UNIVERSIDADE NOVA DE LISBOA

MScCBBi

MASTER IN
COMPUTATIONAL BIOLOGY & BIOINFORMATICS

NUNO FRANCISCO PEREIRA DE BRAGANÇA
MEng in Chemical Engineering

Knowledge transfer for biopharmaceutical production: data analysis of different process development campaigns

Sep, 2025

itqb nova
NOVA SCHOOL OF SCIENCE & TECHNOLOGY
NOVA MEDICAL SCHOOL
NOVA IMS Information Management School
DESDE 1902 INSTITUTO DE HIGIENE E MEDICINA TROPICAL UNIVERSIDADE NOVA DE LISBOA

UNIVERSIDADE NOVA
DE LISBOA

Knowledge transfer for biopharmaceutical production: data analysis of different process development campaigns

by

Nuno Francisco Pereira de Bragança

Dissertation presented for obtaining the Master of Science Degree in

Computational Biology and Bioinformatics

Supervisor: Prof. Dr. Rafael Sousa Costa

Co-Supervisor: Prof. Dr. Rui Manuel Freitas Oliveira

Examination Committee

Prof.ª Dr.ª Isabel Cristina de Almeida Pereira da Rocha

Prof. Dr. Francisco Jorge Dias Oliveira Fernandes

Prof. Dr. Rafael Sousa Costa

September 2025

Knowledge transfer for biopharmaceutical production: data analysis of different process development campaigns

Acknowledgments

I would like to thank my friends and family for their unconditional support throughout this journey. Your unwavering belief in my abilities and your constant encouragement have been my greatest source of strength. Thank you for being my pillars of support during both the challenging and triumphant moments.

A special acknowledgment goes to my supervisor, Prof. Rafael Costa, and my co-supervisor, Prof. Rui Oliveira. Your guidance, mentorship, and constructive feedback have been instrumental in shaping my research work. Thank you for your patience and for pushing me to strive for excellence. Your expertise and support have made a profound impact on my academic journey.

I am also deeply thankful to the incredible team at the Systems Biology and Engineering (SBE) group where I had the privilege to be. Your valuable insights and collaboration have significantly enriched my research experience, and I am grateful for the knowledge and skills I have gained from all of you.

ABSTRACT

This study investigates knowledge transfer in the biopharmaceutical sector, in the topic of monoclonal antibody production in bioreactors. The main objective of this work was to establish a data processing pipeline, which included the initial extraction of amino acid sequences, allocation of titer values, calculation of stoichiometric coefficients, and determination of feed conditions that enables the *in silico* generation of synthetic datasets, relating to monoclonal antibodies (mAb) production in chinese hamster ovary (CHO) cells, thereby enhancing subsequent process development campaigns. A dynamic model proposed in literature and open-source modeling methods developed were employed to establish a robust pipeline allowing efficient data flow.

Results showcase the functionality of the developed pipeline with the retrieval of a feasible list of 72 mAb sequences from a curated database of bioactive molecules and categorisation by low and high titer. Additionally, 1,080 synthetic datasets were generated and further processed using Principal Component Analysis (PCA), revealing two distinct clusters of mAbs, which underscores the robustness of the proposed methodology. This processed data is positioned for integration into hybrid models, aimed at optimizing predictions for future development campaigns and facilitating knowledge transfer within the biopharmaceutical industry.

Keywords: monoclonal antibody, biopharmaceutical production, dynamic model, Python, CHO cells, knowledge transfer

RESUMO

Este estudo investiga a transferência de conhecimento no setor biofarmacêutico, no tema produção de anticorpos monoclonais (mAbs) em bioreatores. O principal objetivo deste trabalho foi de desenvolver uma *data pipeline* para a análise de dados, que inclui a extração inicial de sequências de aminoácidos, alocação de valores de titulação, cálculo de coeficientes estequiométricos e determinação de condições de alimentação que permitem a geração *in silico* de conjuntos de dados sintéticos relevantes da produção de mAbs em células *chinese hamster ovary* (CHO), por consequente melhorando as campanhas de desenvolvimento de processos subsequentes. Um modelo dinâmico proposto na literatura e métodos de modulação de código aberto foram utilizados para estabelecer uma *data pipeline* que permite o eficiente fluxo de dados.

Os resultados mostram a funcionalidade da *pipeline* desenvolvida com a extração de 72 sequências de mAbs, categorizadas por baixa ou alta titulação. Adicionalmente, 1080 conjuntos de dados sintéticos foram gerados e processados usando Análise de Componentes Principais (PCA), revelando dois aglomerados de mAbs, que sublinham a robustez da metodologia aqui proposta. Estes dados ficam posicionados para serem integrados em modelos híbridos, com o objetivo para otimizar previsões para futuras campanhas de desenvolvimento e facilitar a transferência de conhecimento na indústria biofarmacêutica.

Termos chave: anticorpos monoclonais, produção biofarmacêutica, modelo dinâmico, Python, células CHO, transferência de conhecimento

List of Contents

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ADC	antibody-drug conjugate
API	Application programming interface
CBM	constraint-based modeling
CD	circular dichroism
CDRs	complementarity determining regions
CH1	constant heavy chain domains 1
CH2	constant heavy chain domains 2
CH3	constant heavy chain domains 3
ChemEMBL	Chemical EMolecule Bioinformatics Library
CHO	Chinese hamster ovary
CL	constant light chain domain
CSV	Comma-separated values file
EGFR	epidermal growth factor receptor
Fab	variable region
FBA	Flux Balance Analysis
Fc	constant region
FT-IR	Fourier-transform infrared spectroscopy
IEEE 754	IEEE Standard for Floating-Point Arithmetic
IgM	Immunoglobulin M
IgD	Immunoglobulin D
IgA	Immunoglobulin A
IgG	Immunoglobulin G
IgE	Immunoglobulin E
INN	International Nonproprietary Name
IVP	Initial Value Problem

mAb	Monoclonal Antibody
MARs	matrix-attachment regions
MS-HPLC	high-performance liquid chromatography
NP	nanoparticle
ODEs	ordinary differential equations
PCA	Principal Component Analysis
pip	Python package manager
URL	Uniform resource locator
VH	Variable heavy chain domain
VL	Variable light chain domain
X	Biomass
XML	Extensible Markup Language
WHO	World Health Organisation

Intracellular Species

ACCOA	Acetyl-CoA
ADP	adenosine diphosphate
GDP	guanosine diphosphate
AKG	Alpha-ketoglutarate
AMP	Adenosine Monophosphate
ATP	Adenosine triphosphate
CIT	Citrate
F6P	Fructose-6-Phosphate
G6P	Glucose-6-Phosphate
GAP	Glyceraldehyde-3-Phosphate
GLU	Glucose

MAL	Malate
NAD	Nicotinamide Adenine Dinucleotide
NADP	Nicotinamide Adenine Dinucleotide Phosphate
OXA	Oxaloacetate
PEP	Phosphoenolpyruvate
PYR	Pyruvate
R5P	Ribulose-5-Phosphate
SUC	Succinate
X5P	Xylulose-5-Phosphate

Amino acids

ALA	Alanine (A)
ARG	Arginine (R)
ASN	Asparagine (N)
ASP	Aspartic Acid (D)
CYS	Cysteine (C)
EGLN	Extracellular Glutamine (Q)
EGLU	Extracellular Glutamic Acid (E)
GLY	Glycine (G)
HIS	Histidine (H)
ILE	Isoleucine (I)
LEU	Leucine (L)
LYS	Lysine (K)
MET	Methionine (M)
PHE	Phenylalanine (F)
PRO	Proline (P)

SER	Serine (S)
THR	Threonine (T)
TYR	Tyrosine (Y)
VAL	Valine (V)

Other extracellular species

EGLC	Extracellular Glucose
EPYR	Extracellular Pyruvate
LAC	Lactate
NH4	Ammonium

1. INTRODUCTION

1.1 Background

1.1.1 Monoclonal Antibodies (mAbs)

Immunoglobulins—also known as antibodies because of their capacity to attach to antigens—constitute a cornerstone of the adaptive immune system and are classified into five distinct isotypes: IgM, IgD, IgA, IgG, and IgE [1]. Each isotype plays a unique role in immune responses, contributing to the body's ability to recognize and neutralize pathogens. The diversity among these isotypes allows the immune system to tailor its response to different types of threats, ensuring a robust defense against infections.

Within human plasma, IgG dominates the antibody pool and appears in four subclasses (IgG1–4); it is followed by IgM and IgA while IgD and IgE are present only in trace amounts measured in micro and nanograms per milliliter, respectively [2]. The predominance of IgG reflects its critical role in long-term immunity and memory responses. This is particularly important in the context of vaccinations, where IgG levels can indicate the effectiveness of the immune response.

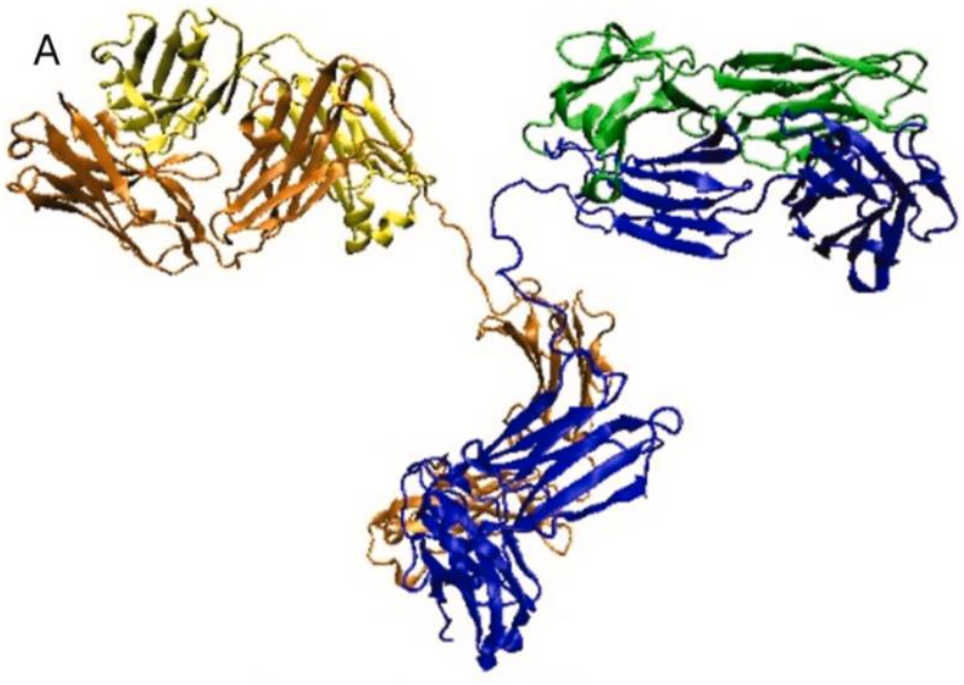

Figure 1.1: 3D Crystal structure of a selected IgG model, the 1IGT

(adapted from [2])

Recent advances in biotechnology have broadened the spectrum of therapeutic modalities beyond traditional small-molecule drugs [3]. This expansion opens avenues for tackling diseases that have previously resisted curative interventions [3]. The development of these new therapies has been driven by a deeper understanding of molecular biology and immunology, which has paved the way for innovative treatment strategies. As a result, researchers are now able to design therapies that are more targeted and effective.

Among the emerging platforms, biologics—particularly therapeutic monoclonal antibodies (mAbs)—have seen a marked rise in clinical utilization over the past ten years [3]. A monoclonal antibody is a Y-shaped glycoprotein built from two identical heavy chains and two identical light chains, all held together by disulfide linkages [4]. This structure allows for specific targeting of antigens, which is essential for effective treatment. The precision of mAbs makes them suitable for a variety of applications, including cancer therapy and autoimmune diseases, where traditional treatments may fall short.

Monoclonal antibodies can be engineered in several formats, including murine, chimeric, humanized, fully human, antibody-drug conjugates, and bispecific constructs [5]. The majority of therapeutic mAbs belong to the IgG1 subclass, which is the most prevalent and thoroughly investigated human IgG isotype [6]; these molecules are sizable, with a molecular mass close to 150 kDa [7]. Their size and structure influence their distribution and efficacy in the body, impacting how they are administered and their pharmacokinetics. This has led to the development of various delivery methods to optimize their therapeutic potential.

Structurally, IgG1 comprises two heavy chains each containing variable (VH) and constant (CH1, CH2, CH3) domains, paired with two light chains bearing variable (VL) and constant (CL) domains [3]. Consequently, an mAb consists of four polypeptide chains: two light chains (~25 kDa each) and two heavy chains (~50 kDa each) [7]. This intricate assembly is crucial for the antibody's functionality, as it determines how well the antibody can bind to its target. The arrangement of these chains also affects the stability and solubility of the antibody in therapeutic formulations.

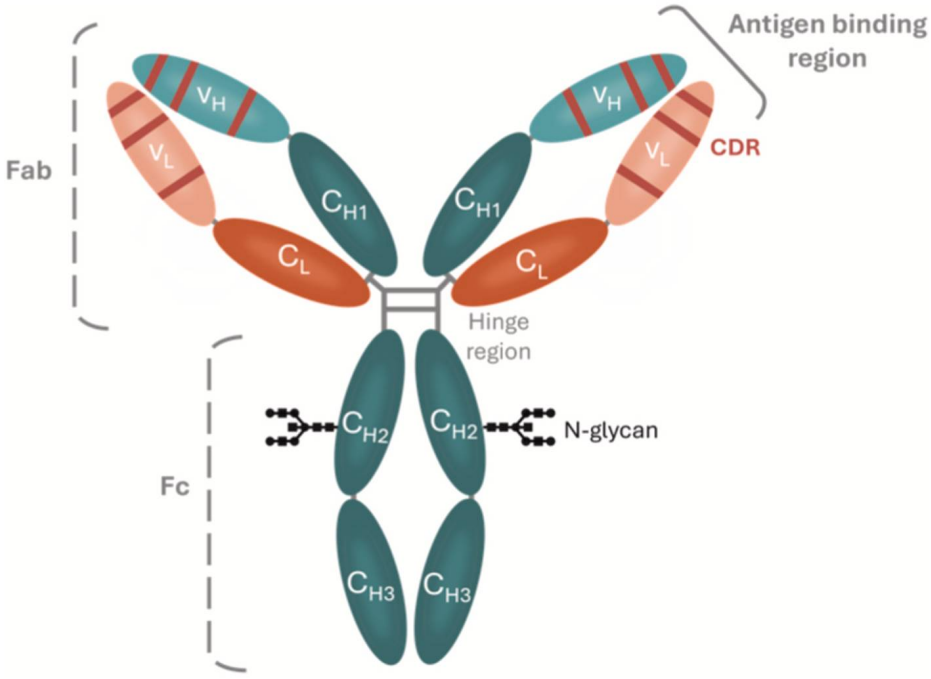

Figure 1.2: Structure of an immunoglobulin G (IgG). The Fab portion is formed by the interaction of the VH and CH1 domains from a heavy chain with a light chain, while the Fc portion is created by the dimerization of the CH2 and CH3 domains. These Fab and Fc portions are linked by the hinge region, which contains several conserved disulfide bridges.

(adapted from [8])

Like naturally occurring antibodies, mAbs are stabilized by disulfide bonds that join the two heavy chains and the two light chains, giving rise to the characteristic "Y" architecture that includes both Fab and Fc regions [6][9]. Each Fab fragment houses three complementarity determining regions (CDRs) that exhibit high sequence variability, thereby endowing the antibody with precise antigen specificity [10]. This specificity is vital for minimizing off-target effects during treatment, ensuring that the therapeutic action is directed at the intended target.

The Fab portion is assembled from the VH, CH1, VL, and CL domains, whereas the Fc region is formed by the pair of CH2 and CH3 domains [3]. This modular arrangement renders IgG1 a dynamic, multi domain protein [3]. The flexibility of this structure allows for adaptability in binding to various antigens, which is particularly beneficial in rapidly evolving pathogens. This adaptability is a key factor in the ongoing development of mAbs against emerging infectious diseases.

Functionally, the Fab segment mediates high affinity binding to target antigens, while the Fc domain orchestrates downstream immune responses and effector activities [3]. These two sections are linked by a hinge region which is essential for the functional versatility of the antibody, allowing it to

engage with multiple targets. The flexibility provided by the hinge is particularly advantageous in complex biological environments where antigens may be presented in various conformations.

The proximal part of the hinge imparts flexibility to the Fab arms, permitting a single IgG1 molecule to engage multiple antigens concurrently [3]. Every chain therefore contains a constant (Fc) region and a variable (Fab) region, the latter dictating antigen recognition [4]. This duality enhances the antibody's ability to respond to diverse pathogens, making it a powerful tool in therapeutic applications.

The ability of monoclonal antibodies to selectively bind protein targets, combined with their prolonged circulatory half-life, makes them attractive candidates for treating a wide array of human diseases [4]. While the variable region supplies the diversity necessary for antigen engagement, the constant region determines the effector functions associated with each antibody isotype [11]. Moreover, pharmacokinetic attributes such as an extended serum half-life enhance their therapeutic appeal [12]. This extended half-life allows for less frequent dosing, improving patient compliance and overall treatment outcomes.

Because of this long half-life, IgG molecules are the preferred immunoglobulin class for developing monoclonal antibody therapeutics [13]. The design of these therapies often involves careful consideration of the antibody's structure and function to maximize efficacy while minimizing side effects. As research continues, the potential for new modifications and enhancements to mAbs is vast, paving the way for next-generation therapies.

In summary, the intricate structure and diverse functionalities of immunoglobulins, particularly monoclonal antibodies, underscore their significance in modern medicine. Their ability to specifically target antigens while engaging the immune system effectively positions them as vital components in the treatment of various diseases.

1.1.1.1 Therapeutic Uses

Therapeutic mAbs have become one of the most successful classes of biopharmaceuticals in modern medicine. By the beginning of 2024 more than 125 distinct mAbs had received regulatory approval, and over 200 additional candidates were progressing through clinical development, underscoring the rapid expansion of this modality [14].

They combine unparalleled target specificity with favorable pharmacokinetic properties, a modular architecture amenable to extensive engineering, and a proven track record across a wide spectrum of diseases. Their continued evolution promises to expand therapeutic options even further, addressing unmet medical needs that were previously considered intractable [9][15][16].

Their impact stems largely from the extraordinary specificity that each antibody can achieve for its target antigen, a property conferred by the highly variable CDRs located in the Fab fragment [9]. Coupled with an intrinsically long serum half-life—often measured in weeks rather than hours—IgG based antibodies can maintain therapeutic concentrations with relatively infrequent dosing [17][18].

The therapeutic landscape for mAbs originally centered on oncology and autoimmune disorders, where early successes such as anti-CD20 (e.g., ocrelizumab) for multiple sclerosis and anti-TNF-α (e.g., infliximab) for rheumatoid arthritis demonstrated the clinical utility of antibody-mediated immune modulation [19][20][21]. Over the past decade, however, the scope of application has broadened dramatically to include chronic non-malignant diseases such as asthma, chronic obstructive pulmonary disease, osteoporosis, and migraine [22][23]. This diversification reflects both advances in antibody engineering—allowing fine-tuning of affinity, effector function, and half-life—and improvements in manufacturing and purification processes that ensure high-quality, homogeneous products [24][25].

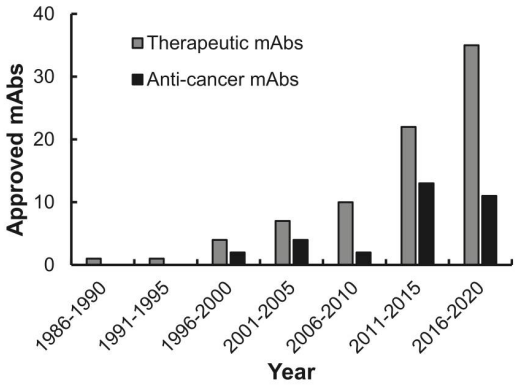

Figure 1.3: Approved therapeutic mAbs by the FDA.

(adapted from [26])

The historical roots of monoclonal technology trace back to the mid 1970s, when Köhler and Milstein pioneered the hybridoma technique that fused antibody-producing B cells with immortal myeloma lines, thereby generating stable clones capable of secreting single-specificity antibodies [27]. This breakthrough catalyzed a paradigm shift in both basic research and clinical therapeutics, making

it possible to produce antibodies with defined specificity at scale [28]. Since then, continuous innovations—including humanization, Fc engineering, and the emergence of multispecific formats—have cemented monoclonal antibodies as a cornerstone of precision medicine [29][30][31].

Adjusting complement activity is crucial for optimizing therapeutic effectiveness while reducing off-target effects in the design of monoclonal antibodies (mAbs). Strategies that enhance complement activity are mainly utilized for mAbs aimed at pathogens, infected cells, tumors, or immunosuppressive cells in the tumor microenvironment. Conversely, strategies that silence complement activity are better suited for mAbs employed in autoimmune or inflammatory conditions, as well as in transplantation scenarios.

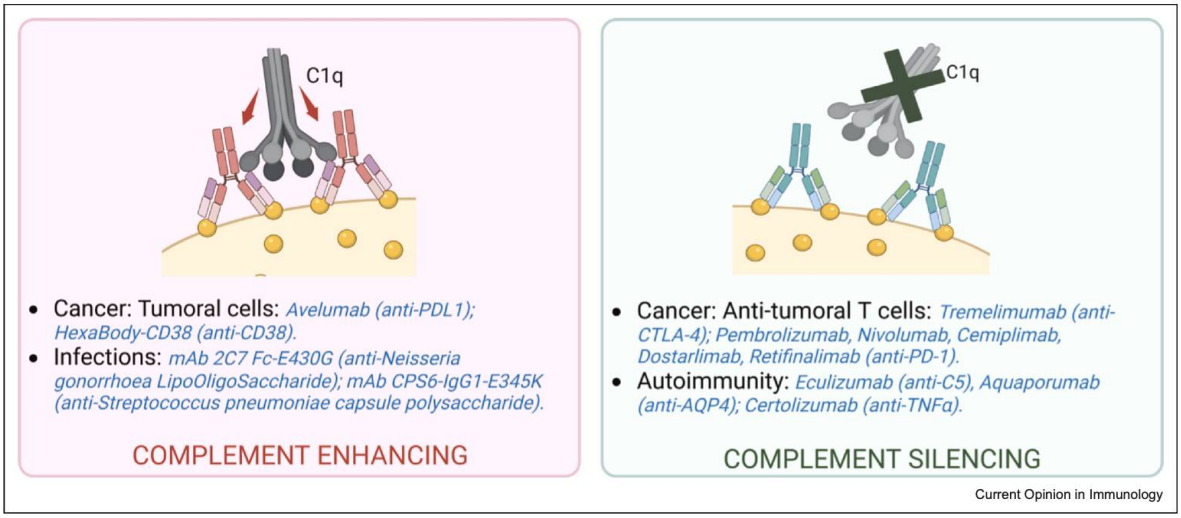

Figure 1.4 Key pathological contexts for mAbs.

Complement activation is either enhanced (left, red) or inhibited (right, green) through the design of monoclonal antibodies (mAbs), along with examples of representative mAbs.

(adapted from [32])

The landscape of oncology has been dramatically reshaped by the introduction of mAbs, which home in on distinct receptors that drive tumor proliferation and survival [33]. These therapies have revolutionized cancer treatment by providing targeted approaches that minimize damage to healthy tissues. By zeroing in on cancer cells—or on molecular markers that accompany them—these engineered proteins boost therapeutic outcomes and open the door to highly personalized treatment regimens [34]. Their remarkable ability to recognize a single epitope endows mAbs with the capacity to modulate cellular processes at the molecular level [33]. This specificity is crucial for enhancing the effectiveness of treatments while reducing side effects.

Among the most celebrated examples, trastuzumab (Herceptin) [35] and pertuzumab (Perjeta) specifically bind to the HER2/neu receptor, delivering clinical benefit in both gastric and breast malignancies [36]. These agents have transformed the management of HER2-positive cancers, leading to improved survival rates. Likewise, cetuximab (Erbitux) and panitumumab (Vectibix) are directed against epidermal growth factor receptor (EGFR), making them valuable tools against non-small-cell lung cancer and colorectal cancer [37]. Their use has been associated with significant improvements in patient outcomes, particularly when combined with other therapies.

Agents such as bevacizumab (Avastin) and other VEGF inhibitors thwart tumor angiogenesis, thereby starving a variety of cancers of their blood supply [38]. This approach not only inhibits tumor growth but also limits the potential for metastasis, making it a critical component of cancer treatment strategies. Targeting immune-related antigens also proves effective: rituximab (Rituxan) attacks CD20 and is employed in chronic lymphocytic leukemia as well as non-Hodgkin lymphoma [39]; blinatumomab (Blincyto) focuses on CD19 for B-cell malignancies [40]. These therapies harness the immune system to fight cancer, showcasing the potential of immunotherapy.

PD-1 inhibitors such as pembrolizumab (Keytruda) and nivolumab (Opdivo) effect on T-cells enhance anti-tumor immunity [41]. This innovative approach has led to remarkable responses in various cancers, including melanoma and lung cancer. In melanoma, CTLA-4 inhibition via ipilimumab (Yervoy) provides a complementary boost to immune activity [41]. The combination of these therapies can lead to synergistic effects, improving overall efficacy.

Other disease-specific antibodies include brentuximab vedotin (Adcetris), which homes in on CD30 in Hodgkin lymphoma [42], and daratumumab (Darzalex), which targets CD38 in multiple myeloma [43]. These targeted therapies exemplify the growing trend of precision medicine in oncology. Emerging targets such as GPC3 for hepatocellular carcinoma and MUC16 for ovarian cancer illustrate the expanding scope of antibody-based therapy [43]. This ongoing research highlights the potential for developing new treatments for previously hard-to-treat cancers.

To amplify their potency, investigators are engineering mAbs as delivery vehicles for radioisotopes, toxins, or chemotherapeutic payloads, and numerous immunoconjugates are now progressing through clinical evaluation [44]. This innovative strategy enhances the therapeutic index of existing drugs, allowing for more effective treatments with fewer side effects. When a cytotoxic drug is chemically tethered to an antibody, the resulting construct is known as an antibody-drug conjugate (ADC) [33]; the antibody component steers the lethal agent directly to malignant cells, acting as a precise targeting guide [33]. This targeted delivery system is particularly advantageous in minimizing systemic toxicity.

Beyond small molecule drugs, antibodies can be linked to nanocarriers—forming mAb NPs—that ferry diverse cargos such as additional drugs, nucleic acids, or imaging probes [33]. These nanoparticle platforms span polymeric, gold, and liposomal formulations, each offering unique advantages for delivery [33]. The versatility of these platforms allows for tailored approaches to treatment, enhancing the efficacy of therapies while improving patient outcomes.

Functionally, ADCs bind to tumor associated antigens on the cell surface, are internalized, and then release their cytotoxic payload intracellularly after the linker is cleaved, ensuring that the drug acts where it is needed most [34][45]. This mechanism of action underscores the importance of targeted therapies in modern oncology,

In theory, mAbs could be engineered to recognize a broad spectrum of biological agents—including bacteria, viruses, fungi, and their toxins. This versatility opens up exciting possibilities for therapeutic interventions across various infectious diseases. Their activity may be exerted directly (for example, by blocking pathogen entry or neutralizing toxins) or indirectly (by shaping inflammatory pathways or enhancing opsonophagocytic clearance) [46][47][48]. This dual mechanism of action allows mAbs to play a multifaceted role in combating infections.

1.1.1.2 Bacterial Applications

To date, the only antibacterial mAbs that have received regulatory approval target secreted exotoxins; the resulting antigen antibody complexes are chiefly cleared by the reticulum-endothelial system [49][50][51][47]. These approved therapies highlight the potential of mAbs in addressing toxin-mediated diseases. A different strategy would involve antibodies that bind to structural components on the bacterial surface—such as membrane proteins—thereby triggering direct killing or immune-mediated cytotoxicity [47] . This approach could enhance the effectiveness of mAbs against a wider range of bacterial pathogens.

Because a single antibody dose can persist for weeks to months, using recombinant antibodies prophylactically might reduce reliance on traditional antimicrobials and lessen selective pressure for cross-resistance, while also sparing the gut flora—a consideration that gains importance as we learn more about the microbiome's role in health and disease [47][52][53]. This potential for long-lasting protection could significantly alter the landscape of infection prevention.

Successful widespread adoption of mAbs for established bacterial infections, however, hinges on rapid and precise pathogen identification [7]. This requirement drives the need for continued investment in validated point-of-care diagnostics (e.g., PCR-based assays and fluorescence in-situ

hybridization) that can deliver timely, pathogen-specific information [54]. The integration of advanced diagnostic tools will be crucial for the effective deployment of mAbs in clinical settings.

Given the durability of a single dose, recombinant antibodies could also serve as prophylaxis, potentially curbing the overall consumption of conventional antibiotics [7]. This shift could help mitigate the growing problem of antibiotic resistance, which poses a significant threat to public health.

Consequently, several clinical development programs are now aiming to prove that anti-bacterial mAbs can protect high risk populations—for instance, preventing Staphylococcus aureus ventilator-associated pneumonia in mechanically ventilated intensive-care patients [54]. These studies are essential for establishing the efficacy and safety of mAbs in vulnerable patient groups.

1.1.1.3 Viral Applications

Monoclonal antibodies can achieve potent, broad-spectrum neutralization by targeting highly conserved viral epitopes [7]. This ability to target conserved regions of viruses enhances the likelihood of effective treatment across different strains. They may also be directed against host-cell receptors or co-receptors that viruses exploit for entry [55]. This strategy can block viral entry at the cellular level, providing a robust defense against infection.

Although still in early stages, antibodies that recognize virus-infected cells and trigger antibody-dependent cellular cytotoxicity represent an additional possible mechanism of action [56]. This approach could enhance the immune response against infected cells, further improving therapeutic outcomes.

To date, licensed mAbs include an HIV rescue therapy for adults and a prophylactic antibody against respiratory syncytial virus (RSV) for high-risk infants [57][58]. Ibalizumab, for example, binds a conformational epitope on the CD4 receptor, blocking HIV-1 entry, while other candidates under investigation either target host antigens or directly bind viral proteins [59][60][61]. These developments illustrate the potential of mAbs in addressing viral infections.

Current viral targets under study also encompass influenza, Ebola, and Zika [48]. Recent outbreaks have underscored the necessity of preparing for emerging infections [48]. The ability to rapidly develop mAbs against new viral threats could be crucial in controlling outbreaks.

For such emergent threats, mAbs could be deployed both as treatment for infected individuals and as targeted prophylaxis to protect persons and interrupt transmission—potentially altering the trajectory of a nascent epidemic [62]. This proactive approach could significantly enhance public health responses to infectious disease outbreaks.

An illustrative case involves ongoing product development against influenza: broadly neutralizing antibodies have shown promise in ferret models and early-phase clinical trials, suggesting they could be employed during the early stages of a pandemic for high-risk groups such as exposed contacts and healthcare workers [62][63][64]. This strategy could provide a critical buffer until more traditional interventions, such as vaccines, become widely available.

In this context, monoclonal antibodies could serve as a bridge until strain-specific vaccines become available [65]. Their rapid deployment in response to emerging viral threats could help mitigate the impact of outbreaks, providing immediate protection to at-risk populations. This flexibility in application underscores the importance of mAbs in the evolving landscape of infectious disease management.

Moreover, the development of mAbs against a variety of pathogens highlights the potential for these therapies to be integrated into broader public health strategies. As we continue to face new and re-emerging infectious diseases, the role of monoclonal antibodies could expand significantly. Their ability to be tailored for specific targets makes them a valuable asset in both therapeutic and preventive settings.

The ongoing research into mAbs also emphasizes the need for collaboration between scientists, clinicians, and public health officials. By working together, these stakeholders can ensure that the most promising mAb candidates are prioritized for development and clinical testing. This collaborative approach will be essential for addressing the challenges posed by infectious diseases in the future.

