IJS, Odsek E6: Komunikacijski sistemi

Local prediction of weather parameters based on historical data

The aim of this project is to provide a short term local forecast of weather parameters using only provided historical data. This data is typically in the form of time series with 15 minute time steps. The goal of this project is to address the problem with multivariate ARIMA (or similar) statistical method to get a weather forecast along with error assessment.

Detailed task:

- A year or more of weather parameter observations are given in form of csv file(s).
- A python script is given that loads files and plots some of the parameters.
- First, select a Python library to perform modelling of the data using ARIMA or similar.
- Then experiment with the library to create some statistical models of the given data. Advice: use only a subset of the provided data for faster development, since complex models can be time demanding to train.
- Write a function to forecast the weather parameters up to 10 time steps into the future for any given time range of data, using a statistical model.
- Write a function to test the prediction - absolute accuracy can be calculated from the differences between the real values and the predictions, but the models can have their own error prediction too (there is no one way to present it), which is also to be tested.
- Write the testing framework, which will test any given model (must be a parameter so that different models can be tested) on a chosen subset of the historical data (the one given in csv files), and will return a measure of model accuracy. E.g. the model predicts temperature for 2 h in advance within 2°C error, 95% of the time (similarly for all the other parameters).
- Finally train the statistical model on all the historical data and test it on the same data using testing the testing framework. Then divide the historical data into training and testing sets (50% each), retrain the model and determine how much the performance of the model trained on less data worsened.
- Optimise the model to minimise both its error and the model complexity (for the latter, e.g. aim for lower numbers in ARIMA description).
- The result may be one or more models of various complexity that predict the given weather parameters.

Matjaž Depolli

Noise correlation in images, sound and accelerometer data
Digital sensors, such as camera, microphone or accelerometer sample a physical quantity and digitise it to create a digital measurement. Both sampling and digitization introduce some noise into the measurement. We wish to discover the typical levels of noise in measurements made by a smartphone: pictures, sound and accelerometer measurements. We classify noise into two categories, correlated and uncorrelated. For example, we might investigate sound sampled with 44 kHz, with samples being represented as 16-bit integers. We might measure some silence to discover that individual sample values seem to be drawn from a normal distribution with mean 0 and some standard deviation. Then we might analyse the recording further to discover what is the correlation between pairs of neighbouring samples. Finally we wish to know the amount of uncorrelated noise in the recording, which we can assume to be the entire noise minus the correlated noise. Accelerometer recordings are similar, but camera recordings are different. Correlated noise on images is between neighbouring pixels (both in horizontal and vertical direction).
Task in details:

- Receive measurements from camera, microphone and accelerometer. Perform the steps below for each type of measurement individually. Analyse each measurement in isolation and combine the results from the measurements of the same type.
- Use python (preferably in jupyter notebooks) as the tool for analysis and visualisation. Use the libraries as you see fit but be sure to know what exactly each imported function does - provide enough comments and descriptions that another person could rewrite your code without the access to the specialised libraries.
- Visualise the measurements and noise distributions. Is noise distributed normally?
- Develop functions that can calculate total, correlated and uncorrelated noise for a supplied measurement.
- Analyse the given measurement to report their average noise per pixel or per sample.
- Visualise the results, e.g. histograms of noise values.
- Analyse and visualise the differences between measurements of the same type.

This task is open-ended, the student(s) may choose the tools for the analysis and types of visualisation somewhat freely, but must make the resulting code easy to read and understand.

Matjaž Depolli

Spodnje ni task, je le opis Meduse, ki je del kasneje definiranih taskov

Medusa: coordinate free implementation of meshless methods

Our team started the development of Medusa library in 2015 to support our research in the field of numerical analysis and to ease the implementation of applied projects. Over time, the interface grew and matured, putting emphasis on modularity, extensibility and reusability. Similarly to many open-source FEM libraries, it relies heavily on the C++ template system and allows the programs to be written independently of the number of spatial dimensions with negligible run-time and memory overhead. Special care is also taken to increase expressiveness and to be able to explicitly translate mathematical notation into program source code. However, source code is still standard compliant C++, which allows the user to use the entirety of the C++ ecosystem. The open-source nature of the library is a novelty compared to the other contemporary libraries. We already utilised Medusa library for solving broad spectra of problems ranging from pure academic experiments like high order solutions of Poisson's equation in 2D, 3D, and 4D, to applied thermo-fluid and thermo-elastic simulations of real life problems.

