetic flag animation as the number of magic points M in each of the 3 dimensions. The error is measured as the maximum error in each vertex and over all the frames in the sequence. The synthetic animation sequence consists of 200 frames.

The EIM has the potential of compressing 3D animation with an exponentially decreasing error with the number of magic points. Combined with the encoding and transmission protocol outlined in the previous section, this could be useful in practical applications. Another benefit of representing the animation using EIM can be to automatically detect loops by only comparing the EIM coefficients. Creating animation loops can be desirable in games and virtual reality to reduce the cost of some repeating animation.

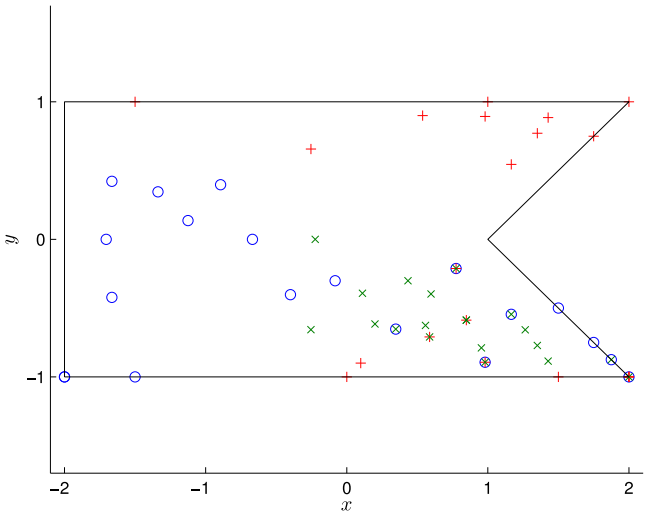

Figure 5.6: The spatial magic points of a synthetic flag animation with 200 frames. EIM chooses different magic points for each of the 3 dimensions, x (blue circles), y (red plus signs), z (green crosses).

5.4 Field reconstruction

In what follows, we will use 3D animation as an example, but the ideas extend to other types of problems. The parameter space will be time and the domain will be related to physical space. The animation might have been computed at some discrete time steps and in some discrete spatial points. This was the case in the compression example. We might want to create a surrogate of the original animation, with the possibility to sample continuously in time and space. By describing the surface as a function in space rather than as a discrete polygon we can avoid sharp edges showing up if the surface is supposed to be smooth. By being able to sample continuously in time we are able to change the playback rate of the animation. But more importantly, this could be used to create computational surrogates for any sufficiently smooth parametric function $\mathcal{G}(\cdot;\cdot)$. It was proposed to apply EIM to this application in [15]. The computational surrogates could be cheaper to use than the original model making them ideal for e.g. optimization and real time applications. It might even be possible to construct good approximations to parametrized partial differential equations. EIM would thus be applied to the same problem as the Reduced Basis method, which EIM was originally developed to solve a sub-problem of. We will now sketch out how the semi-analytical framework could be used to create computational surrogates.

Let us again look at the flag animation. The displacement for a given point t in

time is given by $\mathcal{G}(\cdot;t) : \Omega \to \mathbb{R}^3$. Previously, a discrete domain $\hat{\Omega}$ that consisted of the vertices in the triangulation was used. We now want to use the continuous domain Ω instead in order to be able to sample the displacement in any point. In the quadrature chapter we developed interpolation over the flag-shaped domain. By using the same semi-analytical EIM basis functions and magic points we can interpolate the displacement separately in each spatial dimension, x, y and z. The only difference is that we want to interpolate a component of the displacement, e.g. $\mathcal{G}_x$ rather than the function e^{xy}. The resulting approximation will enable us to sample the animation in any spatial point for a given point in time. With 3 components and M magic points, this will require $3M$ coefficients for each point in time. This should be cheaper than using a fine discrete grid with $\mathcal{M}$ points and a storage cost of $3\mathcal{M}$. Let $\mathcal{N}$ be the original number of snapshots in time. If the development in time of the spatial coefficients are sufficiently smooth, then we can interpolate each of the $3M$ coefficients separately in time. By training EIM in 1D, as with interpolation and quadrature, we can find basis functions and magic points to interpolate each coefficient. Using N magic points in time for each spatial coefficients will require $3MN$ temporal coefficients in total. The ratio between the storage cost of the continuous and discrete versions is thus $\frac{MN}{\mathcal{M}\mathcal{N}}$. To sample the function at a given point in time and space, we first use the temporal coefficients to compute the values of the spatial coefficients. We then use the spatial coefficients to recover the actual displacement. Implementing the method mainly involves reuse of the existing EIM functionality and bookkeeping of the coefficients. The work was started, but we did not have time to test the method properly.

To summarize, we have reused the generic results when interpolating over the flag-shaped domain and in 1D to be able to sample continuously in space and time. The magic points in 1D are used to decide at what points in time the spatial coefficients should be computed. The spatial coefficients are computed using EIM on the flag-shaped domain. The spatial coefficient are then interpolated in time to produce N temporal coefficients for each spatial coefficient. Sampling the surrogate in space and time is done by first using the temporal coefficients to approximate the spatial coefficient, which are again used to sample the actual geometry. By changing the spatial domain (flag) and parameter domain (1D time), the same method could be used for other sufficiently smooth parametrized functions.

If the different components of the generating function $\mathcal{G}(\cdot;\boldsymbol{\mu}) : \Omega \to \mathbb{R}^3$ are not independent, then it might be possible to find alternative ways to using 3 separate approximations as we have done until now. One approach could be to relax the requirement that EIM is to interpolate exactly in each magic point and rather formulate the selection of the coefficients $\underline{\beta}$ as an optimization problem. So instead of solving

$$\sum_{j=1}^{M} q_j(\boldsymbol{x}_i)\beta_j = f(\boldsymbol{x}_i) \qquad i = 1, \ldots, M. \tag{5.2}$$

where q_i are basis functions, f is the target function, $\boldsymbol{x}_i$ is the magic points and $\underline{\beta}$ is the desired coefficients, it might work to solve

$$\underline{\beta} = \arg \min_{\underline{\beta} \in \mathbb{R}^M} \max_{\boldsymbol{x}_i \in \underline{\boldsymbol{x}}} \left\| \sum_{j=1}^{M} q_j(\boldsymbol{x}_i)\beta_j - f(\boldsymbol{x}_i) \right\|_{L^2(\Omega)} \tag{5.3}$$

where $\underline{\boldsymbol{x}}$ are the magic points.

Chapter 6

Solving Partial Differential Equations

In this chapter we present a new collocation method to solve partial differential equations directly on arbitrary domains by using bases given by the semi-analytical formulation of EIM. The required system of linear equations is easy and inexpensive to set up and yields exponential convergence, for smooth problems, in the number of magic points used. Because the procedure uses the strong formulation of the problem, it is straight forward to change the governing equation and the boundary conditions.