The potential applications of monoclonal antibodies in combating bacterial and viral infections are vast and varied. Their ability to provide targeted, effective treatment options could revolutionize the way we approach infectious diseases. As research progresses, the hope is that mAbs will become a cornerstone of both therapeutic and preventive strategies, ultimately improving health outcomes for populations worldwide.

1.1.2 CHO Cells in BioPharma

1.1.2.1 Production of Antibodies

The first approved monoclonal antibody, OKT3, was developed to prevent acute kidney transplant rejection and was created using a murine hybridoma technique [66][67]. This pioneering work laid the foundation for the field of monoclonal antibodies, demonstrating their potential in clinical applications. However, initial commercial successes with mAbs faced limitations due to a restricted selection of suitable myeloma cell lines [68]. Additionally, this method often triggered unwanted immune responses, leading to the production of human anti-mouse antibodies [68]. Such immune reactions can compromise the efficacy of treatments and pose safety risks for patients.

To address these challenges and enhance effector functions, chimeric antibodies (comprising both mouse and human components) were introduced, followed by humanized antibodies (which contain less than 10% non-human sequences) and fully human mAbs [69][67]. This evolution in antibody design reflects a growing understanding of immunogenicity and the need for safer therapeutic options. Over the years, advancements in monoclonal antibody engineering have significantly improved production processes, resulting in more efficient methods and, importantly, human or humanized antibodies that present fewer safety concerns compared to earlier hybridoma-derived products or serum therapies [69][70]. This shift has been crucial for increasing patient acceptance and expanding the therapeutic applications of mAbs.

Several preferred cell lines serve as "bio-factories" for mAb production, with their selection based on characteristics such as adaptability, productivity, and both phenotypic and genotypic stability [71]. The choice of cell line is critical, as it directly impacts the yield and quality of the final product. New expression platforms have emerged, including phage-display libraries, transgenic mouse systems, Chinese hamster ovary (CHO)-based biomanufacturing, highly scalable plant-based platforms, and the option to generate antibodies from human B cells obtained from convalescent donors or vaccinated individuals [72][73][74][75]. This diversity in production methods allows for flexibility in addressing specific therapeutic needs.

CHO cells, derived from Chinese hamster ovarian tissue, were first immortalized by Theodore Puck in 1975 [76]. Today, CHO-derived lines are the predominant mammalian expression system for therapeutic proteins, with approximately 70% of biopharmaceuticals—including antibodies like dupilumab and sarilumab—produced in CHO cells [77][76]. The widespread use of CHO cells is attributed to their ability to perform post-translational modifications similar to those in humans, which is essential for the functionality of therapeutic proteins. Adaptations of the parental CHO line have led to sub-lines such as CHO-K1 and CHO-S; the latter and its derivatives have been engineered for

suspension growth, allowing for higher cell densities and, consequently, greater volumetric yields [78] [76]. This engineering is vital for meeting the increasing demand for mAbs in clinical settings.

Compared to other mammalian hosts, CHO cells offer several advantages: they readily support gene amplification, which directly enhances recombinant protein output and overall yield [79]. Therefore, stable expression systems rely on selecting high-productivity clones to maximize manufacturing efficiency [71]. Their genomic flexibility also allows for extensive cell engineering to increase productivity, although this same adaptability can lead to genetic rearrangements that destabilize the cell line [80]. Even well-selected clones may experience genotypic or phenotypic drift, resulting in product heterogeneity and fluctuations in quality [71]. This variability can complicate regulatory approval processes and affect patient safety.

Transgene silencing is another challenge for maintaining CHO stability. While viral promoters are often used to boost expression during cell line development, they are susceptible to silencing, which can reduce productivity over time [81]. The incorporation of novel expression enhancers—such as matrix-attachment regions (MARs)—presents a promising strategy to maintain long-term stability and reduce variability among transfected lines [82]. This innovation could lead to more consistent production processes and higher-quality mAbs.

Scaling up antibody production necessitates the expansion of cell banks; cells can be cultivated in suspension cultures or spinner flasks under controlled temperature and CO_2 conditions to sustain exponential growth and achieve high cell densities [71]. Following upstream cultivation, downstream processing begins with clarification and purification to remove cells and isolate the target protein [71]. This step is crucial for ensuring that the final product is free from contaminants that could affect its safety and efficacy.

Most production runs utilize a fed-batch mode, where nutrients, antifoam agents, and other supplements are added according to process requirements [83]. This approach allows for continuous nutrient supply, which is essential for maintaining optimal cell growth and productivity throughout the culture period. After harvesting, the broth undergoes clarification—typically through centrifugation and sequential filtration—to eliminate cells and debris, resulting in a clarified feedstock suitable for capture chromatography without risking column fouling [83]. This step is critical for ensuring that the purification process is efficient and that the final product meets stringent quality standards.

The final downstream steps often involve ultrafiltration/diafiltration (UF/DF). This stage concentrates the antibody to its therapeutic concentration and facilitates buffer exchange into a formulation optimal for storage [84]. The ability to concentrate the antibody effectively is vital for achieving the desired dosage forms for clinical use. Once purified and concentrated, the antibody

product must be characterized for its molecular profile, structural integrity, post-translational modifications, and functional activity [71]. This comprehensive characterization is essential for ensuring that the mAb is safe and effective for patient use.

Structural characterization employs a range of analytical techniques. Primary structure verification is routinely conducted using mass spectrometry coupled with high-performance liquid chromatography (MS-HPLC), which provides detailed information about the amino acid sequence and any modifications that may have occurred during production [85]. Secondary structure assessment utilizes methods such as Fourier-transform infrared (FT-IR) spectroscopy or circular dichroism (CD) [85]. These techniques help to confirm the proper folding and stability of the antibody, which are critical for its functionality. Insights into tertiary structure can be obtained from intrinsic tryptophan fluorescence measurements [86]. Understanding the three-dimensional conformation of the antibody is crucial for predicting its interactions with target antigens and for optimizing its therapeutic efficacy.

The production of monoclonal antibodies involves a complex interplay of biological engineering, cell culture technology, and analytical characterization. The advancements in these areas have significantly enhanced the efficiency and safety of mAb production, paving the way for their widespread use in treating various diseases. As the field continues to evolve, ongoing research and innovation will be essential for addressing the challenges associated with mAb manufacturing and ensuring that these valuable therapeutics remain accessible to patients in need.

1.1.2.2. Titer determination

Production runs in bioreactors are evaluated by measuring the product titer (g/L) and the volumetric productivity (g/L/day); these metrics help keep the process under control and can even improve overall product quality [71].

To determine the titer of monoclonal antibodies, a systematic approach is used that involves quantifying mAb concentrations in samples from bioreactors [87].

First, a sample of the culture is diluted to reduce the concentration of mAbs to a measurable level. This diluted sample is then transferred to a multi-well plate, which allows for the simultaneous analysis of multiple samples [87] .

Next, the plate is placed in a specialized instrument that measures how well the mAb binds to a sensor surface. This binding is directly related to the concentration of the mAb in the sample. To ensure accurate measurements, a set of known standards is included for comparison, allowing for calibration of the results [87].

The entire process is relatively quick, taking about 30 minutes to analyze a full plate of samples. Additionally, while the primary focus is on determining the titer, other aspects such as the integrity and purity of the mAb can also be assessed using complementary techniques. This comprehensive approach enables researchers to effectively evaluate mAb concentrations and their quality in biopharmaceutical development [87].

In the context of mAb titer determination, a high titer is generally considered to be up to 10 g/L, while a low titer is defined as being below 0.5 g/L [87] [88]. This classification helps in assessing the efficiency of the production process and the potential yield of mAbs in biopharmaceutical applications.

1.1.3. Modelling of biological systems

Mathematical modeling can be employed to explain or predict the behavior of a system, particularly in the context of biological processes. It serves as a powerful tool for understanding both internal and external cell interactions and how these interactions influence cell metabolism. By quantifying these relationships, researchers can gain insights into the dynamics of cellular functions and the overall behavior of biological systems.

This interest in mathematical modeling has stimulated the development of genome-scale networks, which allow researchers to perform *in silico* simulations of complex biological systems. These networks provide a framework for analyzing how various components of a biological system interact and how these interactions affect metabolic pathways. By simulating these processes, scientists can explore the effects of different variables on metabolic flux distributions, leading to a deeper understanding of cellular metabolism and its regulation [89].

1.1.3.1. Constraint-based models

Metabolic-network modeling can draw on detailed knowledge of enzyme mechanisms together with experimental measurements to construct a dynamic representation of the system. Such models describe how metabolite concentrations evolve over time by solving systems of ordinary differential equations (ODEs) [90]. Each ODE incorporates an initial metabolite concentration, a reaction-rate expression, and the associated kinetic parameters [90]. This approach treats cell metabolism modeling as a problem defined by the dynamics of the underlying biochemical processes, yielding explicit time-course solutions for both transient behaviors and steady-state equilibria from

any chosen set of initial metabolite levels [90]. Kinetic rate laws are derived from biochemical and mechanistic insights, and the resulting fluxes emerge directly from those laws combined with the equilibrium metabolite concentrations [90].

These frameworks have been successfully applied to relatively small, well-characterized pathways in organisms such as Saccharomyces cerevisiae [91] and Escherichia coli [92]. However, extending dynamic representations to genome-scale networks is often hampered by the scarcity of experimentally determined kinetic data needed to formulate accurate rate laws [90]. Defining a dynamic model involves specifying the interaction mechanisms that underpin the network and choosing an appropriate form for the kinetic rate expressions [90]. The model is then expressed as a set of (typically nonlinear) ODEs that trace the temporal trajectories of the system's variables.

Nonetheless, this modeling paradigm demands extensive datasets—kinetic constants, enzyme abundances, and other parameters—that are not always accessible [93]. Parameterizing large-scale models can therefore be labor-intensive and computationally demanding [90]. When experimental time-course data are available, the model outputs can be validated against multiple measurement points [90]. This validation process is crucial for ensuring the reliability and accuracy of the model, ultimately enhancing its utility in predicting metabolic behaviors in various biological contexts. In a deterministic, continuous framework, ordinary differential equations integrate the network's stoichiometry, the specified initial conditions, and the mathematical formulas that define each reaction's rate such that:

$$\begin{aligned} \frac{dx(t)}{dt} &= S \cdot r\left(x(t),\, u(t),\, p\right) \\ y(t) &= g(x(t),\, u(t),\, p) \\ with\, x(0) &= x_0\left(p\right) \end{aligned}$$

(Eq. 1)

where:

• S is a $i \times j$ stoichiometric matrix whose entries are the coefficients that describe how each of the j reactions consumes or produces each of the i metabolites in the network.

• $x(t)$ is an i-dimensional state-vector that holds the time-dependent concentrations (or amounts) of all metabolites.

• $r(\cdot)$ is a j-dimensional vector of reaction rate expressions. Each element of r is a function of the current state $x(t)$, any external inputs $u(t)$, and a set of kinetic parameters p

• $g(\cdot)$ is a mapping that links the internal state $x(t)$ to observable model outputs $y(t)$ [94].

For some models the ODE system is augmented with algebraic constraints (resulting in a differential algebraic system) or with additional differential equations that capture phenomena such as changes in compartment volume or intracellular dilution [94].

One of the most common strategies for constructing a dynamic system is the bottom-up, or forward, approach. This method assembles all system components and their interconnections by providing mechanistic descriptions of each entity—from molecular actors to whole-organism processes—and then validates and refines the model against experimental data [90]. In kinetic-model networks, the building blocks are the rate laws and the associated parameter values, which can be sourced from the literature, laboratory measurements, or public data repositories [90]. When some parameters are missing or only loosely bounded, dedicated parameter-estimation techniques become necessary, which is a frequent requirement of bottom-up workflows [90]. This approach allows for a comprehensive understanding of the system by starting from the fundamental components and building up to the overall dynamics.

Conversely, top-down approaches begin with experimental observations and adjust existing models to accurately reproduce the measured data [90]. This method enables the incorporation of previously unknown mechanisms, interactions, or properties that were not captured in the original formulation [90]. The prevailing trend in top-down modeling is to devise objective functions that allow models to represent high-level biological mechanisms with comparatively simple rate expressions while preserving the essential connections among network elements [95]. This approach is particularly useful for refining models based on empirical data, ensuring that they remain relevant and accurate in light of new findings.

1.1.3.2. Dynamic models

In contrast to dynamic formulations, constraint-based approaches limit the description of a metabolic network to its reaction stoichiometry and reversibility, assuming the system operates at steady state and therefore cannot capture transient dynamics [90]. These models are expressed as linear equation systems that are typically under-determined—the number of unknown fluxes exceeds the number of independent equations—resulting in infinitely many mathematically valid flux vectors [90]. To obtain a single, biologically plausible solution, additional restrictions (constraints) and a chosen objective function are imposed, allowing for the identification of an optimal flux distribution [96].

The core of constraint-based modeling (CBM) rests on the stoichiometric matrix and enforces limits derived from physical and chemical principles such as mass balance, thermodynamic feasibility, and permissible flux ranges [90]. Solutions are steady-state flux configurations that lie within a feasible region often visualized as a “flux hypercone” [97]. When applied to a stationary system, CBM

predictions generally agree well with experimental measurements and, crucially, bypass the arduous task of defining kinetic rate laws and estimating kinetic parameters. Only minimal information—upper and lower bounds for each reaction flux—is required to infer system-level properties [90]. The most prevalent algorithm for extracting an optimal flux pattern is Flux Balance Analysis (FBA) [98].

Although CBM does not incorporate explicit time-course metabolite concentrations or transient behavior, it eliminates the need for detailed mechanistic knowledge, relying solely on flux bounds [90]. Consequently, the solution space of a dynamic model is a subset of the broader constraint-based solution space, as the former adds extra constraints derived from kinetic data [99]. Nonetheless, as the demand for richer representations of biological networks grows, kinetic information is increasingly being integrated into constraint-based frameworks [90]. This integration enhances the predictive power of CBM while maintaining its computational efficiency, making it a valuable tool in systems biology.

1.2 Related-work

1.2.1 Model describing mAb production

Robitaille et al. (2015) describes a comprehensive kinetic framework that couples extracellular nutrient uptake, intracellular metabolic fluxes (glycolysis, pentose-phosphate pathway, TCA cycle) and the biosynthetic demands of monoclonal antibody synthesis [100]. The model was calibrated on four experimental cultivations—two batch and two fed batch runs—using two distinct serum-free media. By employing a single set of kinetic parameters, the model accurately reproduces viable-cell growth, substrate consumption, metabolite accumulation, and mAb titer across all conditions. This demonstrates that a unified mechanistic representation can reliably predict both cell physiology and product formation, providing a valuable basis for *in silico* evaluation of media formulations and feeding strategies in recombinant CHO processes.

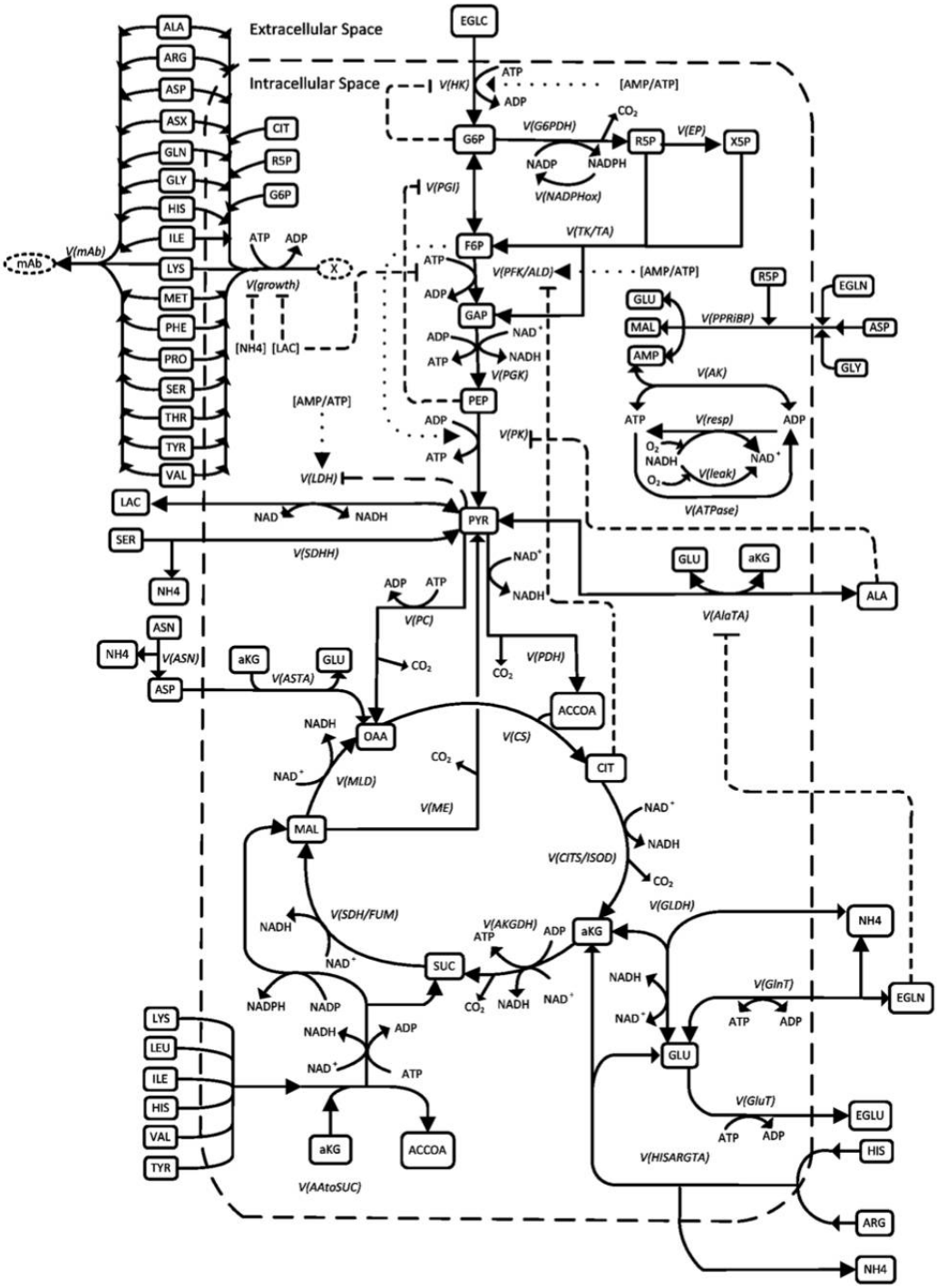

Figure 1.5 Simplified diagram of the metabolic network described by the model. Solid arrows denote the biochemical reactions (glycolysis, pentose-phosphate pathway, TCA cycle, and mAb related biosynthetic steps). Dashed lines indicate inhibition mechanisms, while dotted lines represent activation mechanisms.

(adapted from [100])

1.3 Motivation and Objectives

The primary objective of this work is to develop a robust data processing pipeline for knowledge transfer in the production of monoclonal antibodies. In the biopharma sector, the acquisition of high quality experimental data is hampered by several challenges: proprietary data are often confidential, and obtaining experimental measurements for each of the 72 distinct mAb molecules under study is impractical. This scarcity of data limits the ability to build generalized predictive models for mAb manufacturing and underscores the need for alternative, data efficient approaches. To address these gaps, the goal is to establish an in silico production simulation data pipeline that generates and analyses synthetic datasets for 72 mAbs. These datasets will be used to demonstrate the feasibility of the data pipeline and to illustrate how pre processed, synthetic data can serve as a foundation for training hybrid models that predict and optimize mAb production processes.

1.4 Structure of the thesis

Introduction - The opening chapter sets the stage for the work. It begins with a Background subsection that introduces three foundational topics. First, it explains monoclonal antibodies and their therapeutic applications. Next, it describes chinese hamster ovary (CHO) cells and their role in biopharmaceutical protein production, covering both the basic process of protein synthesis and the methods used to determine titers. Finally, it surveys the landscape of mathematical modeling, contrasting constraint-based approaches with dynamic models. After establishing this context, the Related-work subsection reviews existing models that describe mAb production. The chapter then moves to Motivation and Objectives, clarifying why the study is needed and what it aims to achieve.

Materials and Methods - This chapter details the resources and techniques employed throughout the research. It starts with a description of the specific mAbs under investigation. The CHO virtual lab notebook follows, outlining the kinetic equations that govern the system, the ODEs derived from those kinetics, and the formulation of the initial value problem that drives simulations. The next section, python environments, lists the software packages and computational setup used for analysis. Finally, biochemical calculation methods delves into the derivation of stoichiometric coefficients and the computation of reacted amounts, providing the quantitative backbone for later results.

Results - The findings are presented in a logical progression. An initial data flows subsection explains the supporting shell scripts that automate parts of the data pipeline. This is followed by a presentation of the amino acid sequences relevant to the mAbs studied. The mAb titer allocation section breaks down how sequence information translates into model equations and the resulting titer

values. Subsequent subsections report the calculated stoichiometric coefficients and the feed conditions—including feed composition. The chapter then introduces synthetic experimental data, describing new inputs allowing for design of experiments studies and comparing high versus low titer datasets. The reacted amounts portion showcases the implemented calculation functions and visualizes example outcomes. Data preprocessing is addressed through data normalisation. Finally, a principal component analysis section details the implementation of PCA, reports the extracted principal components and scores, and displays clustering visualizations that highlight patterns in the data.

Conclusions - The closing chapter synthesizes the work's main takeaways in concluding remarks, reflecting on how the objectives were met and what insights were gained. It ends with a future work segment that outlines promising directions for extending the research, such as refining models, expanding experimental validation, or exploring additional bioprocess variables.

2. METHODOLOGY AND PROPOSED SOLUTION

To achieve the results presented in this thesis, various methodologies were integrated. This included a list of specific monoclonal antibodies (mAbs) selected for detailed study, along with a Jupyter notebook capable of generating synthetic datasets that represent the production of these mAbs. Additionally, specific Python environments were used to run the necessary scripts and analyses, and biochemical calculation methods were employed to process and analyze the generated data.

2.1 mAb information

The list of monoclonal antibodies utilized to test the data pipeline of this thesis was obtained from the publication by Walsh et al. (2022), which has information about the recent biopharmaceuticals approved in the United States and the European Union up until 2022 [101]. The information is organized using International Nonproprietary Names (INNs), which are essential in the pharmaceutical industry for providing a standardized, unique identifier for active ingredients [102]. Managed by the World Health Organization (WHO) since its establishment in 1950, the INN system aims to create a universal nomenclature for pharmaceutical substances. INNs are considered public property, allowing for unrestricted use, which is crucial for healthcare professionals, researchers, and regulatory bodies worldwide [102].

2.2 CHO virtual lab notebook

A Jupyter notebook previously developed at the SBE lab in Nova FCT, Lisbon in order to produce syntentic datasets for another case scenario was used as a first guideline [103]. The scheme of this valuable material is depicted below.

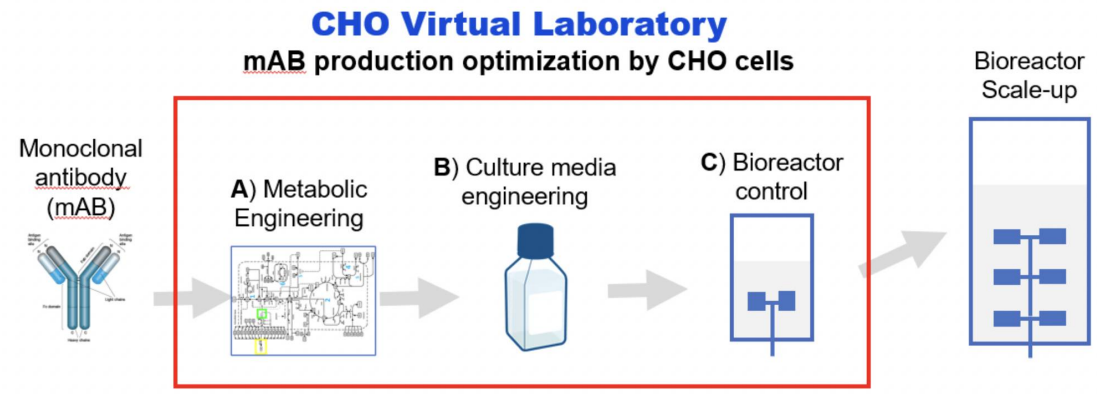

Figure 2.1: Scheme of Jupyter Notebook used as guideline

This was capable of generating a dataset for one mAb experiment using the metabolic model developed in [100]. This model was simulated by adjusting two key parameters: the pre-induction feeding rate and the post-induction feeding rate. The dynamic model consists of 25 extracellular species and 21 intracellular species. Over a period of 240 hours, concentrations of the species were recorded as time series, with samples collected every 24 hours. The duration of the simulation and the frequency of data collection could be modified, and in this thesis it was kept at a period of 240 hours and sample collection every 12 hours to ensure a wider data range.

2.2.1 Kinetic Equations

The kinetic equations used to test the data pipeline were the ones described in [100]. For example, the kinetic equation for the monoclonal antibody production rate (V_{MAB}) is expressed as follows:

$$V_{MAB} = V_{maxMAB} \times \left(\frac{LYS}{K_{mLYSMAB}+LYS}\right) \times \left(\frac{ILE}{K_{mILEMAB}+ILE}\right) \times \left(\frac{VAL}{K_{mVALMAB}+VAL}\right) \times \left(\frac{TYR}{K_{mTYRMAB}+TYR}\right) \times \left(\frac{ASN}{K_{mASNMAB}+ASN}\right)$$
$$\times \left(\frac{ASP}{K_{mASPMAB}+ASP}\right) \times \left(\frac{ALA}{K_{mALAMAB}+ALA}\right) \times \left(\frac{ARG}{K_{mARGMAB}+ARG}\right) \times \left(\frac{EGLN}{K_{mEGLNMAB}+EGLN}\right) \times \left(\frac{EGLU}{K_{mEGLUMAB}+EGLU}\right)$$
$$\times \left(\frac{GLY}{K_{mGLYMAB}+GLY}\right) \times \left(\frac{HIS}{K_{mHISMAB}+HIS}\right) \times \left(\frac{PHE}{K_{mPHEMAB}+PHE}\right) \times \left(\frac{PRO}{K_{mPROMAB}+PRO}\right) \times \left(\frac{THR}{K_{mTHRMAB}+THR}\right)$$
$$\times \left(\frac{ATP}{K_{mATPMAB}+ATP}\right)$$

(Eq. 2)

where V_{maxMAB} is the maximum rate of mAb production, *LYS, ILE, VAL, TYR, ASN, ASP, ALA, ARG, EGLN, EGLU, GLY, HIS, PHE, PRO, THR*, and *ATP* are the concentrations of the respective species *SPE*, and $K_{mSPEMAB}$ represent a constant of a certain species *SPE* in the production reaction of the mAb.

2.2.2 Ordinary Differential Equations (ODEs)

The ODEs used to test the data pipeline were the ones described in [100]. For instance, the ODE representing the rate of change of Asparagine is defined as follows:

$$\frac{d[\mathrm{ASN}]}{dt} = -\left(\frac{V_{maxASN} \times \mathrm{ASN}}{K_{mASN} + \mathrm{ASN}}\right) - 3.6 \times 10^{-4} \times V_{MAB} - 4.51 \times 10^{-5} \times \mu$$

(Eq. 3)

where V_{maxASN} represent the maximum rate of Asparagine consumption, K_{mASN} is a constant specific to Asparagine and μ represents the specific growth rate of the biomass, indicating how quickly the biomass is increasing or decreasing over time.

2.2.3 Initial Value Problem

An initial value problem (IVP) for ordinary differential equations specifies a system of time-dependent equations and the state of the system at an initial time [104]. Formally, it is the pair

$$\frac{dx}{dt} = f(t, x), \quad x(t_0) = x_0$$

(Eq. 4)

where $x(t)$ is the vector of state variables, $f(t,x)$ defines their time derivatives, t_0 is the initial time, and x_0 is the initial state vector. The IVP formulation uniquely determines the time evolution of the system and is what numerical integrators solve to produce $x(t)$ on the requested time interval [104].

The system was set to be integrated on the interval 0–240 h using the explicit Runge–Kutta 4(5) integrator from *scipy.integrate.solve_ivp* [105]. The initial state vector $x(0)$ consisted of the measured initial concentrations/amounts for all modeled species. Solutions were returned at 21 equally spaced output times on 0–240 h, which corresponded with sample collection every 12 hours.

2.3 Python virtual environments

To execute the Jupyter notebook, it was necessary to create a dedicated Python environment using the *venv* module to accommodate the outdated library versions from the guideline Jupyter notebook which was developed before 2023. This was accomplished by first establishing a new virtual environment with the command *python3 -m venv old_env*, which created a directory named *old_env* containing the necessary files for the isolated environment. After creating the environment, it was activated using the command source *old_env/bin/activate*. Within this activated environment, the required packages were installed using *pip*, specifying the necessary versions to ensure compatibility with the original code. For instance, the command *pip install package_name==version* was used for each package.

In line with this approach, another virtual environment was created for processing new data, which could utilize the latest versions of the required libraries. This environment was established using the same command *python3 -m venv new_env* and activated similarly. Within this environment, the latest versions of the necessary packages were installed, allowing for efficient data processing without compatibility issues. This setup facilitated the analysis of new data while maintaining the integrity of the original code. The main version of the packages for *old_env* include *SciPy* 1.15.3, *NumPy* 1.23.5, *matplotlib* 3.6.0 and *pandas* 2.2.2. For the new_env, the versions are *SciPy* 1.16.1, *NumPy* 2.2.6, *matplotlib* 3.10.3 and *pandas* 2.3.1.

Therefore, each Jupyter notebook could be launched within their environment allowing the Python code to run as intended. This method ensured that all dependencies were correctly managed, enabling the notebook to function as originally designed.

2.4 Biochemical Calculation Methods

By applying fundamental principles of chemistry, various biochemical calculation methods are employed both before and after the *in silico* simulations in the production of monoclonal antibodies. Prior to the simulations, stoichiometric coefficients involving the amino acids are determined to establish the necessary ratios for the reactions involved in antibody synthesis. Following the simulations, the reacted amounts are calculated.

2.4.1 Feed concentrations

The ratios of specific amino acid counts relative to the count of Alanine can be calculated. The feed concentration of Alanine, C_{ALA} is set to 20 mM, in line with the order of magnitude presented in [100], which is used as a scaling factor for the calculated ratios using the expression:

$$C_j = \frac{Y_{j/mAb}}{Y_{ALA/mAb}} \cdot C_{ALA}$$

(Eq. 5)

where C_j is the feed concentration of another amino acid *j*. $Y_{j/mAb}$ is the amount of amino acid j in the mAb sequence and $Y_{ALA/mAb}$ the amount of ALA in the mAb sequence.

2.4.2 Stoichiometric Coefficients

To calculate the molecular weight of each mAb, specifically immunoglobulin G (IgG), and the coefficients for each amino acid, the process begins with counting the amino acids in the light and heavy chains, referred to as C_L and C_H. The total counts of amino acids are determined by summing the counts from each chain. The total number of bonds formed is equal to the sum of the total counts from both chains.

$$\text{Total Bonds} = C_L + C_H$$

(Eq. 6)

Next, the amount of water released during bond formation is calculated using the formula

$$\text{Water Released} = -(\text{Total Bonds}) \times 18.01528$$

(Eq. 7)

which provides the mass of water released in grams per mole.