J. Slak and G. Kosec, 'Medusa: A C++ Library for Solving PDEs Using Strong Form Mesh-free Methods', *ACM Trans. Math. Softw.*, vol. 47, no. 3, pp. 1–25, Sep. 2021, doi: 10.1145/3450966.

https://e6.ijs.si/medusa/wiki/index.php/Medusa

Miscellaneous Medusa library improvements

The aim of this project is for student to add some new functionality or improve some existing aspect of the Medusa C++ library. Compared to the projects that follow, this project will require the student to familiarise himself with the source code of Medusa in greater detail, to be able to build on top of it.

There are numerous possible tasks a student can take on in the scope of the project, subject to discussion. Some examples are described below.:

- While Medusa has several domain shapes and the methods of their discretisation implemented, some methods (for example, higher dimensional spheres) are currently lacking. Interested student could implement those missing parts.
- For approximation purposes, Medusa supports different basis functions (Radial Basis Functions, monomials etc.). Some of the more popular basis functions are not yet included, which could become a student's task.
- When solving PDEs numerically using Medusa, one is often interested about certain quantitative properties of the solution procedure, such as condition numbers of the matrices involved or the quality metrics of the discretisation. While they are still relatively simple to compute, such functionality is not already a part of Medusa. A student could add it.
- Due to how the operators are defined, special care must be taken in order to use Medusa on problems that consist of two coupled computational domains. This could be simplified by appropriately modifying the existing implementation of the operators, which an interested student could work on.

Andrej Pozun

Numerically Solving Navier Stokes equation with GPUs
The aim of the project is for the student to solve the Navier-Stokes equation with implicit time stepping using the Medusa library. The numerical solution procedure is already established and provides stable results on the host CPU. The student's task is to port the most time-consuming parts, i.e. the solution of the global sparse system, to the GPUs.
The task in detail:

- Receive a source code in C++ that solves a particular problem.
- The portion of code where a large system of linear equations (represented with a sparse matrix) is solved is clearly marked.
- Select a C++ library that enables one to work with (nvidia-based) GPU’s
- Rewrite the marked portion of code to use GPU instead of CPU.
- Compare the results obtained with the original program to the results obtained by the modified program and confirm only minimal differences can be detected, confirming the correctness of the modified code.
- Compare the execution times of the original and modified program.
- Analyse performance and accuracy tradeoffs for using single- and double-precision floating point variables.

The student will receive an account on a Linux workstation with 2× Nvidia RTX 3090’s to use for their development and testing.

Andrej Pozun

Analysis of using ghost nodes
Ghost nodes are a technique used for discretizing Neumann boundary conditions in FDM. To be able to use the central difference for the first derivative, an additional point, called ghost node, is introduced outside the domain boundary. The unknown function value at the ghost node is added as a variable. At the boundary node, the Neumann condition is enforced, as well as the equation itself (two equations for two unknowns, the ghost and the boundary function value). While the ghost node positioning is more or less fixed in the classic FDM the meshless approach allows for more freedom. The aim of this project is to analyse the behaviour of this technique as the setup changes.
The task in greater detail:

- The student will be making use of the https://e6.ijs.si/medusa/ C++ library and should first familiarise himself/herself with its basic usage.
- The student will be provided with a preliminary c++ script that implements ghost nodes in Poisson equation solution.
- Suitably modifying the supplied script, the student will then analyse the changes in accuracy and stability caused by changing the ghost node positioning. This part of the project is open in a sense that the student can choose what ghost node positioning strategy to use and how to best isolate the behaviour in order to study the effects.

Miha Rot

FDM on scattered nodes

Finite Differences are a well known method for discrete approximations of a derivative or a more general differential operator. They can work in arbitrary dimensions and are usually applied to simple geometries such as cartesian grids. On the other hand, interpolation of scattered (irregular) data is currently a popular topic of research and application. We can combine the two methods to generalise the Finite Difference approach to irregular data, by interpolating scattered values to a grid, where the usual Finite Differences can be applied. Such a method could then be used to numerically solve (partial) differential equations on irregular domains. The aim of this project is to analyse the properties of the resulting method (accuracy, stability etc.).