6.1 Derivation

Finite Element Methods (FEM) and Spectral methods are two common approaches for solving partial differential equations that both utilize a weak formulation and a Galerkin projection [21]. The weak formulation lowers the smoothness requirement on the basis functions, allowing e.g. the Poisson problem to be solved with linear basis functions. The weak formulation also allows for easy stitching of subdomains. Linear FEM is easily applied to a variety of domains but it has a slow convergence rate. Spectral methods have fast convergence, but might require complex partitioning and mapping of the domain to a reference domain.

Collocation is an alternative method to weak formulations, where the system of linear equations are built by requiring the problem to be satisfied directly in a set of points. Difference methods, where the derivatives are approximating using simple difference schemes involving neighbouring points, is a similar approach, only that the solution is exclusively given in the points. Collocation methods, however, typically use polynomials over the domain, so that the solution can be

sampled on the whole domain as is the case with finite element and spectral element based methods. The use of collocation is often limited to simple domains such as squares or tensor product domains. Stitching together multiple domains can also be challenging if a point lie on the boundary of two or more subdomains. This is due to the fact that continuity of the derivatives also needs to be enforced for second order-problems.

In this chapter we present a new collocation method based on the semi-analytical formulation of EIM. It will be defined globally on the whole domain and yields exponential convergence for sufficiently smooth problems. Because the method is defined globally there is no need, given of course that the domain is not too complicated, to partition the domain or to map it to a reference domain. We will require the approximation produced by EIM to fulfil the problem in the magic points. Because our basis functions often are analytical, we can easily obtain the necessary differentiation related to the governing equation or the boundary conditions. We will now demonstrate the method in the domain Ω by solving Poisson's equation, but because we use the strong formulation directly it is straight forward to change the governing equation. We consider

$$-\nabla^2 u = f \tag{6.1}$$

with Dirichlet boundary condition

$$u|_{\Gamma_D} = g_D, \tag{6.2}$$

and Neumann boundary condition

$$\frac{\partial u}{\partial n}|_{\Gamma_N} = g_N. \tag{6.3}$$

This can be expressed succinctly as

$$\Psi u = g \tag{6.4}$$

with

$$\Psi_i = \begin{cases} -\nabla^2 & \text{if } \boldsymbol{x}_i \in \Omega \backslash \partial\Omega \\ I & \text{if } \boldsymbol{x}_i \in \Gamma_D \\ \frac{\partial}{\partial n} & \text{if } \boldsymbol{x}_i \in \Gamma_N \end{cases}, \tag{6.5}$$

where I is the identity operator, i.e. $Iu = u$, and

$$g_i = \begin{cases} f & \text{if } \boldsymbol{x}_i \in \Omega \backslash \partial\Omega \\ g_D & \text{if } \boldsymbol{x}_i \in \Gamma_D \\ g_N & \text{if } \boldsymbol{x}_i \in \Gamma_N \end{cases}. \tag{6.6}$$

The collocation method is straight forward requiring the governing equation and boundary conditions to be satisfied in the magic points given by the EIM. We start by assuming that the solution u can be approximated with

$$u \approx u_M = \sum_{i=1}^{M} \alpha_i \mathcal{G}(\cdot; \boldsymbol{\mu}_i), \tag{6.7}$$

and insert u_M in the problem stated above

$$\begin{bmatrix} \Psi_1\mathcal{G}(\boldsymbol{x}_1;\boldsymbol{\mu}_1) & \cdots & \Psi_1\mathcal{G}(\boldsymbol{x}_1;\boldsymbol{\mu}_M) \\ \vdots & \ddots & \vdots \\ \Psi_M\mathcal{G}(\boldsymbol{x}_M;\boldsymbol{\mu}_1) & \cdots & \Psi_M\mathcal{G}(\boldsymbol{x}_M;\boldsymbol{\mu}_M) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_M \end{bmatrix} = \begin{bmatrix} g_1(\boldsymbol{x}_1) \\ \vdots \\ g_M(\boldsymbol{x}_M) \end{bmatrix} \tag{6.8}$$

where Ψ_i is the operator given by the governing equation or boundary conditions at magic point $\boldsymbol{x}_i$, with the accompanying load function g_i.

Changing the differential operator Ψ_i and load function g_i to other types of governing equations or boundary conditions is straight forward, as long as we are able to evaluate the result in the magic points. Changing the domain is only related to applying the EIM to the new domain.

Remark 6.1. We choose to use the generating function $\mathcal{G}(\cdot;\cdot)$ directly rather than the q or h-basis functions since the semi-analytical EIM ultimately uses this basis. This results in cheaper evaluation of the basis function because they consist of only a single function rather than a linear combination, and we avoid any round off error that could occur in the change of basis. The computational cost of building the system of linear equations is thus $\mathcal{O}(M^2)$ function evaluations. And with exponential convergence in M, the actual cost of building and solving the system of linear equations will be quite inexpensive. However, as before this system is prone to be ill-conditioned.

6.2 Numerical results

A simple test problem is to construct a 1D problem with the analytical solution $u(x) = 1 - x$. This was done by letting $f = 0$, with homogeneous Dirichlet boundary condition at $x = 1$ and Neumann boundary condition at $x = -1$. The The grid spacing was set to $\Delta = 2 \cdot 10^{-2}$. As expected, EIM solves this to machine

precision with 2 magic points. In 2D the problem with the analytical solution $u(x, y) = x(1-x)y(1-y)$ is solved to machine precision with 15 magic points. It was solved on a square domain with the grid spacing set to $\Delta = 5 \cdot 10^{-2}$. Homogeneous Dirichlet boundary conditions were used. Including more magic points can lead to rounding errors. In both the 1D and 2D examples we can easily print the solution as e.g. $u(x) = 1.00 - 1.00x$, in the same way we print the basis functions. This could be used to find suggestions for the analytical solution of simple problems.

We observe that the EIM collocation is competitive with that of a spectral method in both 1D and 2D in Figures 6.1 and 6.2. Both cases use inhomogeneous Dirichlet boundary conditions. The collocation is then applied to a line, a square, a semicircle, a circle and a flag. The examples uses inhomogeneous Dirichlet boundary condition. In the collocation framework it is simple to apply other types of boundary conditions, which is why an example using mixed Dirichlet and Neumann boundary conditions is included in Figure 6.9.

Overall the results look promising with exponential convergence down to machine precision, but as with the quadrature there are some issues. These will be discussed in the next section.