The weighted counts for the light and heavy chains are computed by multiplying the count of each amino acid by its respective molecular weight, such that if C_A is the count of amino acid A and MW_A is its molecular weight, then

$$\text{Weighted Count} = C_A \times MW_A$$

(Eq. 8)

The total molecular weight of IgG is then obtained by summing the weighted counts of both chains and adding the water released, followed by multiplying the result by 2:

$$\text{MW(IgG)} = \left(\sum \text{Weighted Counts}_{\text{Light Chain}} + \sum \text{Weighted Counts}_{\text{Heavy Chain}} + \text{Water Released}\right) \times 2$$

(Eq. 9)

Finally, the coefficients for each amino acid in the light chain are calculated by dividing each amino acid count by the total IgG molecular weight.

$$\text{Coefficient for Amino Acid A} = \frac{C_A}{\text{MW(IgG)}}$$

(Eq. 10)

This results in a set of coefficients that can be utilized in the dynamic model.

To calculate the coefficients for ATP in relation to the molecular weight of immunoglobulin G, *MW(IgG)*, first the following intermediate steps are performed:

1. The ATP coefficient is determined by taking the total number of amino acids from both the light and heavy chains, multiplying this sum by 2, and then dividing by the IgG molecular weight.

$$\text{ATP Coefficient} = \frac{(C_L + C_H) \times 2}{\text{MW(IgG)}}$$

(Eq. 11)

2. The GDP coefficient is calculated by taking the total number of amino acids from both chains, multiplying this sum by 4, subtracting 8, and then dividing by the IgG molecular weight.

$$\text{GDP Coefficient} = \frac{((C_L + C_H) \times 4) - 8}{\text{MW(IgG)}}$$

(Eq. 12)

3. The ADP coefficient is simply calculated by dividing 4 by the IgG molecular weight.

$$\text{ADP Coefficient} = \frac{4}{\text{MW(IgG)}}$$

(Eq. 13)

Finally, the total coefficient for ATP is obtained by summing the ATP, GDP, and ADP coefficients.

2.4.3 Reacted Amounts

The variables further analysed in this thesis excluded the intracellular species. Let $Ra_{S,t}$ denote the reacted amount of species S at time t, quantifying how much of the species has reacted or been consumed in the system. To calculate $Ra_{S,t}$ first $S_t \cdot V_t$ can be derived which calculates the total amount of species S present in the system at time t based on its concentration and the current volume. Furthermore, $S_0 \cdot V_0$ gives the initial amount of species S, while $(V_t - V_0) \cdot S_{feed\text{-}conc}$ calculates the amount added due to the feed, considering the change in volume from the initial state to the current state, $S_{feed\text{-}conc}$ being the concentration of species S in the feed being introduced into the system. Therefore, the reacted amounts for each extracellular species $Ra_{S,t}$ can be calculated as follows:

$$Ra_{S,t} = S_t \cdot V_t - (S_0 \cdot V_0 + (V_t - V_0) \cdot S_{feed-conc})$$

(Eq. 14)

By subtracting the total amount of species S that was initially present and added through the feed from the current total amount in the system, the equation effectively quantifies the amount of

species *S* that has reacted or been consumed. This calculation is essential for understanding the dynamics of the reactions and the consumption of extracellular species over time.

For the biomass, $Ra_{Xv,t}$, since there is no biomass being added, this can be done as follows:

$$Ra_{Xv,t} = Xv_{,t} \cdot V_{,t} - (Xv_{,0} \cdot V_{,0})$$

(Eq. 15)

In what concerns the product, $Ra_{mAb,t}$, the calculation simplifies even further, since there is also no initial amount:

$$Ra_{mAb} = mAb_{,t} \cdot V_{,t}$$

(Eq. 16)

2.4.3.1 Data normalisation for Principal Component Analysis

The reacted amounts data was also analysed to assess skewness. If the data has a long tail on one side, either left or right, it may indicate skewness. In a skewed distribution, the mean is typically pulled in the direction of the skew [106]. Specifically, if the mean is greater than the median, the data is positively skewed; conversely, if the mean is less than the median, it is negatively skewed. When dealing with highly skewed data, normalization becomes essential to enhance the performance of statistical analyses and machine learning models.

One effective data normalization technique is the “Yeo-Johnson transformation”, which is an extension of the “Box-Cox transformation” that can handle both positive and negative values [106]. This technique was chosen because it is particularly ideal for datasets with reacted amounts since the values can change between positive and negative several times within one experiment.

This allows the data to be processed using Principal Component Analysis (PCA), a statistical technique used to reduce the dimensionality of data while preserving as much variance as possible [107]. In this case, the number of principal components is set to two, facilitating a two-dimensional representation of the data, which makes it easier to visualize and interpret. The principal components are designated as PC1 and PC2, representing the directions in which the data varies the most. The corresponding principal component scores are calculated for each observation, indicating their position in this reduced-dimensional space. Additionally, the explained variance ratio for each principal component is provided, which quantifies the proportion of the total variance in the dataset that is

captured by each component. This information is crucial for understanding how well the PCA has summarized the original data, ultimately enabling it to be integrated into a hybrid model.

3. RESULTS

This section aims to present the code developed as part of this thesis, which is archived at *https://codeberg.org/nfb/mabs*.

3.1 Data Flow Diagram

The directories containing the thesis were designed in a way for them to be able to be moved with ease. This involved not containing system files in the directories themselves, but instead only the versions of the *Python* environments which can be installed again via a *pip* command, and also by not including the full user-specific absolute paths on the python code, but instead the relative paths. The data pipeline was organised in 7 different sections with the potential for real laboratory data to be included, which are depicted below in figure 3.1.

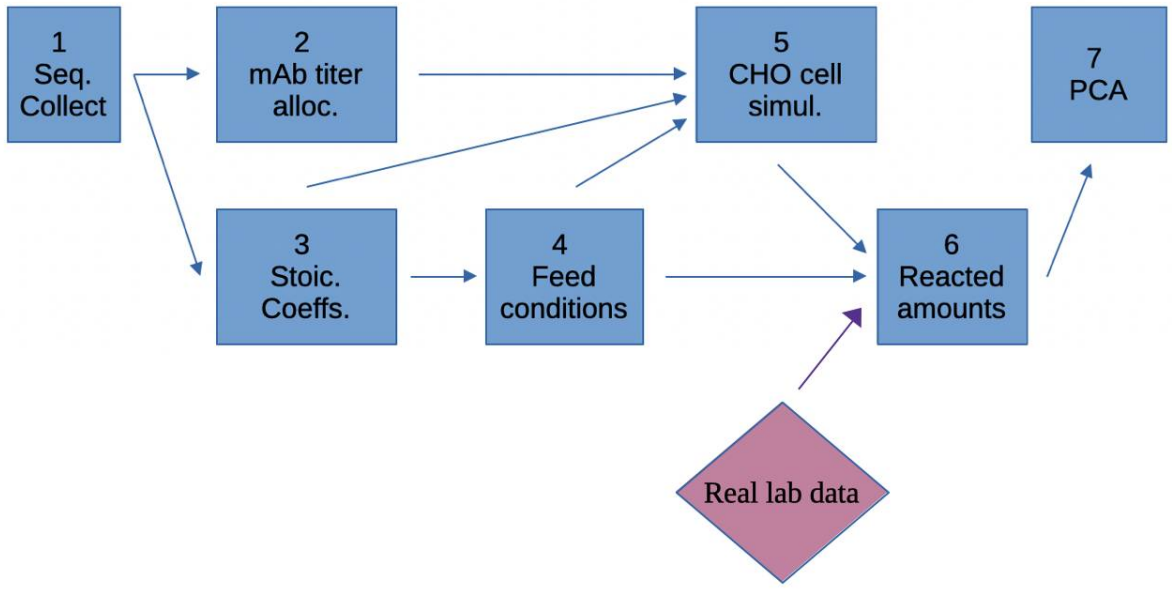

Figure 3.1: Data pipeline scheme. The arrows represent the flow of data. "1Seq Collect" represents the extraction of sequences implemented in section 3.2 of this document. "mAb titer aloc." represents the allocation of a titer value to each mAb implemented in 3.3. "3Stoic. Coeffs." refers to the calculation of stoichiometric coefficients in 3.4. "4Feed Conditions" mentions the calculated feed conditions for the mAbs and is shown in 3.5. "5CHO cell simul." represents the in silico simulation of the synthetic datasets in 3.6. "6Reacted amounts" calculates the reacted amounts of the extracellular species in 3.7 and "7PCA" implements data normalisation and Principal Component Analysis from the reacted amounts data in 3.9. "Real lab data" is not part of this thesis.

3.1.1 Shell Scripts

Shell scripts designed to be run by a Unix shell were developed for a smoother operation. These involved allowing opening jupyter notebooks in different python environments and creating the folders relating to the mAbs names ahead of the simulations according to a pre-defined list.

The script *open_jupyter.sh* (Annex 1.1) activates the virtual environment located at the specified path. It prompts the user to provide the path to the Jupyter Notebook folder, which on the most common desktops can be done by dragging the folder to the prompt. It then captures the user's input with the command read *jupyter_path*, which stores the path in the variable *jupyter_path*. The script assumes that each folder (which is one section of the thesis) contains only one file with the .ipynb extension, having the same file name as the folder itself. Finally, the script launches the Jupyter Notebook application.

The script *create_folders.sh* (Annex 1.2) prompts the user to provide the path to a "low titer value" directory and captures the input using the command *read low_value_dir*. It does similarly for a "high titer value" directory. Next, the user is requested to enter a list of names and values of vmaxmab in the format "name,value," indicating that the user should press Ctrl+D when finished. The script then enters a loop that reads user input until the end of the list is reached. For each line of input, it splits the line into a name and a value using the comma as a delimiter. It checks if the value is below a threshold and if the condition is met, the script creates a new folder with the name of the mAb in the low value directory using the name provided, ensuring the directory structure is created and outputs a message indicating the creation of the folder. If the value is higher than the defined threshold, it creates a new folder with the mAb name in the high value directory instead, following the same process.

3.2 Amino Acid Sequences

3.2.1 Extracting Monoclonal Antibody INNs

This section of thesis starts by presenting python code designed to extract unique words from PDF documents, specifically targeting words that end with the substring "mab". The implementation utilizes the *PyPDF2* library for reading PDF files and the RE library for regular expression operations.

The core function, *extract_unique_mab_words*, accepts a single parameter, *pdf_path*, which specifies the location of the PDF file to be processed. The function initializes a set called unique_words to store the unique occurrences of words that meet the specified criteria.

The PDF file is opened in binary read mode using a context manager to ensure proper resource management. A *PdfReader* object is instantiated to facilitate the extraction of text from each page of the document. The script iterates through all pages, extracting text content where available.

To identify words that end with "mab", a regular expression is employed. The pattern used is designed to match any word character sequence followed by "mab", optionally followed by specific punctuation marks or whitespace. The matches found are then added to the *unique_words* set, ensuring that only distinct entries are retained. Finally, the function returns the set of unique words ending with "mab".

Upon execution of this code, with the publication by Walsh et al. (2022) used as example in this thesis, it resulted in a total of 155 INNs of monoclonal antibodies.

3.2.2 Monoclonal Antibody Identifiers from ChEMBL API

The code was then designed to retrieve unique mAb identifiers from the ChEMBL database using its API.

First, it loops through the list of unique mAb names. For each name, it constructs a URL to query the ChEMBL API, appending the mAb name to the base URL. The code then sends a GET request to this URL to fetch data.

Once the response is received, it parses the XML content using the ElementTree library. It searches for elements that contain the mAb identifiers and retrieves the text of each identifier. These

identifiers are then stored in a dictionary, where the ChEMBL ID serves as the key and the corresponding mAb name as the value.

Finally, the code prints the count of successfully retrieved identifiers from the dictionary.

With the example of this thesis, the goal was to obtain 155 identifiers, corresponding to the list of mAbs. However, the final count was 129, indicating that some requested data may have not been available or returned by the API.

A new pandas DataFrame named *df* was created. This DataFrame is initialized with a dictionary that contains four key-value pairs. The first key, 'Name', is assigned the values from *mabs_ids*, which is expected to be a collection where the values represent the names of mAbs. The second key, 'ChemEMBL ID', is initialized with a list of *None* values, with the length equal to the number of entries in *mabs_ids*. This column is intended to store ChemEMBL identifiers for the corresponding mAbs.

3.2.3 Extraction and Filtering of Chain Sequences

On the same *df* Dataframe, a third column 'Light Chain' is also initialized with *None* values, indicating that the light chain information for each mAb will be populated later. Similarly, the fourth key, 'Heavy Chain', follows the same pattern, initialized with *None* values to hold the heavy chain information for each mAb.

The provided code snippet iterates over a collection of monoclonal antibody identifiers stored in *mabs_ids*. For each identifier, it constructs a URL that points to the corresponding biotherapeutic data on the ChEMBL API. A GET request is then sent to this URL using the requests library.

Upon receiving a response, the code checks if the status code is 200, indicating a successful request. If successful, the response content is parsed as XML using the ElementTree library.

Two lists, *light_chain_sequences* and *heavy_chain_sequences*, are initialized to store the sequences of the light and heavy chains, respectively. The code then iterates through each *biocomponent* found in the XML structure. For each, it retrieves the *description* and *sequence* elements.

The code checks if the description indicates a 'light chain' or 'heavy chain'. If the description contains 'light chain', the corresponding sequence is appended to the *light_chain_sequences* list. Similarly, if the description indicates a 'heavy chain', the sequence is added to the

heavy_chain_sequences list. An exception handling mechanism is in place to catch any *AttributeError* that may arise if the description or sequence is None.

The code updates the DataFrame *df* with the retrieved sequences. If any light chain sequences are found, the first sequence is assigned to the appropriate row in the DataFrame based on the matching ChemEMBL ID. If no light chain sequences are found, a message is printed. The same process is repeated for heavy chain sequences. If the initial API request fails, an error message is printed, indicating the received status code.

The provided code snippet creates a new DataFrame named *filtered_df* by filtering the existing DataFrame *df*. This filtering process selects only those rows where both the 'Light Chain' and 'Heavy Chain' columns contain non-null values. The condition is specified using the *notna()* method, which checks for the presence of valid (non-null) entries in each of the specified columns.

The logical operator & is used to combine the two conditions, ensuring that only rows meeting both criteria are included in the new DataFrame. The code then prints the contents of *filtered_df*, which for the example of this thesis returned a total of 72 entries. This filtered DataFrame consists solely of monoclonal antibodies that possess complete light and heavy chain information, facilitating further analysis or processing of these relevant entries. The complete Python code is displayed in Annex 2.1.

Table 3.1: Structure of initial data frame with the sequences of amino acids of each mAb in the Light and Heavy Chain respectively

	Name	ChemEMBL ID	Light Chain	Heavy Chain
0	satralizumab	CHEMBL3833307	DIQMTQSPSSLSASVGDSVTITCQASTDISSHLNWYQQKPGKAPEL...	QVQLQESGPGLVKPSETLSLTCAVSGHSISHDHAWSWVRQPPGEGL...
1	naxitamab	CHEMBL4297984	EIVMTQTPATLSVSAGERVTITCKASQSVSNDVTWYQQKPGQAPRL...	QVQLVESGPGVVQPGRSLRISCAVSGFSVTNYGVHWVRQPPGKGLE...
2	emicizumab	CHEMBL3833393	DIQMTQSPSSLSASVGDRVTITCKASRNIERQLAWYQQKPGQAPEL...	QVQLVQSGSELKKPGASVKVSCKASGYTFTDNNMDWVRQAPGQGLE...
3	ravulizumab	CHEMBL3989986	DIQMTQSPSSLSASVGDRVTITCGASENIYGALNWYQQKPGKAPKL...	QXVQLVQSGAEVKKPGASVKVSCKASGHIFSNYWIQWVRQAPGQGL...
4	secukinumab	CHEMBL1743068	EIVLTQSPGTLSLSPGERATLSCRASQSVSSSYLAWYQQKPGQAPR...	EVQLVESGGGLVQPGGSLRLSCAASGFTFSNYWMNWVRQAPGKGLE...
...	...	...	...	...
67	durvalumab	CHEMBL3301587	EIVLTQSPGTLSLSPGERATLSCRASQRVSSSYLAWYQQKPGQAPR...	EVQLVESGGGLVQPGGSLRLSCAASGFTFSRYWMSWVRQAPGKGLE...
68	dostarlimab	CHEMBL4298124	DIQLTQSPSFLSAYVGDRVTITCKASQDVGTAVAWYQQKPGKAPKL...	EVQLLESGGGLVQPGGSLRLSCAASGFTFSSYDMSWVRQAPGKGLE...
69	olaratumab	CHEMBL1743049	EIVLTQSPATLSLSPGERATLSCRASQSVSSYLAWYQQKPGQAPRL...	QLQLQESGPGLVKPSETLSLTCTVSGGSINSSSYYWGWLRQSPGKG...
70	tixagevimab	CHEMBL4650265	EIVLTQSPGTLSLSPGERATLSCRASQSVSSSYLAWYQQKPGQAPR...	QMQLVQSGPEVKKPGTSVKVSCKASGFTFMSSAVQWVRQARGQRLE...
71	loncastuximab	CHEMBL4297238	EIVLTQSPAIMSASPGERVTMTCSASSGVNYMHWYQQKPGTSPRRW...	QVQLVQPGAEVVKPGASVKLSCKTSGYTFTSNWMHWVKQAPGQGLE...

72 rows × 4 columns

All 72 rows can be viewed in Annex 3.1 which also contains other variables mentioned in the next sub-chapters.

3.3 mAb titer allocation

Since this data pipeline is designed to process several mAb molecules, the V_{maxmab} value will be different each time a simulation is done with another molecule. This data is often kept with a proprietary license and not available to the public by biopharma companies. Therefore, in order to create and test this section of the data pipeline, each mAb is allocated a value for V_{maxmab} according to its amino acid sequence.

A preprocessing step is necessary to ensure that the sequences consist solely of valid amino acid letters. It employs the *str.replace('1', 'X')* method to replace all instances of the character '1' with 'X', as the presence of '1' is not representative of any standard amino acid.

Then, a new column called 'Combined Chains' was created by concatenating the 'Light Chain' and 'Heavy Chain' columns within the DataFrame *df*. This operation merges the amino acid sequences from both chains into a unified sequence for each monoclonal antibody.

3.3.1 Sequence encodes

A function was created to convert an amino acid sequence into a numerical value using the IEEE 754 floating-point representation. The *amino_acid_to_ieee754* function takes an amino acid sequence as input and performs several key operations:

1. Mapping Amino Acids to Binary: A dictionary is defined to map each amino acid to a corresponding 5-bit binary string. This mapping includes standard amino acids as well as a placeholder for unknown amino acids.
2. Padding the Sequence: A nested function, *pad_sequence*, ensures that the input sequence is padded with 'X' characters until its length is a multiple of 12. This is necessary for subsequent processing.
3. Splitting the Sequence: Another nested function, *split_sequence*, divides the padded sequence into segments of 12 characters each.
4. Conversion to IEEE 754: For each 12-character segment, the function constructs a 64-bit binary string by prefixing '0000' to the concatenated binary representations of the amino acids.
5. Calculating the Value: The function computes the numerical value based on the IEEE 754 format. The final value for each segment is accumulated into a total sum.

The function ultimately returns the total value derived from the entire amino acid sequence.

A new column named 'IEEE754' was created in the DataFrame *df* by applying the *amino_acid_to_ieee754* function to the 'Combined Chains' column. This operation processes each amino acid sequence in the 'Combined Chains' column, converting it into a numerical value represented in the IEEE 754 floating-point format.

Finally, the expression *df['IEEE754'].is_unique* is used to determine whether all values in the 'IEEE754' column of the DataFrame (df) are unique, which was true for the example of this thesis.

3.3.2 Model Equation

In this section, the data is preprocessed to prepare for building a binary classification model resembling the following sigmoid function:

$$p = \frac{1}{1 + e^{-(b_0 + b \cdot aa)}}$$

(Eq. 17)

where p is the probability of high mAb titer, aa is a decimal number corresponding to an amino acid sequence and b_0 is the value for the intercept of the sigmoid equation it is part of and b is the coefficient for aa.

The following steps are undertaken:

1. Extracting Values: The numerical representations from the 'IEEE754' column of the DataFrame *df* are converted into a list named *values*.
2. Scaling the Values: A scaling factor of 10^{263} is applied to each value in the list. This scaling is performed to adjust the magnitude of the values, since the IEEE754 are extremely small which can be detrimental for numerical stability during model training. The scaled values are stored in a new list called *scaled_values*.
3. Standardizing the Values: The mean and standard deviation of the scaled values are calculated using the *statistics* module. The values are transformed to have a mean of 0, such that half of the values would be above 0 and half below.
4. Saving the Standardized Values: This results in a new list of *standardized_values*, which is then added to the DataFrame as a new column.

These preprocessing steps ensure that the data is appropriately scaled and standardized, which is crucial for the effective performance of the binary classification model. Then, the following steps are done to create a training and a test set for the low titer mAbs and high titer mAbs:

1. Setting the Threshold: A threshold value of 0 is defined to categorize the standardized values into binary classes.
2. Creating a Value Dictionary: A dictionary named *value_dict* is constructed to assign a label of 0 or 1 to each standardized value based on whether it is below or above the threshold. Values less than the threshold are assigned 0, while those equal to or greater than the threshold are assigned 1.
3. Preparing the Feature Set and Target Variable: The original standardized values are used as features *X*, reshaped to fit the requirements of the *scikit-learn* library. The corresponding binary labels from *value_dict* are assigned as the target variable *y*.
4. Splitting the Data: The dataset is divided into training and testing sets using a 70-30 split, ensuring that the model can be evaluated on unseen data.

The model was then instantiated and trained using the training data, with the parameters of the equation retrieved, becoming

$$p = \frac{1}{1 + e^{-(-0.584 + 2.912 \cdot aa)}} \quad \text{(Eq. 18)}$$

where -0.584 is the intercept value and 2.912 is the coefficient for *aa*.

The equation can be visualized through a plot of the predicted high mAb titer probabilities below, with the *matplotlib.pyplot* library imported to facilitate the creation of visualizations. The original standardized values of sequence encodes are the data points.

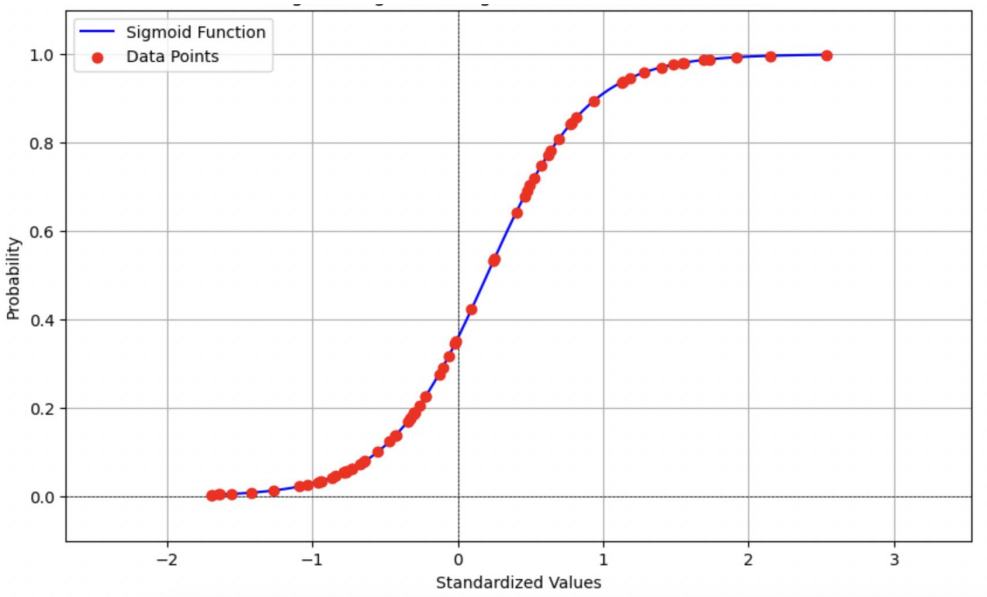

Figure 3.2: Sigmoid function derived

The function characterises the original points below and above a titer threshold.

This highlights the equation capability to model the relationship between the standardized values and the corresponding probabilities of mAb titer.

3.3.3 Values allocated

The V_{maxmab} is calculated and rounded to four decimal places according to this formula developed as part of this thesis:

$$V_{\text{maxmab}} = V_{\text{min}} \times (1 + 0.1 \times random\_float) + 10 \times V_{\text{min}} \times \frac{1}{1 + e^{-(b_0 + b \cdot aa)}}$$

(Eq. 19)

where

- V_{min} is set to 0.005 in order to allow for V_{maxmab} to be within the order of magnitude presented in [100],
- *random_float* is a real number uniformly distributed between 10 and 20 to introduce variability into the calculation,

- *aa* is a decimal number corresponding to an amino acid sequence,
- b_0 is the value for the intercept of the sigmoid equation it is part of and *b* is the coefficient for *aa*.

The results are saved in a new column of the DataFrame, each row corresponding to the mAb the value was assigned. It is then exported to a file *vmaxmab.csv* in order for the data to flow to the next sections. The complete Python code is displayed in Annex 2.2.

Table 3.2: Values calculated of V_{maxmab} for selected mAbs, with the intermediate value steps described in the section 3.3.2 Model Equation

	Name	Combined Chains	IEEE754	standardized_values	p	vmaxmab
0	satralizumab	DIQMTQSPSSLSASVGDSVTITCQASTDISSHLNWYQQKPGKAPEL...	1.821900e-261	-0.303158	0.187502	0.0143
1	naxitamab	EIVMTQTPATLSVSAGERVTITCKASQSVSNDVTWYQQKPGQAPRL...	2.191522e-261	-0.010434	0.351173	0.0113
2	emicizumab	DIQMTQSPSSLSASVGDRVTITCKASRNIERQLAWYQQKPGQAPEL...	2.809370e-261	0.478874	0.692325	0.0634
3	ravulizumab	DIQMTQSPSSLSASVGDRVTITCGASENIYGALNWYQQKPGKAPKL...	1.826830e-261	-0.299254	0.189240	0.0111
4	secukinumab	EIVLTQSPGTLSLSPGERATLSCRASQSVSSSYLAWYQQKPGQAPR...	1.873438e-261	-0.262343	0.206285	0.0129
...	...	...	...	...	...	...
67	durvalumab	EIVLTQSPGTLSLSPGERATLSCRASQRVSSSYLAWYQQKPGQAPR...	2.323348e-261	0.093967	0.423147	0.0142
68	dostarlimab	DIQLTQSPSFLSAYVGDRVTITCKASQDVGTAVAWYQQKPGKAPKL...	2.071646e-261	-0.105370	0.291038	0.0144
69	olaratumab	EIVLTQSPATLSLSPGERATLSCRASQSVSSYLAWYQQKPGQAPRL...	4.114448e-262	-1.420177	0.008844	0.0144
70	tixagevimab	EIVLTQSPGTLSLSPGERATLSCRASQSVSSSYLAWYQQKPGQAPR...	1.231277e-261	-0.770906	0.055806	0.0150
71	loncastuximab	EIVLTQSPAIMSASPGERVTMTCSASSGVNYMHWYQQKPGTSPRRW...	1.789118e-261	-0.329120	0.176255	0.0108

72 rows × 6 columns

The full table can be consulted in Annex 3.1

3.4 Stoichiometric Coefficients

In order to calculate the stoichiometric coefficients, code was first developed for analyzing amino acid sequences and calculating the weighted counts of each amino acid based on their molecular weights.

The molecular weights of various amino acids are stored in a dictionary called *tabulated_MW*, where each key represents a single-letter code for an amino acid, and the corresponding value indicates its molecular weight in grams per mole. This dictionary includes the common amino acids such as Alanine with a molecular weight of 89.093 g/mol and Tryptophan with a molecular weight of 204.228 g/mol.

Two primary functions are defined within the code:

- *count_amino_acids(sequence)*: This function takes a sequence of amino acids as input and counts the occurrences of each amino acid using the *Counter* class from the *collections* module. If the input sequence is *NaN* or *None*, the function returns an empty dictionary, indicating that no counts can be performed.

- *calculate_weighted_counts(counts, mw_dict)*: This function computes the weighted counts of amino acids by multiplying the count of each amino acid (obtained from the first function) by its corresponding molecular weight from the *tabulated_MW* dictionary. The result is a new dictionary containing the weighted counts for each amino acid present in the input counts.

The DataFrame containing information about various antibodies was loaded. The developed code begins with the initialization of a results list, where an empty list named results is created to store the outcomes of the analysis for each antibody. Next, the code iterates over the rows of the DataFrame using the *iterrows()* method. For each row, the name of the antibody is extracted, and the count_amino_acids function is called for both the Light Chain and Heavy Chain sequences. This function counts the occurrences of each amino acid in the respective chains, returning a dictionary of counts.

Following this, the total counts are calculated by summing the values of the dictionaries returned by the *count_amino_acids* function. Specifically, *total_light_chain_count* holds the sum of counts for the light chain, while *total_heavy_chain_count* contains the sum of counts for the heavy chain.

Next, the code was developed taking into consideration the equations and expressions mentioned in the Methods section, as follows:

1. Water Molecules Calculation: The variable *delete_waters* is calculated to account for the water molecules released during the formation of peptide bonds. This is done by taking the negative sum of the total counts of amino acids in both the light and heavy chains, multiplying by the molecular weight of water (approximately 18.01528 g/mol).

2. Weighted Counts Calculation: The *calculate_weighted_counts* function is called for both the light and heavy chains to compute the weighted counts of amino acids. This results in two dictionaries: *light_chain_weighted* and *heavy_chain_weighted.*

3. IgG Molecular Weight Calculation: The molecular weight of IgG (*igg_mw*) is calculated by summing the weighted counts of both chains and adding the *delete_waters* value. The total is then multiplied by 2 to account for the dimeric nature of IgG.

4. Coefficients Calculation: A dictionary named *mmolg* is created, where each amino acid count from the light chain is divided by the calculated *igg_mw.*

5. Results Compilation: A new dictionary named *data* is created to store the results for the current antibody. It includes the coefficients for the amino acids (or *None* if an amino acid is not present) and the antibody name. This dictionary is then appended to the results list.

6. The ATP coefficients calculation: a check is done to ensure that the molecular weight of IgG (*igg_mw*) is not equal to zero, which prevents division by zero errors. If this condition is satisfied, the coefficients are calculated as follows: *ampcoef*, representing the contribution of AMP, is calculated with the total number of amino acids in both the light and heavy chains, then doubling that value to reflect the dimeric nature of IgG and dividing by the molecular weight of IgG. Next, the *gdpcoef,* which represents the contribution of GDP, is calculated by applying a factor of four to the total amino acid count, subtracting a specific adjustment value and dividing by the molecular weight of IgG. Lastly, the *adpcoef*, representing the contribution of ADP, is calculated based on a fixed contribution related to the molecular weight of IgG. The total ATP coefficient is then computed by summing *ampcoef, gdpcoef,* and *adpcoef*, and this value is stored in the data dictionary under the key 'ATP'.

A new DataFrame *dfcoefs* is then populated with the results obtained from the previous calculations. containing the antibody names, ATP coefficients, and the coefficients for each amino acid. The columns are as follows:

- Name: The name of the mAb.
- ATP: The calculated ATP coefficient.
- A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y: These represent the one-letter codes for the amino acids, which will hold the corresponding ODE coefficients derived from the light chain counts.

The final line of code exports the DataFrame *dfcoefs* to a CSV file named *4coefs.csv.* The complete Python code is displayed in Annex 2.3.