The task in greater detail:

- Student will be making use of the https://e6.ijs.si/medusa/ C++ library and should first familiarize himself/herself with its basic usage.
- Afterwards, a preliminary c++ script with FDM on scattered nodes will be supplied that the student should understand in greater detail.
- Suitably modifying the supplied script, student will then perform analyses of the method. This part of the project is open in a sense that the student can choose which aspects of the method he/she wants to focus on. Some ideas are:
    - Studying convergence orders of the resulting method by calculating the error of differential operator approximation for different functions, domain discretisations and parameter choices.
    - An interesting question would be how scattered FDM works near irregular boundaries, which should be studied in greater detail.
    - Extending the mentioned studies to solution of simple PDEs, such as Poisson's equation with different boundary conditions.
    - Besides approximation accuracy, stability of the procedure can also be studied, by, for example, analysing the condition numbers and the spectra of the matrices involved in the process.
- If time allows it, student can also:
    - Further improve the supplied script, either by speeding up the computation time or making it extendable/easier to use by packing it inside its own class, for example.
    - Perform some analytical computations - likely, the method can be analysed "on paper" for some very simple cases.

Andrej Pozun

Upwind stabilisation of Navier Stokes equation

The simulation of fluid flow with a direct solution of the Navier-Stokes equation becomes unstable as we progress towards higher Reynolds numbers. The instabilities stemming from the non-linear advection term blow up in time and lead to a divergent solution. One of the possible solutions is to use the well-known upwind stabilisation technique, originally developed in connection with the FDM by Courant. In a meshless setup the upwind magnitude can be dynamically controlled to

minimise the numerical dissipation. The aim of this project is to implement an adaptive upwind, solve the Lid driven problem, and analyse the effect of upwind magnitude on eigenspectra of the related global systems.
The task in greater detail:

- The student will be making use of the https://e6.ijs.si/medusa/ C++ library and should first familiarise himself/herself with its basic usage.
- Afterwards, a preliminary c++ script that solves the lid-driven cavity flow without stabilisation will be provided.
- The student should examine the provided literature and implement the upwind stabilisation.
- The student should verify that the solution is not significantly degraded in comparison to non-stabilised solution and examine how changes to upwind parameterization impact the degradation.
- With the newly acquired knowledge the student should then attempt to extend the analysis to flow regimes where non-stabilised solution is no longer adequate. The effects of stabilisation in this regime should be further studied with the help of eigenvalue decomposition on the corresponding dynamical system matrix.

Miha Rot

Domain decomposition and distributed computing

The aim of this project is to implement a simple domain decomposition technique on domain discretized with scattered nodes. Our recently developed parallel scattered node positioning algorithm already generates connected sets of nodes of similar sizes, which could be used as building blocks for subdomains balanced in size. The student's task here is to appropriately mark the nodes on borders between domains and solve PDE of his choice using one of the meshless methods supported in Medusa.
The task in detail:

- A C++ program is provided that loads a single set of nodes from a file - the discretization of the domain, and solves a given problem on that domain.
- Sets of nodes, distributed into two or more connected areas (subdomains) are provided as files (the same format as in the first bullet).
- The original program is to be modified in such a way that distributed sets of nodes are loaded and the problem is solved with a selected domain decomposition method inserted into the solution procedure.
- Note that the goal of this project is to prepare for making the algorithm work in parallel. This project, however, only requires the solution procedure to be performed sequentially on a single CPU. The goal is only for the subdomains to be solved separately.
- Compare the results of the original and modified program and confirm only minor differences due to the change of solution procedure are present.

https://gitlab.com/e62Lab/2019_medusa_domain_decomposition/-/tree/master?ref_type=heads

Gregor Kosec

Oversampled RBF-FD

RBF-FD (Radial Basis Function-generated Finite Difference) is a numerical method for approximating differential operators on scattered (irregular) nodes. A typical use case of such a method is numerical solution of Partial Differential Equations (PDEs) on irregular domains. Typically, the domain is first discretised into a set of N points X. RBF-FD provides us with a recipe to discretise the PDE at each of the N points resulting in N equations for N unknowns (function values of the solution at each point of X). Upon solving this (sparse) linear system, we obtain a numerical solution to the given PDE.

While RBF-FD has some provable stability properties, issues in that last step can still arise - the resulting system can be ill-conditioned or does not possess the properties for optimal convergence rates of iterative solvers (sparse linear systems are often solved iteratively). For this reason, an improved "Oversampled RBF-FD" has been proposed recently, where we demand, the discretised PDE to hold at a set of M points Y, where X \subset Y. This results in M equations for N unknowns, where M>N i.e. an overdetermined system. We can then look for its solution in a least-squares sense.