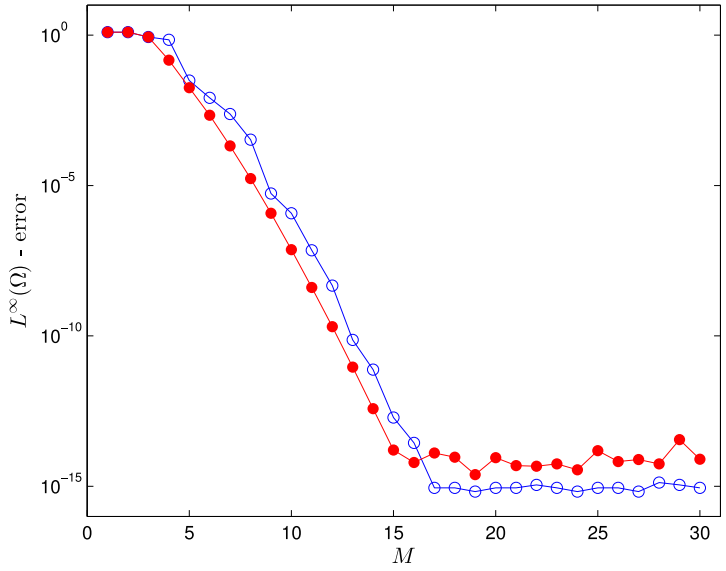

Figure 6.1: The performance of EIM collocation (blue circles) is comparable to that of a spectral method (red dots) in 1D. Homogeneous Dirichlet boundary conditions were used.

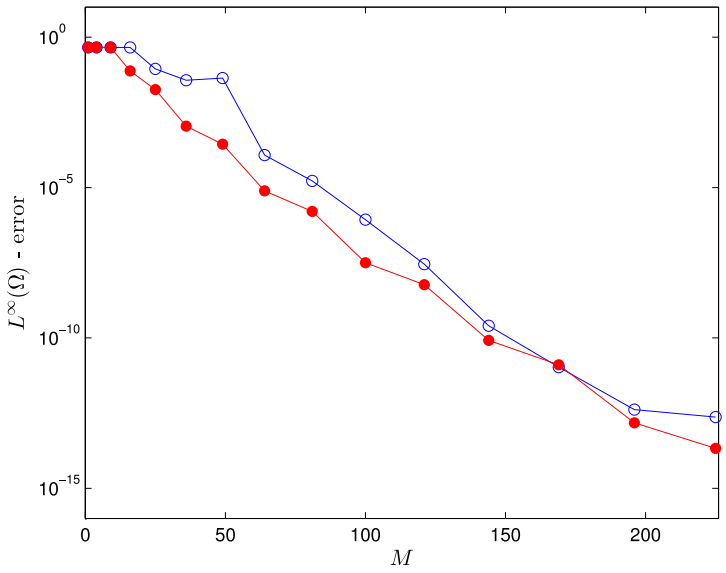

Figure 6.2: The performance of EIM collocation (blue circles) is comparable to that of a spectral method based on tensor product GLL-points (red dots) on a square. Homogeneous Dirichlet boundary conditions were used.

6.3 Issues

The EIM collocation uses the same EIM bases as the EIM quadrature, the only difference is how they are used. The problems experienced earlier, with EIM choosing the same magic points twice is therefore also present here. This will lead to singular collocation matrices for the triangle and the semicircle. However, there is also another serious problem as shown in Figure 6.10, the error suddenly spikes in the convergence plot. As with the quadrature, this can partially be mitigated, as shown in Figures 6.11 and 6.12, by using EIM with $\mathcal{G}$-basis functions rather than the q-basis functions. However, this does not solve the problem and is one of the issues that must be addressed if the method is to be developed further.

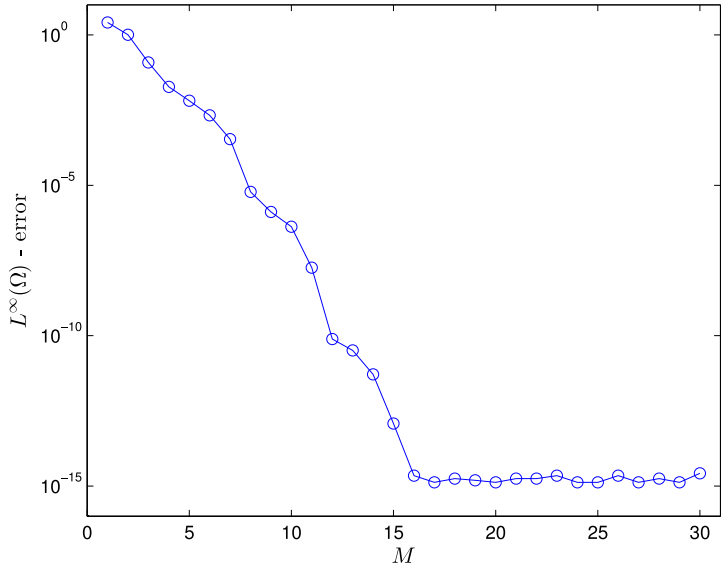

Figure 6.3: Convergence of the collocation error on a line as a function of magic points M. The solution is $u(x) = e^x \sin(x)$, the load function and inhomogeneous Dirichlet boundary conditions are chosen accordingly.

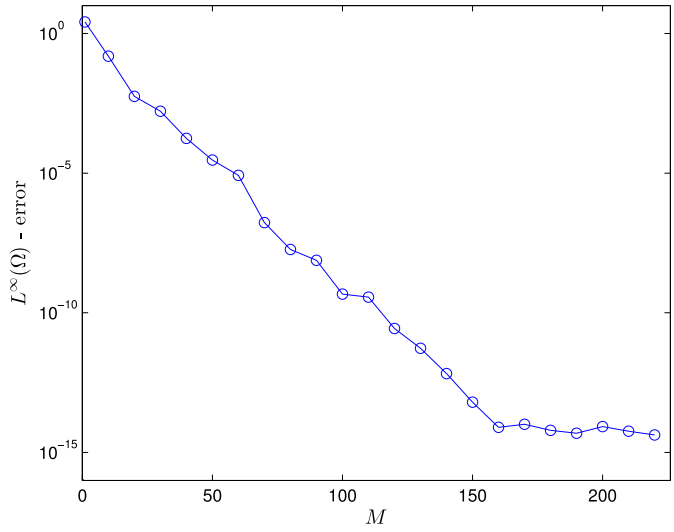

Figure 6.4: Convergence of the collocation error on a square domain as a function of magic points M. The solution is $u(x, y) = e^x \sin(y)$, the load function and inhomogeneous Dirichlet boundary conditions are chosen accordingly.

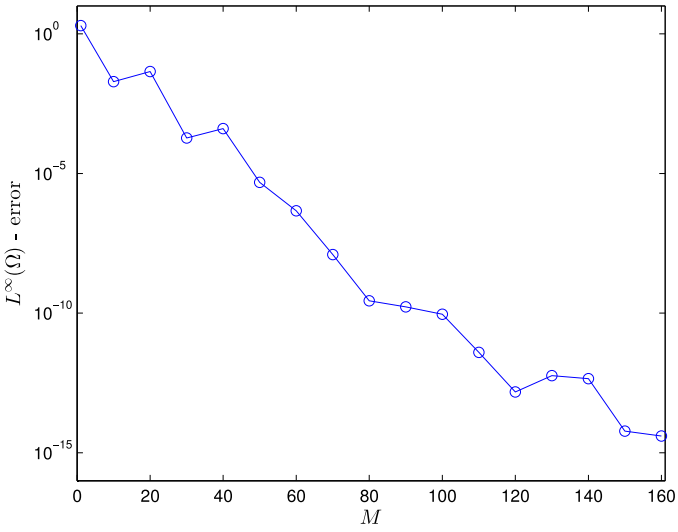

Figure 6.5: Convergence of the collocation error on a triangular domain as a function of magic points M. The solution is $u(x, y) = e^x \sin(y)$, the load function and inhomogeneous Dirichlet boundary conditions are chosen accordingly.