Table 3.3: Representation of a selection of calculated stoichiometric coefficients

	Name	ATP	A	C	D	E	F	G	H	I	...	M	N	P	Q
0	satralizumab	0.027466	0.000091	0.000035	0.000070	0.000084	0.000056	0.000105	0.000028	0.000049	...	0.000007	0.000049	0.000070	0.000091
1	naxitamab	0.027397	0.000097	0.000035	0.000055	0.000083	0.000069	0.000083	0.000014	0.000042	...	0.000007	0.000048	0.000062	0.000104
2	emicizumab	0.027179	0.000090	0.000034	0.000076	0.000083	0.000048	0.000083	0.000014	0.000055	...	0.000007	0.000041	0.000083	0.000110
3	ravulizumab	0.027384	0.000097	0.000035	0.000069	0.000069	0.000055	0.000104	0.000014	0.000048	...	0.000007	0.000069	0.000076	0.000090
4	secukinumab	0.027234	0.000095	0.000041	0.000054	0.000081	0.000054	0.000101	0.000014	0.000041	...	NaN	0.000034	0.000081	0.000095
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
67	durvalumab	0.027290	0.000096	0.000034	0.000062	0.000082	0.000055	0.000096	0.000014	0.000041	...	NaN	0.000034	0.000082	0.000096
68	dostarlimab	0.027320	0.000097	0.000035	0.000062	0.000069	0.000062	0.000090	0.000028	0.000042	...	NaN	0.000035	0.000076	0.000097
69	olaratumab	0.027323	0.000115	0.000034	0.000054	0.000082	0.000054	0.000082	0.000014	0.000041	...	NaN	0.000048	0.000088	0.000095
70	tixagevimab	0.027340	0.000095	0.000034	0.000055	0.000082	0.000055	0.000109	0.000020	0.000041	...	NaN	0.000034	0.000075	0.000089
71	loncastuximab	0.027393	0.000097	0.000035	0.000048	0.000083	0.000042	0.000104	0.000028	0.000042	...	0.000028	0.000042	0.000076	0.000069

The full table can be consulted in Annex 3.2.

3.5 Feed conditions

It is important to define realistic feed conditions for the *in silico* simulations in order to ensure they represent correctly what occurs in the real bioreactor, as well as ensuring datasets with group of simulations with the same feed conditions can be separated from one another.

3.5.1 Feed composition

The ratios of specific amino acid counts relative to the count of Alanine were calculated and a new DataFrame was created to store these ratios. The feed concentration of Alanine, C_{ALA} is set to 20 mM, which will be used as a scaling factor for the calculated ratios, as explained with the expression in the Methods section.

A list named *columns_to_ratio* is defined, containing the one-letter codes of amino acids for which the ratios will be calculated. A new DataFrame named *ratio_df* is created by performing the following operations:

1. The *div()* method is used to divide the counts of each amino acid in the specified columns by the counts of Alanine in the same row. The *axis=0* parameter indicates that the division should be performed row-wise.
2. The result is then multiplied by C_{ala} to scale the ratios.
3. The columns of the *ratio_df* DataFrame are renamed to indicate that they represent ratios. This is done using a list comprehension that prefixes each column name with "A_".

4. Adding the Name Column: The mAb name column from the original DataFrame *df* is added to the end of the *ratio_df* DataFrame. This allows for easy identification of the corresponding antibody for each row of ratios.

5. Displaying the New DataFrame: The *dropna()* method is called on ratio_df to remove any rows that contain NaN values. This ensures that only complete data is displayed.

This process results in a new DataFrame that contains the scaled ratios of preferred amino acid concentrations relative to Alanine.

Table 3.4: Amino acid concentrations relative to Alanine

A_K	A_L	A_M	A_N	A_P	A_Q	A_R	A_S	A_T	A_V	A_W	A_Y	name
16.923077	24.615385	1.538462	10.769231	15.384615	20.000000	7.692308	50.769231	27.692308	21.538462	3.076923	15.384615	satralizumab
17.142857	17.142857	1.428571	10.000000	12.857143	21.428571	10.000000	44.285714	25.714286	25.714286	2.857143	14.285714	naxitamab
20.000000	23.076923	1.538462	9.230769	18.461538	24.615385	13.846154	44.615385	26.153846	21.538462	3.076923	15.384615	emicizumab
18.571429	24.285714	1.428571	14.285714	15.714286	18.571429	7.142857	40.000000	27.142857	21.428571	2.857143	12.857143	ravulizumab
17.333333	20.000000	1.333333	9.333333	14.666667	20.000000	12.000000	40.000000	24.000000	20.000000	2.666667	14.666667	Adalimumab
20.000000	23.076923	1.538462	7.692308	16.923077	24.615385	10.769231	55.384615	29.230769	21.538462	3.076923	15.384615	Bamlanivimab
20.000000	24.615385	1.538462	9.230769	18.461538	24.615385	9.230769	52.307692	29.230769	18.461538	3.076923	13.846154	atoltivimab
17.333333	17.333333	1.333333	8.000000	16.000000	20.000000	8.000000	42.666667	24.000000	20.000000	4.000000	10.666667	enfortumab
21.538462	23.076923	3.076923	12.307692	16.923077	21.538462	7.692308	50.769231	26.153846	23.076923	3.076923	15.384615	crizanlizumab
13.750000	18.750000	1.250000	8.750000	15.000000	17.500000	10.000000	38.750000	22.500000	20.000000	3.750000	10.000000	brodalumab
16.000000	21.333333	1.333333	9.333333	14.666667	18.666667	9.333333	44.000000	22.666667	18.666667	4.000000	9.333333	amivantamab

It is important to agree on a unique feed composition, in order to then create synthetic datasets using the same conditions. Therefore the points can be plotted in order to analyze the dispersion of the values, with their mean and standard deviation. The ratio *A_N,* Alanine concentration relative to Asparagine, for all mAbs from the example of this thesis was plotted below, where the 72 mAbs are represented in the *Index* axis.

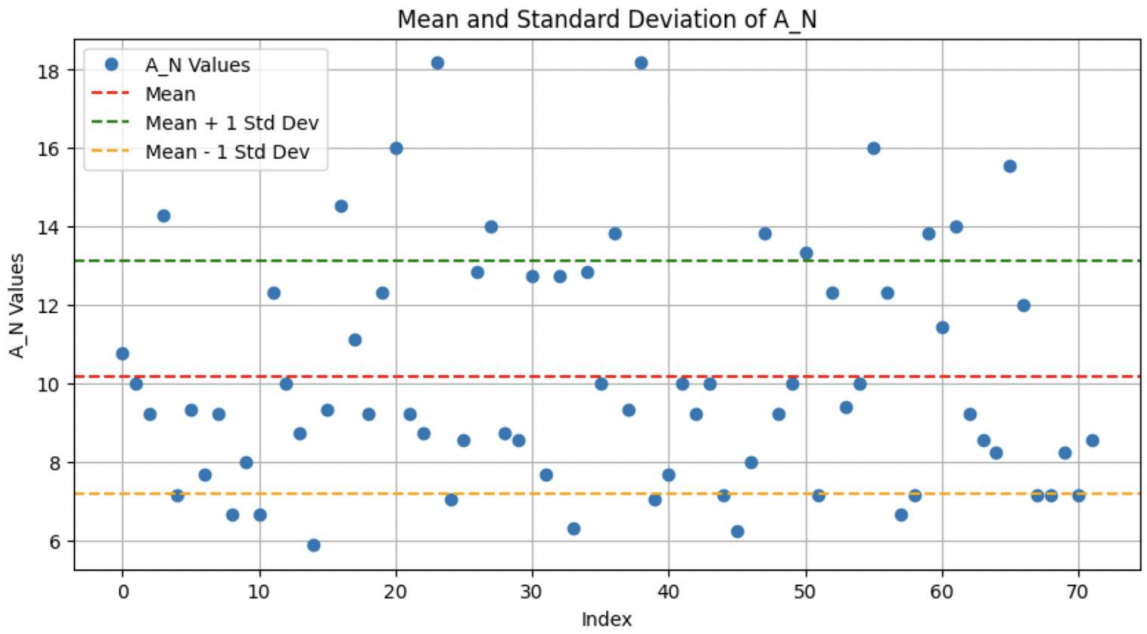

Figure 3.3: Plot of Asparagine concentrations relative to Alanine. The central red dashed line represents the mean value of all the concentrations across the 72 mAbs, which are represented in the Index axis. The upper green dashed line and the lower yellow dashed line represent the mean plus and minus 1 standard deviation respectively.

Then, an implementation to remove outliers that fall outside one standard deviation from the mean was developed for the *new_ratio_df* DataFrame. Each column in the DataFrame is processed through a series of operations. First, a loop iterates over each column, ensuring that the 'name' column is skipped since it does not contain numerical data relevant for statistical analysis. For each applicable column, the mean and standard deviation are calculated using the *mean()* and *std()* methods, respectively, while ignoring any *NaN* values.

The column is then updated using the *where()* method, which retains values that meet the specified condition and replaces those that do not with *None* (or *NaN*). After executing this code, *new_ratio_df* will contain None for any values that fall outside the range defined by one standard deviation from the mean for each respective column, effectively filtering the data to focus on values that are more representative of the central tendency of the dataset.

The new mean values for each column in the *new_ratio_df* DataFrame are calculated using the *mean()* method, while excluding the 'name' column with *drop(columns=['name'])*. This results in a Series object containing the mean values for the relevant columns. A new DataFrame, *new_mean_df*, is then created by passing this Series to *pd.DataFrame()* and transposing it with the *.T* method, resulting in a single row format. Consequently, *new_mean_df* will contain one row with the mean values for each amino acid ratio. The complete Python code is displayed in Annex 2.4.

The original and the modified mean values for the example of this thesis are plotted below for every amino acid relative to Alanine.

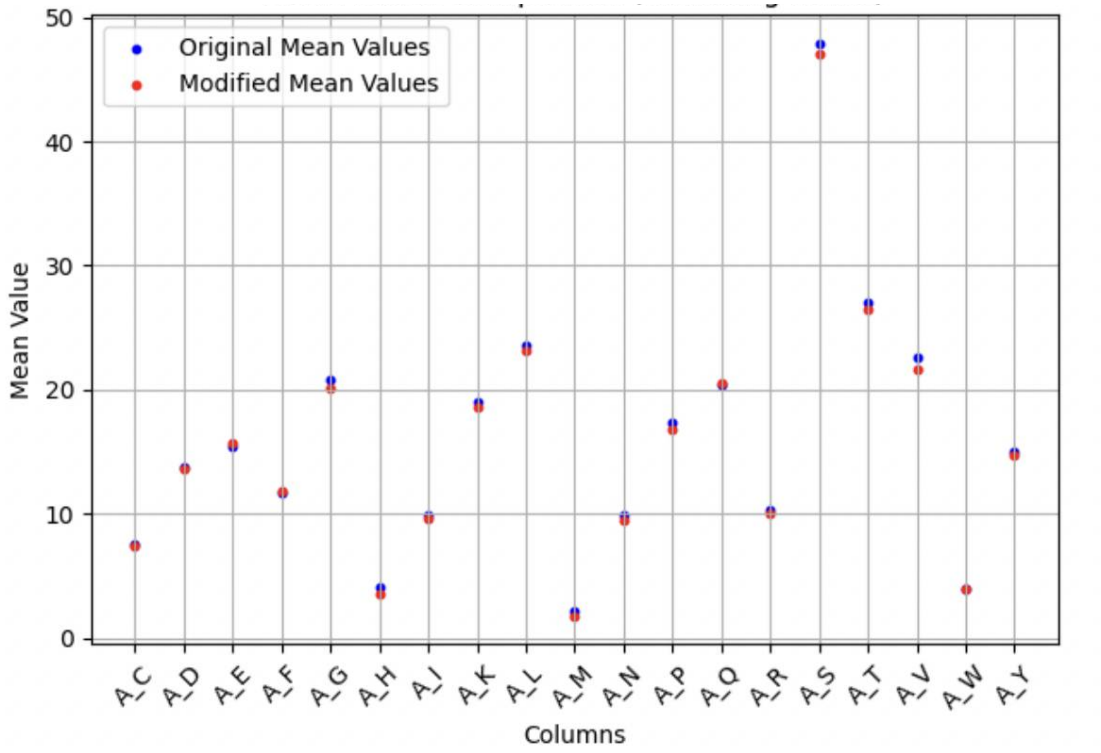

Figure 3.4: Mean values for preferential amino acid feed compositions across all mAbs

In the specific example of this thesis we can see that there was no significant change to the mean values. The feed composition was therefore chosen as in the following table. For the other species the pre-default in the guideline Jupyter notebook was assumed.

Table 3.5: Feed concentrations (mM) used for the in silico simulations of a bioreactor

ALA	ARG	ASN	ASP	CYS	EGLC	EGLN	EGLU	EPYR	GLY	HIS	ILE
20	14	14	18	10	1892	0	0	39	32	6	12
LAC	LEU	LYS	MET	NH4	PHE	PRO	SER	THR	TYR	VAL	
0	34	26	2	0	16	26	64	32	4	36	

3.5.2 Feed rate tests

The acceptable feed rate range for the process can be determined through a systematic trial-and-error approach. This method involves conducting multiple tests with varying feed rates to observe if the effects on the time course simulations of mAb yield and biomass growth resemble the ones presented in literature. In what concerns the example of this thesis, this was achieved and a design of experiments within these values were selected (more information on the next sub-chapter).

3.6 Synthetic experimental data

From the *4comp.csv* file, the feed concentration values are loaded and transformed into a list using *tolist()*, resulting in *values_list*. A NumPy array is then created from values_list using *np.array()*.

In order to not execute the Jupyter notebook one by one for each mAb and for each design of experiment, a strategy was developed. First, the list of monoclonal antibody names is also retrieved from a csv file. Then it is possible to operate the simulator by range index of mAb through created variables *a* and *b*. So, in the example of this thesis, the widest range would be *a=0* and *b=71*. These variables define the range of rows to be processed from the loaded *df_cleaned* DataFrame

The *mabnames_list* is created by selecting the 'Name' column from *df_cleaned* and extracting the values from rows indexed from *a* to *b*. The *tolist()* method converts this selection into a list of monoclonal antibody names.

The DataFrame *dfmb* is created by reading the previously created csv file with the stoichiometric coefficients. The code retrieves the values for each amino acid and ATP based on the current mAb name. A loop iterates through each name in *mabnames_list.* For each antibody name, these values for *ATP, A, R, N, D, C, G, H, I, L, K, M, F, P, S, T, W, V, Q,* and *E* are extracted using the *set_value* function. Each value is obtained by filtering the *dfmb* DataFrame for the current antibody name and accessing the corresponding column. The full code is displayed in Annex 2.5.

3.6.1 New Input for design of experiments

The *doedf* DataFrame is now modifiable with information about the feed rates and induction time. A loop iterates through each row of the *doedf* DataFrame using the *iterrows()* method. During each iteration, several actions are performed. First, the variable *value1* is assigned the value from the

'Feed_pre_ind' column of the current row, converted to a float. Similarly, *value2* is assigned the value from the 'Feed_post_ind' column, also converted to a float. The variable time of induction, *tind*, is assigned the value from the 'T_ind' column of the current row, converted to an integer.

Next, two count calculations are performed: count1 is calculated as *tind - 1*, representing the number of time points before the time of induction, *tind*, while count2 is calculated as *241 - tind*, representing the number of time points after *tind*, taking into consideration the final time of the experiment is 240 hours.

The *Feed_Rate* array is created using the *np.array()* function from the NumPy library, since this is the way it is loaded into the model. This array is constructed by concatenating two lists: the first list contains *value1* repeated *count1* times, which represents the feed rate each hour before the specified time index, while the second list contains *value2* repeated *count2* times, representing the feed rate each hour after the specified time index. This approach effectively generates an array that reflects the feed rates over every hour of the relevant time periods. The full code is displayed in Annex 2.5.

For the example in this thesis the design of experiments was done as follows.

Table 3.6: Design of experiments for this thesis

Time of induction (hours)	Pre-Induction Feed Rate (L/h)	Post-Induction Feed Rate (L/h)
72	0.055	1.65
84	0.082	2.452
84	0.028	0.847
84	0.028	2.452
84	0.082	0.847
96	0.1	1.65
96	0.01	1.65
96	0.055	0.3
96	0.055	1.65
96	0.055	3.00
108	0.028	0.847
108	0.028	2.452
108	0.082	0.847
108	0.082	2.452
120	0.055	1.65

The datasets are then automatically saved accordingly within each mAb name directory, which were created before using the shell scripts mentioned in the previous sections.

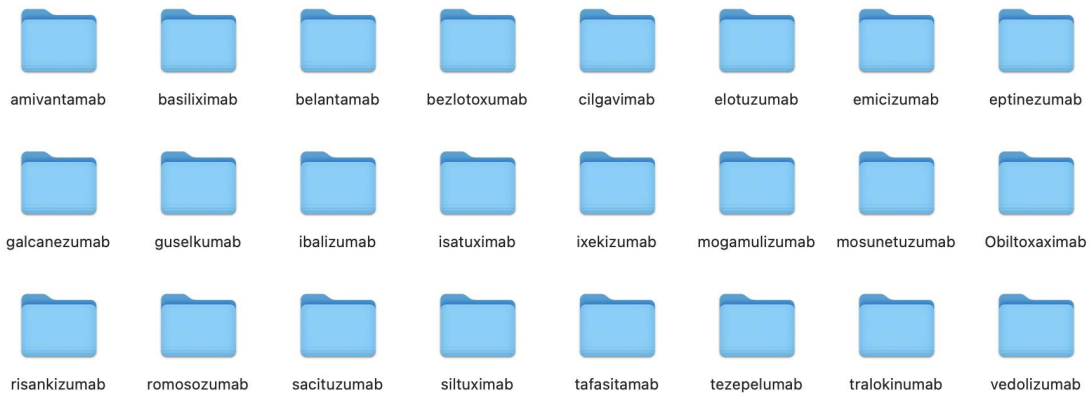

Figure 3.5: Screenshot of the organisation of directories for mAb simulation data

Each folder contains a csv with the experiment simulation, with the file name containing the design of experiment information as well as the date and time stamp of the simulation.

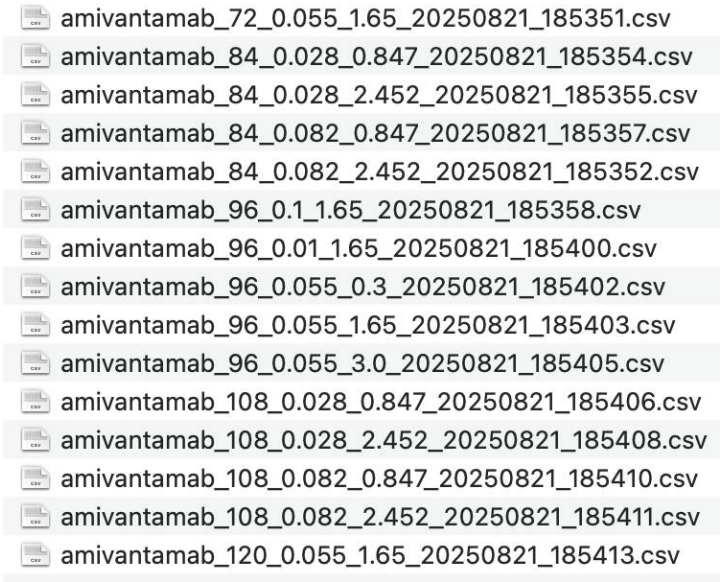

Figure 3.6: Screenshot of the examples of contents of one mAb directory

For the example in this thesis, with the first conditions on the design of experiments, the results for intracellular metabolites in low titer mAbs were as follows:

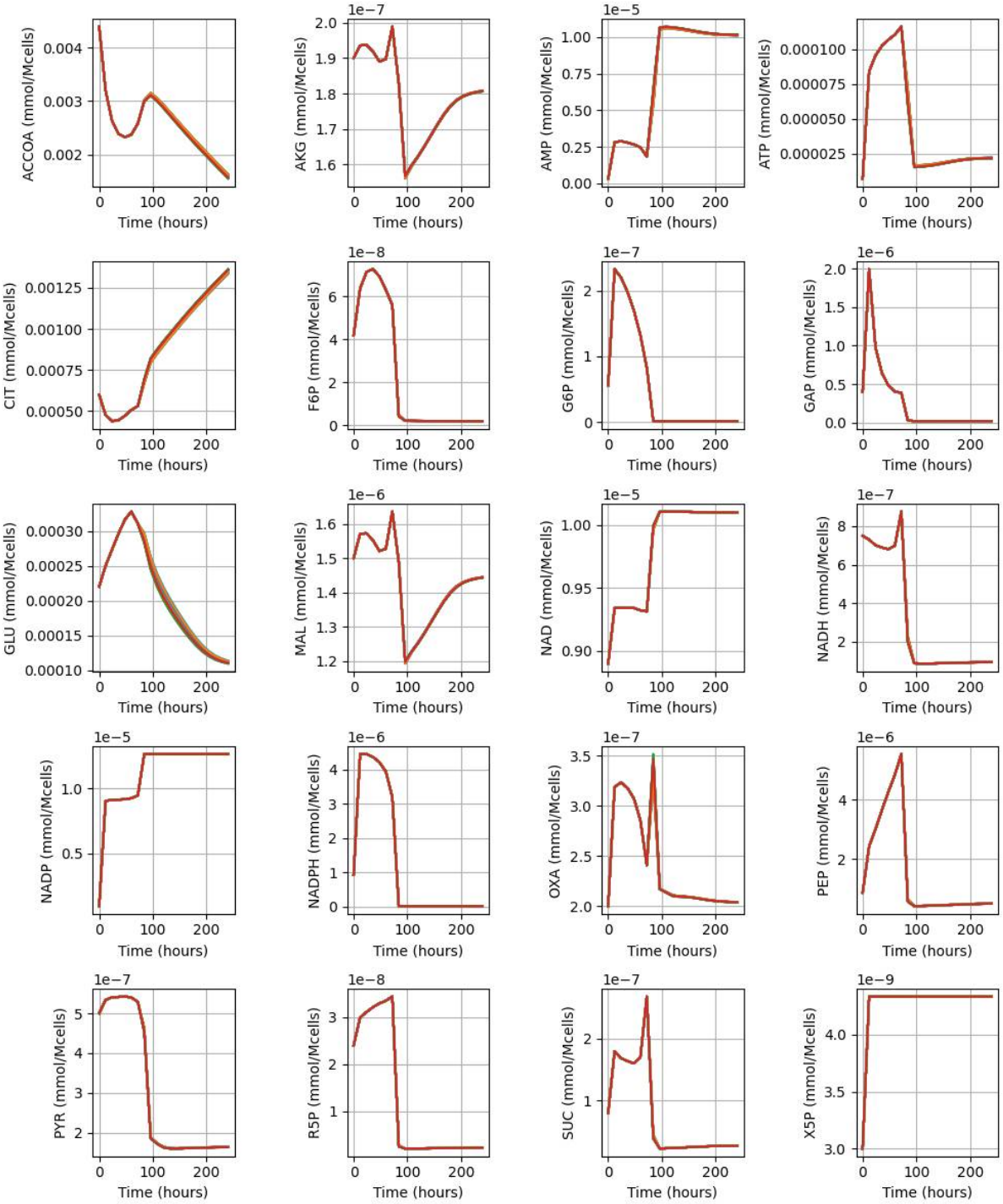

Figure 3.7: Intracellular species simulation results with time of induction of 72 hours, 0.055 L/h of pre-induction feed rate and 1.65 L/h post-induction feed rate in low titer mAbs.

Some results resemble the ones obtained in [100] such as with PYR where the value remains roughly constant after induction, except with MAL increasing after induction whereas in [100] there is a decrease, and with AKG there is an increase while in [100] shows a constant slightly decreasing trend.

The extracellular metabolites (except amino acids) results are presented below:

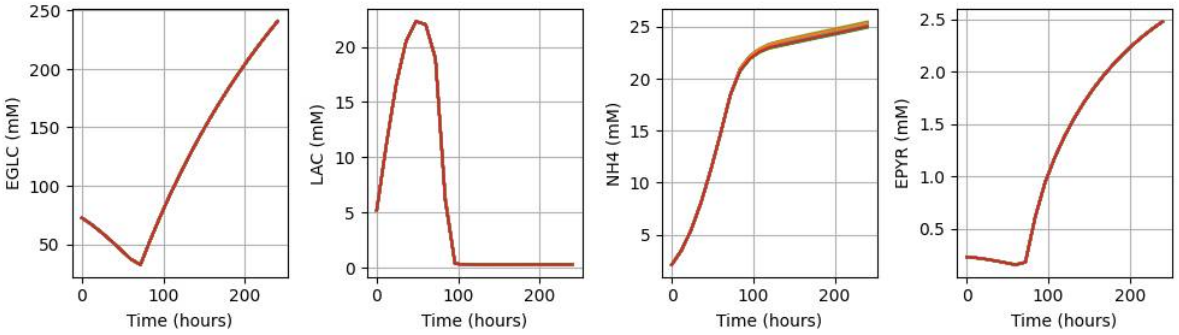

Figure 3.8: Extracellular metabolites (except amino acids) results with time of induction of 72 hours, 0.055 L/h of pre-induction feed rate and 1.65 L/h post-induction feed rate in low titer mAbs.

The observed slight increase in NH_4 levels is concerning, since elevated NH_4 concentrations can be detrimental to the CHO cells health and overall bioreactor performance. However these results of increasing NH_4 concentration are on par with the ones reported by [100].

3.6.2 High vs Low Titer datasets

A high titer and low titer classification helps in assessing the potential yield in mAbs production, with high titer at around 10 g/L and low titer possibly at levels of 0.5 g/L [87] [88]. As explained in the previous sections, for each mAb, a different value of V_{maxmab} was retrieved from the file *vmaxmab.csv.* This resulted in a considerable difference for the yields of the two groups of mAbs, which are plotted below as an example for this thesis:

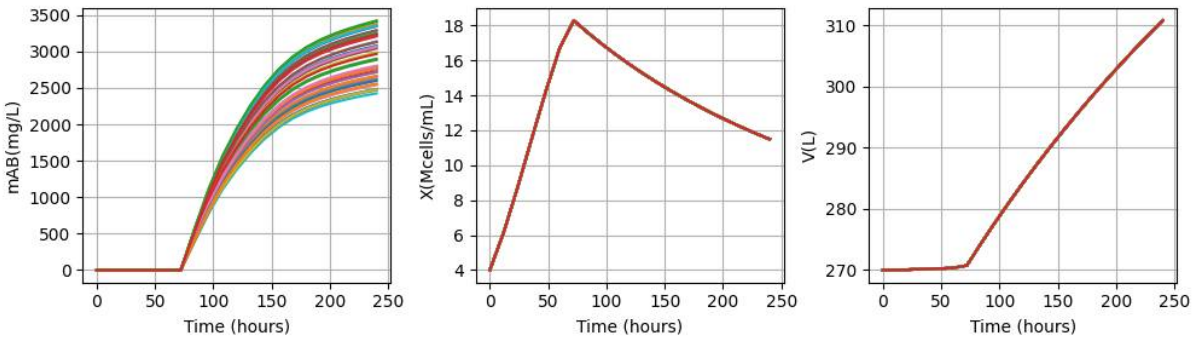

Figure 3.9: Simulation results for low titer mAbs with time of induction of 72 hours, 0.055 L/h of pre-induction feed rate and 1.65 L/h post-induction feed rate

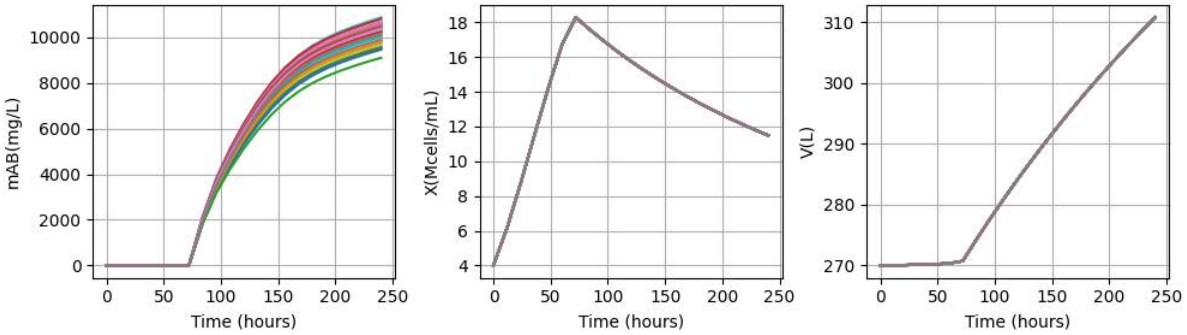

Figure 3.10: Simulation results for high titer mAbs with time of induction of 72 hours, 0.055 L/h of pre-induction feed rate and 1.65 L/h post-induction feed rate

As originally intended, it can be observed that there is a sharp difference between the yields of the two groups of mAbs. The biomass increases to a peak before decreasing with time, as expected, since in mAb production it is paramount to optimize for the yield of mAb instead of biomass growth.

3.7 Reacted Amounts

In the loop, each file path in the *file_list* is processed sequentially. The first step involves extracting key parameters from the filename using the *os.path.basename* function, which retrieves the base name of the file (i.e., the last segment of the path) indicating the mAb name, but also *t_ind* represents the time of induction, *feed_preind* corresponds to the pre-induction feed rate, and *feed_postind* indicates the post-induction feed rate.

Following the extraction of these parameters, the data from the csv file is loaded into a DataFrame. the code identifies all column names in the DataFrame that end with '(mM)', which correspond to the concentrations of the extracellular species. This is accomplished using a list comprehension.

Next, a dictionary named *new_dfs* is created to hold new DataFrames for each of the identified (mM) columns. This structure will facilitate the organization of data. The code then iterates through each column. For each (mM) column, the corresponding feed concentration is extracted from another DataFrame, *df_comp*.

3.7.1 Function Implementation for calculations

A function named *calculate_row* was designed to perform the reacted amounts calculations for each row of the DataFrame, focusing on the biochemical species concentrations and their associated volumes. Within the function, the following calculations are performed:

1. Extracting Values: The function begins by extracting the current species concentration *spe_t* from the specified (mM) column and the corresponding volume *v_t* from the 'V(L)' column of the current row.
2. Determining Initial Volume: The initial volume at the specified time index *t_ind* is retrieved from the DataFrame. If no matching time index is found, it defaults to zero.
3. Calculating Quantities:

- Quantity Not Reacted, *a_nr*: This represents the amount of the species currently in the system that has not yet reacted, calculated by multiplying the current concentration by the current volume.
- Initial Quantity Introduced, *a_b*: This is the amount of the species introduced at the beginning of the experiment, calculated by multiplying the initial concentration *spe_i* by the initial volume *v_i.*

- Quantity Introduced in the Feed, *a_feed*: This quantity accounts for the amount introduced during the feeding process, calculated as the difference between the current volume and the initial volume, multiplied by the feed concentration.
- Calculating the Reaction Amount, *ra*: Finally, the function calculates the net amount of the species that has reacted, defined as the quantity not reacted minus the sum of the initial quantity and the quantity introduced in the feed.

The function returns a Pandas Series containing the calculated values: *a_nr, a_feed*, and *ra*.

The next steps involve creating a new DataFrame to store the calculated values for each biochemical species represented by the (mM) columns. This process is crucial for organizing the results of the calculations performed by the *calculate_row* function.

1. Creating a New DataFrame: For each (mM) column, a new DataFrame named *new_df* is initialized. This DataFrame includes the time column 't(h)', the current species concentration *mM_col*, and the volume column 'V(L)'.
2. For the case of the mAb product and biomass specifically, a new column, 'mAB', is calculated by multiplying the concentration of mAb (column 'mAB(mg/L)') by the volume (column 'V(L)'), and the 'X' column is computed by taking the product of the biomass (column 'X(Mcells/mL)') and the volume, then subtracting the initial concentration multiplied by the initial volume.
3. Creating an Empty Dictionary: An empty dictionary named data is initialized to hold the calculated values for the reaction amounts, *ra* from each of the (mM) DataFrames stored in *new_dfs*.
4. Looping Through Each Column: A loop iterates through the indices of the *mM_columns* list. For each index, the corresponding column name is used to extract the *ra* values from the respective DataFrame in *new_dfs* and store them in the data dictionary.
5. Creating a New DataFrame: A new DataFrame, *new_dataframe*, is created from the data dictionary, which now contains the reaction amounts for each biochemical species.
6. Adding the Time Column: The 't(h)' column from the original DataFrame (df) is added as the first column in *new_dataframe.*

Finally, the new DataFrame is saved to a csv file at the specified path. The original file path from the *in silico* simulation is used for the new csv file name. The full code is displayed in Annex 2.6.