First aim of this project is to implement the oversampled RBF-FD in C++ (Much of the required machinery, such as the domain discretisation and operator approximation is already implemented in our library) and applying it to solving PDEs, mainly those, where stability issues often arise (for example, Neumann boundary conditions on irregular domains).

The task in greater detail:

- Student will be making use of the https://e6.ijs.si/medusa/ C++ library and should first familiarize himself/herself with its basic usage.
- Afterwards, a C++ script for solution of the Poisson equation with RBF-FD will be supplied that the student should understand in greater detail.
- Additionally, the idea of oversampled RBF-FD will be explained in more detail.
- The student will be expected to modify the given script appropriately and implement oversampled RBF-FD method.
- The student will then perform analyses of the method. This part of the project is open in a sense that the student can choose which aspects of the method he/she wants to focus on. Examples include:
    - Analysing the stability of the method, by looking at the properties of the matrices involved in the solution procedure, especially the final sparse matrix (which is rectangular in the oversampled version).
    - The first bullet point will be mainly performed experimentally on known difficult cases - difficult geometries, non-optimal discretisation sets and Neumann boundary conditions.
    - Computational complexity of the method can be analysed as well and compared with some (simple) stabilisation procedures in the usual RBF-FD.
- If time allows it, student can also:
    - Work on further improving his/her oversampled RBF-FD implementation (make the code more efficient or extendable).

Andrej Pozun

Stencil selection in meshless methods

The stencil selection is a crucial part in meshless discretisation, and it is far from understood. From the early days of meshless evolution, two approaches have been commonly used. While some authors constructed stencil from nodes that lie within a certain radius others selected a certain number of nearest nodes. Recently, more sophisticated stencils have been proposed where stencil nodes are quasi-uniformly distributed around the central node. The aim of this project is to experiment with different stencil selection strategies on solving Poisson's equation.

The task in greater detail:

- The student will be making use of the https://e6.ijs.si/medusa/ C++ library and should first familiarise himself/herself with its basic usage.
- Afterwards, a C++ script for solution of the Poisson equation with RBF-FD will be supplied that the student should understand in greater detail.
- The student will be provided with literature about existing stencil selection strategies.
- The student will then implement and analyse the effects of different stencil selection strategies. This part of the project is somewhat open in a sense that the student can choose which strategies to focus on and implement original ideas if they arise.

Miha Rot

Python toy model RBF-FD implementation

There are multiple variations on the radial basis function-generated finite difference (RBF-FD) method ranging from the use of exotic basis functions e.g. divergence free RBF, to completely different strategies of imposing the approximation constraints e.g. oversampled RBF-FD. While it is beneficial from the performance standpoint to have the production code implemented in C++ it is much faster to prototype and try new variations in an interpreted language like python. The aim of this project is to build a RBF-FD framework in python that allows for a simple adaptation of its constituent parts.

The task in greater detail:

- The student will be provided with an introduction to how the RBF-FD method works and the existing C++ implementation.
- Afterwards, the student will work on designing an adaptable framework architecture and implement the basic RBF-FD algorithm.
- If the time permits the adaptable framework should then be used to implement one of the modern variations to the algorithm and test the improvement on a simple test case e.g. the Poisson equation.

Miha Rot

Implicit surface fill algorithm

Using the zero isosurface of a scalar field is one of the most general and dimension agnostic ways to describe a surface while also possessing multiple beneficial properties. The most important being that implicit surfaces can be trivially combined to form a new implicit surface, and that it is relatively simple to calculate the surface normal. Those properties can be used to solve problems that arise while attempting to discretize complex surfaces with point clouds that are suitable for use with meshless methods for differential operator approximation. The general idea is to combine the signed distance fields (SDF) describing separate objects into a single implicit surface and use those same SDFs as potential fields to move discretization points along that surface into problematic areas like re-entrant corners.The aim of this projects is to develop an algorithm that discretizes an arbitrary composition of implicit surfaces with meshless ready point clouds.

The task in greater detail:

- The student will implement SDFs for some basic primitives (e.g. a circle and a square), functionality to combine them and explore their prospects for use as potential fields.
- Afterwards, the student will work on developing a general algorithm that can discretize the complex surfaces created by combining those primitives.
- The work can be continued in multiple directions at the students discretion. Either to optimise the algorithm (translating points along implicit surfaces is expensive and only required for the most problematic parts), generalise the algorithm to other dimensions, discretize a real (non-primitive generated) complex surface, implement it in C++ and/or use it to solve a PDE.

Miha Rot