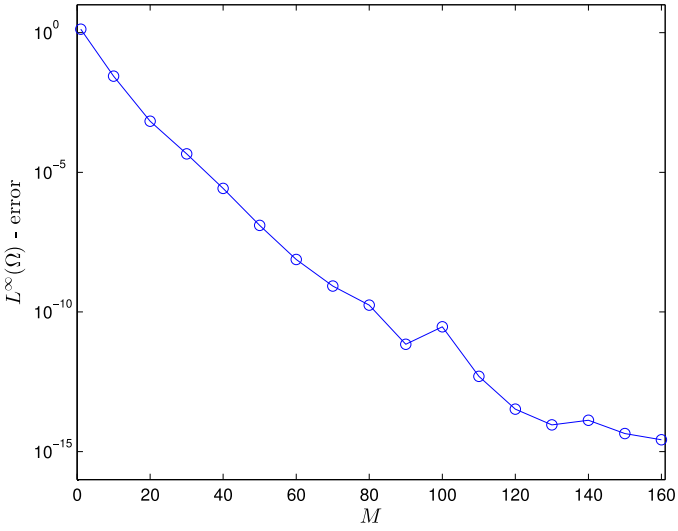

Figure 6.6: Convergence of the collocation error on a semicircular domain as a function of magic points M. The solution is $u(x, y) = e^x \sin(y)$, the load function and inhomogeneous Dirichlet boundary conditions are chosen accordingly.

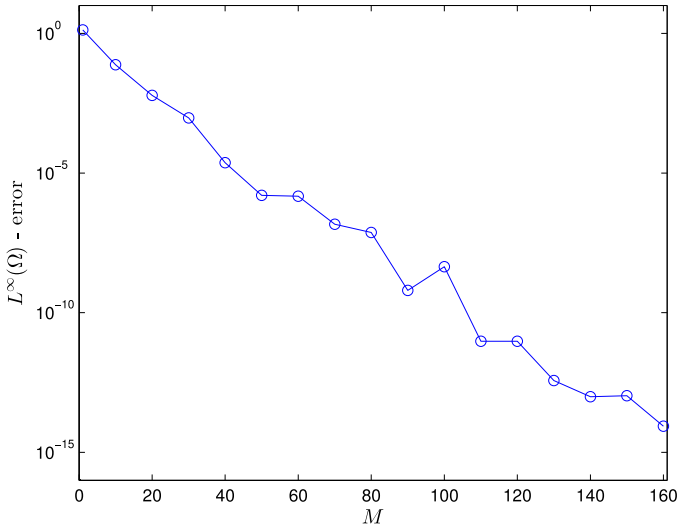

Figure 6.7: Convergence of the collocation error on a circular domain as a function of magic points M. The solution is $u(x, y) = e^x \sin(y)$, the load function and inhomogeneous Dirichlet boundary conditions are chosen accordingly.

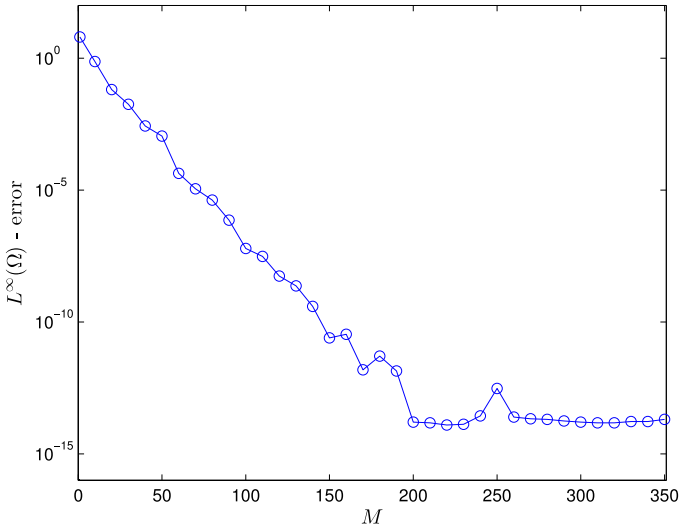

Figure 6.8: Convergence of the collocation error on a flag-shaped domain as a function of magic points M. The solution is $u(x, y) = e^x \sin(y)$, the load function and inhomogeneous Dirichlet boundary conditions are chosen accordingly.

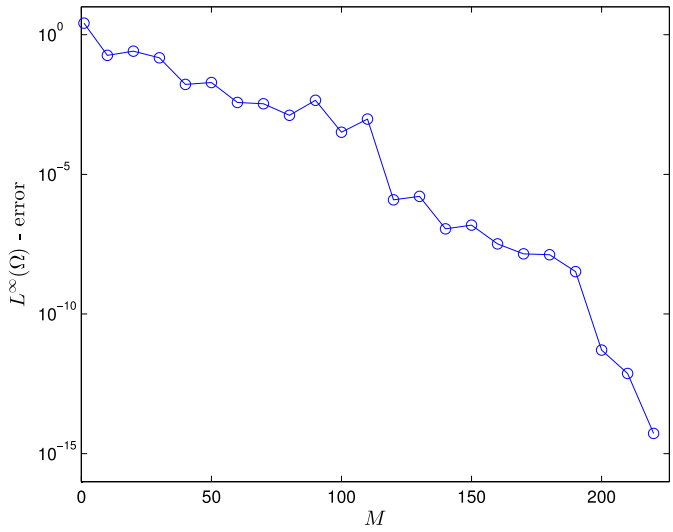

Figure 6.9: Convergence of the collocation error on a square domain with mixed Dirichlet and Neumann boundary conditions. The solution is $u(x, y) = e^{x}\sin(y)$, the load function and boundary conditions are chosen accordingly.