3.7.2 Example visualisations

The obtained results for the amino acids can be visualised in the following figure:

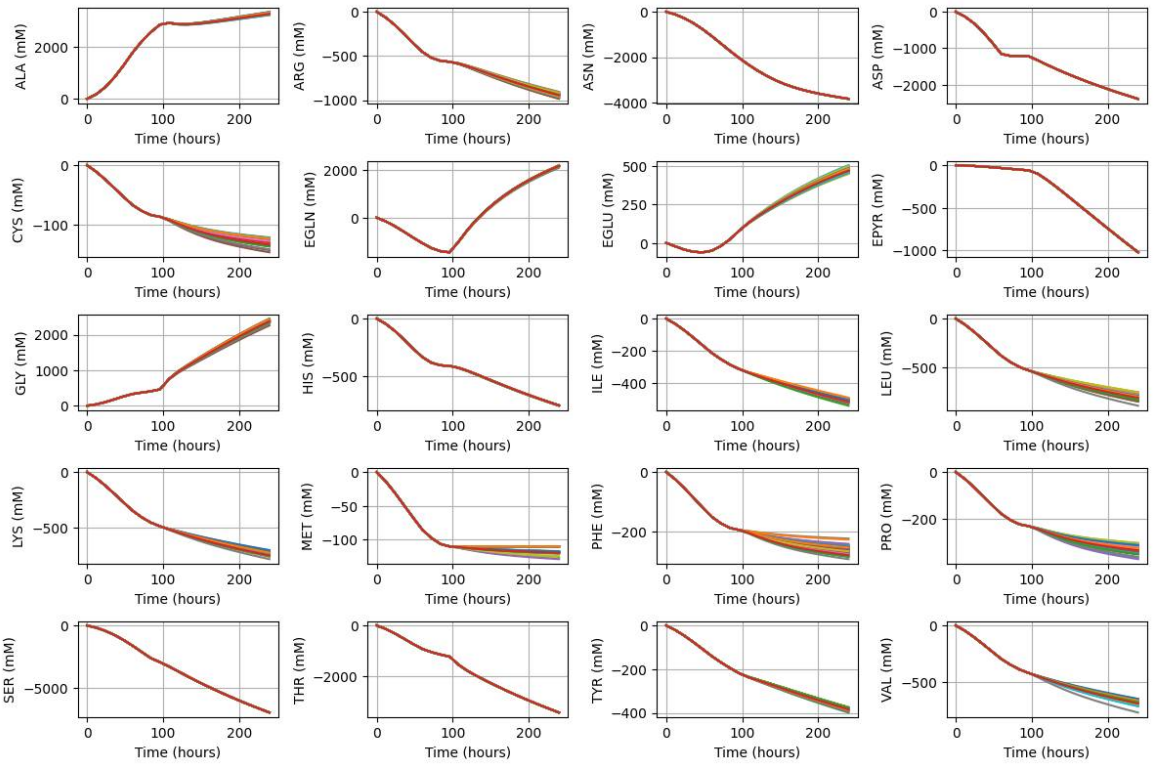

Figure 3.11: Amino Acids reacted amounts for low titer mAbs

For the high titer mAbs, the visualizations obtained are presented below:

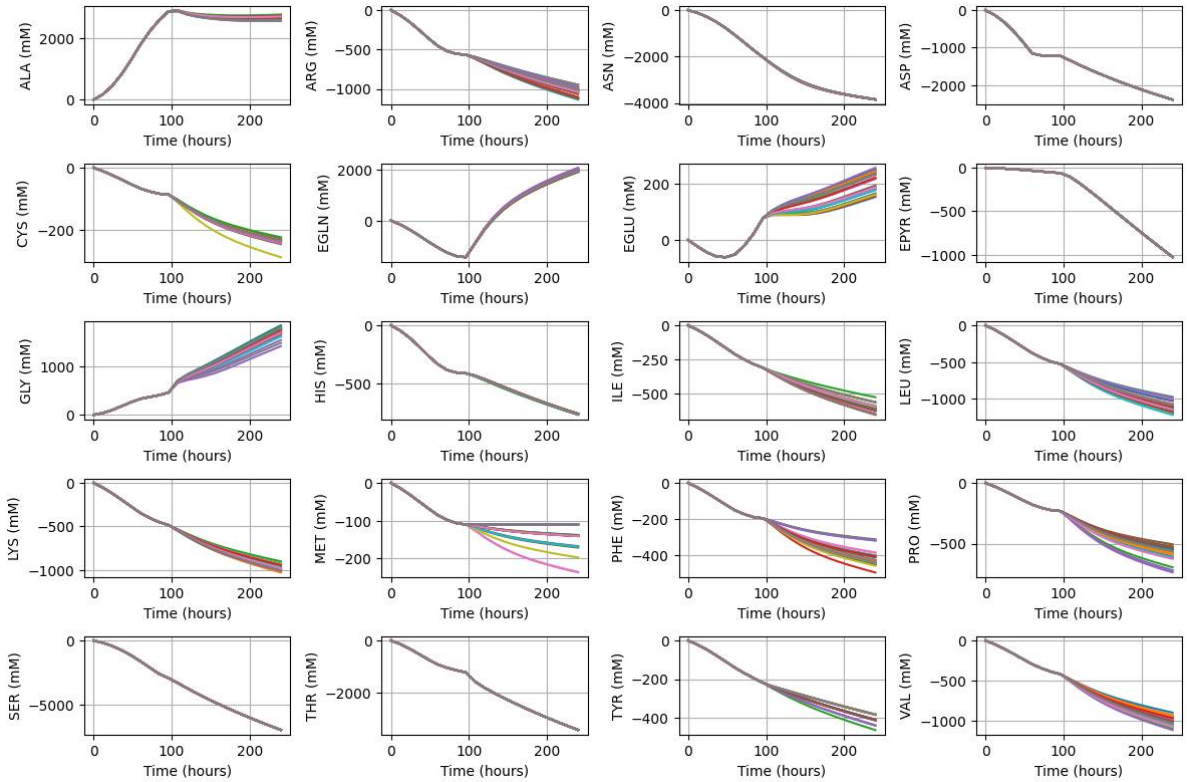

Figure 3.12: Amino Acids reacted amounts for high titer mAbs

When comparing both figures, it can be observed that there is a generalised higher uptake of aminoacids for the high titer mAbs, more prominent for CYS, ILE, VAL, PRO and LYS. This is expected, in order to produce a higher yield.

For the extracellular species which are not amino acids, the results were as follows:

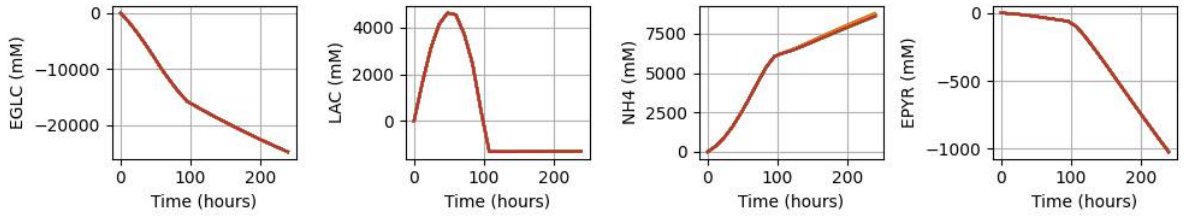

Figure 3.13: extracellular species reacted amounts (except amino acids) - low titer mAbs

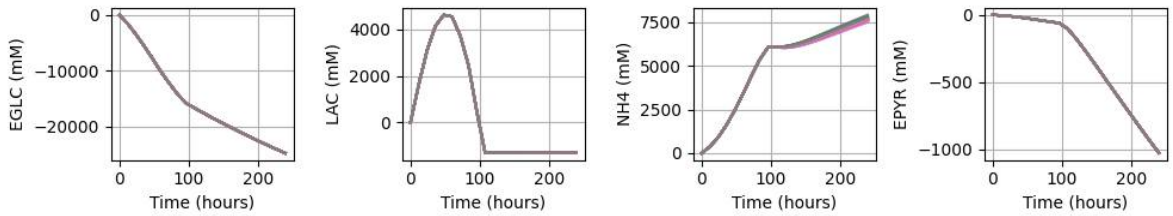

Figure 3.14: extracellular species reacted amounts (except amino acids) - high titer mAbs

Therefore when not taking into consideration the amino acids there was no noticeable difference in the reacted amounts of extracellular species in mAbs of different yields.

3.8 Data Normalisation using *PowerTransformer*

The csv dataset with the reacted amounts was loaded into a DataFrame, and a function collected all the file paths to csv datasets in the selected folders, denoted as *file_list*, in order for a loop to be iterated over the list of file paths.

As expected at least the data in the column ALA (mM), representing alanine concentrations, appears to be positively skewed for most mAbs simulations.

The data transformation process was conducted using the *PowerTransformer* from the *Scikit-learn* library, specifically employing the Yeo-Johnson method.

An empty DataFrame, *transformed_df*, was then established to store the transformed data. To maintain the temporal context of the dataset, the 't(h)' column from the original DataFrame, *df*, was added back to *transformed_df*.

Subsequently, the code iterated over each column in the original DataFrame. The 't(h)' column was excluded from transformation to preserve its original values. For each remaining column, the data was reshaped into a 2D array, which is a requirement for the transformation process. The transformation was applied using the *fit_transform* method of the *PowerTransformer*.

The transformed data was then added to the new DataFrame, ensuring it was flattened back to a 1D array to fit the DataFrame structure. The full code is displayed in Annex 2.7.

3.9 Principal Component Analysis

Principal Component Analysis (PCA) code was developed to reduce the dimensionality of the reacted amounts dataset, facilitating the visualization and interpretation of the underlying patterns in the biochemical species data for future input in hybrid models.

3.9.1 Function Implementation

The first step involved loading the data including only the extracellular species. The relevant features were extracted from the *transformed_df* DataFrame. Next, PCA was applied to the standardized data. The number of principal components was specified as two, allowing for a two-dimensional representation of the data, using *sklearn.decomposition import PCA* and *pca.fit_transform(x).* Following the PCA transformation, a new DataFrame, *principal_df*, was created to store the principal components, labeled as PC1 and PC2:

To facilitate further analysis and interpretation, the scores scores columns were added do the original *transformed_df,* being which was populated with the original time values and the corresponding principal component scores. The explained variance ratio of the first principal component was inserted in the *PC1* row, column *Scores1*, and for the second principal component the same line of thought was done. This was achieved to make it possible to have a glimpse of the PCA implementation within only one csv file per experiment.

Table 3.7: Format of the reacted amounts datasets with PCA information now included. The columns Scores1 and Scores2 indicate the respective principal component scores from t(h)=0 up until t(h)=240. The Principal Components themselves are included on the last rows as PC1 and PC2 from columns ALA(mM) up until VAL(mM). The explained variance ratio of each principal component x is at the intersection of the corresponding Scores and PC cell.

t(h)	ALA(mM)	ARG(mM)	ASN(mM)	ASP(mM)	...	THR(mM)	TYR(mM)	VAL(mM)	Scores1	Scores2
0.0	-2.249253	2.192152	2.077094	2.243849	...	2.156620	2.060194	2.103110	9.653686	-2.508759
12.0	-2.099656	1.963839	1.792081	2.014835	...	1.825552	1.808255	1.811271	8.673034	-1.802713
24.0	-1.823212	1.647034	1.544847	1.689422	...	1.517845	1.535448	1.522731	7.5718	-0.862782
36.0	-1.415652	1.265673	1.284444	1.269223	...	1.211964	1.245259	1.230716	6.296664	0.187925
48.0	-0.894108	0.857744	1.011252	0.767123	...	0.929062	0.950044	0.946145	4.897833	1.238743
...	...	...	...	...	...	...	...	...	...	...
216.0	0.594648	-0.987783	-0.975267	-0.989897	...	-1.036284	-1.096840	-1.048647	-4.828946	-0.944471
228.0	0.604736	-1.053478	-0.998504	-1.064392	...	-1.109833	-1.187094	-1.110795	-5.101996	-1.05782
240.0	0.615887	-1.117366	-1.019765	-1.136678	...	-1.180959	-1.274766	-1.170513	-5.364611	-1.164569
PC1	-0.193917	0.207929	0.208828	0.206135	...	0.208835	0.208432	0.208812	0.914431	
PC2	0.259892	-0.070653	0.001104	-0.101331	...	-0.006689	0.009415	0.005910		0.067472

The final step involved saving the computed scores from the Principal Component Analysis to a csv file. The *transformed_df* DataFrame, which contained the time values and the principal component scores, was then exported to a csv file using the *to_csv* method. The full code is displayed in Annex 2.7

3.9.2 Principal Components and Scores

The subsequent phase of the analysis involved aggregating the PCA scores from all the experiments with the same conditions, with the multiple mAb csv files stored in a list called *file_list2*. This process aimed to compile the data into a single DataFrame, referred to as *doe1*, which would facilitate comparative analysis across different datasets. Initially, an empty DataFrame, *doe1*, was created to hold the aggregated results, and a loop was then initiated to iterate over each filename in *file_list2*.

The scores at hourly observations close to the time of induction were set to be extracted to the new *doe1,* where each row contained the PC1 score and PC2 score of each mAb at a specific time.

Table 3.8: Principal Component scores at selected observations (8th and 9th)

	odesivimab	romosozumab	tezepelumab	amivantamab	elotuzumab	isatuximab
8	[0.7714293996320931, 2.396893464539581]	[0.8137954342826618, 2.3998696540453244]	[0.6367353509076812, -2.543988022497122]	[0.8201851633089989, 2.428646073606352]	[0.8042446421417614, 2.4133109430912825]	[0.8306868842659758, 2.3850724493055995]
9	[-0.6607633397749787, 1.113515657090332]	[-0.6230846226768076, 1.11599430244606]	[-0.7637873136045568, -1.2460565521635345]	[-0.6162880972743127, 1.1546206055353472]	[-0.6275034699235859, 1.1347902427750762]	[-0.5986788437695628, 1.1094063709857536]

3.9.3 Clustering Visualisations

The analysis progressed to the application of K-means clustering on the aggregated PCA scores stored in the *doe1* DataFrame. This step aimed to identify distinct clusters within the data, providing insights into the underlying patterns and relationships among the biochemical species.

To prepare the data for clustering, the coordinates of the PCA scores were extracted and organized into a NumPy array. Each column in *doe1* represented a different mAb. K-means clustering was then applied, specifying the number of clusters as two. Following the clustering, a scatter plot was created to visualize the PCA scores and their corresponding cluster labels.

Each point in the scatter plot was colored based on its cluster label of titer, with points assigned to the first cluster displayed in blue and those in the second cluster in red. As an example this

was done below for the experiment with time of induction 96, pre-induction feed rate of 0.055 L/h and post induction feed rate of 3.0 L/h, with the observation just before the induction.

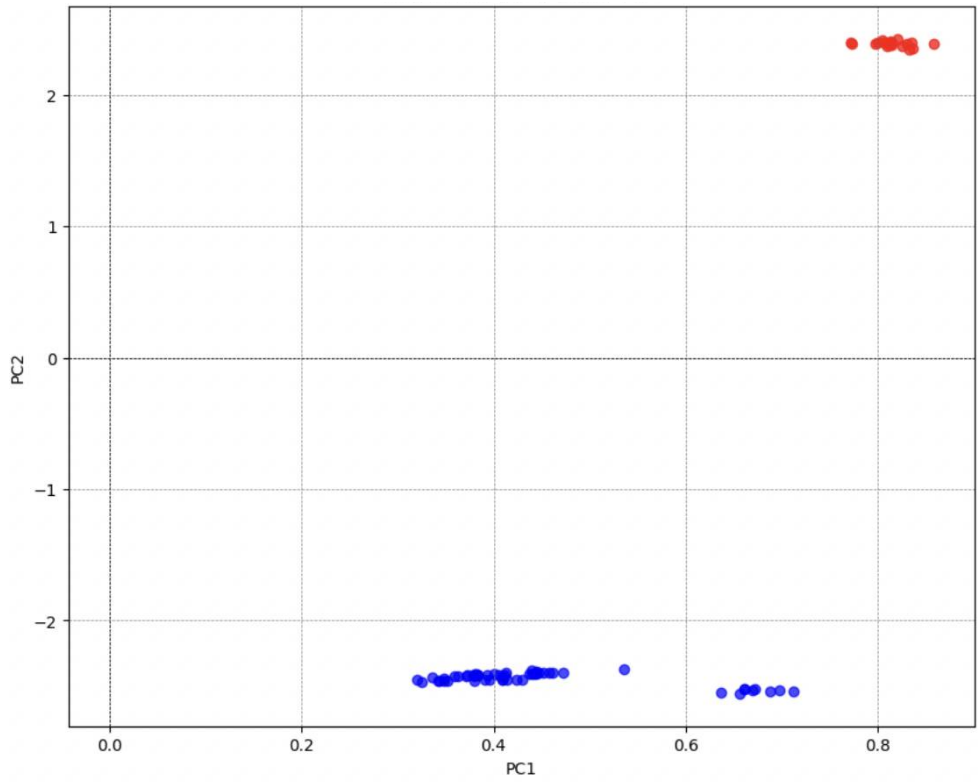

Figure 3.15: Plot of PCA Scores for a selected experiment at the time of 96 hours.

In this example it is possible to observe that from the 72 mAbs, 62 were clustered in the correct group. The detailed results with the mAbs names are presented below.

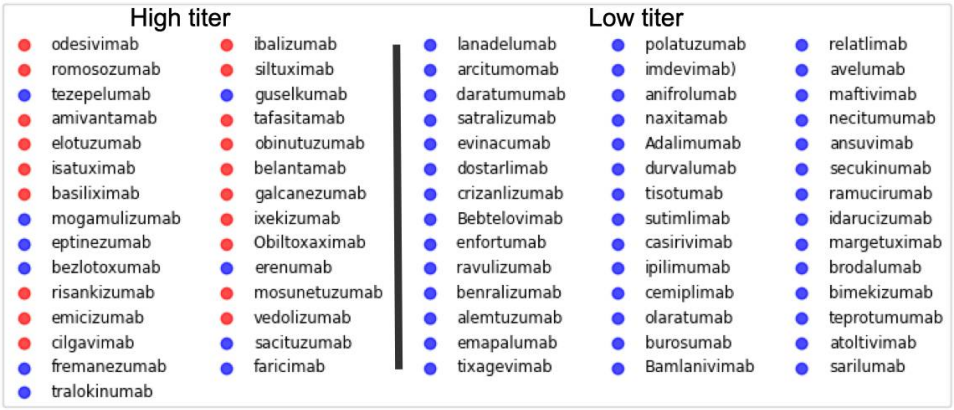

Figure 3.16: Names of mAbs matched within their cluster

The vertical line separates the two obtained groups. Red dots were mAbs previously defined as high titer and blue were low titer.

For the following observation at 108 hours, the same number of correct mAbs was achieved, with a different scatter plot:

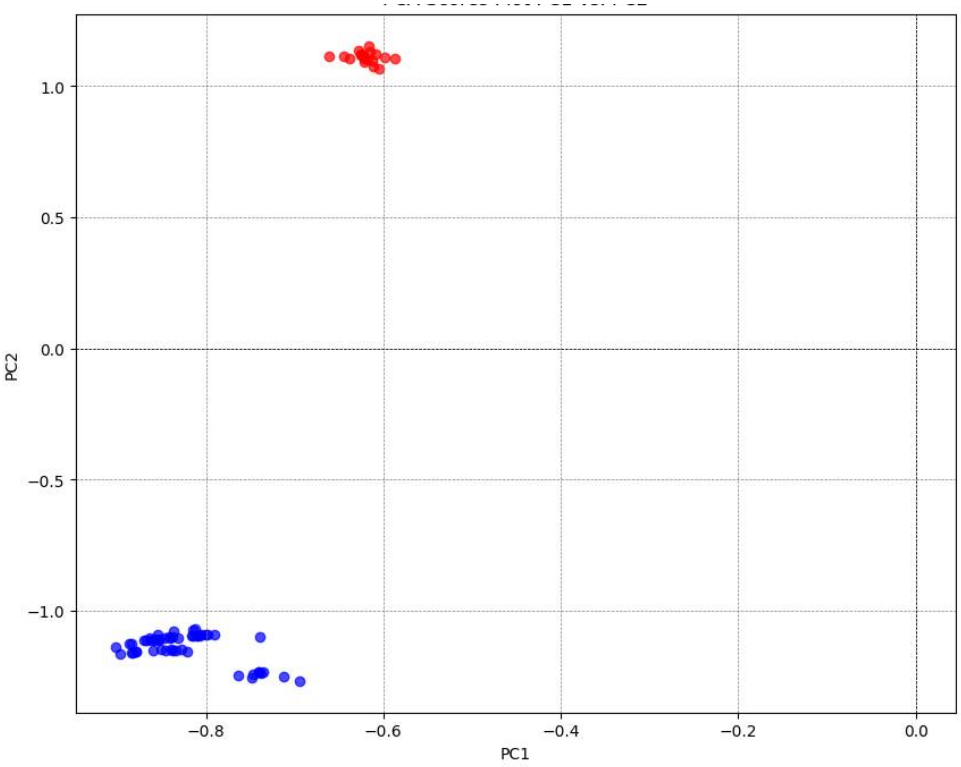

Figure 3.17: Plot of PCA Scores for a selected experiment at the time of 108 hours.

This comprehensive approach to clustering and PCA allowed for the identification of distinct groupings within the data, achieving a low titer and high titer clusters, with only a few mAbs being clustered in the wrong group.

4. CONCLUSIONS

The objectives of this thesis were achieved by developing the data pipeline and exemplifying its execution, by successfully establishing a comprehensive organisation into seven distinct sections, facilitating efficient management and navigation.

4.1 Concluding remarks

The thesis included the automated creation of directories based on the names of monoclonal antibodies (mAbs) and their classification into low or high titer groups, achieved through the use of shell scripts.

Additionally, a specialized tool was developed to extract information on monoclonal antibodies, beginning with the identification of International Nonproprietary Names (INNs) from scientific literature. This process successfully retrieved 155 INNs, which were subsequently linked to 129 ChEMBL IDs. From these, 72 amino acid sequences were obtained via the ChEMBL API, with a clear distinction made between light and heavy chain sequences.

Furthermore, a novel method was introduced to assign a value for Vmaxmab based on the amino acid sequence. This method utilized a sigmoid function to categorize the mAbs into low or high titer groups. This approach effectively addressed a significant challenge in data processing, as such information is often not publicly accessible from biopharmaceutical companies.

Code was developed to calculate the stoichiometric coefficients and integrate them into the ordinary differential equations of the dynamic model. This foundational step facilitated the accurate representation of the system dynamics.

The preferred feed conditions were derived from the amino acid sequences, ensuring that the simulations were grounded in biologically relevant parameters. Additionally, a code was created to enable the design of experiments simulations, which included saving the results in synthetic datasets organized by the corresponding mAb names in csv files. This process exemplified the generation of 1,080 datasets, paving the path for analysis.

The reacted amounts were calculated based on the synthetic datasets and the specified feeds, allowing for a detailed examination of the system's behavior. To further analyze the data, normalization was performed, taking into account the inherent characteristics of the datasets. Principal Component Analysis (PCA) was conducted, revealing two distinct clusters of mAbs.

4.2 Future work

The thesis could be further enhanced with greater availability of information from biopharma processes, which would facilitate a more comprehensive analysis and improved the overall outcomes of the research, such as information about mAb titers, even if it was not for all molecules, or more recent dynamic models.

Each time there is a process development campaign, the real laboratory measured data from a successful one with similar conditions could be input in this data pipeline combined with the synthetic data generated by the dynamic model. Afterwards deep learning models can be created for each observation, thus doing knowledge transfer to more accurately predict the process development campaigns that follow on.

Furthermore, in what concerns the computation architecture, the data pipeline could be containerized using orchestrations tools such as Docker and Kubernetes in order for new features and new data structures to be readily continuously integrated and continuously developed for each section without compromising the others, in line with the software development cycle of today in biopharma.

BIBLIOGRAPHY

[1] Waller-Pulido, A., Jiménez-Pérez, M. I., Gonzalez-Sanchez, F. A., Rojo-Gutierrez, R. P., Torres-Anguiano, E., Aleman-Aguilar, J. P., & Garcia-Varela, R. (2023). Production of monoclonal antibodies for therapeutic purposes: A review. International Immunopharmacology, 120, 110376. https://doi.org/10.1016/j.intimp.2023.110376

[2] Chrone, V. G., Jespersen, J. C., Asani, D. C., Trier, N. H., Ray, S., Berthias, F., Willemoës, M., Nordvang, A. H., Frederiksen, J. L., Houen, G., & Højrup, P. (2025). Native structure of the monoclonal therapeutic CD20 antibody ocrelizumab. Biochimica et Biophysica Acta – Proteins and Proteomics, 1873, 141084. https://doi.org/10.1016/j.bbapap.2025.141084

[3] Tokunaga, Y.; Takeuchi, K. Role of NMR in High Ordered Structure Characterization of Monoclonal Antibodies. Int. J. Mol. Sci. 2021, 22, 46. http://dx.doi.org/10.3390/ijms22010046

[4] Song, X., Tian, H., Jing, R., Liu, K., Xu, K., Guo, L.* & Zhang, G. (2025). Navigating the frontier: Advances in monoclonal antibody purity control. Protein Expression and Purification, 232, 106725. https://doi.org/10.1016/j.trac.2025.118385

[5] Waller-Pulido, A., Jiménez-Pérez, M. I., Gonzalez-Sanchez, F. A., Rojo-Gutierrez, R. P., Torres-Anguiano, E., Aleman-Aguilar, J. P., & Garcia-Varela, R. (2023). Production of monoclonal antibodies for therapeutic purposes: A review. International Immunopharmacology, 120, 110376. https://doi.org/10.1016/j.intimp.2023.110376

[6] V. Bayer, An overview of monoclonal antibodies, Semin. Oncol. Nurs. 35 (5) (2019), 150927, https://doi.org/10.1016/J.SONCN.2019.08.006.

[7] Alez-Martin, L., Hirschler, E., Houzé, P., Potier, N., Mignet, N., Leize-Wagner, E., François, Y-N., & Gahoual, R. (2025). Characterization and quantification of therapeutic monoclonal antibodies, anti-drug antibodies and their interactions for clinical applications. Trends in Analytical Chemistry, 192, 118385. https://doi.org/10.1016/j.trac.2025.118385

[8] Tokunaga, Y., & Takeuchi, K. (2021). Role of NMR in High Ordered Structure Characterization of Monoclonal Antibodies. International Journal of Molecular Sciences, 22, 46. DOI:10.3390/ijms22010046.

[9] Alez-Martin, L., Hirschler, E., Houzé, P., Potier, N., Mignet, N., Leize-Wagner, E., François, Y.–N., & Gahoual, R. (2025). Characterization and quantification of therapeutic monoclonal antibodies, anti-drug antibodies and their interactions for clinical applications. Trends in Analytical Chemistry, 192, 118385. DOI:10.1016/j.trac.2025.118385.

[10] M. L. Chiu, D. R. Goulet, A. Teplyakov, G. L. Gilliland, Antibody structure and function: the basis for engineering therapeutics, Antibodies 8 (2019) 55, https://doi.org/10.3390/antib8040055.

[11] B.S. Moorthy, B. Xie, E.M. Moussa, L.K. Iyer, S. Chandrasekhar, J.P. Panchal, E.M. Topp, Structure of monoclonal antibodies, in: AAPS Advances in the Pharmaceutical Sciences Series, Vol. 19, SpringerVerlag, 2015, pp. 81–89, doi:10.1007/978-1-4939-2543-8_6.

[12] F. Breedveld, Therapeutic monoclonal antibodies, Lancet 355 (2000) 735–740, https://doi.org/10.1016/S0140-6736(00)01034-5.

[13] J. T. Ryman, B. Meibohm, Pharmacokinetics of monoclonal antibodies, CPT Pharmacometrics Syst. Pharmacol. 6 (2017) 576–588, https://doi.org/10.1002/psp4.12224.

[14] Crescioli, S., Kaplon, H., Chenoweth, A., Wang, L., Visweswaraiah, J., & Reichert, J. M. (2024). Antibodies to watch in 2024. mAbs, 16, 2297450. DOI:10.1080/19420862.2023.2297450.

[15] Chiu, M. L., Goulet, D. R., Teplyakov, A., & Gilliland, G. L. (2019). Antibody structure and function: the basis for engineering therapeutics. Antibodies, 8, 55. DOI:10.3390/antib8040055.

[16] Husain, B., & Ellerman, D. (2018). Expanding the Boundaries of Biotherapeutics with Bispecific Antibodies. BioDrugs, 32, 441–464. DOI:10.1007/s40259-018-0299-9.

[17] Breedveld, F. (2000). Therapeutic monoclonal antibodies. Lancet, 355, 735–740. DOI:10.1016/S0140-6736(00)01034-5.

[18] Ryman, J. T., & Meibohm, B. (2017). Pharmacokinetics of monoclonal antibodies. CPT: Pharmacometrics & Systems Pharmacology, 6, 576–588. DOI:10.1002/psp4.12224.

[19] Chrone, V. G., Jespersen, J. C., Asani, D. C., Trier, N. H., Ray, S., Berthias, F., Willemoës, M., Nordvang, A. H., Frederiksen, J. L., Houen, G., & Højrup, P. (2025). Native structure of the monoclonal therapeutic CD20 antibody ocrelizumab. Biochimica et Biophysica Acta – Proteins and Proteomics, 1873, 141084. DOI:10.1016/j.bbapap.2025.141084.

[20] Braun, J., & Sieper, J. (2003). Overview of the use of the anti-TNF agent infliximab in chronic inflammatory diseases. Expert Opinion on Biological Therapy, 3, 141–168. DOI:10.1517/14712598.3.1.141.

[21] Lamb, Y. N. (2022). Ocrelizumab: A review in multiple sclerosis. Drugs, 82, 323–334. DOI:10.1007/s40265-022-01672-9.

[22] Nixon, J., Newbold, P., Mustelin, T., Anderson, G. P., & Kolbeck, R. (2017). Monoclonal antibody therapy for the treatment of asthma and chronic obstructive pulmonary disease with eosinophilic inflammation. Pharmacology & Therapeutics, 169, 57–77. DOI:10.1016/j.pharmthera.2016.10.016.

[23] LiverTox: Clinical and Research Information on Drug-Induced Liver Injury. (2012). National Institute of Diabetes and Digestive and Kidney Diseases.

[24] Song, X., Tian, H., Jing, R., Liu, K., Xu, K., Guo, L.*, & Zhang, G. (2025). Navigating the frontier: Advances in monoclonal antibody purity control. Protein Expression and Purification, 232, 106725.DOI:10.1016/j.trac.2025.118385.

[25] Hanack, K., Messerschmidt, K., & Listek, M. (2016). Antibodies and selection of monoclonal antibodies. In Advances in Experimental Medicine and Biology (Vol. 917, pp. 11–22). DOI:10.1007/978-3-319-32805-8_2.

[26] Si, Y., Melkonian, A. L., Curry, K. C., Xu, Y., Tidwell, M., Liu, M., Zaky, A. F., & Liu, X. (2020). Monoclonal antibody-based cancer therapies. Chemical Engineering Journal, 404, 126835. https://doi.org/10.1016/j.cjche.2020.11.009

[27] Köhler, G., & Milstein, C. (1975). Continuous cultures of fused cells secreting antibody of predefined specificity. Nature, 256, 495–497. DOI:10.1038/256495a0.