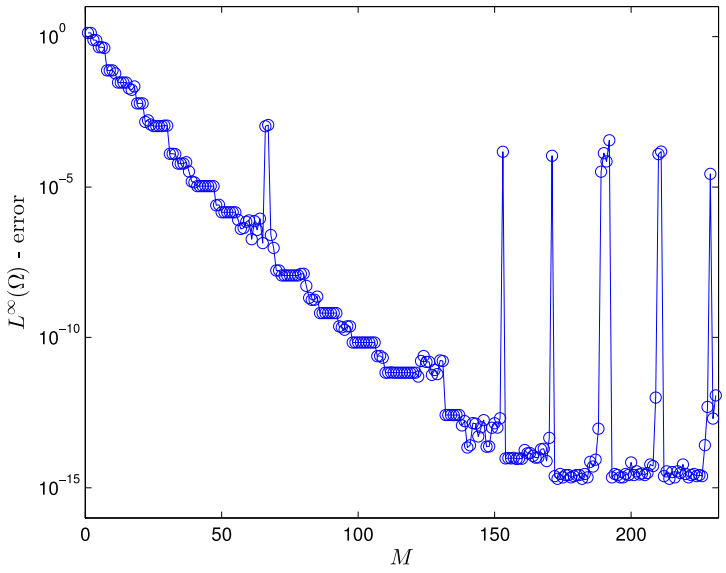

Figure 6.10: Detailed convergence of the collocation error on a triangular domain using EIM with q-basis functions.

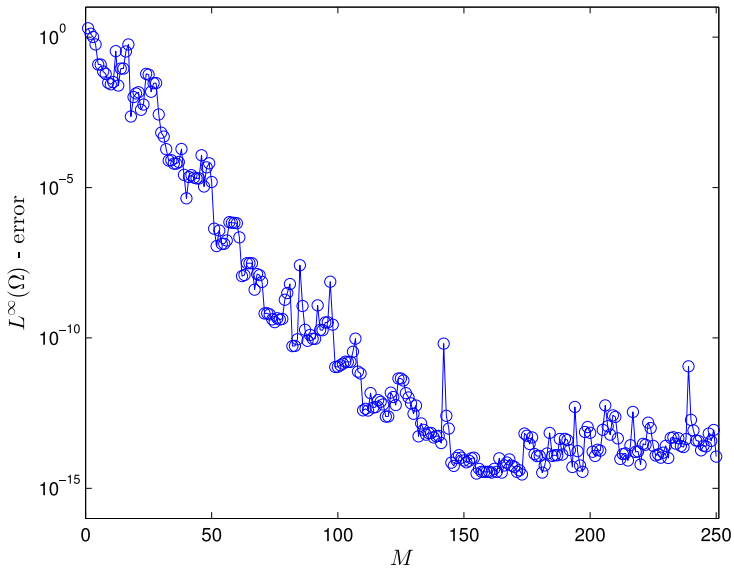

Figure 6.11: Detailed convergence of the collocation error on a triangular domain using EIM with $\mathcal{G}$-basis functions.

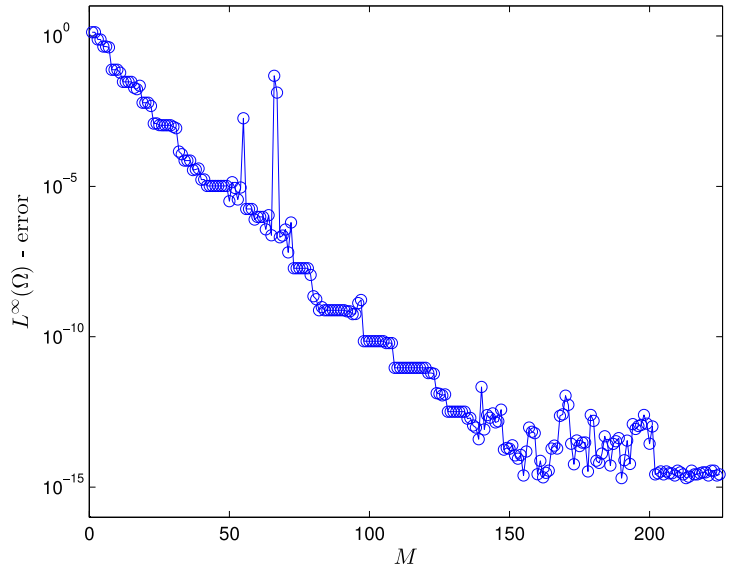

Figure 6.12: Detailed convergence of the collocation error on a circular domain using EIM with $\mathcal{G}$-basis functions.

Chapter 7

Inversion

In this chapter we will briefly look at two types of inversion. This to illustrate that the EIM might not be limited to the uses previously discussed. The first type of inversion will construct an EIM basis that is independent of the target data, and then compare the EIM representation of the target data to find the best match. It will be applied to image recognition. The second approach will assume that we can build an EIM basis that will interpolate our data exactly. The EIM representation will thus give the desired answer directly. This approach will be applied to sound recognition. Although the methods show some potential, they also seem very sensitive to noise.

7.1 Introduction

Inversion is the process of finding $\boldsymbol{\mu}$ given the output $f(\cdot;\boldsymbol{\mu})$. In many cases f^{-1} is not given explicitly. One approach to find the approximation to the inverse is to compute $f(\cdot;\boldsymbol{\mu})$ for a set of $\boldsymbol{\mu}$'s, storing each pair $(f(\cdot;\boldsymbol{\mu}_i),\boldsymbol{\mu}_i)$ in the set of pairs $\mathcal{S} = \{(f(\cdot;\boldsymbol{\mu}_i),\boldsymbol{\mu}_i)\}\forall i$. The solution $\boldsymbol{\mu}_i$ is given by the closest match

$$\boldsymbol{\mu}_i = \arg\min_{\boldsymbol{\mu}_i \in \Xi} \|f(\cdot;\boldsymbol{\mu}) - f(\cdot;\boldsymbol{\mu}_i)\|_{L^\infty(\Omega)} \tag{7.1}$$

if the difference is less than a tolerance *tol*

$$\min_{\boldsymbol{\mu}_i \in \Xi} \|f(\cdot;\boldsymbol{\mu}) - f(\cdot;\boldsymbol{\mu}_i)\|_{L^\infty(\Omega)} \leq tol \tag{7.2}$$

otherwise there is no solution. An alternative to comparing f directly is to compute the EIM representation, i.e. $f(\cdot) \approx \sum_{i=1}^{M} \alpha_i \mathcal{G}(\cdot;\boldsymbol{\mu}_i)$ and the compare coefficients α_i instead. We will apply this method to image recognition along the lines of [17].

The second approach assumes that the interpolation is exact

$$f(\cdot) = \sum_{i=1}^{M} \alpha_i \mathcal{G}(\cdot; \boldsymbol{\mu}_i). \tag{7.3}$$

This is a serious limitations, but it allows us to only train the system on the underlying functions $\mathcal{G}(\cdot; \boldsymbol{\mu}_i)$ we expect to find. In the case of noise free electromagnetic or sound waves we only have to train with the sine and cosine functions in the desired frequency range. Due to phase shifts, we need $2N$ different magic points to exactly interpolate N frequencies. This is regardless of the target frequencies, because our EIM is only trained for those frequencies. Because EIM interpolates functions in the approximation space exactly, we should be able to reconstruct noise free signals exactly. It should also be possible to train EIM using more complex combinations of sine and cosine or other functions, but phase shifts might become an issue.

7.2 Representing images

Images are often represented as an array of color squares called pixels as shown in Figure 7.1. When training EIM, sampling a function $\mathcal{G}(\boldsymbol{x}; \boldsymbol{\mu})$ will be to select an image based on the parameter $\boldsymbol{\mu}$ and then select a pixel in the given image given by the point $\boldsymbol{x}$. The domain is thus the uniform 2D square given by an image, while the parameter space is the index in the set of images.

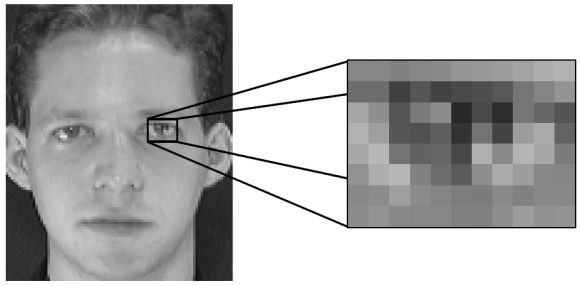

Figure 7.1: Images are often represented as an array of individual squares of colors called pixels.

7.3 Image recognition

We will here apply EIM to image recognition along the lines of [17]. We therefore use the same library of face images, which is from [23]. The library consists of 10 variations of 40 different subjects. We will use all 10 variations of the 20 first subjects to train EIM. We will then use a single variation of the remaining 20 subjects to create a coefficient profile, which we will compare the remaining 180 images (9 variations, 20 subjects) against.

If we choose the closest match of all 180 images, using the $L^1(\Omega)$-norm, then 59 out of 180 images was labelled correctly or a 33% success ratio. The $L^1(\Omega)$-norm seemed to produce better answers than the $L^\infty(\Omega)$-norm.

In practice it would be more desirable to set a tolerance, as in (7.2), and only accept those images where the error is lower than the tolerance. We can achieve that all images within the tolerance is recognized correctly, but then only 7 images out of 180 are chosen or 4%. We have no good way of selecting the tolerance a priori, making it quite likely that false positives will be introduced. Choosing the tolerance will thus become a trade-off between how many faces are recognized and how many faces that are incorrectly recognized. At at a certain level we are able to recognize 34 faces correctly or (19%), but we also have 33 images that are falsely identified.

In [17] they crop the images before they are processed. In Figure 7.2 we can see that a lot of the magic points lie in the vicinity of the ears or even where the clothes show up in the images. This can act as an distraction away from the core features, and thus if the selection criteria was fine tuned together with cropping the images, EIM might become a more viable alternative.

7.4 Representing sound

A pure tone can be represented with a sine function, and to account for phase shifts, we can represent it as a linear combination of sine and cosine, see Figure 7.3. In practice sound is sampled discretely at uniform time intervals. A common sampling rate is 44100 samples per second (44100Hz). This is because humans can hear sounds up to about 20kHz, and, by the Nyquist sampling theorem, we then need to sample at twice that rate. We will generate the sound synthetically and sample it at 44100Hz.

Each pure tone is included in the EIM training set as a pair of sine and cosine functions. Since we have two functions per frequency, we use an index set to describe the sounds in our parameter space rather than using the frequencies. The domain is time. Sampling a function $\mathcal{G}(t;\boldsymbol{\mu})$ is thus the same as first choosing the sound $\boldsymbol{\mu}$ and then its contribution at time t.

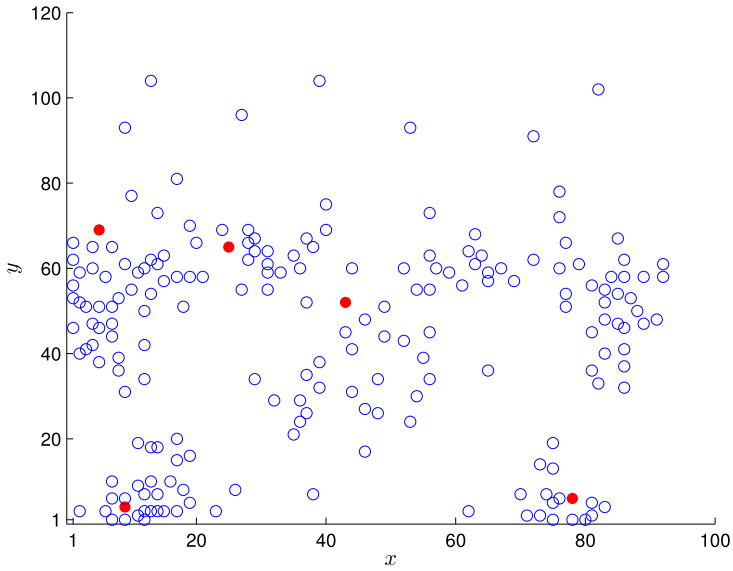

Figure 7.2: Distribution of the first 50 spatial magic points (first five marked with red dots) of the face sequence consisting of 20 subjects and 10 variations per subject. There are clusters of points around the eyes, nose and mouth in the center of the plot, but the majority of points lie nearby the ears, neck and shoulders which include some of the clothing the subjects are wearing.

7.5 Sound recognition

Our synthetic sounds will be the fundamental frequency of keys on a piano. The fundamental frequency of the nth key of a piano is given by

$$f(n) = 2^{\frac{n-49}{12}} \cdot 440\text{Hz}. \tag{7.4}$$

By applying a moving window in time we can achieve temporal resolution. This is done by first looking at the samples 1 through 441, then 2 through 442 and so on. Because the window size is 441, we train EIM on a domain with 441 points. Where there is silence or in the transition between different tones, there will be an area that does not just consist of pure tones but a mix of silence and different tones: within a single window there is first one or more tones then silence before some other tones. EIM is very sensitive to noise in the data, and the situation just described will not work properly. Therefore, a filter step is included: we assume that no more than say N instruments contribute significantly (more than some threshold) at any given time. If that is the case, we enforce that no instrument is contributing. By training on the forth octave of a piano, containing 12 keys from

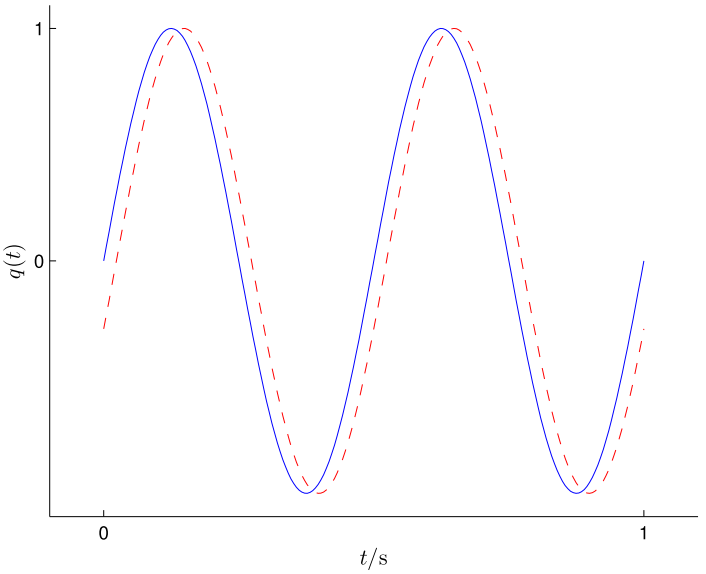

Figure 7.3: Pure tones can be represented as sine waves. The frequency is equal to the number of periods per second, which in this case makes the frequency 2Hz. Phase shifts (dashed line compared to solid line) can be represented as a linear combination of sine and cosine waves.