[28] Murphy, K. M., & Weaver, C. (2022). Janeway's Immunobiology (9th ed.). Garland Science/W. W. Norton.

[29] Hafeez, U., Gan, H. K., & Scott, A. M. (2018). Monoclonal antibodies as immunomodulatory therapy against cancer and autoimmune diseases. Current Opinion in Pharmacology, 41, 114–121. DOI:10.1016/j.coph.2018.05.010.

[30] Houen, G. (2022). Methods in Molecular Biology (Vol. 2313, pp. 1–25). DOI:10.1007/978-1-0716-1450-1_1.

[31] Kaplon, H., Muralidharan, M., Schneider, Z., & Reichert, J. M. (2020). Antibodies to watch in 2020. mAbs, 12, 1703531. DOI:10.1080/19420862.2019.1703531.

[32] Amen, A., Thielens, N., Dumestre-Pérard, C., & Clavarino, G. (2025). Eliciting or silencing complement activation: Strategies for optimizing monoclonal antibodies functions. Current Opinion in Immunology, 96, 102638. https://doi.org/10.1016/j.coi.2025.102638

[33] Kapare, H., Bhosale, M., & Bhole, R. (2025). Navigating the future: Advancements in monoclonal antibody nanoparticle therapy for cancer. Journal of Drug Delivery Science and Technology, 104, 106495. https://doi.org/10.1016/j.jddst.2024.106495

[34] S. Ponziani, G. Di Vittorio, G. Pitari, A.M. Cimini, M. Ardini,
Antibody-drug conjugates: the new frontier of chemotherapy, Int. J. Mol. Sci. 21 (15) (2020) 55 10, https://doi.org/10.3390/ijms21155510.

[35] J.J. Gemmete, S.K. Mukherji, Trastuzumab (Herceptin), AJNR.
American J. Neuroradiol. 32 (8) (2011) 1373–1374, https://doi.org/10.3174/ajnr.A2619.

[36] J. Tabernero, P.M. Hoff, L. Shen, A. Ohtsu, M.A. Shah, A. Siddiqui,
Pertuzumab, trastuzumab, and chemotherapy in HER2-positive gastric/
gastroesophageal junction cancer,

International Gastric Cancer Association and the Japanese Gastric Cancer Associ. 26 (1) (2023) 123–131, https://doi.org/10.1007/s10120-022-01335-4.

[37] S. Siena, A. Sartore-Bianchi, F. Di Nicolantonio, J. Balfour, A. Bardelli,
Biomarkers predicting clinical outcome of epidermal growth factor receptor-targeted therapy in metastatic colorectal cancer, J. Natl. Cancer Inst. 101 (19) (2009) 1308–1324, https://doi.org/10.1093/jnci/djp280.

[38] Gerriets V, Kasi A. Bevacizumab. [Updated 2023 Aug 28]. In: StatPearls [Internet]. Treasure Island (FL): StatPearls Publishing; 2025 Jan-. Available from: https://www.ncbi.nlm.nih.gov/books/NBK482126/

[39] E. Dotan, C. Aggarwal, M.R. Smith,
Impact of Rituximab (Rituxan) on the treatment of B-cell non-Hodgkin's lymphoma. P & T: a P eer-Reviewed, J. Formulary Manag. 35 (3) (2010) 148–157.

[40] R. Burt, D. Warcel, A.K. Fielding,
Blinatumomab, a bispecific B-cell and T-cell engaging antibody, Hum. Vaccines Immunother. 15 (3) (2019) 594–602, https://doi.org/10.1080/21645515.2018.1540828.

[41] Y. Shiravand, F. Khodadadi, S.M. Amin Kashani, S.R. ,
Immune checkpoint inhibitors in cancer therapy, Curr. Oncol. 29 (5) (2022) 3044–3060, https://doi.org/10.3390/curroncol29050247.

[42] C. Vaklavas, A. Forero-Torres,
Safety and efficacy of brentuximab vedotin in patients with Hodgkin lymphoma Therapeutic A dv. Hematology. 3 (4) (2012) 209–225, https://doi.org/10.1177/2040620712443076.

[43] L.A. Raedler,
Darzalex (Daratumumab): first anti-CD38 monoclonal antibody approved for patients with rela psed multiple myeloma, American Health. Drug Benefits. 9 (Spec Feature) (2016) 70–73.

[44] D.C. Julien, S. Behnke, G. Wang, G.K. Murdoch, R.A. Hill,
Utilization of monoclonal antibody-targeted nanomaterials in the treatment of cancer, mAbs 3 (5) (2011) 467–478, https://doi.org/10.4161/mabs.3.5.16089.

[45] T.C. Ezike, U.S. Okpala, U.L. Onoja, C.P. Nwike,
Advances in drug delivery systems, challenges and future directions, Heliyon 9 (6) (2023) e174 88, https://doi.org/10.1016/j.heliyon.2023.e17488.

[46] Saylor C, Dadachova E, Casadevall A. Monoclonal antibody-based therapies for microbial diseases. Vaccine 2009;27S. G38e46.

[47] Wang-Lin SX, Balthasar JP. Pharmacokinetic and pharmacodynamic considerations for the use of monoclonal antibodies in the treatment of bacterial infections. Antibodies 2018;7:5.

[48] Salazar G, Zhang N, Fu TM, An Z. Antibody therapies for the prevention and treatment of viral infections. npj Vaccines 2017;2:19.

[49] European Medicines Agency; Committee for Medicinal Products for Human Use. Zinplavad Available at: http://www.ema.europa.eu/docs/en_GB/document_library/EPAR-*Summary_for_the_public/human/004136/WC500222645.pdf.

[50] US Food and Drug Administration. Drugs@FDA. FDA approved drug products: Raxibacumab. Available at: https://www.accessdata.fda.gov/drugsatfda_docs/label/2012/125349s000lbl.pdf.

[51] US Food and Drug Administration. Drugs@FDA. FDA approved drug products: Anthim. Available at: https://www.accessdata.fda.gov/drugsatfda_docs/label/2016/125509lbl.pdf.

[52] François B, Barraud O, Jafri HS. Antibody-based therapy to combat Staphylococcus aureus infections. Clin Microbiol Infect 2017;23:219e21.

[53] Lynch SV, Pedersen O. The human intestinal microbiome in health and disease. N Engl J Med 2016;375:2369e79.

[54] Caliendo AM, Gilbert DN, Ginocchio CC, Hanson KE, May L, Quinn TC, et al. Better tests, better care: improved diagnostics for infectious diseases. Clin Infect Dis 2013;57:S139e70.

[55] Graham BS, Ambrosino DM. History of passive antibody administration for prevention and treatment of infectious diseases. Curr Opin HIV AIDS 2015;10: 129e34.

[56] Soares MM, King SW, Thorpe PE. Targeting inside-out phosphatidylserine as a therapeutic strategy for viral diseases. Nat Med 2008;14:1357e62.

[57] US Food and Drug Administration. Drugs@FDA. FDA approved drug products: Trogarzo. Available at: https://www.accessdata.fda.gov/drugsatfda_docs/label/2018/761065lbl.pdf.

[58] European Medicines Agency; Committee for Medicinal Products for Human Use. Synagisd Available at: http://www.ema.europa.eu/docs/en_GB/document_library/EPAR*-_Summary_for_the_public/human/000257/WC500056736.pdf.

[59] Henrich TJ, Kuritzkes DR. HIV-1 entry inhibitors: recent development and clinical use. Curr Opin Virol 2013;3:51e7.

[60] Jacobson JM, Lalezari JP, Thompson MA, et al. Phase 2a study of the CCR5 monoclonal antibody Antimicrob Agents Chemother 2010;54:4137e42.

[61] Klein F, Mouquet H, Dosenovic P, Scheid JF, Scharf L, Nussenzweig MC. Antibodies in HIV-1 vaccine development and therapy. Science 2013;341:1199e204.

[62] Marston HD, Paules CI, Fauci AS. Monoclonal antibodies for emerging infectious diseases borrowing from history. N Engl J Med 2018;378:1469e72.

[63] Friesen RHE, Koudstaal W, Koldijk MH, Weverling GJ, Brakenhoff JPJ, Lenting PJ, et al. New class of monoclonal antibodies against severe influenza: prophylactic and therapeutic efficacy in ferrets. PLoS One 2010;5, e9106.

[64] Paules CI, Lakdawala S, McAuliffe JM, Paskel M, Vogel L, Kallewaard NL, et al. The hemagglutinin system antibody MEDI8852 prevents and controls disease and limits transmission of pandemic influenza viruses. J Infect Dis 2017;216:356e65.

[65] Sparrow E, Friede M, Sheikh M, Torvaldsen S, Newall AT. Passive immunisation for influenza through antibody therapies,. Vaccine 2016;34:5442e8.

[66] Norman DJ, Shield III CF, Barry JM, Henell K, Funnell MB, Lemon J. Therapeutic use of OKT3 monoclonal antibody for acute renal allograft rejection. Nephron 1987;46:41e7.

[67] Buss NA, Henderson SJ, McFarlane M, de Haan L. Monoclonal antibody therapeutics: history and future. Curr Opin Pharmacol 2012;12:615e22.

[68] Pelfrene, E., Mura, M., Cavaleiro Sanches, A., & Cavaleri, M. (2019). Monoclonal antibodies as anti-infective products: A promising future? Clinical Microbiology and Infection, 25(1), 60-64. https://doi.org/10.1016/j.cmi.2018.04.024

[69] Graham BS, Ambrosino DM. History of passive antibody administration for prevention and treatment of infectious diseases. Curr Opin HIV AIDS 2015;10: 129e34.

[70] Saylor C, Dadachova E, Casadevall A. Monoclonal antibody-based therapies for microbial diseases. Vaccine 2009;27S. G38e46.

[71] Waller-Pulido, A., Jiménez-Pérez, M. I., Gonzalez-Sanchez, F. A., Rojo-Gutierrez, (2023). Production of monoclonal antibodies for therapeutic purposes: A review. International Immunopharmacology, 120, 110376. https://doi.org/10.1016/j.intimp.2023.110376

[72] Tiller T, Meffre E, Yurasov S, Tsuiji M, Nussenzweig MC, Wardemann H. Efficient generation of monoclonal antibodies from single human B cells by single cell RT-PCR and expression vector cloning. J Immunol Methods 2008;329: 112e24.

[73] Wang Q, Yang H, Liu X, Dai L, Ma T, Qi J, et al. Molecular determinants of human neutralising antibodies isolated from a patient infected with Zika virus. Sci Transl Med 2016;8. 369ra179.

[74] Wrammert J, Koutsonanos D, Li G-M, Edupganti S, Sui J, Morrissey M, et al. Broadly cross-reactive antibodies dominate the human B cell response against 2009 pandemic H1N1 influenza virus infection. JExp Med 2011;208:181e93.

[75] Hey A. History and practice: antibodies in infectious diseases. Microbiol Spectr 2015;3. AID-0026-2014.

[76] B. Tihanyi, L. Nyitray, Recent advances in CHO cell line development for recombinant protein production, Drug Discov. Today: Technol. 38 (2020) 25–34, doi:10.1016/J.DDTEC.2021.02.003.

[77] V. G. Dhara, H. M. Naik, N. I. Majewska, M. J. Betenbaugh, Recombinant antibody production in CHO and NS0 cells: differences and similarities, BioDrugs 32 (6) (2018) 571–584, https://doi.org/10.1007/s40259-018-0319-9.

[78] S. M. Noh, S. Shin, G. M. Lee, 2019. Cell line development for therapeutic protein production, in: G. M. Lee, H. Faustrup Kildegaard, S. Y. Lee, J. Nielsen, G. Stephanopolous (Eds.), Cell Culture Engineering. doi:10.1002/9783527811410.ch2.

[79] J. Dumont, D. Euwart, B. Mei, S. Estes, R. Kshirsagar, Human cell lines for biopharmaceutical manufacturing: history, status, and future perspectives, Crit. Rev. Biotechnol. 36 (6) (2016) 1110–1122, https://doi.org/10.3109/07388551.2015.1084266.

[80] H. Dahodwala, K. H. Lee, The fickle CHO: a review of the causes, implications, and potential alleviation of the CHO cell line instability problem, Curr. Opin. Biotechnol. 60 (2019) 128–137, https://doi.org/10.1016/J.COPBIO.2019.01.011.

[81] N. Romanova, T. Noll, Engineered and natural promoters and chromatin-modifying elements for recombinant protein expression in CHO cells, Biotechnol. J. 13 (3) (2018), https://doi.org/10.1002/biot.201700232.

[82] J. H. Gao, T. Y. Wang, M. Y. Zhang, F. Shi, S. Z. Gu, Identification of consensus sequence from matrix attachment regions and functional analysis of its activity in stably transfected Chinese hamsterovary cells, J. Cell. Biochem. 120 (8) (2019) 13985–13993, https://doi.org/10.1002/jcb.28673.

[83] O. Yang, S. Prabhu, M. Ierapetritou, Comparison between batch and continuous monoclonal antibody production and economic analysis, Ind. Eng. Chem. Res. 58 (15) (2019) 5851–5863, https://doi.org/10.1021/acs.iecr.8b04717.

[84] B. Kelley, Downstream processing of monoclonal antibodies: current practices and future opportunities, in: Process Scale Purification of Antibodies, John Wiley & Sons, Ltd., 2017, pp. 1–21, https://doi.org/10.1002/9781119126942.ch1.

[85] X. Wang, Z. An, W. Luo, N. Xia, Q. Zhao, Molecular and functional analysis of monoclonal antibodies in support of biologics development, Protein Cell 9 (1) (2018) 74–85, https://doi.org/10.1007/s13238-017-0447-x.

[86] C. Y. Huang, M. C. Hsieh, Q. Zhou, Application of tryptophan fluorescence bandwidth-maximum plot in analysis of monoclonal antibody structure, AAPS PharmSciTech 18 (3) (2017) 838–845, https://doi.org/10.1208/s12249-016-0568-1.

[87] Legmann, R., Schreyer, H. B., Combs, R. G., McCormick, E. L., Russo, A. P., & Rodgers, S. T. (2009). A predictive high-throughput scale-down model of monoclonal antibody production in CHO cells. Biotechnology and Bioengineering, 104(5), 1020-1030. https://doi.org/10.1002/bit.22474

[88] Vishwanathan, N., Le, H., Jacob, N. M., Tsao, Y.-S., Ng, S.-W., Loo, B., Liu, Z., Kantardjieff, A., & Hu, W.-S. (2014). Transcriptome dynamics of transgene amplification in Chinese hamster ovary cells. Biotechnology and Bioengineering, 111(3), 518–528. https://doi.org/10.1002/bit.25112

[89] McCloskey, D., Palsson, B. Ø., and Feist, A. M. (2013). Basic and applied uses of genome-scale metabolic network reconstructions of Escherichia coli. Mol. Syst. Biol. 9:661. doi: 10.1038/msb.2013.18

[90] Kim OD, Rocha M and Maia P (2018) A Review of Dynamic Modeling Approaches and Their Application in Computational Strain Optimization for Metabolic Engineering. Front. Microbiol. 9:1690. doi: 10.3389/fmicb.2018.01690

[91] Rizzi, M., Baltes, M., Theobald, U., and Reuss, M. (1997). In vivo analysis of metabolic dynamics in saccharomyces cerevisiae: II. Mathematical Model. Biotechnol. Bioeng. 55, 592–608.

[92] Chassagnole, C., Noisommit-Rizzi, N., Schmid, J. W., Mauch, K., and Reuss, M. (2002). Dynamic modeling of the central carbon metabolism of Escherichia coli. Biotechnol. Bioeng. 79, 53–73. doi: 10.1002/bit.10288

[93] Smallbone, K., Simeonidis, E., Swainston, N., and Mendes, P. (2010). Towards a genome-scale kinetic model of cellular metabolism. BMC Syst. Biol. 4:6. doi: 10.1186/1752-0509-4-6

[94] Almquist, J., Cvijovic, M., Hatzimanikatis, V., Nielsen, J., and Jirstrand, M. (2014). Kinetic models in industrial biotechnology improving cell factory performance. Metab. Eng. 24, 38–60. doi: 10.1016/j.ymben.2014.03.007

[95] Bruggeman, F. J., and Westerhoff,. H. V. (2007). The nature of systems biology. Trends Microbiol. 15, 45–50. doi: 10.1016/j.tim.2006.11.003

[96] Lewis, N. E., Nagarajan, H., and Palsson, B. O. (2012). Constraining the metabolic genotype-phenotype relationship using a phylogeny of in silico methods. Nature Rev. Microbiol. 10, 291–305. doi: 10.1038/nrmicro2737

[97] Wagner, C., and Urbanczik, R. (2005). The geometry of the flux cone of a metabolic network. Biophys. J. 89, 3837–3845. doi: 10.1529/biophysj.104.055129

[98] Orth, J. D., Thiele, I., and Palsson, B. Ø. (2010). What is flux balance analysis? Nat. Biotechnol. 28, 245–248. doi: 10.1038/nbt.1614

[99] Machado, D., Costa, R. S., Ferreira, E. C., Rocha, I., and Tidor, B. (2012). Exploring the gap between dynamic and constraint-based models of metabolism. Metab. Eng. 14, 112–119. doi: 10.1016/j.ymben.2012.01.003

[100] Robitaille J, Chen J, Jolicoeur M (2015) A Single Dynamic Metabolic Model Can Describe mAb Producing CHO Cell Batch and Fed-Batch Cultures on Different Culture Media. PLoS ONE 10(9): e0136815. doi:10.1371/journal.pone.0136815

[101] Walsh, G., Walsh, E. Biopharmaceutical benchmarks 2022. Nat Biotechnol 40, 1722–1760 (2022). https://doi.org/10.1038/s41587-022-01582-x

[102] World Health Organization. "International Nonproprietary Names (INNs)." Accessed January 10, 2025. https://www.who.int/teams/health-product-and-policy-standards/inn/.

[103] Pinto J, Ramos JRC, Costa RS, Rossell S, Dumas P and Oliveira R (2023) Hybrid deep modeling of a CHO-K1 fed-batch process: combining first-principles with deep neural networks. Front. Bioeng. Biotechnol. 11:1237963. doi: 10.3389/fbioe.2023.1237963

[104] Hairer, E., & Wanner, G. (1996). Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems.

[105] SciPy developers. SciPy v1.15.3 Reference Guide — scipy.integrate.solve_ivp. (https://docs.scipy.org/doc//scipy-1.2.3/reference/generated/scipy.integrate.solve_ivp.html).

[106] Weisberg, Sanford. "Yeo-Johnson power transformations." Department of Applied Statistics, University of Minnesota. Retrieved June 1.2003 (2001)

[107] Howley, T., Madden, M.G., O'Connell, ML., Ryder, A.G. (2006). The Effect of Principal Component Analysis on Machine Learning Accuracy with High Dimensional Spectral Data. In: Macintosh, A., Ellis, R., Allen, T. (eds) Applications and Innovations in Intelligent Systems XIII. SGAI 2005. Springer, London. https://doi.org/10.1007/1-84628-224-1_16

ANNEXES

Annex 1

Annex 1.1

open_jupyter.sh

```
#!/bin/bash

# Set the root path
root_path=/Users/

# Activate the virtual environment
source "$root_path/simuls/mabs/bin/activate"

# Prompt the user for the path to the Jupyter Notebook
echo "Please drag the Jupyter Notebook folder"
read jupyter_path

jupyter_name="$(basename "$jupyter_path").ipynb"
jupyter_new_path="$jupyter_path/$jupyter_name"

# Start Jupyter Notebook and open a specific notebook
jupyter notebook "$jupyter_new_path"
```

Annex 1.2

create_folders.sh

```
#!/bin/bash

# Prompt the user for the path to the Jupyter Notebook
echo "Please drag the low titer folder"
read low_value_dir

# Prompt the user for the path to the Jupyter Notebook
echo "Please drag the high titer folder "
read high_value_dir

# Prompt the user for input
echo "Please enter the list of names and values (format: name,value). Press Ctrl+D when done:"

# Read user input until EOF (Ctrl+D)
while IFS= read -r line; do
    # Split the line into name and value
    IFS=',' read -r name value <<< "$line"

    # Check if the value is below 0.02
    if (( $(echo "$value < 0.02" | bc -l) )); then
        # Create a new folder in the low titer value directory
        mkdir -p "$low_value_dir/$name"
        echo "Created folder in low titer value directory: $low_value_dir/$name"

    else
        # Create a new folder in the high titer value directory
        mkdir -p "$high_value_dir/$name"
        echo "Created folder in high titer value directory: $high_value_dir/$name"
    fi
done
```

Annex 1.3

Common first cell for all jupyter notebooks

```
import os
import pandas as pd
import glob


# Get the user's home directory
user_path = os.path.expanduser("~")

# Construct the absolute path
root_path = os.path.join(user_path, "Desktop", "IDE_Dev_Sections")
```

Annex 2

Annex 2.1

1Seq_Collect.ipynb
Cell #1

```
import PyPDF2
import re

def extract_unique_mab_words(pdf_path):
    unique_words = set()

    # Open the PDF file
    with open(pdf_path, 'rb') as file:
        reader = PyPDF2.PdfReader(file)

        # Iterate through each page
        for page in reader.pages:
            text = page.extract_text()
            if text:
                # Find all words ending with 'mab' possibly followed by specified symbols
                matches = re.findall(r'\b\w*mab[),;\'"\\.:\s]?\b', text)
                unique_words.update(matches)

    return unique_words

# Example usage
unique_mab_words = extract_unique_mab_words(pdf_path)
print(unique_mab_words)
```

1Seq_Collect.ipynb
Cell #2

```
import requests
import xml.etree.ElementTree as ET


mabs_ids = {}
for mab in unique_mab_words:
    url = 'https://www.ebi.ac.uk/chembl/api/data/chembl_id_lookup/search?q='+mab  #
Replace with your API URL
    response = requests.get(url)
    root = ET.fromstring(response.content)
    for item in root.findall('.//chembl_id_lookup'):
        chemid=item.find('chembl_id').text
    mabs_ids[chemid] = mab
    print(len(mabs_ids))
```

1Seq_Collect.ipynb
Cell #3

```
df = pd.DataFrame({
    'Name': mabs_ids.values(),
    'ChemEMBL ID': [None] * len(mabs_ids),
    'Light Chain': [None] * len(mabs_ids),
    'Heavy Chain': [None] * len(mabs_ids),
})

# Display the populated DataFrame
print(df)

# Populate the 'ChemEMBL iD' column based on the names
for index, row in df.iterrows():
    name = row['Name']
    # Check if the name exists in the mabs_ids dictionary
    for chem_id, chem_name in mabs_ids.items():
        if name == chem_name:
            df.at[index, 'ChemEMBL ID'] = chem_id  # Update the ChemEMBL iD column

# Display the updated DataFrame
print(df)
```

1Seq_Collect.ipynb
Cell #4

```
import requests
import xml.etree.ElementTree as ET


for id in mabs_ids:
    url = 'https://www.ebi.ac.uk/chembl/api/data/biotherapeutic/'+id
    response = requests.get(url)


    if response.status_code == 200:
        root = ET.fromstring(response.content)


        # Initialize lists to store light chain and heavy chain sequences
        light_chain_sequences = []
        heavy_chain_sequences = []

        # Iterate through each biocomponent
        for biocomponent in root.findall('.//biocomponent'):
            description = biocomponent.find('description')
            sequence = biocomponent.find('sequence')

            # Check if description is 'Light chain'
            try:
                if description is not None and 'light chain' in description.text.lower():
                    if sequence is not None:
                        light_chain_sequences.append(sequence.text)


            # Check if description is 'Heavy chain'
                elif description is not None and 'heavy chain' in description.text.lower():
                    if sequence is not None:
                        heavy_chain_sequences.append(sequence.text)
```

1Seq_Collect.ipynb
Cell #4

```
        except AttributeError:
            # Handle the error (e.g., log it, print a message, etc.)
            print("An error occurred: description or sequence is None.")

    # Output the results
    if light_chain_sequences:
        for index, row in df.iterrows():
          if row['ChemEMBL ID'] == id:
              df.at[index, 'Light Chain'] = light_chain_sequences[0]
    else:
        print('No Light chain found.')

    if heavy_chain_sequences:
        for index, row in df.iterrows():
          if row['ChemEMBL ID'] == id:
              df.at[index, 'Heavy Chain'] = heavy_chain_sequences[0]
    else:
        print('No Heavy chain found.')
else:
  print(f"Error: Received status code {response.status_code}")
```

1Seq_Collect.ipynb
Cell #5

```
# Filtering the DataFrame
filtered_df = df[df['Light Chain'].notna() & df['Heavy Chain'].notna()]

print(filtered_df)
filtered_df.to_csv('out.csv', index=False)
```

Annex 2.2

2mAb_titer_alloc.ipynb
Cell #1

```
import os
import glob


comp_location = "1Seq_Collect/out.csv"
path = os.path.join(root_path, comp_location)
import pandas as pd

df = pd.read_csv(path)
df['Heavy Chain'] = df['Heavy Chain'].str.replace('1', 'X')
# Combining the columns
df['Combined Chains'] = df['Light Chain'] + df['Heavy Chain']
```

2mAb_titer_alloc.ipynb
Cell #2

```
def amino_acid_to_ieee754(amino_acid_sequence):
    # Define the mapping of amino acids to binary strings
    amino_acid_mapping = {
        'A': '00000',  # Alanine
        'C': '00001',  # Cysteine
        'D': '00010',  # Aspartic acid
        'E': '00011',  # Glutamic acid
        'F': '00100',  # Phenylalanine
        'G': '00101',  # Glycine
        'H': '00110',  # Histidine
        'I': '00111',  # Isoleucine
        'K': '01000',  # Lysine
        'L': '01001',  # Leucine
        'M': '01010',  # Methionine
        'N': '01011',  # Asparagine
        'P': '01100',  # Proline
        'Q': '01101',  # Glutamine
        'R': '01110',  # Arginine
        'S': '01111',  # Serine
        'T': '10000',  # Threonine
        'V': '10001',  # Valine
        'W': '10010',  # Tryptophan
        'Y': '10011',  # Tyrosine
        'X': '10100'   # Unknown or any other amino acid
    }

    # Function to pad the sequence with 'X' until its length is a multiple of 12
    def pad_sequence(sequence, multiple=12):
        while len(sequence) % multiple != 0:
            sequence += 'X'
        return sequence
    # Pad the amino acid sequence
    amino_acid_sequence = pad_sequence(amino_acid_sequence)

    # Function to split the sequence into parts of length 12
    def split_sequence(sequence, part_length=12):
        return [sequence[i:i + part_length] for i in range(0, len(sequence), part_length)]
```

2mAb_titer_alloc.ipynb
Cell #2

```
# Split the amino acid sequence
parts = split_sequence(amino_acid_sequence)

total_value = 0  # Initialize total sum
for part in parts:
    # Check if the part length is 12
    if len(part) != 12:
        raise ValueError("Amino acid sequence must be exactly 12 characters long.")

    # Convert the amino acid sequence to a binary string
    binary_string = '0000' + ''.join(amino_acid_mapping[aa] for aa in part)

    # Ensure the binary string is exactly 64 bits
    if len(binary_string) != 64:
        raise ValueError("The binary string must be exactly 64 bits long.")

    # IEEE 754 components
    sign = int(binary_string[0], 2)  # 1 bit
    exponent_bits = binary_string[1:12]  # 11 bits
    mantissa_bits = binary_string[12:]  # 52 bits

    # Calculate the exponent
    exponent = int(exponent_bits, 2) - 1023  # Subtract the bias (1023)

    # Calculate the mantissa
    mantissa = 1  # Implicit leading 1
    for i, bit in enumerate(mantissa_bits):
        mantissa += int(bit) * (2 ** -(i + 1))

    # Calculate the final value
    value = (-1) ** sign * mantissa * (2 ** exponent)
    total_value += value  # Add to total sum

return total_value
```

2mAb_titer_alloc.ipynb
Cell #3

```
# Apply the function and create a new column "IEEE754"
df['IEEE754'] = df['Combined Chains'].apply(amino_acid_to_ieee754)
df['IEEE754'].is_unique
values= df['IEEE754'].tolist()
#scaling
# Scaling factor
scaling_factor = 10**263

# Scale the values
scaled_values = [value * scaling_factor for value in values]
import statistics

# Standardizing the values
mean = statistics.mean(scaled_values)

stdev = statistics.stdev(scaled_values)

standardized_values = [(x - mean) / stdev for x in scaled_values]

df['standardized_values'] = standardized_values
threshold = 0

# Create a dictionary to assign 0 or 1 based on the threshold
value_dict = {value: (0 if value < threshold else 1) for value in standardized_values}

# Count the number of values below the threshold
count_below_threshold = sum(1 for value in standardized_values if value < threshold)
```

2mAb_titer_alloc.ipynb
Cell #4

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np

# Prepare the feature set (X) and target variable (y)
# Here, we can use the original values as features
X = np.array(list(value_dict.keys())).reshape(-1, 1)  # Reshape for sklearn
y = np.array(list(value_dict.values()))

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create and train the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Get coefficients
coef = model.coef_[0][0]
intercept = model.intercept_[0]
print(f'The intercept is: {intercept}')
print(f'The coefficient for x is: {coef}')
```

2mAb_titer_alloc.ipynb
Cell #5

```
import random
norms_dict = {}
df['vmaxmab'] = 0
Vmin=0.005

# Iterate over the DataFrame
# df['p']
for index, row in df.iterrows():
    start = row['IEEE754']
    mab_feature=start*10**263
    random_float = random.uniform(10, 20)
    standardized_mab_feature = (mab_feature-mean)/stdev
    X = np.array(list([standardized_mab_feature])).reshape(-1, 1)
    vmaxmab=round(Vmin*(1+0.1*random_float)+10*Vmin*model.predict(X)[0],4)
    float(vmaxmab)
    df.at[index, 'vmaxmab'] = float(vmaxmab)
df.to_csv('vmaxmab.csv', index=False)
df1 = df[['Name'] + df.columns[1:4].tolist()]
df2 = df[['Name'] + df.columns[4:].tolist()]
```

Annex 2.3

3Stoic_coeffs.ipynb
Cell #1

```
import os
import glob

comp_location = "1Seq_Collect/out.csv"
path = os.path.join(root_path, comp_location)
import pandas as pd

df = pd.read_csv(path)
tabulated_MW = {
    'A': 89.093,   # Alanine
    'C': 121.158,  # Cysteine
    'D': 133.104,  # Aspartic Acid
    'E': 147.131,  # Glutamic Acid
    'F': 165.192,  # Phenylalanine
    'G': 75.067,   # Glycine
    'H': 155.156,  # Histidine
    'I': 131.174,  # Isoleucine
    'K': 146.189,  # Lysine
    'L': 131.174,  # Leucine
    'M': 149.207,  # Methionine
    'N': 132.119,  # Asparagine
    'P': 115.132,  # Proline
    'Q': 146.146,  # Glutamine
    'R': 174.203,  # Arginine
    'S': 105.093,  # Serine
    'T': 119.119,  # Threonine
    'V': 117.148,  # Valine
    'W': 204.228,  # Tryptophan
    'Y': 181.191,  # Tyrosine
}
```

3Stoic_coeffs.ipynb
Cell #2

```
from collections import Counter

# Function to count amino acids
def count_amino_acids(sequence):
    if pd.isna(sequence):  # Check if the sequence is NaN or None
        return {}  # Return an empty dictionary if it is
    return dict(Counter(sequence))

# Function to calculate weighted counts
def calculate_weighted_counts(counts, mw_dict):
    weighted_counts = {}
    for amino_acid, count in counts.items():
        if amino_acid in mw_dict:
            weighted_counts[amino_acid] = count * mw_dict[amino_acid]
    return weighted_counts
results = []

# Iterate over each row in the DataFrame
for index, row in df.iterrows():
    name = row['Name']

    #count each aminoacid, count = mmol/mmol of IgG
    light_chain_counts = count_amino_acids(row['Light Chain'])
    heavy_chain_counts = count_amino_acids(row['Heavy Chain'])

    total_light_chain_count = sum(light_chain_counts.values())
    total_heavy_chain_count = sum(heavy_chain_counts.values())

    #count the waters released when bonds form (heavy_count_sum-light_count_sum)
    delete_waters = -(total_light_chain_count + total_heavy_chain_count) * 18.01528
```

3Stoic_coeffs.ipynb
Cell #2

```
    # Calculate weighted counts for light and heavy chains, multiply the tabulated MW of each
aminoacid and sum it all
    light_chain_weighted = calculate_weighted_counts(light_chain_counts, tabulated_MW)
    heavy_chain_weighted = calculate_weighted_counts(heavy_chain_counts, tabulated_MW)

    #That's the igg MW g/mol
    igg_mw = (sum(light_chain_weighted.values()) + sum(heavy_chain_weighted.values()) +
delete_waters) * 2

    #Divide each aminoacid count by the igg MW g/mol to get the ODEs coefs
    mmolg = {key: value / igg_mw for key, value in light_chain_counts.items()}

    # Create a new row for the results
    data = {col: mmolg.get(col, None) for col in columns[2:]}  # Get values from mmolg or
None if not present
    data['Name'] = name  # Set the Name

    results.append(data)

    #Deal with the ATPs
    if igg_mw!=0:
      ampcoef=((total_light_chain_count+total_heavy_chain_count)*2)/igg_mw
      gdpcoef=(((total_light_chain_count+total_heavy_chain_count)*4)-8)/igg_mw
      adpcoef=4/igg_mw
      data['ATP'] = ampcoef+gdpcoef+adpcoef
```

3Stoic_coeffs.ipynb
Cell #3

```
# Populate the DataFrame with the results
columns = ['Name', 'ATP', 'A', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'V',
'W', 'Y']
dfcoefs = pd.DataFrame(results, columns=columns)
dfcoefs
dfcoefs.to_csv('coefs.csv', index=False)
```

Annex 2.4

4Feed_conditions.ipynb
Cell #1

```
import os
import glob


comp_location = "3Stoic_coeffs/coefs.csv"
path = os.path.join(root_path, comp_location)
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv(path, index_col=False)
#Assuming CAla = 20, and the rate constants, calculate Cjs
Cala = 20

# List of columns to calculate ratios
columns_to_ratio = ['C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'V','W','Y']

# Create a new DataFrame for the ratios
ratio_df = df[columns_to_ratio].div(df['A'], axis=0)*Cala

# Optionally, rename the columns to indicate they are ratios
ratio_df.columns = [f"A_{col}" for col in columns_to_ratio]

# Add the 'name' column at the end
ratio_df['name'] = df['Name']

# Display the new DataFrame with the ratios
ratio_df.dropna()
```

4Feed_conditions.ipynb
Cell #2

```
# Define the column name and the values
column_name = 'A_N'  # Change this to 'A_C' or any other name as needed

# Calculate mean and standard deviation, ignoring NaN values
mean_value = ratio_df[column_name].mean()
std_dev = ratio_df[column_name].std()

# Print mean and standard deviation
print(f'Mean: {mean_value}')
print(f'Standard Deviation: {std_dev}')

# Plotting
plt.figure(figsize=(10, 5))
plt.plot(ratio_df[column_name], marker='o', linestyle='', label=f'{column_name} Values')  #
Only markers
plt.axhline(mean_value, color='red', linestyle='--', label='Mean')
plt.axhline(mean_value + std_dev, color='green', linestyle='--', label='Mean + 1 Std Dev')
plt.axhline(mean_value - std_dev, color='orange', linestyle='--', label='Mean - 1 Std Dev')

plt.title(f'Mean and Standard Deviation of {column_name}')
plt.xlabel('Index')
plt.ylabel(f'{column_name} Values')
plt.legend()
plt.grid()
plt.show()
# Calculate the mean values for each column
mean_values = ratio_df.drop(columns=['name']).mean()

# Create a new DataFrame with the mean values as a single row
mean_df = pd.DataFrame(mean_values).T  # Transpose to get a single row
```

4Feed_conditions.ipynb

Cell #3

```
# Filter the DataFrame to remove outliers
# Create a new DataFrame by copying the original
new_ratio_df = ratio_df.copy()

# Apply the condition to the new DataFrame
for column in new_ratio_df.columns:
    if column != 'name':  # Skip the 'name' column
        mean_value = new_ratio_df[column].mean()
        std_dev = new_ratio_df[column].std()
        new_ratio_df[column] = new_ratio_df[column].where(
            (new_ratio_df[column] >= mean_value - std_dev) &
            (new_ratio_df[column] <= mean_value + std_dev),
            None
        )
# Calculate the mean values for each column
new_mean_values = new_ratio_df.drop(columns=['name']).mean()

# Create a new DataFrame with the mean values as a single row
new_mean_df = pd.DataFrame(new_mean_values).T  # Transpose to get a single row
```

Annex 2.5

5Cho_cell_simul.ipynb
Cell #1

```
import pandas as pd
a = 0
b = 71
doedf = pd.DataFrame(columns=['T_ind', 'Feed_pre_ind', 'Feed_post_ind'])
doedf.loc[len(doedf)] = ['72','0.055','1.65']
doedf.loc[len(doedf)] = ['84','0.082','2.452']
doedf.loc[len(doedf)] = ['84', '0.028', '0.847']
doedf.loc[len(doedf)] = ['84', '0.028', '2.452']
doedf.loc[len(doedf)] = ['84', '0.082', '0.847']
doedf.loc[len(doedf)] = ['96', '0.1', '1.65']
doedf.loc[len(doedf)] = ['96', '0.01', '1.65']
doedf.loc[len(doedf)] = ['96', '0.055', '0.3']
doedf.loc[len(doedf)] = ['96', '0.055', '1.65']
doedf.loc[len(doedf)] = ['96', '0.055', '3']
doedf.loc[len(doedf)] = ['108', '0.028', '0.847']
doedf.loc[len(doedf)] = ['108', '0.028', '2.452']
doedf.loc[len(doedf)] = ['108', '0.082', '0.847']
doedf.loc[len(doedf)] = ['108', '0.082', '2.452']
doedf.loc[len(doedf)] = ['120', '0.055', '1.65']
doedf

import os
import glob
root_path="/Users/"
comp_location = "4Feed_conditions/4comp.csv"
comp_path = os.path.join(root_path, comp_location)
coefs_location = "3Stoic_coeffs/coefs.csv"
coefs_path = os.path.join(root_path, coefs_location)
titer_location = "2mAb_titer_alloc/vmaxmab.csv"
titer_path = os.path.join(root_path, titer_location)
url = coefs_path
url2 = titer_path
url3 = comp_path
url41 = ''
# Feed Concentration
df = pd.read_csv(url3, header=None)
values_list = df.iloc[1].astype(float).tolist()
Conc_Pre=np.array(values_list);
Conc_Post=Conc_Pre;

#now define kinetics(t,state,tind,G_mult,G_val) and ode(t,state)
```

5Cho_cell_simul.ipynb
Cell #2

```
import pandas as pd
import numpy as np

df_cleaned = pd.read_csv(url2)
mabnames_list = df_cleaned['Name'].iloc[a:b].tolist()
print(mabnames_list)
df_cleaned

for mabname in mabnames_list:
    dfmb = pd.read_csv(url, index_col=False)

    def set_value(value):
        if value is None or (isinstance(value, float) and np.isnan(value)):
            return 0
        return value

    atp_value = set_value(dfmb[dfmb['Name'] == mabname]['ATP'].values[0])
    a_value = set_value(dfmb[dfmb['Name'] == mabname]['A'].values[0])
    r_value = set_value(dfmb[dfmb['Name'] == mabname]['R'].values[0])
    n_value = set_value(dfmb[dfmb['Name'] == mabname]['N'].values[0])
    d_value = set_value(dfmb[dfmb['Name'] == mabname]['D'].values[0])
    c_value = set_value(dfmb[dfmb['Name'] == mabname]['C'].values[0])
    g_value = set_value(dfmb[dfmb['Name'] == mabname]['G'].values[0])
    h_value = set_value(dfmb[dfmb['Name'] == mabname]['H'].values[0])
    i_value = set_value(dfmb[dfmb['Name'] == mabname]['I'].values[0])
    l_value = set_value(dfmb[dfmb['Name'] == mabname]['L'].values[0])
    k_value = set_value(dfmb[dfmb['Name'] == mabname]['K'].values[0])
    m_value = set_value(dfmb[dfmb['Name'] == mabname]['M'].values[0])
    f_value = set_value(dfmb[dfmb['Name'] == mabname]['F'].values[0])
    p_value = set_value(dfmb[dfmb['Name'] == mabname]['P'].values[0])
    s_value = set_value(dfmb[dfmb['Name'] == mabname]['S'].values[0])
    t_value = set_value(dfmb[dfmb['Name'] == mabname]['T'].values[0])
    w_value = set_value(dfmb[dfmb['Name'] == mabname]['W'].values[0])
    v_value = set_value(dfmb[dfmb['Name'] == mabname]['V'].values[0])
    q_value = set_value(dfmb[dfmb['Name'] == mabname]['Q'].values[0])
    e_value = set_value(dfmb[dfmb['Name'] == mabname]['E'].values[0])
```

5Cho_cell_simul.ipynb
Cell #2

```
# Set vmaxmab value
vmm = set_value(df_cleaned[df_cleaned['Name'] == mabname]['vmaxmab'].values[0])
VmaxMAB = vmm

if vmm > 0.02:
    folder="high"
else:
    folder="low"

#for DOE conditions
for index, row in doedf.iterrows():
    # Set the variables for each row
    value1 = float(row['Feed_pre_ind'])
    value2 = float(row['Feed_post_ind'])
    tind = int(row['T_ind'])

    count1 = tind-1
    count2 = 241-tind

    # Create the Feed_Rate array by multiplying the values by their counts and then scaling
    Feed_Rate = np.array([value1] * count1 + [value2] * count2)


    #solve DE (IVP) using Rung-Kutta 4/5
    abs_tol = 1e-8
    rel_tol = 1e-7
    sol = solve_ivp(ode, time_span, x_init, method='LSODA',t_eval =t_points, atol =
abs_tol, rtol = rel_tol)
```

5Cho_cell_simul.ipynb
Cell #2

```
    from datetime import datetime
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')  # Format:
YYYYMMDD_HHMMSS
    # Create the filename with the mabname and timestamp
    url42 = f'{folder}/{mabname}/{mabname}_{tind}_{value1}_{value2}
_{timestamp}.csv'
    url4 = f'{url41}{url42}'
    f = open(url4, 'w')

    # create the csv writer
    writer = csv.writer(f)
    writer.writerow(["t(h)","ACCOA(mmol.10-6cells)","ADP(mmol.10-
6cells)","AKG(mmol.10-6cells)","AMP(mmol.10-6cells)","ATP(mmol.10-
6cells)","CIT(mmol.10-6cells)","F6P(mmol.10-6cells)","G6P(mmol.10-
6cells)","GAP(mmol.10-6cells)","GLU(mmol.10-6cells)","MAL(mmol.10-
6cells)","NAD(mmol.10-6cells)","NADH(mmol.10-6cells)","NADP(mmol.10-
6cells)","NADPH(mmol.10-6cells)","OXA(mmol.10-6cells)","PEP(mmol.10-
6cells)","PYR(mmol.10-6cells)","R5P(mmol.10-6cells)","SUC(mmol.10-
6cells)","X5P(mmol.10-
6cells)","ALA(mM)","ARG(mM)","ASN(mM)","ASP(mM)","CYS(mM)","EGLC(mM)","
EGLN(mM)","EGLU(mM)","EPYR(mM)","GLY(mM)","HIS(mM)","ILE(mM)","LAC(m
M)","LEU(mM)","LYS(mM)","MET(mM)","NH4(mM)","PHE(mM)","PRO(mM)","SER(
mM)","THR(mM)","TYR(mM)","VAL(mM)","mAB(mg/L)","X(Mcells/mL)","V(L)"])
    writer.writerow([t[0], ACCOA[0], ADP[0], AKG[0], AMP[0], ATP[0], CIT[0], F6P[0],
G6P[0], GAP[0], GLU[0], MAL[0], NAD[0], NADH[0], NADP[0], NADPH[0], OXA[0],
PEP[0], PYR[0], R5P[0], SUC[0], X5P[0], ALA[0], ARG[0], ASN[0], ASP[0], CYS[0],
EGLC[0], EGLN[0], EGLU[0], EPYR[0], GLY[0], HIS[0], ILE[0], LAC[0], LEU[0],
LYS[0], MET[0], NH4[0], PHE[0], PRO[0], SER[0], THR[0], TYR[0], VAL[0], 0.00, X[0],
V[0]])
    for i in range(1,21):
writer.writerow([t[i],ACCOA[i],ADP[i],AKG[i],AMP[i],ATP[i],CIT[i],F6P[i],G6P[i],GAP[
i],GLU[i],MAL[i],NAD[i],NADH[i],NADP[i],NADPH[i],OXA[i],PEP[i],PYR[i],R5P[i],S
UC[i],X5P[i],ALA[i],ARG[i],ASN[i],ASP[i],CYS[i],EGLC[i],EGLN[i],EGLU[i],EPYR[i],
GLY[i],HIS[i],ILE[i],LAC[i],LEU[i],LYS[i],MET[i],NH4[i],PHE[i],PRO[i],SER[i],THR[i],
TYR[i],VAL[i],mAB[i],X[i],V[i]]
    # close the file
    f.close()
```

Annex 2.6

6Reacted_calc.ipynb
Cell #1

```
import os
import pandas as pd
import glob

comp_location = "4Feed_conditions/4comp.csv"
comp_path = os.path.join(root_path, comp_location)

df_comp = pd.read_csv(comp_path)

folder_name = "5Cho_cell_simul/high"
folder_path = os.path.join(root_path, folder_name)

from pathlib import Path

file_list = []

for dirpath, dirnames, filenames in os.walk(folder_path):
    for filename in filenames:
        if '.DS_Store' not in filename:
            file_list.append(os.path.join(dirpath, filename))
```

6Reacted_calc.ipynb
Cell #2

```
for full_path in file_list:

    t_ind = float(os.path.basename(full_path).split('_')[1])
    feed_preind = float(os.path.basename(full_path).split('_')[2])
    feed_postind = float(os.path.basename(full_path).split('_')[3].replace('.csv', ''))

    df = pd.read_csv(full_path)

    # Assuming df_comp and df are already defined DataFrames

    # Get all column names that end with (mM)
    mM_columns = [col for col in df.columns if col.endswith('(mM)')]

    # Create a dictionary to hold new DataFrames for each (mM) column
    new_dfs = {}

    # Iterate through each (mM) column
    for mM_col in mM_columns:
        spe_feed_conc = df_comp[mM_col.replace("(mM)", "")]
        spe_i = df[mM_col].iloc[0]
        v_i = df['V(L)'].iloc[0]

        # Function to perform calculations for each row
        def calculate_row(row):
            spe_t = row[mM_col]
            v_t = row['V(L)']
            v_ind = df.loc[df['t(h)'] == t_ind, 'V(L)'].values[0] if not df.loc[df['t(h)'] ==
t_ind].empty else 0

            # Quantidade que está no sistema mas ainda não reagiu
            a_nr = spe_t * v_t

            # Quantidade introduzida no início
            a_b = spe_i * v_i

            # Quantidade introduzida no feed
            a_feed = (v_t - v_i) * spe_feed_conc

            # Calculate ra
            ra = a_nr - (a_b + a_feed)

            return pd.Series([a_nr, a_feed, ra])
```

6Reacted_calc.ipynb
Cell #2

```
    # Create a new DataFrame with the calculated values
    new_df = pd.DataFrame()
    new_df['t(h)'] = df['t(h)']
    new_df[mM_col] = df[mM_col]
    new_df['V(L)'] = df['V(L)']

    # Apply the function and create new columns
    new_df[['a_nr', 'a_feed', 'ra']] = df.apply(calculate_row, axis=1)
    # Formatting
    new_df['a_feed'] = new_df['a_feed'].apply(lambda x: x.item())
    new_df['ra'] = new_df['ra'].apply(lambda x: x.item())

    # Store the new DataFrame in the dictionary
    new_dfs[mM_col] = new_df

# Now new_dfs contains a DataFrame for each (mM) column
# Create an empty dictionary to hold the new DataFrame data
data = {}

# Loop through each column name in mM_columns
for n in range(len(mM_columns)):
    column_name = mM_columns[n]
    # Extract the values from the corresponding DataFrame
    data[column_name] = new_dfs[column_name]['ra']

# Create a new DataFrame from the dictionary
new_dataframe = pd.DataFrame(data)

# Add 't(h)' as the first column
new_dataframe.insert(0, 't(h)', df['t(h)'].values)  # Insert at index 0

new_dataframe['mAB']=df['mAB(mg/L)']*df['V(L)']

spe_i = df['X(Mcells/mL)'].iloc[0]
v_i = df['V(L)'].iloc[0]
new_dataframe['X']=df['X(Mcells/mL)']*df['V(L)'] - spe_i*v_i

ficheiro=full_path
parts = ficheiro.rsplit('/', 3)
# Combine the last three parts to get the desired path
new_path = parts[-3] + '/' + parts[-2] + '/' + parts[-1]
new_dataframe.to_csv(f'{new_path}', index=False)
```

Annex 2.7

7PCA.ipynb
Cell #1

```
folder_name = "6Reacted_calc/high"
folder_path = os.path.join(root_path, folder_name)

from pathlib import Path

file_list = []

for dirpath, dirnames, filenames in os.walk(folder_path):
    for filename in filenames:
        if '.DS_Store' not in filename:
            file_list.append(os.path.join(dirpath, filename))

df = pd.read_csv(file_list[0])
df
```

7PCA.ipynb
Cell #2

```
import numpy as np
from sklearn.preprocessing import PowerTransformer
from sklearn.decomposition import PCA

for full_path in file_list:
    ##Data Normalisation
    df = pd.read_csv(full_path)
    transformer = PowerTransformer(method='yeo-johnson')

    # Create an empty DataFrame to store transformed data
    transformed_df = pd.DataFrame()

    # Optionally, add the 't(h)' column back to the transformed DataFrame
    transformed_df['t(h)'] = df['t(h)']

    # Iterate over each column in the original DataFrame
    for column in df.columns:
        if column != 't(h)':  # Skip the 't(h)' column
            # Convert the column to a 2D array and transform it
            data = np.array(df[column]).reshape(-1, 1)
            transformed_data = transformer.fit_transform(data)

            # Add the transformed data to the new DataFrame
            transformed_df[column] = transformed_data.flatten()  # Flatten to 1D for DataFrame

    ##PCA
    # Step 2: Standardize the data (excluding the time column)
    features = transformed_df.columns[1:]  # Exclude the time column
    x = transformed_df[features].values

    # Step 3: Apply PCA
    pca = PCA(n_components=2)  # Specify the number of principal components
    principal_components = pca.fit_transform(x)

    # Step 4: Create a DataFrame for the principal components
    principal_df = pd.DataFrame(data=principal_components, columns=['PC1','PC2'])
```

7PCA.ipynb
Cell #2

```
    #scores in last columns
    transformed_df['Scores1'] = principal_components[:, 0]  # 2nd column (index 1)
    transformed_df['Scores2'] = principal_components[:, 1]  # 3rd column (index 2)

    #PCs in last rows
    import warnings
    warnings.simplefilter(action='ignore', category=FutureWarning) # Suppress
FutureWarnings
    # Add PC1
    new_row = [None] * transformed_df.shape[1]  # Start with a row of None
    new_row[0] = "PC1"  # Insert "PC1" in the first column
    new_row[1:len(pca.components_[0]) + 1] = pca.components_[0]  # Insert array values into
columns 2 to 26
    transformed_df.loc[len(df)] = new_row  # Add the row to the DataFrame

    # Add PC2
    new_row = [None] * transformed_df.shape[1]  # Start with a new row of None
    new_row[0] = "PC2"  # Insert "PC2" in the first column
    new_row[1:len(pca.components_[1]) + 1] = pca.components_[1]  # Insert array values into
columns 2 to 26
    transformed_df.loc[len(df) + 1] = new_row  # Add the row to the DataFrame

    #explained variance
    transformed_df.at[21, "Scores1"] = pca.explained_variance_ratio_[0]
    transformed_df.at[22, "Scores2"] = pca.explained_variance_ratio_[1]

    ##save scores onto csvs
    ficheiro=full_path
    parts = ficheiro.rsplit('/', 3)  # Split into four parts to capture the last three segments
    # Combine the last three parts to get the desired path
    new_path = parts[-3] + '/' + parts[-2] + '/' + parts[-1]
    transformed_df.to_csv(f'{new_path}', index=False)
```

7PCA.ipynb
Cell #3

```
doe = '_96_0.055_3.0_'
observ = [8,9]
folder_name = "7PCA"
folder_path2 = os.path.join(root_path, folder_name)
from pathlib import Path
import os

file_list2 = []

for dirpath, dirnames, filenames in os.walk(folder_path2):
    for filename in filenames:
        if '.DS_Store' not in filename and doe in filename:
            file_list2.append(os.path.join(dirpath, filename))

doe1 = pd.DataFrame()

for filename_pca in file_list2:
    df = pd.read_csv(filename_pca)
    a = df.iloc[observ[0]][1:].tolist()
    b = df.iloc[observ[1]][1:].tolist()

    name = filename_pca.rsplit('/', 3)[2]
    # Initialize the column if it doesn't exist
    if name not in doe1.columns:
        doe1[name] = None

    # Assign the lists to the specified rows
    doe1.at[observ[0], name] = a
    doe1.at[observ[1], name] = b
```

7PCA.ipynb
Cell #4

```
from sklearn.cluster import KMeans
# Prepare data for clustering
coordinates = np.array([doe1[column].iloc[0] for column in doe1.columns])

# Apply K-means clustering
kmeans = KMeans(n_clusters=2)  # Specify the number of clusters
kmeans.fit(coordinates)
labels = kmeans.labels_

# Create scatter plot
plt.figure(figsize=(10, 8))


# Adding labels and title
plt.title('PCA Scores Plot PC1 vs. PC2')
plt.xlabel('PC1')
plt.ylabel('PC2')
# Set x-axis limits
#plt.xlim(-6, -4)
plt.axhline(0, color='black', linewidth=0.5, ls='--')
plt.axvline(0, color='black', linewidth=0.5, ls='--')
plt.grid(color='gray', linestyle='--', linewidth=0.5)

# Plot each point with its cluster label
for i, column in enumerate(doe1.columns):
    coords = coordinates[i]
    plt.scatter(coords[0], coords[1], label=column, alpha=0.7,
                color='blue' if labels[i] == 0 else 'red')

# Custom legend placement with multiple columns
plt.legend(title='Points', ncol=5, loc='upper center', bbox_to_anchor=(0.5, -0.1),
fontsize='small')