C4 at 262Hz to B4 at 494Hz, using 12 pairs of sine and cosine to account for phase shifts, and applying the filtering rule that no more than 6 tones should be playing at any one time we get the results in Figure 7.4. Except that we use 441 samples to produce a single point in the figure, the inversion reproduces the synthetic signal correctly.

This application has just been a frequency analysis of a simple signal. In principle we could choose a more complex generating function. It could perhaps be extended to recognize musical instruments rather than just pure tones. Actual musical instruments are of course more complex with a fundamental frequency, overtones and other variations.

Both of the inversion applications are very sensitive to noise, the work done in [19] to perform noisy regression using EIM, might be useful for further work in this area. It could be interesting to compare existing frequency analysis methods such as the Discrete Fourier Transform (DFT) to the one using EIM.

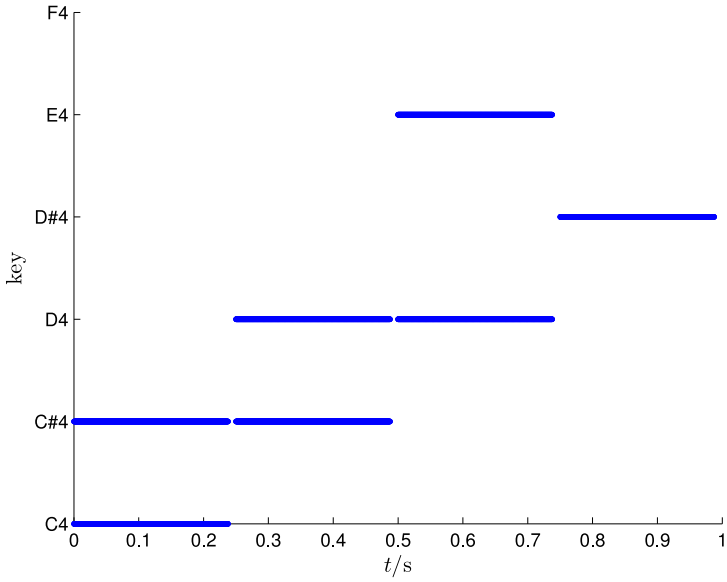

Figure 7.4: Tones recognized at different points in time by EIM inversion.

Chapter 8

Software implementation

8.1 General comments

The computational framework was developed alongside the semi-analytical representation of EIM, and many of the abstractions are equivalent with the actual implementation. This is achieved through extensive use of object orientation. Object oriented programming is a paradigm where variables and functions that are related are grouped into classes. EIM itself is implemented once, and all of the different applications looked at in this thesis is achieved by changing the spatial domain, parameter domain and function space according to the problem. Additional functionality, like collocation, then made use of the single implementation of EIM. This made it possible to quickly implement new applications.

The EIM implementation knows how to work with an abstract notion of the spatial domain, parameter domain and function space. Each specific implementation of e.g. the domain has to conform to the abstract domain. In object orientated programming this is called interfaces. This way each function space knows how to integrate and differentiate the functions in its space. This creates a clear division between EIM and the problem specific code. Furthermore, the function space was implemented using the fundamental principle of functional programming: that calling a function has no side effects and that the input is the only thing that affects the output. This is, of course, the same as with mathematical functions and made a lot of sense when implementing a function space. The key practical advantage with this is that a function space will not change after its creation, so as long as all parties have access to the function space, they will be able to send meaningful messages using the encoding $(\underline{\alpha}, \underline{\boldsymbol{\mu}})$ describe earlier. Any information related to EIM or particular applications like compression is not needed.

The encoding of functions as $(\underline{\alpha}, \underline{\boldsymbol{\mu}})$ offers a simple way to represent the function as human readable text. This was exploited to generate the tables containing basis

```
FunctionSpace:
  sample(alphas,mus,points)
  max(alphas,mus,omega)
  differentiate(alphas,mus,points)
  integrate(alphas,mus,omega)
  printFunction(alphas,mus)

SpatialDomain:
  numPoints()
  getPoint(index)
  numSurfaces()
  sampleSurface(index,t)
  discretize(numPoints)

ParameterDomain:
  numParameters()
  getParameter(index)

EIM:
  interpolate{Q,H}(f)
  sample{Q,H}(coefficients,points)
  [alphas mus] = encode{Q,H}(f)
  sample(alphas,points)
  greedy(numPoints)
  ascending(numPoints)
```

Figure 8.1: An overview of some of the main classes and functions. The `ParameterDomain`, `SpatialDomain` and `ParameterDomain` classes act as interfaces to abstract their respective functionalities. This is done so that a single implementation of the `EIM` class can work with all the different problems presented in this thesis. The α-coefficients are central in the implementation of `FunctionSpace`. This is to isolate the functions that span the space in order to make it easier to perform analytical operations.

functions for the various domains in this thesis.

Some of the operations are suited for parallelization. Sampling a function in a set of points are independent of each other, so is finding max when training EIM. Some use of parallelization was applied to produce results faster in this thesis, and is definitely something that should be explored if developing the semi-analytical framework further.

8.2 Acknowledgements

Almost all of the code and results have been developed using MathWorks MATLAB. Maplesoft Maple was used to obtain some of the analytical results. MATLAB code related to GLL and spectral methods were adapted from code handed out in the course MA8502 at the Norwegian University of Science and Technology [15]. The flag from the compression chapter was modelled and animated using Autodesk Maya and exported via the OBJ format. The animation was never used (it was replaced by synthetic animation in order to have greater control), but the geometry was used. The code to handle 3D objects in the OBJ format in MATLAB was written by Dirk-Jan Kroon and downloaded from the MATLAB file exchange. All other code was written in MATLAB for the purpose of this thesis.

Chapter 9

Conclusions and final remarks

This thesis have been based on the Empirical Interpolation Method and some of the suggested applications presented in [15]. We have proposed a new practical approach to deal with basis functions analytically called the semi-analytic formulation of EIM. Based on this formulation we have developed new ways to deal with the integration of basis functions when developing EIM quadrature rules. This included integrating polynomials exactly over arbitrary simple polygons. We also propose a new way to solve partial differential equations by combining EIM and collocation. While the results from both the quadrature and collocation are promising, with exponential convergence for sufficiently smooth problems, there are also issues with the error that must be addressed if these methods are developed further. In the case of collocation, we have no measure as to how many points we need to include for the problem to make sense, i.e. have enough points in the interior and at the boundary to fulfil the governing equation and boundary conditions. Regardless of the application, the interpolation error in the magic points seems to grow as the number of magic points increases. This is quite alarming as the error should be zero in the magic points by construction. At this point it is unknown whether these issues are related to round-off errors, ill-conditioned matrices, implementation bugs, or some other error. These issues must be investigated if the methods are to be developed further. In addition, the test cases are quite simple and do not test the limits of what functions and domains the EIM can handle.

In addition to enable basis functions to be handled analytically, the semi-analytic EIM makes it easy to apply EIM to a variety of problems. We have looked at compression of 3D animation, which yields exponential convergence for sufficiently smooth data. The method should work equally well for other types of smooth data. EIM was also applied to image recognition and signal recognition. Both methods

did work to some extent, but was not flexible in what data they can work with. This suggests that the smoothness requirement of EIM might be too strict for EIM to work properly for some practical applications, especially with noisy data. The methods would need to be further developed and compared to existing solutions to say something conclusive.

Field reconstruction was briefly mentioned in the compression chapter. The semi-analytic EIM could offer ways to create computational surrogates that can be sampled continuously in time and space. If the system to be modelled has a vector output, e.g. a displacement in 3D, then the dimensions could be handled separately by applying EIM once for each dimension. Alternatively, EIM could perhaps be changed to choose the closest interpolation via optimization rather than requiring the interpolation to be exact in the interpolation points.

Handling the basis functions of the EIM analytically shows great potential if the issues encountered can be fixed. Alternatively, it might be possible to circumvent them by working only with polynomials rather than any function generated by $\mathcal{G}(\cdot;\cdot)$.

If we want to reduce the computational cost there are several techniques that might help. In [10], the parameter domain is partitioned into subdomains before EIM is trained separately on each subdomain. This can lead to smaller EIM bases, but requires us to know which one to use when interpolating. This is straight forward when using EIM in the traditional way, i.e. when the approximation space is spanned by the parametric function to be interpolated, but it is not obvious if this would work for generic bases as well. An approach to deal with greedy algorithms in higher dimensions is presented in [12]. Practical implementations can be improved further by using optimizations like parallelization.

Bibliography

[1] T. O. Aanonsen, *Empirical interpolation with application to reduced basis approximations*, Master Thesis, Norwegian University of Science and Technology, (2009).

[2] H. Antil, S. E. Field, F. Herrmann, R. H. Nochetto, and M. Tiglio, *Two-step greedy algorithm for reduced order quadratures*, CoRR, abs/1210.0577 (2012).

[3] M. Barrault, Y. Maday, N. C. Nguyen, and A. T. Patera, *An 'empirical interpolation' method: application to efficient reduced-basis discretization of partial differential equations*, Comptes Rendus Mathematique, 339 (2004), pp. 667 – 672.

[4] C. Bernardi and Y. Maday, *Spectral Methods. In Handbook of Numerical Analysis. Volume V: Techniques of Scientific Computing (Part 2)*, Elsevier, 1997.

[5] C. Canuto, M. Hussaini, A. Quarteroni, and T. Zang, *Spectral Methods in Fluid Dynamics*, Springer, 1993.

[6] S. Chaturantabut and D. C. Sorensen, *Discrete empirical interpolation for nonlinear model reduction*, in Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on, IEEE, 2009, pp. 4316–4321.

[7] P. Chen, A. Quarteroni, and G. Rozza, *A weighted empirical interpolation method: a priori convergence analysis and applications*, (2013).

[8] J. L. Eftang, M. A. Grepl, and A. T. Patera, *A posteriori error bounds for the empirical interpolation method*, Comptes Rendus Mathematique, 348 (2010), pp. 575–579.

[9] J. L. Eftang, M. A. Grepl, A. T. Patera, and E. M. Rønquist, *Approximation of parametric derivatives by the empirical interpolation method*, Foundations of Computational Mathematics, (2011), pp. 1–25.

[10] J. L. Eftang and B. Stamm, *Parameter multi-domain 'hp'empirical interpolation*, International Journal for Numerical Methods in Engineering, 90 (2012), pp. 412–428.

[11] M. A. Grepl, Y. Maday, N. C. Nguyen, and A. T. Patera, *Efficient reduced-basis treatment of nonaffine and nonlinear partial differential equations*, ESAIM: Mathematical Modelling and Numerical Analysis, 41 (2007), pp. 575–605.

[12] J. S. Hesthaven, B. Stamm, and S. Zhang, *Efficient greedy algorithms for high-dimensional parameter spaces with applications to empirical interpolation and reduced basis methods*, tech. rep., DTIC Document, 2011.

[13] F. James, *Monte carlo theory and practice*, Reports on Progress in Physics, 43 (1980), p. 1145.

[14] E. Kreyszig, *Advanced Engineering Mathematics*, 9 ed., 2006.

[15] Y. Maday, N. C. Nguyen, A. T. Patera, and G. S. H. Pau, *A general multipurpose interpolation procedure: the magic points*, Communications on pure and applied analysis, Volume 8, Number 1, January 2009.

[16] N. Nguyen, A. Patera, and J. Peraire, *A 'best points' interpolation method for efficient approximation of parametrized functions*, International journal for numerical methods in engineering, 73 (2008), pp. 521–543.

[17] N. Nguyen and J. Peraire, *An interpolation method for the reconstruction and recognition of face images*, tech. rep., DTIC Document, 2007.

[18] R. Pasquetti and F. Rapetti, *Spectral element methods on unstructured meshes: which interpolation points?*, Numer. Algorithms, 55 (2010), pp. 349–366.

[19] A. T. Patera and E. M. Rønquist, *Regression on parametric manifolds: Estimation of spatial fields, functional outputs, and parameters from noisy data.*, C. R., Math., Acad. Sci. Paris, 350 (2012), pp. 543–547.

[20] A. T. Patera and G. Rozza, *Reduced basis approximation and a posteriori error estimation for parametrized partial differential equations*, Version 1.0, Copyright MIT 2006, to appear in (tentative rubric) MIT Pappalardo Graduate Monographs in Mechanical Engineering.

[21] A. Quarteroni and A. Valli, *Numerical Approximation of Partial Differential Equations*, Springer, 1994.

[22] E. M. Rønquist, *Numerical solution of partial differential equations. Lecture Notes (course MA8502)*, 2012.

[23] F. Samaria and A. Harter, *Parameterisation of a stochastic model for human face identification*, 2nd IEEE Workshop on Applications of Computer Vision, (1994).

[24] D. Wirtz, D. C. Sorensen, and B. Haasdonk, *A-posteriori error estimation for DEIM reduced nonlinear dynamical systems*, SRC SimTech Preprint Series, (2012).