plt.show()
```

Annex 3

Annex 3.1

Name	ChemEMBL ID	Light Chain	Heavy Chain	Combined Chains	IEEE754	standardized_values	p	vmaxmab
satralizumab	CHEMBL3833307	DIQMTQSPSSLSASV	QVQLQESGPGLVKPSETLS	DIQMTQSPSSLSASVGDSV	1.8219000427447365e-261	-0.30315828076461737	0.18750193947452787	0.012
naxitamab	CHEMBL4297984	EIVMTQTPATLSVSAG	QVQLVESGPGVVQPGRSLF	EIVMTQTPATLSVSAGERVT	2.191522347535003e-261	-0.010433584064177732	0.35117275551526717	0.0117
emicizumab	CHEMBL3833393	DIQMTQSPSSLSASV	QVQLVQSGSELKKPGASVK	DIQMTQSPSSLSASVGDRV	2.809369512060386e-261	0.47887434324162637	0.6923253117801564	0.0604
ravulizumab	CHEMBL3989986	DIQMTQSPSSLSASV	QXVQLVQSGAEVKKPGASV	DIQMTQSPSSLSASVGDRV	1.8268297001676168e-261	-0.29925420772776834	0.1892400985380648	0.0107
secukinumab	CHEMBL1743068	EIVLTQSPGTLSLSPC	EVQLVESGGGLVQPGGSLR	EIVLTQSPGTLSLSPGERAT	1.8734375678272018e-261	-0.2623428153477878	0.20628485025336762	0.014
Adalimumab	CHEMBL5314652	DIQMTQSPSSLSASV	EVQLVESGGGLVQPGRSLR	DIQMTQSPSSLSASVGDRV	9.866410140847924e-262	-0.9646468940384361	0.03252663929258673	0.0144
Bamlanivimab	CHEMBL4650379	DIQMTQSPSSLSASV	QVQLVQSGAEVKKPGSSVK	DIQMTQSPSSLSASVGDRV	6.072180704255146e-262	-1.2651332680057004	0.01382074824418844	0.01
atoltivimab	CHEMBL4298183	DIQMTQSPSSLSASV	QVQLVESGGGVVQPGRSLF	DIQMTQSPSSLSASVGDRI	1.2788502676041485e-262	-1.644744070727636	0.004618196532373355	0.015
anifrolumab	CHEMBL2364653	EIVLTQSPGTLSLSPC	EVQLVQSGAEVKKPGESLK	EIVLTQSPGTLSLSPGERAT	2.0413272648184175e-261	-0.12938151946828513	0.2768234467258838	0.0147
enfortumab	CHEMBL3301579	DIQMTQSPSSVSAS	EVQLVESGGGLVQPGGSLR	DIQMTQSPSSVSASVGDRV	6.380363511588344e-263	-1.6954937297449169	0.003986254833220596	0.0131
sacituzumab	CHEMBL5314875	DIQLTQSPSSLSASV	QVQLQQSGSELKKPGASVK	DIQLTQSPSSLSASVGDRV	2.866092511330487e-261	0.5237964773825875	0.719469066348334	0.0627
crizanlizumab	CHEMBL4297734	DIQMTQSPSSLSASV	QVQLVQSGAEVKKPGASVK	DIQMTQSPSSLSASVGDRV	1.1082273852213763e-261	-0.8683558067512144	0.04260592967563273	0.0133
ipilimumab	CHEMBL5314775	QGQSGSCRTQLYGY	QVQLVESGGGVVQPGRSLF	QGQSGSCRTQLYGYNLCP	1.402392635434823e-262	-1.6349600556170194	0.004751035454766845	0.0133
brodalumab	CHEMBL1742996	EIVMTQSPATLSVSPG	QVQLVQSGAEVKKPGASVK	EIVMTQSPATLSVSPGERAT	1.1476716536840615e-261	-0.8371176715719482	0.046474877772961394	0.0145
guselkumab	CHEMBL2364648	QSVLTQPPSVSGAPC	EVQLVQSGAEVKKPGESLK	QSVLTQPPSVSGAPGQRVT	3.238253522233454e-261	0.8185317182715701	0.858163604626574	0.0608
amivantamab	CHEMBL4297774	DIQMTQSPSSVSAS	QVQLVQSGAEVKKPGASVK	DIQMTQSPSSVSASVGDRV	2.783362321105148e-261	0.45827778565909016	0.6794040089530777	0.0618
belantamab	CHEMBL4298208	DIQMTQSPSSLSASV	QVQLVQSGAEVKKPGSSVK	DIQMTQSPSSLSASVGDRV	2.8321192598461224e-261	0.4968911483269417	0.7033870481082775	0.0608
idarucizumab	CHEMBL3544996	DVVMTQSPLSLPVTL	QVQLQESGPGLVKPSETLS	DVVMTQSPLSLPVTLGQPA	1.2370264636737353e-261	-0.7663525707813059	0.05650933514281124	0.0111
maftivimab	CHEMBL4298184	DIQMTQSPSSLSASV	EVQLVESGGGLVQPGGSLR	DIQMTQSPSSLSASVGDRV	6.380109303380447e-263	-1.695495742962723	0.003986231556468926	0.0147
polatuzumab	CHEMBL3301582	DIQLTQSPSSLSASV	EVQLVESGGGLVQPGGSLR	DIQLTQSPSSLSASVGDRV	1.011298043323442e-261	-0.9451196054741999	0.03436446124282224	0.0132
obinutuzumab	CHEMBL1743048	DIVMTQTPLSLPVTP	QVQLVQSGAEVKKPGSSVK	DIVMTQTPLSLPVTPGEPAS	3.182405327750843e-261	0.7743023912549929	0.8417519210788889	0.0619
margetuximab	CHEMBL2364649	DIVMTQSHKFMSTS	QVQLQQSGPELVKPGASLK	DIVMTQSHKFMSTSVGDRV	1.8186705783772325e-261	-0.3057158752745562	0.18636992848969686	0.0107
imdevimab)	CHEMBL4650249	QSALTQPASVSGSPC	QVQLVESGGGVVQPGRSLF	QSALTQPASVSGSPGQSIT	1.9226780043806072e-261	-0.22334654356253483	0.22549840865774917	0.0104
tafasitamab	CHEMBL4298047	DIVMTQSPATLSLSP	EVQLVESGGGLVKPGGSLK	DIVMTQSPATLSLSPGERAT	2.930605539432190 7e-261	0.5748879735925194	0.74849690561791	0.0626
necitumumab	CHEMBL1743047	EIVMTQSPATLSLSPC	QVQLQESGPGLVKPSQTLS	EIVMTQSPATLSLSPGERAT	1.0184261283431765e-261	-0.9394744739482744	0.03491416235148323	0.0127
sarilumab	CHEMBL2108730	DIQMTQSPSSVSAS	EVQLVESGGGLVQPGRSLR	DIQMTQSPSSVSASVGDRV	1.280834826124946e-261	-0.7316582640652917	0.06213464624460076	0.0138
arcitumomab	CHEMBL1743090	QTVLSQSPAILSASP	EVKLVESGGGLVQPGGSLR	QTVLSQSPAILSASPGEKV	1.7896520212628867e-261	-0.3286973031264313	0.17643413795817547	0.0127
vedolizumab	CHEMBL1743087	DVVMTQSPLSLPVTP	QVQLVQSGAEVKKPGASVK	DVVMTQSPLSLPVTPGEPA	2.9897422957901523e-261	0.6217216977600982	0.7732922086876416	0.0644
daratumumab	CHEMBL1743007	EIVLTQSPATLSLSPG	EVQLLESGGGLVQPGGSLR	EIVLTQSPATLSLSPGERATL	1.6629193795292744e-261	-0.4290640116089386	0.13788469816372417	0.0137
siltuximab	CHEMBL1743070	QIVLIQSPAIMSASPG	EVQLVESGGKLLKPGGSLK	QIVLIQSPAIMSASPGEKVT	3.0791476211582348e-261	0.6925268045207041	0.8073970622181164	0.0612
faricimab	CHEMBL4297750	DIQLTQSPSSLSASV	EVQLVESGGGLVQPGGSLR	DIQLTQSPSSLSASVGDRV	8.300432727117581e-262	-1.0886654550216706	0.02289266942913518	0.0113
elotuzumab	CHEMBL1743010	DIQMTQSPSSLSASV	EVQLVESGGGLVQPGGSLR	DIQMTQSPSSLSASVGDRV	3.195725860103285e-261	0.78485167018709	0.8458012052555179	0.0647
romosozumab	CHEMBL2107874	DIQMTQSPSSLSASV	EVQLVQSGAEVKKPGASVK	DIQMTQSPSSLSASVGDRV	4.628773284698148e-261	1.9197625434903298	0.9933534375169212	0.0645
Bebtelovimab	CHEMBL4802210	QSALTQPASVSGSPC	QITLKESGPTLVKPTQTLTLT	QSALTQPASVSGSPGQSIT	1.767446325558516e-261	-0.3462832427622238	0.16911570264309628	0.0143
cilgavimab	CHEMBL4650258	DIVMTQSPDSLAVSL	EVQLVESGGGLVKPGGSLR	DIVMTQSPDSLAVSLGERA	3.388542220412187e-261	0.9375537930266002	0.8953620311996273	0.0617
galcanezumab	CHEMBL3707328	DIQMTQSPSSLSASV	QVQLVQSGAEVKKPGSSVK	DIQMTQSPSSLSASVGDRV	4.080094214737719e-261	1.4852327214029386	0.9768333769451705	0.0607
casirivimab	CHEMBL4650248	DIQMTQSPSSLSASV	QVQLVESGGGLVKPGGSLR	DIQMTQSPSSLSASVGDRV	1.8225000346471638e-261	-0.30268311342383597	0.18771283381626927	0.0115
ramucirumab	CHEMBL1743062	DIQMTQSPSSVSASI	EVQLVQSGGGLVKPGGSLR	DIQMTQSPSSVSASIGDRV	1.2171216277481658e-261	-0.7821163301206958	0.0541110956920593	0.0119
alemtuzumab	CHEMBL5314722	DIQMTQSPSSLSASV	QVQLQESGPGLVRPSQTLS	DIQMTQSPSSLSASVGDRV	1.399935057122137e-261	-0.6373360910151897	0.08021163543807154	0.0117
tralokinumab	CHEMBL1743081	SYVLTQPPSVSVAPG	QVQLVQSGAEVKKPGASVK	SYVLTQPPSVSVAPGKTAR	5.40416712927073e-261	2.5338405496990997	0.998882147800803	0.0617
isatuximab	CHEMBL3545131	DIVMTQSHLSMSTSL	QVQLVQSGAEVAKPGTSVK	DIVMTQSHLSMSTSLGDPV	3.7018990734976165e-261	1.185718713242887	0.9463126904897934	0.061
avelumab	CHEMBL3833373	QSALTQPASVSGSPC	EVQLLESGGGLVQPGGSLR	QSALTQPASVSGSPGQSIT	1.922678004587051e-261	-0.2233465433990405	0.22549840874090082	0.0118
sutimlimab	CHEMBL4297832	QIVLTQSPATLSLSPG	EVQLVESGGGLVKPGGSLR	QIVLTQSPATLSLSPGERAT	1.6147289875405383e-261	-0.46722869403193923	0.12519706740496328	0.012
bimekizumab	CHEMBL4297700	AIQLTQSPSSLSASV	EVQLVESGGGLVQPGGSLR	AIQLTQSPSSLSASVGDRV	8.962283600821427e-262	-1.0362497610132115	0.02656746045273476	0.011
mogamulizumab	CHEMBL1743041	EIVLTQSPGTLSLSPC	QAQVVESGGGVVQSGRSL	EIVLTQSPGTLSLSPGERAT	2.7113349879503824e-261	0.4012352885515792	0.6421992207190471	0.0618
teprotumumab	CHEMBL1743079	EIVLTQSPATLSLSPG	QVELVESGGGVVQPGRSQI	EIVLTQSPATLSLSPGERATL	1.8312318649681474e-261	-0.29576788577327534	0.19080268482720236	0.0106
tisotumab	CHEMBL4594344	DIQMTQSPPSLSAS	EVQLLESGGGLVQPGGSLR	DIQMTQSPPSLSASAGDRV	1.3532126919724396e-261	-0.674338160398781	0.07261294488650703	0.0139
cemiplimab	CHEMBL4297723	DIQMTQSPSSLSASV	EVQLLESGGVLVQPGGSLR	DIQMTQSPSSLSASVGDSI	2.3550746716908036e-262	-1.559511805741182	0.005911542423159843	0.0108
lanadelumab	CHEMBL3545189	DIQMTQSPSTLSASV	EVQLLESGGGLVQPGGSLR	DIQMTQSPSTLSASVGDRV	2.1272799173875555e-261	-0.061310778531328644	0.3182021373912741	0.0146
relatlimab	CHEMBL3990044	EIVLTQSPATLSLSPG	QXVQLQQWGAGLLKPSET	EIVLTQSPATLSLSPGERATL	1.3838263309965177e-261	-0.6500934974493822	0.07751314879723505	0.0103
ibalizumab	CHEMBL1743029	DIVMTQSPDSLAVSL	QVQLQQSGPEVVKPGASV	DIVMTQSPDSLAVSLGERV	4.391962108175224e-261	1.7322184507108598	0.9885793337933311	0.0643
risankizumab	CHEMBL3990029	DIQMTQSPSSLSASV	QVQLVQSGAEVKKPGSSVK	DIQMTQSPSSLSASVGDRV	4.338927260220229e-261	1.6902171710776166	0.9871125933198245	0.0605
mosunetuzumab	CHEMBL4297788	DIVMTQSPDSLAVSL	EVQLVQSGAEVKKPGASVK	DIVMTQSPDSLAVSLGERA	2.5247443170650494e-261	0.25346363940006666	0.538573243298084	0.0636
ansuvimab	CHEMBL4594388	DIQMTQSPSSLSASV	EVQLVESGGGLIQPGGSLR	DIQMTQSPSSLSASVGDRI	1.6675323248139048e-261	-0.42541076073138295	0.13915420660121539	0.0143
fremanezumab	CHEMBL4297756	EIVLTQSPATLSLSPG	EVQLVESGGGLVQPGGSLR	EIVLTQSPATLSLSPGERATL	3.0142271091162375e-261	0.6411125988549663	0.7830386483859616	0.0644
Obiltoxaximab	CHEMBL3544926	DIQMTQSPSSLSASV	QVQLQQSGPELKKPGASVK	DIQMTQSPSSLSASVGDRV	3.6425365055220472e-261	1.1387061561520264	0.9389169337604604	0.0615
eptinezumab	CHEMBL3833320	QVLTQSPSSLSASVG	EVQLVESGGGLVQPGGSLR	QVLTQSPSSLSASVGDRVT	4.1556800150695135e-261	1.5450933688536908	0.9804669718505277	0.0608
basiliximab	CHEMBL5314724	QIVLTQSPAIMSASPC	EVQLQQSGTVLARPGASVK	QIVLTQSPAIMSASPGEKVT	3.8234570645389224e-261	1.2819873247361027	0.9588938800400432	0.0643
bezlotoxumab	CHEMBL2108670	EIVLTQSPGTLSLSPC	EVQLVQSGAEVKKSGESLK	EIVLTQSPGTLSLSPGERAT	2.51471551191693e-261	0.24552126442105454	0.5328205837731694	0.0646
odesivimab	CHEMBL4298188	DIVMTQSPDSLAVSL	EVQLVESGGGLVQPGGSLR	DIVMTQSPDSLAVSLGERA	4.16323212933388e-261	1.5510743129927034	0.98079775601757	0.0648
emapalumab	CHEMBL3989977	NFMLTQPHSVSESPC	EVQLLESGGGLVQPGGSLF	NFMLTQPHSVSESPGKTVT	1.5104212272468642e-261	-0.5498358773641071	0.10113579266880322	0.0142

benralizumab	CHEMBL1742991	DIQMTQSPSSLSASV	EVQLVQSGAEVKKPGASVK	DIQMTQSPSSLSASVGDRV	2.18185490697225e-261	-0.018089774099049057	0.34610974911698716	0.0124
evinacumab	CHEMBL3545191	DIQMTQSPSTLSASV	EVQLVESGGGVIQPGGSLR	DIQMTQSPSTLSASVGDRV	1.2850972946743569e-261	-0.7282825754308369	0.06271904345339334	0.011
burosumab	CHEMBL3707326	AIQLTQSPSSLSASV	QVQLVQSGAEVKKPGASVK	AIQLTQSPSSLSASVGDRV	1.2170742598412454e-261	-0.7821538434309233	0.054105504644634476	0.0127
tezepelumab	CHEMBL3707229	SYVLTQPPSVSVAPG	QMQLVESGGGVVQPGRSL	SYVLTQPPSVSVAPGQTAR	3.9721904079146678e-261	1.3997776265243347	0.9704808340166278	0.0628
ixekizumab	CHEMBL1743034	DIVMTQTPLSLSVTP	QVQLVQSGAEVKKPGSSVK	DIVMTQTPLSLSVTPGQPA	3.6271842353682003e-261	1.1265478297572002	0.9368544988758875	0.0606
erenumab	CHEMBL3833329	QSVLTQPPSVSAAPG	QVQLVESGGGVVQPGRSLR	QSVLTQPPSVSAAPGQKVT	4.915114593244376e-261	2.1465323341948244	0.99655492044827	0.0638
durvalumab	CHEMBL3301587	EIVLTQSPGTLSLSPG	EVQLVESGGGLVQPGGSLR	EIVLTQSPGTLSLSPGERAT	2.323348138003 77e-261	0.09396667542276467	0.42314723316752934	0.0111
dostarlimab	CHEMBL4298124	DIQLTQSPSFLSAYV	EVQLLESGGGLVQPGGSLR	DIQLTQSPSFLSAYVGDRV	2.0716463991774668e-261	-0.1053700913308285	0.29103752382460185	0.0138
olaratumab	CHEMBL1743049	EIVLTQSPATLSLSPG	QLQLQESGPGLVKPSETLS	EIVLTQSPATLSLSPGERATL	4.114448368811169e-262	-1.4201771051027772	0.008843704545484178	0.0109
tixagevimab	CHEMBL4650265	EIVLTQSPGTLSLSPG	QMQLVQSGPEVKKPGTSVK	EIVLTQSPGTLSLSPGERAT	1.2312765315892372e-261	-0.7709062654682051	0.055806471196015346	0.013
loncastuximab	CHEMBL4297238	EIVLTQSPAIMSASPG	QVQLVQPGAEVVKPGASVK	EIVLTQSPAIMSASPGERVT	1.789117745151663 3e-261	-0.3291204264352393	0.1762551692015338	0.013

Annex 3.2

	Name	ATP	A	C	D	E	F	G	H	I
0	satralizumab	0.027466	0.000091	0.000035	0.000070	0.000084	0.000056	0.000105	0.000028	0.000049
1	naxitamab	0.027397	0.000097	0.000035	0.000055	0.000083	0.000069	0.000083	0.000014	0.000042
2	emicizumab	0.027179	0.000090	0.000034	0.000076	0.000083	0.000048	0.000083	0.000014	0.000055
3	ravulizumab	0.027384	0.000097	0.000035	0.000069	0.000069	0.000055	0.000104	0.000014	0.000048
4	secukinumab	0.027234	0.000095	0.000041	0.000054	0.000081	0.000054	0.000101	0.000014	0.000041
5	Adalimumab	0.027420	0.000103	0.000034	0.000062	0.000062	0.000048	0.000090	0.000014	0.000048
6	Bamlanivimab	0.027391	0.000089	0.000034	0.000061	0.000061	0.000055	0.000082	0.000014	0.000048
7	atoltivimab	0.027313	0.000090	0.000034	0.000062	0.000062	0.000062	0.000083	0.000021	0.000062
8	anifrolumab	0.027351	0.000103	0.000034	0.000062	0.000083	0.000062	0.000096	0.000014	0.000048
9	enfortumab	0.027516	0.000104	0.000035	0.000063	0.000063	0.000069	0.000104	0.000014	0.000049
10	sacituzumab	0.027397	0.000103	0.000034	0.000076	0.000062	0.000055	0.000082	0.000021	0.000055
11	crizanlizumab	0.027306	0.000089	0.000034	0.000082	0.000075	0.000055	0.000096	0.000021	0.000041
12	ipilimumab	0.027568	0.000091	0.000046	0.000059	0.000078	0.000059	0.000189	0.000013	0.000046
13	brodalumab	0.027300	0.000111	0.000035	0.000062	0.000076	0.000062	0.000090	0.000014	0.000035
14	guselkumab	0.027683	0.000118	0.000035	0.000049	0.000063	0.000028	0.000132	0.000021	0.000035
15	amivantamab	0.027371	0.000103	0.000034	0.000062	0.000062	0.000069	0.000090	0.000021	0.000055
16	belantamab	0.027298	0.000075	0.000034	0.000068	0.000062	0.000055	0.000082	0.000021	0.000048
17	idarucizumab	0.027850	0.000094	0.000052	0.000115	0.000105	0.000094	0.000157	0.000042	0.000052
18	maftivimab	0.027532	0.000090	0.000035	0.000063	0.000063	0.000063	0.000083	0.000014	0.000049
19	polatuzumab	0.027449	0.000090	0.000034	0.000083	0.000083	0.000062	0.000090	0.000014	0.000041
20	obinutuzumab	0.027372	0.000068	0.000034	0.000062	0.000082	0.000048	0.000103	0.000021	0.000048
21	margetuximab	0.027291	0.000089	0.000034	0.000075	0.000062	0.000069	0.000082	0.000034	0.000041
22	imdevimab)	0.027702	0.000111	0.000035	0.000042	0.000062	0.000028	0.000118	0.000021	0.000042
23	tafasitamab	0.027248	0.000075	0.000034	0.000054	0.000088	0.000061	0.000088	0.000020	0.000041
24	necitumumab	0.027527	0.000117	0.000035	0.000055	0.000083	0.000055	0.000097	0.000021	0.000041
25	sarilumab	0.027455	0.000097	0.000035	0.000062	0.000069	0.000062	0.000097	0.000014	0.000049
26	arcitumomab	0.027176	0.000097	0.000035	0.000055	0.000069	0.000042	0.000090	0.000028	0.000062
27	vedolizumab	0.027358	0.000068	0.000034	0.000061	0.000075	0.000054	0.000109	0.000020	0.000041
28	daratumumab	0.027465	0.000110	0.000034	0.000055	0.000083	0.000055	0.000083	0.000014	0.000041
29	siltuximab	0.027376	0.000097	0.000034	0.000048	0.000076	0.000041	0.000097	0.000014	0.000048
30	faricimab	0.027349	0.000075	0.000034	0.000068	0.000062	0.000062	0.000082	0.000021	0.000048
31	elotuzumab	0.027329	0.000089	0.000034	0.000076	0.000062	0.000048	0.000089	0.000021	0.000048
32	romosozumab	0.027249	0.000075	0.000034	0.000075	0.000062	0.000055	0.000096	0.000014	0.000048
33	Bebtelovimab	0.027679	0.000132	0.000035	0.000049	0.000070	0.000035	0.000104	0.000021	0.000042
34	cilgavimab	0.027353	0.000094	0.000034	0.000060	0.000081	0.000047	0.000087	0.000013	0.000040
35	galcanezumab	0.027422	0.000083	0.000035	0.000076	0.000062	0.000056	0.000104	0.000021	0.000049

	Name	ATP	A	C	D	E	F	G	H	I
36	casirivimab	0.027411	0.000090	0.000034	0.000076	0.000069	0.000055	0.000096	0.000014	0.000055
37	ramucirumab	0.027555	0.000104	0.000035	0.000084	0.000056	0.000070	0.000111	0.000014	0.000056
38	alemtuzumab	0.027285	0.000075	0.000034	0.000068	0.000068	0.000048	0.000082	0.000021	0.000062
39	tralokinumab	0.027581	0.000118	0.000035	0.000062	0.000069	0.000028	0.000125	0.000021	0.000049
40	isatuximab	0.027413	0.000090	0.000034	0.000076	0.000062	0.000055	0.000090	0.000028	0.000048
41	avelumab	0.027762	0.000111	0.000035	0.000042	0.000063	0.000028	0.000118	0.000021	0.000035
42	sutimlimab	0.027363	0.000090	0.000035	0.000048	0.000069	0.000048	0.000083	0.000028	0.000041
43	bimekizumab	0.027240	0.000082	0.000034	0.000075	0.000075	0.000054	0.000082	0.000020	0.000048
44	mogamulizumab	0.027406	0.000095	0.000034	0.000055	0.000082	0.000055	0.000109	0.000014	0.000041
45	teprotumumab	0.027294	0.000110	0.000034	0.000055	0.000082	0.000055	0.000082	0.000014	0.000034
46	tisotumab	0.027414	0.000104	0.000035	0.000062	0.000069	0.000055	0.000083	0.000014	0.000048
47	cemiplimab	0.027430	0.000090	0.000035	0.000070	0.000056	0.000077	0.000090	0.000021	0.000049
48	lanadelumab	0.027321	0.000089	0.000034	0.000062	0.000069	0.000055	0.000082	0.000014	0.000048
49	relatlimab	0.027278	0.000110	0.000034	0.000055	0.000083	0.000055	0.000083	0.000014	0.000048
50	ibalizumab	0.027184	0.000081	0.000034	0.000068	0.000075	0.000048	0.000088	0.000014	0.000034
51	risankizumab	0.027299	0.000096	0.000034	0.000076	0.000062	0.000062	0.000076	0.000027	0.000048
52	mosunetuzumab	0.027296	0.000089	0.000034	0.000068	0.000075	0.000055	0.000082	0.000014	0.000048
53	ansuvimab	0.027615	0.000118	0.000035	0.000063	0.000063	0.000056	0.000090	0.000028	0.000049
54	fremanezumab	0.027277	0.000096	0.000034	0.000048	0.000082	0.000055	0.000089	0.000014	0.000041
55	Obiltoxaximab	0.027316	0.000069	0.000034	0.000069	0.000069	0.000048	0.000096	0.000014	0.000055
56	eptinezumab	0.027617	0.000091	0.000049	0.000077	0.000063	0.000056	0.000105	0.000021	0.000035
57	basiliximab	0.027335	0.000104	0.000035	0.000049	0.000076	0.000042	0.000090	0.000021	0.000049
58	bezlotoxumab	0.027349	0.000096	0.000034	0.000055	0.000082	0.000055	0.000103	0.000014	0.000041
59	odesivimab	0.027401	0.000089	0.000034	0.000062	0.000082	0.000048	0.000089	0.000014	0.000041
60	emapalumab	0.027637	0.000096	0.000034	0.000055	0.000076	0.000034	0.000096	0.000021	0.000041
61	benralizumab	0.027298	0.000068	0.000034	0.000068	0.000068	0.000055	0.000096	0.000021	0.000055
62	evinacumab	0.027375	0.000089	0.000034	0.000062	0.000068	0.000055	0.000082	0.000014	0.000048
63	burosumab	0.027463	0.000097	0.000035	0.000076	0.000062	0.000069	0.000090	0.000014	0.000049
64	tezepelumab	0.027450	0.000118	0.000035	0.000062	0.000069	0.000028	0.000111	0.000028	0.000028
65	ixekizumab	0.027232	0.000062	0.000034	0.000062	0.000068	0.000062	0.000096	0.000034	0.000048
66	erenumab	0.027559	0.000103	0.000034	0.000048	0.000055	0.000027	0.000130	0.000014	0.000041
67	durvalumab	0.027290	0.000096	0.000034	0.000062	0.000082	0.000055	0.000096	0.000014	0.000041
68	dostarlimab	0.027320	0.000097	0.000035	0.000062	0.000069	0.000062	0.000090	0.000028	0.000042
69	olaratumab	0.027323	0.000115	0.000034	0.000054	0.000082	0.000054	0.000082	0.000014	0.000041
70	tixagevimab	0.027340	0.000095	0.000034	0.000055	0.000082	0.000055	0.000109	0.000020	0.000041
71	loncastuximab	0.027393	0.000097	0.000035	0.000048	0.000083	0.000042	0.000104	0.000028	0.000042

	Name	K	L	M	N	P	Q	R	S	T	V	W	Y
0	satralizumab	0.000077	0.000112	0.000007	0.000049	0.000070	0.000091	0.000035	0.000230	0.000126	0.000098	0.000014	0.000070
1	naxitamab	0.000083	0.000083	0.000007	0.000048	0.000062	0.000104	0.000048	0.000215	0.000125	0.000125	0.000014	0.000069
2	emicizumab	0.000090	0.000103	0.000007	0.000041	0.000083	0.000110	0.000062	0.000200	0.000117	0.000096	0.000014	0.000069
3	ravulizumab	0.000090	0.000117	0.000007	0.000069	0.000076	0.000090	0.000035	0.000193	0.000131	0.000104	0.000014	0.000062
4	secukinumab	0.000068	0.000122	NaN	0.000034	0.000081	0.000095	0.000068	0.000223	0.000115	0.000088	0.000014	0.000068
5	Adalimumab	0.000090	0.000103	0.000007	0.000048	0.000076	0.000103	0.000062	0.000207	0.000124	0.000103	0.000014	0.000076
6	Bamlanivimab	0.000089	0.000102	0.000007	0.000034	0.000075	0.000109	0.000048	0.000246	0.000130	0.000096	0.000014	0.000068
7	atoltivimab	0.000090	0.000110	0.000007	0.000041	0.000083	0.000110	0.000041	0.000234	0.000131	0.000083	0.000014	0.000062
8	anifrolumab	0.000069	0.000124	NaN	0.000034	0.000076	0.000096	0.000069	0.000221	0.000124	0.000090	0.000014	0.000062
9	enfortumab	0.000090	0.000090	0.000007	0.000042	0.000083	0.000104	0.000042	0.000222	0.000125	0.000104	0.000021	0.000056
10	sacituzumab	0.000096	0.000103	NaN	0.000034	0.000076	0.000096	0.000041	0.000213	0.000117	0.000117	0.000014	0.000076
11	crizanlizumab	0.000096	0.000103	0.000014	0.000055	0.000075	0.000096	0.000034	0.000226	0.000116	0.000103	0.000014	0.000068
12	ipilimumab	0.000072	0.000137	NaN	0.000046	0.000091	0.000111	0.000072	0.000267	0.000117	0.000091	0.000020	0.000085
13	brodalumab	0.000076	0.000104	0.000007	0.000049	0.000083	0.000097	0.000056	0.000215	0.000125	0.000111	0.000021	0.000056
14	guselkumab	0.000084	0.000111	NaN	0.000035	0.000111	0.000077	0.000028	0.000237	0.000139	0.000118	0.000028	0.000063
15	amivantamab	0.000083	0.000110	0.000007	0.000048	0.000076	0.000096	0.000048	0.000227	0.000117	0.000096	0.000021	0.000048
16	belantamab	0.000096	0.000116	0.000007	0.000055	0.000075	0.000103	0.000041	0.000219	0.000123	0.000089	0.000021	0.000075
17	idarucizumab	0.000147	0.000209	0.000010	0.000052	0.000126	0.000126	0.000084	0.000325	0.000178	0.000199	0.000021	0.000105
18	maftivimab	0.000083	0.000118	0.000007	0.000042	0.000069	0.000111	0.000049	0.000250	0.000125	0.000090	0.000014	0.000063
19	polatuzumab	0.000096	0.000117	NaN	0.000055	0.000076	0.000103	0.000034	0.000227	0.000117	0.000103	0.000014	0.000062
20	obinutuzumab	0.000089	0.000137	0.000014	0.000055	0.000089	0.000082	0.000041	0.000205	0.000109	0.000123	0.000014	0.000075
21	margetuximab	0.000096	0.000075	0.000014	0.000041	0.000069	0.000089	0.000048	0.000192	0.000144	0.000130	0.000014	0.000069
22	imdevimab)	0.000090	0.000090	0.000007	0.000049	0.000097	0.000076	0.000021	0.000257	0.000146	0.000111	0.000028	0.000069
23	tafasitamab	0.000081	0.000136	0.000020	0.000068	0.000081	0.000088	0.000047	0.000210	0.000115	0.000095	0.000014	0.000068
24	necitumumab	0.000076	0.000117	0.000007	0.000041	0.000083	0.000083	0.000055	0.000214	0.000124	0.000090	0.000014	0.000069
25	sarilumab	0.000090	0.000104	0.000007	0.000042	0.000076	0.000104	0.000042	0.000243	0.000111	0.000097	0.000021	0.000062
26	arcitumomab	0.000090	0.000069	0.000014	0.000062	0.000083	0.000055	0.000048	0.000249	0.000145	0.000076	0.000035	0.000069
27	vedolizumab	0.000089	0.000116	0.000007	0.000048	0.000089	0.000095	0.000048	0.000218	0.000109	0.000123	0.000014	0.000075
28	daratumumab	0.000076	0.000117	NaN	0.000048	0.000089	0.000096	0.000062	0.000213	0.000117	0.000096	0.000021	0.000062
29	siltuximab	0.000083	0.000097	0.000028	0.000041	0.000076	0.000083	0.000041	0.000241	0.000110	0.000103	0.000021	0.000090
30	faricimab	0.000089	0.000103	NaN	0.000048	0.000075	0.000103	0.000034	0.000233	0.000130	0.000109	0.000021	0.000068
31	elotuzumab	0.000096	0.000089	0.000007	0.000034	0.000076	0.000103	0.000041	0.000206	0.000131	0.000124	0.000021	0.000076
32	romosozumab	0.000089	0.000117	0.000007	0.000048	0.000075	0.000096	0.000048	0.000213	0.000130	0.000096	0.000014	0.000075
33	Bebtelovimab	0.000083	0.000083	0.000007	0.000042	0.000097	0.000077	0.000021	0.000257	0.000146	0.000104	0.000021	0.000070
34	cilgavimab	0.000094	0.000114	0.000013	0.000060	0.000067	0.000094	0.000040	0.000215	0.000107	0.000107	0.000020	0.000081
35	galcanezumab	0.000104	0.000104	0.000007	0.000042	0.000083	0.000090	0.000042	0.000215	0.000125	0.000097	0.000014	0.000076
36	casirivimab	0.000090	0.000110	0.000007	0.000062	0.000076	0.000103	0.000034	0.000193	0.000131	0.000096	0.000014	0.000069
37	ramucirumab	0.000098	0.000098	0.000007	0.000049	0.000077	0.000098	0.000035	0.000202	0.000118	0.000104	0.000021	0.000056
38	alemtuzumab	0.000103	0.000103	0.000007	0.000068	0.000075	0.000103	0.000048	0.000199	0.000130	0.000096	0.000014	0.000062
39	tralokinumab	0.000083	0.000083	NaN	0.000042	0.000118	0.000069	0.000035	0.000180	0.000139	0.000139	0.000028	0.000062
40	isatuximab	0.000083	0.000090	0.000014	0.000034	0.000076	0.000096	0.000048	0.000207	0.000124	0.000124	0.000014	0.000083
41	avelumab	0.000090	0.000076	0.000007	0.000056	0.000097	0.000076	0.000028	0.000250	0.000153	0.000118	0.000021	0.000076
42	sutimlimab	0.000090	0.000124	0.000007	0.000041	0.000090	0.000090	0.000041	0.000242	0.000131	0.000090	0.000021	0.000083
43	bimekizumab	0.000088	0.000109	0.000007	0.000041	0.000075	0.000095	0.000054	0.000204	0.000122	0.000109	0.000027	0.000054
44	mogamulizumab	0.000075	0.000123	NaN	0.000034	0.000082	0.000089	0.000061	0.000225	0.000116	0.000095	0.000014	0.000068
45	teprotumumab	0.000089	0.000117	NaN	0.000034	0.000089	0.000096	0.000062	0.000220	0.000117	0.000096	0.000027	0.000062
46	tisotumab	0.000090	0.000104	0.000007	0.000041	0.000083	0.000111	0.000048	0.000235	0.000117	0.000083	0.000014	0.000076
47	cemiplimab	0.000070	0.000111	0.000007	0.000063	0.000083	0.000090	0.000049	0.000223	0.000139	0.000097	0.000014	0.000056
48	lanadelumab	0.000096	0.000103	0.000007	0.000041	0.000069	0.000103	0.000041	0.000220	0.000137	0.000096	0.000027	0.000069
49	relatlimab	0.000069	0.000131	NaN	0.000055	0.000083	0.000096	0.000062	0.000213	0.000117	0.000083	0.000021	0.000062

	Name	K	L	M	N	P	Q	R	S	T	V	W	Y
50	ibalizumab	0.000095	0.000115	0.000014	0.000054	0.000061	0.000102	0.000048	0.000217	0.000115	0.000115	0.000020	0.000088
51	risankizumab	0.000096	0.000096	0.000007	0.000034	0.000076	0.000082	0.000055	0.000220	0.000124	0.000117	0.000021	0.000062
52	mosunetuzumab	0.000095	0.000123	0.000007	0.000055	0.000068	0.000095	0.000061	0.000211	0.000123	0.000109	0.000020	0.000061
53	ansuvimab	0.000083	0.000104	0.000007	0.000056	0.000076	0.000090	0.000049	0.000215	0.000118	0.000111	0.000014	0.000063
54	fremanezumab	0.000089	0.000124	NaN	0.000048	0.000082	0.000082	0.000055	0.000206	0.000124	0.000096	0.000014	0.000089
55	Obiltoxaximab	0.000082	0.000110	0.000007	0.000055	0.000069	0.000110	0.000062	0.000206	0.000124	0.000103	0.000021	0.000069
56	eptinezumab	0.000091	0.000112	NaN	0.000056	0.000070	0.000098	0.000035	0.000223	0.000126	0.000133	0.000014	0.000077
57	basiliximab	0.000097	0.000083	0.000028	0.000035	0.000069	0.000083	0.000042	0.000250	0.000125	0.000090	0.000021	0.000076
58	bezlotoxumab	0.000076	0.000117	NaN	0.000034	0.000076	0.000096	0.000062	0.000227	0.000124	0.000096	0.000021	0.000069
59	odesivimab	0.000096	0.000116	0.000007	0.000062	0.000075	0.000096	0.000041	0.000219	0.000116	0.000116	0.000021	0.000082
60	emapalumab	0.000076	0.000076	0.000014	0.000055	0.000103	0.000076	0.000041	0.000255	0.000145	0.000103	0.000028	0.000069
61	benralizumab	0.000089	0.000110	0.000007	0.000048	0.000075	0.000103	0.000041	0.000205	0.000137	0.000096	0.000014	0.000075
62	evinacumab	0.000096	0.000110	0.000007	0.000041	0.000068	0.000103	0.000048	0.000233	0.000123	0.000089	0.000021	0.000075
63	burosumab	0.000090	0.000111	NaN	0.000042	0.000076	0.000097	0.000042	0.000229	0.000118	0.000104	0.000014	0.000062
64	tezepelumab	0.000076	0.000083	NaN	0.000048	0.000111	0.000076	0.000035	0.000208	0.000131	0.000138	0.000035	0.000062
65	ixekizumab	0.000089	0.000123	0.000007	0.000048	0.000082	0.000089	0.000062	0.000219	0.000116	0.000123	0.000014	0.000062
66	erenumab	0.000096	0.000103	NaN	0.000062	0.000109	0.000075	0.000027	0.000205	0.000151	0.000109	0.000027	0.000062
67	durvalumab	0.000075	0.000123	NaN	0.000034	0.000082	0.000096	0.000068	0.000212	0.000116	0.000096	0.000021	0.000068
68	dostarlimab	0.000097	0.000111	NaN	0.000035	0.000076	0.000097	0.000035	0.000201	0.000139	0.000104	0.000028	0.000076
69	olaratumab	0.000075	0.000115	NaN	0.000048	0.000088	0.000095	0.000061	0.000211	0.000109	0.000095	0.000020	0.000061
70	tixagevimab	0.000075	0.000116	NaN	0.000034	0.000075	0.000089	0.000068	0.000225	0.000116	0.000095	0.000020	0.000068
71	loncastuximab	0.000083	0.000083	0.000028	0.000042	0.000076	0.000069	0.000055	0.000229	0.000125	0.000097	0.000021	0.000076

UNIVERSIDADE NOVA
DE LISBOA

2025

Knowledge transfer for biopharmaceutical production: data analysis of different process development campaigns

NUNO FRANCISCO PEREIRA DE BRAGANÇA
MASTER IN COMPUTATIONAL BIOLOGY & BIOINFORMATICS

UNIVERSIDADE NOVA DE LISBOA